次世代スパコンのための仮想計算機環境の研究開発

品川高廣

東京大学情報基盤センター

１．はじめに

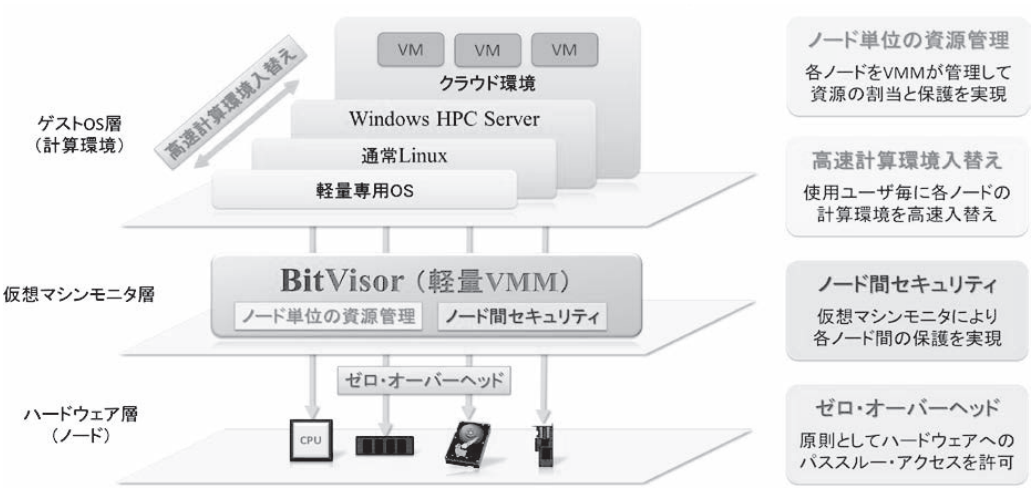
近年のスーパーコンピュータは多数のノードの集合体として構成されることが多い。例えば，「京」は 102 ノード×864 ラックで計 88,128 台のノードから構成されている。また，「TSUBAME2」の Thin ノードは 44 ラック 1,408 台のノードから構成されている。スーパーコンピュータ全体としての性能を向上させるためには，これらの個々のノードの性能を最大限に引き出すことが重要となっている。

一方で，スーパーコンピュータのユーザやそのアプリケーションはますます多様化してきており，要求される性能は数値計算の種類ごとに大きく性質が異なることが考えられる。例えば，従来のスーパーコンピュータで主流のアプリケーションである数値計算を考えてみても，CPU やメモリ，ディスク，ネットワークのいずれがボトルネックになるかは，数値計算の性質毎に異なることが考えられる。また，近年はスーパーコンピュータとクラウドの融合が進んでおり，Hadoop 等の新しい性質のアプリケーションのサポートや IaaS/PaaS といったサービスの提供をおこなっているところもある。

このような多様化した要求に答えるためには，スーパーコンピュータが単一の環境を提供しているだけでは難しくなってくる。従来のスーパーコンピュータでは Linux などのコモディティなオペレーティングシステムを一般的な設定で構成したものを提供していることが多く，ユーザが自由にオペレーティングシステムを変えることは難しい。従って，ユーザ毎の要求に応じてオペレーティングシステムの種類や構成を自由に変更出来るようにすることで，より柔軟なスーパーコンピュータを実現することができると考えられる。例えば，Linux の代わりに Windows HPC を使ったり，特殊なライブラリをインストールして使ったり，システムやオペレーティングシステムカーネルのパラメータをチューニングしたり，専用の超軽量オペレーティングシステムを用いることでオーバーヘッドを極限まで減らしたりといったことを実現することができるようになる。

しかしオペレーティングシステムの種類や構成の変更をユーザが自由におこなえるようにすると，ユーザ間でのセキュリティが問題となる。一般にオペレーティングシステムは全てのハードウェア資源にアクセス可能であるため，オペレーティングシステムを一般ユーザに開放してしまうと，他のユーザが利用するノードとの隔離や共有ストレージのセキュリティなどを確実に実現することが難しくなる。従って，各ノードの性能低下を最小限に抑えつつセキュリティを実現する仕組みが必要である。

本研究では，我々が研究開発してきた仮想マシンモニタ「BitVisor」を応用して，オペレーティングシステムへのオーバーヘッドを極めて低く抑えてスーパーコンピュータ環境でも十分な性能で利用できるようにしつつ，仮想マシンモニタにより必要最小限のセキュリティを確実に確保することにより，多様な用途に応じたノード性能の最適化とセキュリティの確保を両立することを目指す（図１）。目的・用途に応じたシステムの最適化を可能にするために，ノード



第 1 図: BitVisor を活用した仮想計算機環境。

スーパーコンピュータ環境において，高速計算環境入れ替えやノード間のセキュリティなどを実現するために，軽量な仮想マシンモニタ（VMM）である BitVisor を導入する。BitVisor は管理を行うための専用 VMM で，同時に動作する OS を 1 つに限定する代わりに，仮想化のオーバーヘッドを非常に低く押さえることが出来る。

の部分集合を仮想スーパーコンピュータとみなし，それらをユーザが直接制御できるようにする。一方で，システムのセキュリティを実現するために，仮想マシンモニタによって各ノードのハードウェアへのアクセスを必要最小限だけ監視してアクセス制御をおこなう。また，オペレーティングシステムを各ノードに高速にデプロイするための機能を実現し，仮想スーパーコンピュータの管理機能の枠組みを提供することを目指す。

2．オペレーティングシステム透過なデプロイ手法

今回のプロジェクトでは，主にオペレーティングシステムを各ノードに高速にデプロイするための機能に関する研究開発をおこなった。

スーパーコンピュータでユーザが自由にオペレーティングシステムを入れ替えて使用できるようにするためには，オペレーティングシステムのデプロイを容易に行える仕組みをサポートすることが重要になる。従来のオペレーティングシステムデプロイ手法では，デプロイに非常に時間がかかってしまったり，手法がオペレーティングシステムに依存してしまったりといった問題点があった。例えば，最も単純な手法では，オペレーティングシステムのイメージをサーバから各ノードのローカルディスクにコピーするという手法がある。この場合，ネットワーク帯域とイメージサイズにもよるが，数分から数十分の時間がかかってしまう。また，イメージコピーをした後に，サーバマシンを再起動する際にも更に数分の時間がかかってしまう。このようなオペレーティングシステムデプロイにかかる時間は，ユーザにとって待ち時間となってしまうため，非常に不便である。

OS streaming deployment [13]では，オペレーティングシステム自身に改変を加える事でオペレーティングシステムデプロイの待ち時間を削減することが可能である。この手法では，まずネットワークブートをおこない，その後オペレーティングシステムイメージをバックグラウンドでローカルディスクにコピーする。これにより，オペレーティングシステムの高速な起動

とオペレーティングシステムデプロイが終了した後のオーバーヘッドが低い計算環境を両立することが出来る。しかし，この手法ではオペレーティングシステムの機能や設定に大きく依存するため，オペレーティングシステム透過性が失われてしまう。オペレーティングシステム透過性が確保されていないと，管理者にとってのみならずユーザにとっても大きな制約となる。例えば，ユーザはオペレーティングシステムを更新したりする際に，専用のドライバの互換性をテストしたり検証したりする手間が必要となる。予め設定したオペレーティングシステムイメージを管理者が配る手法もあるが，ユーザが利用できるオペレーティングシステムに制限が生じてしまう。

仮想マシンモニタを使うことによって，オペレーティングシステム透過なオペレーティングシステムデプロイ手法を実現することは比較的容易に可能である。仮想マシンモニタは，オペレーティングシステムに依存せずにホストオペレーティングシステムを用いてバックグランドでイメージコピーをおこなうことが容易にできるため，高速なオペレーティングシステム起動と透過的なオペレーティングシステムデプロイを両立することが出来る。しかし，仮想マシンモニタでは一定のオーバーヘッドが避けられず，スーパーコンピュータでの利用においては問題となる。

本プロジェクトでは，オペレーティングシステム透過でかつオーバーヘッドの低い高速オペレーティングシステムデプロイシステムを実現する研究開発をおこなった。本システムでは，仮想マシンモニタのレベルで OS streaming deployment をおこなうことで，オペレーティングシステム透過なオペレーティングシステムデプロイを実現する。一方で，この仮想マシンモニタはオペレーティングシステムからハードウェアへのアクセスを仮想化せずになるべくパススルーするようにすることで，仮想化のオーバーヘッドを必要最小限に抑える。

３．関連研究

オペレーティングシステムを各ノードに高速にデプロイする手法としては，OpenStack Nova [12]のようなイメージコピーの技術を用いたものが考えられる。しかし，この手法はコピーにかかる時間が非常に長くなるという問題点がある。例えば，30 GB のディスクイメージ（Amazon EC2 における Windows Server 2008 のデフォルトイメージサイズ）を 130 MB/sec（15,000 rpm の SAS ディスクと 10 GB Ethernet を使用）で転送した場合，イメージコピーには約 5 分かかる。SSD を使うことでコピー時間は更に削減できるが，複数のノードに同時にデプロイする場合には，ネットワークやサーバの性能がネックとなる。イメージコピー後に必要となるリブート時間は，特にサーバマシンのようなファームウェアの初期化に時間がかかるマシンでは問題となる。Linux の Kickstart のようなネットワークインストールの手法では，オペレーティングシステムのファイルをネットワークからコピーすることにより，イメージコピーと似たような効果を得ることが出来る。しかし，この手法はオペレーティングシステム依存であるほか，やはりコピー操作に多くの時間がかかる。

ネットワークブートでは，オペレーティングシステムを速やかにブートすることが出来る。しかし，この手法ではオペレーティングシステムイメージをローカルディスクにコピーしないため，ディスク I/O が発生するたびにネットワークへリダイレクトするためのコストが常にかかり続ける。ネットワークブートとローカルディスクへのキャッシュを組み合わせた手法［14］では，I/O のオーバーヘッドを削減できるが，ディスクアクセスのたびにキャッシュの有効期

限をチェックする必要があり，I/O のレイテンシを削減することが難しくなる。

従来の仮想化技術をスーパーコンピュータに応用すると，仮想化によるオーバーヘッドが問題となる。例えば，Xen [1]や KVM [2]，VMWare [3]等の仮想マシンモニタを使用すると，仮想マシン間のスケジューリングやソフトウェアによるデバイスの仮想化などによって大きなオーバーヘッドが発生する。仮想化によるオーバーヘッドを削減する技術としては，para-virtualization [1]や I/O pass-through [4,5]などが挙げられるが，これらの技術を用いてもスーパーコンピュータで用いられるアプリケーションのように CPU や I/O を酷使する場合には，一定のオーバーヘッドが避けられないことが知られている [6,7,8]。このオーバーヘッドの原因としては，lock-holder preemption problem [9]やキャッシュ汚染，ネステッドページングや割り込み処理におけるオーバーヘッド[10,11]等が挙げられる。

仮想化のオーバーヘッドを無くす技術としては，オペレーティングシステムをデプロイするときだけ仮想化技術を使用して，必要がなくなったらアンインストールするという手法が考えられる。近年の仮想マシンモニタは raw disk や virtual-to-physical 変換をサポートしているため，オペレーティングシステムを導入した後に仮想インスタンスを物理インスタンスに変換することは可能である。しかし，仮想マシンモニタのアンインストールにはオペレーティングシステムの再起動が必要であり，ダウンタイムが生じてしまう。近年の研究では，オペレーティングシステムのハイバネーション機能を活用することで，よりシームレスな変換をおこなうことが可能であることが示されている [15]。しかし，この手法ではオペレーティングシステムに修正が必要であるほか，変換に 90 秒程度の時間がかかってしまう。

物理デバイスと全く同じ仕様の仮想デバイスを作成することで，オペレーティングシステム透過性を実現するという手法も理論的には考えられるが，チップセットや ACPI の機能など，すべての物理ハードウェアの機能を正確に再現するのは難しい。また，物理デバイスと仮想デバイスの内部状態を同期することは容易ではない。従って，仮想インスタンスを物理インスタンスにシームレスに変換することは難しい。

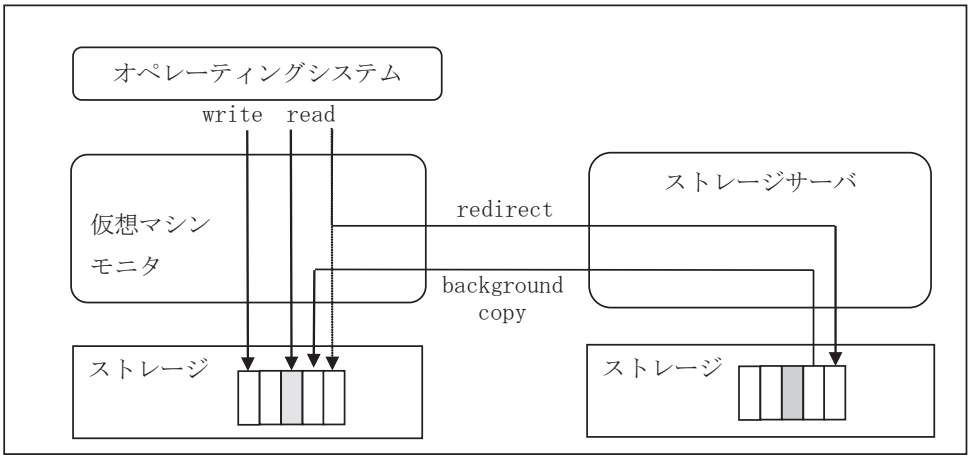
NoHype [16]は，ゲストオペレーティングシステムを起動した後に仮想化レイヤを削除することで，仮想マシンモニタへの攻撃を防止する手法である。Microvisor [17]は実行時に脱仮想化することでオンラインメンテナンスを容易にする手法である。また，パススルー方式を採用した仮想マシンモニタ [18, 19]でも脱仮想化を実現することが可能である。しかし，これらの手法では，仮想マシンモニタでオペレーティングシステムのデプロイを実現することは容易ではない。

４．設計

４．１．全体設計

本章では，本プロジェクトでの提案システムの設計について述べる。本システムでは，初期化フェーズとデプロイフェーズの２つに大きく分けられる。

初期化フェーズでは，仮想マシンモニタがまず対象ノードで起動する。仮想マシンモニタはローカルディスクや USB メモリから起動することもできるが，本システムでは管理の容易性を考えて仮想マシンモニタ自身をネットワークブートする。仮想マシンモニタ自身が高速に起動するようにするために，仮想マシンモニタのサイズをなるべく小さくするとともに，初期化のプロセスを並列化することで起動時間の最適化を図っている。仮想マシンモニタは，自身に割



第２図: BitVisor を用いたオペレーティングシステムのデプロイ方式。

オペレーティングシステムからのストレージの読み込み要求（read）は，その内容が既にローカルディスクに存在すれば，ローカルのストレージから読み込まれる（図の灰色のブロック）。そうでない場合は，BitVisor がストレージサーバに要求を redirect し，ストレージサーバから内容を読み込む。それとは別に，BitVisor がバックグラウンドでストレージサーバからローカルストレージへのコピーをおこない（background copy），ローカルストレージにオペレーティングシステムのイメージをコピーしていく。最終的には，全てのデータがローカルディスクから読み書きされるようになり，オーバーヘッドを削減できる。

り当てられた専用 NIC の初期化のみをおこない，その他のデバイスはゲストオペレーティングシステムに初期化をおこなわせるために，仮想マシンモニタでは未初期化のままにしておく。実際の起動時間は数秒程度である。このフェーズでは，ローカルストレージは空っぽのままでありオペレーティグシステムのイメージは含まれていない。

デプロイフェーズでは，仮想マシンモニタは主に２つの処理をおこなう。まず（1）読み込み時コピーによってサーバからローカルディスクにブロックの読み込みをおこなう。また，（2）バックグラウンドコピーによって，能動的にブロックを埋める処理をおこなう。読み込み時コピー処理によって，オペレーティングシステムがすみやかに起動することを可能にしつつ，バックグランドコピーによってディスク読み込みのオーバーヘッドを最終的に削減する。読み込み時コピー処理では，仮想マシンモニタはゲストオペレーティングシステムから要求されたディスクブロックが空だった場合にはサーバからデータを取得する。それから仮想マシンモニタは，サーバから受け取ったデータをあたかもローカルディスクから読み込んだかのようにしてゲストオペレーティングシステムに返す（第図２の redirect）。同時に仮想マシンモニタは，サーバから受け取ったデータを将来の使用のためにローカルディスクにも書き込んでおく。その他の場合は，仮想マシンモニタはゲストオペレーティングシステムからの I/O 要求をローカルディスクにパススルーして，ローカルディスクが直接その要求を処理できるようにする。

読み込み時コピーと平行して，仮想マシンモニタはバックグラウンドコピーによって能動的にからのディスクブロックを埋めていく処理をおこなう（第図２の background copy）。これによって，ローカルディスクは徐々にサーバのディスクイメージのデータで埋められていき，最終的にはゲストオペレーティングシステムはローカルディスクと同等の性能を得ることができるようになる。この処理では，ゲストオペレーティングシステムと仮想マシンモニタが同時に

ローカルディスクにアクセスすることになる。そこで，仮想マシンモニタの処理によってゲストオペレーティングシステムの性能が影響をうけることをなるべく避けるために，仮想マシンモニタはバックグラウンドコピーの速度を適切に調整する。

４．２．デバイスメディエータによる I/O 仲介

デバイスメディエータは提案方式を実現する上で鍵となるコンポーネントである。本方式では，オーバーヘッドを削減するためにゲストオペレーティングシステムに対して物理ハードウェアのインターフェイスを直接見せつつ，仮想マシンモニタがゲストオペレーティングシステムとデバイスを共有する必要がある。デバイスメディエータは，ゲストオペレーティングシステムと仮想マシンモニタからの I/O 要求を仲介する役割を果たす。I/O 仲介には３つの操作がある。I/O 解釈，I/O リダイレクション，I/O 多重化である。I/O 解釈ではゲストオペレーティングシステムからの一連の I/O 要求を観察して内容を解釈する。I/O リダイレクションでは，ゲストオペレーティングシステムからの I/O 要求を捕捉して，サーバへリダイレクトする。I/O 多重化では，仮想マシンモニタからの I/O 要求をゲストオペレーティングシステムからの I/O 要求の合間に挿入する。I/O 解釈は，I/O リダイレクションと I/O 多重化の基礎となる機構であり，I/O リダイレクションは読み込み時コピー，I/O 多重化はバックグラウンドコピーを実現するための機構である。

I/O 解釈：I/O 解釈では，ゲストオペレーティングシステムからの I/O アクセスを正確に制御するために，I/O のコンテキストを決定する。例えば，ディスクコントローラでは，コマンドやステータス，データといったコンテキスト情報を取得する。コマンドは，操作の種類（読み込みか書き込みか），論理ブロックアドレス，セクタ数などを含む。ステータスはデバイスがビジーかどうかなどの情報を含む。データは，実際に転送されるデータに関するものであり，DMA のバッファアドレスなどが含まれる。

デバイスメディエータは，これらの情報をプログラム I/O（PIO）やメモリマップド I/O（MMIO）などを監視しながら，デバイスの仕様やメモリ上のデータ構造などに基づいて解釈することで，これらの情報を取得することが出来る。これらの操作はデバイス依存であるが，多くのデバイスにおいて基本的な考え方は共通である。また，デバイスメディエータはデバイスドライバに比べて操作の内容が少ないため，完全なデバイスドライバよりも小さくシンプルに実現することが出来る。

I/O リダイレクション：I/O リダイレクションは，空のディスクブロックへのアクセスをサーバへのアクセスへリダイレクトするために使われる。デバイスメディエータは，(1)アクセスするブロックに関する情報の取得，(2)対応するデータをサーバから取得，(3)データをゲストオペレーティングシステムに渡す，といった処理をおこなう必要がある。

デバイスメディエータは，まず I/O 解釈の機構を用いてコマンドの内容を取得する。この内容に基づいて，仮想マシンモニタはブロックが空かどうかを判断する。もしブロックが空であれば，対応するデーバをサーバから取得する。このとき，ゲストオペレーティングシステムからデバイスへのアクセスを一時的にブロックする。その後，仮想マシンモニタはサーバから取得したデータをゲストオペレーティングシステムの DMA バッファにコピーする。

ここで，デバイスメディエータは割り込みを生成する必要がある。割り込みコントローラを仮想化すると仮想マシンモニタの構造が複雑になるため，提案手法では物理デバイスに対してダミーのリクエストを送ることにより，デバイス自身に割り込みを発生させることによって，

割り込みコントローラを仮想化することなく割り込みの挿入を可能にしている。

I/O 多重化：I/O 多重化は，バックグラウンドコピーに利用する。I/O 多重化を用いると，ゲストオペレーティングシステムがデバイスを制御しつつ，仮想マシンモニタもそのデバイスを使用することが可能になる。デバイスメディエータは，(1)I/O 要求を挿入する適切なタイミングを見つける，(2)デバイスのステータスをエミュレーションして要求を処理する，(3)リクエストキューを管理する，といった処理をおこなう必要がある。

デバイスメディエータは，まず I/O 要求をタイムシェアリング方式で挿入する適切なタイミングを探す。もしデバイスがゲストオペレーティングシステムからの要求を処理していた場合には，デバイスメディエータはその要求が処理されるまで待つ。処理が完了したら，デバイスメディエータは仮想マシンモニタからの I/O 要求をデバイスに送る。一貫性を維持するために，ゲストオペレーティングシステムからデバイスを見た時には，デバイスはビジー状態ではなくレディ状態であるように見せる必要がある。従って，ゲストオペレーティングシステムは新たに I/O 要求を発行してくる可能性がある。仮想マシンモニタは，この I/O 要求を一時的にキューに入れておき，仮想マシンモニタの I/O 処理が完了した後にこのキューに入っている要求をデバイスに送る。

４．３．バックグラウンドコピー

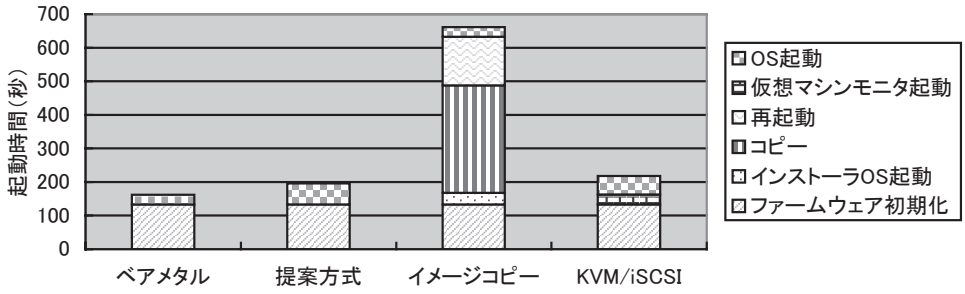
ローカルディスク全体をオペレーティングシステムのイメージで埋めるためには，仮想マシンモニタは能動的に空のディスクブロックに対応するデータをサーバからバックグラウンドで取得する必要がある。ディスク書き込みとネットワークからのデータ取得の速度は異なるので，仮想マシンモニタはそれぞれスレッドを用いてデータの処理をおこなう。取得スレッドはサーバからのデータの取得をおこない，書き込みスレッドがディスクへのデータ書き込みをおこなう。

ゲストオペレーティングシステムからのアクセスと平行して仮想マシンモニタがバックグラウンドでコピーをおこなうと一貫性の問題が生じる可能性がある。例えば，仮想マシンモニタがディスクブロックを埋めようとしている最中にゲストオペレーティングシステムがそのブロックに書き込みをおこない，その後サーバからのデーバを書き込んでしまうと，ゲストオペレーティングシステムから書き込まれたデータが消えてしまう。このような問題を回避するために，仮想マシンモニタはビットマップを管理してデータの一貫性が失われないようにする。

仮想マシンモニタは，ゲストオペレーティングシステムのディスク性能に影響を与えないようにするために，ゲストオペレーティングシステムの挙動を監視して，ゲストオペレーティングシステムが I/O 要求を頻繁に行っている時にはバックグラウンドコピーのスピードを落とすような処理をおこなう。

５．実験

本システムの性能を評価するための実験をおこなった。実験は全て東京大学の柏キャンパスに設置されたスーパーコンピュータ上でおこなった。このマシンは，富士通の PRIMAGY RX200 S6 で，それぞれ Intel Xeon X5680（3.33GHz，6 コア，ハイパースレッディング無効）を 2 個，96 GB のメモリ，Mellanox MT26428 InfiniBand カード（4X QDR），および Seagate Constellation.2 ST9500620NS（500GB，7200rpm）SATA ハードディスクを搭載している。また，Intel 82575EM ギガビットイーサネットを２つ搭載しており，１つを仮想マシンモニタ専用に割り当てた。こ

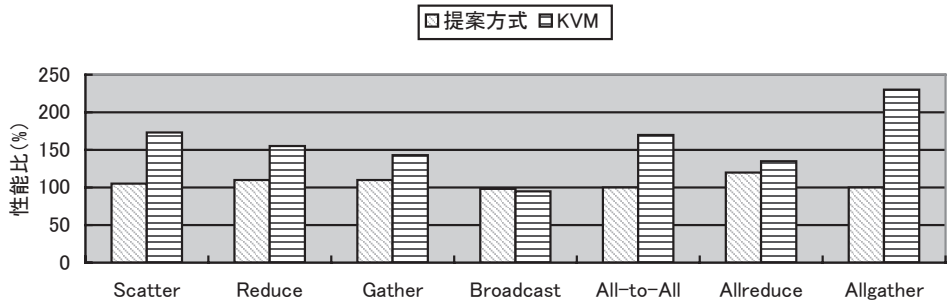


第３図: オペレーティングシステム起動時間。

れらのマシンは富士通 SR-S348TC1 ギガビットイーサネットスイッチと Mellanox Grid Director 4036E InfiniBand スイッチで接続されている。すべての実験において，Ubuntu 14.04 (Linux カーネル 3.13.0) を使用した。比較実験では，KVM (Linux カーネル 3.9.0) に Exit-less Interrupt(ELI)[11]のパッチを適用したものを使用した。

第３図にオペレーティングシステムの起動時間を測定した結果を示す。実験では，仮想マシンモニタの無いベアメタルマシン上の起動時間と提案方式での起動時間を測定した。また，比較のためにイメージコピー方式での総起動時間と KVM/iSCSI 上での起動時間も測定した。今回使用したマシン上でのファームウェア初期化には 133 秒かかった。ベアメタルマシン上では，オペレーティングシステムの起動に 29 秒かかった。提案方式では，63 秒かかっている。これは初回起動時にはディスクのデータをネットワークから取得する必要があるためである。一方，イメージコピー方式では 544 秒かかっている (インストーラの起動に 50 秒，ディスクイメージのコピーに 320 秒，再起動に 145 秒，ローカルディスクからのオペレーティングシステム起動に 29 秒)。従って，提案方式はイメージコピーと比べて 8.6 倍高速にオペレーティングシステムを起動することができている。KVM/iSCSI では，KVM 自身を起動するのに 30 秒かかっている。ゲストオペレーティングシステムの起動には 55 秒かかっており，提案方式は KVM/iSCSI と同等の起動性能を実現できていることが分かる。なお，KVM では，OS 稼働中に定常的に仮想化のオーバーヘッドがかかるのに対し，提案方式では仮想化のオーバーヘッドは非常に低い。

第４図に MPI ベンチマークの結果を示す。実験では，提案方式と KVM 上のそれぞれにおいて OSU Micro-Benchmarks を走らせて，MPI 通信のレイテンシを測定した。この実験は，InfiniBand で接続された富士通 PRIMAGY RX200 10 台のマシン環境を用いておこなった。OS は MPICH2 を使うように設定されている。第４図の結果の数値は，ベアメタルマシン上で実行した場合の結果との比であらわしている。図からわかるように，提案方式ではベアメタルマシン上に非常に近い MPI 性能が実現できている。一方，KVM 上では比較的大きなオーバーヘッドが発生している。例えば、Allgather の実験では，KVM のレイテンシはベアメタルと比べて 235%増加しているのに対し，提案方式ではほぼゼロオーバーヘッドが達成できている。Allreduce の実験では，提案方式でも 22%ほどのオーバーヘッドがかかっている。これは，InfiniBand のレイテンシのオーバーヘッドによるものと考えられる。しかし，全体としては，提案方式は非常にオーバーヘッドの低い方式を実現できていると言える。



第４図: MPI ベンチマーク。

６．まとめ

本プロジェクトでは，次世代スパコンのための仮想計算機環境の実現に向けて，オペレーティングシステムの入れ替えを高速に行えるようにするためのデプロイシステムの研究開発をおこなった。軽量な仮想マシンモニタを用いてオペレーティングシステムのデプロイを高速に行えるようにすることで，スパコン環境においてユーザが自由にオペレーティングシステムを入れ替えて計算を行いやすくすることを目指した。実験の結果，仮想化のオーバーヘッドは従来方式に比べて大幅に削減できることがわかった。

参　考　文　献

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In Proceedings of the 19th ACM Symposium on Operating Systems Principles, SOSP ’03, pages 164-177, Oct. 2003.

[2] A. Kivity. kvm: the Linux Virtual Machine Monitor. In Proceedings of the 2007 Ottawa Linux Symposium, pages 225-230, July 2007.

[3] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. 2001. Virtualizing I/O Devices on VMware Workstation’s Hosted Virtual Machine Monitor. In Proceedings of the General Track: 2001 USENIX Annual Technical Conference, Yoonho Park (Ed.). USENIX Association, Berkeley, CA, USA, 1-14.

[4] H. Raj and K. Schwan. High Performance and Scalable I/O Virtualization via Self-Virtualized Devices. In Proceedings of the 16th International Symposium on High Performance Distributed Computing, HPDC ’07, pages 179-188, June 2007.

[5] Paul Willmann, Jeffrey Shafer, David Carr, Aravind Menon, Scott Rixner, Alan L. Cox, and Willy Zwaenepoel. Concurrent Direct Network Access for Virtual Machine Monitors. In Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture, HPCA ’07, pages 306-317, Feb. 2007.

[6] S. K. Barker and P. Shenoy. Empirical Evaluation of Latency-sensitive Application Performance in the Cloud. In Proceedings of the 1st annual ACM SIGMM Conference on

Multimedia Systems, MMSys ’10, pages 35-46, Feb. 2010.
[7] Alexandru Iosup, Simon Ostermann, M. Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick H.J. Epema. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. IEEE Transactions on Parallel and Distributed Systems, 22(6):931-945, June 2011.
[8] Vlad Nae, Radu Prodan, Thomas Fahringer, and Alexandru Iosup. The Impact of Virtualization on the Performance of Massively Multiplayer Online Games. In Proceedings of the 8th Annual Workshop on Network and Systems Support for Games, NetGames ’09, page 9, Nov. 2009.
[9] V. Uhlig, J. LeVasseur, E. Skoglund, and U. Dannowski. Towards Scalable Multiprocessor Virtual Machines. In Proceedings of the 3rd Virtual Machine Research and Technology Symposium, pages 43 - 56, May 2004.
[10] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne. Accelerating Two-Dimensional Page Walks for Virtualized Systems. In Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XIII, pages 26-35, Mar. 2008.
[11] Abel Gordon1, Nadav Amit, Nadav Har’El, Muli Ben-Yehuda, Alex Landau1, Assaf Schuster, and Dan Tsafrir. ELI: Bare-metal Performance for I/O Virtualization. In Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XVII, pages 411- 422, Mar. 2012.
[12] Openstack. http://www.openstack.org/.
[13] C. David, G.-E. Luis, and R. Sean. OS Streaming Deployment. In Proceedings of the 29th IEEE International Performance Computing and Communications Conference (IPCCC), pages 169 -179, Dec. 2010.
[14] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam. The Collective: A Cache-Based System Management Architecture. In Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI), pages 259-272, May 2005.
[15] Thawan Kooburat and Michael Swift. The Best of Both Worlds with OnDemand Virtualization. In Proceedings of the 13th Workshop on Hot Topics in Operating Systems, HotOS XIII, May 2011.
[16] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. NoHype: Virtualized Cloud Infrastructure without the Virtualization. In Proceedings of the 37th Annual International Symposium on Computer Architecture, ISCA ’10, pages 350-361, June 2010.
[17] D. E. Lowell, Y. Saito, and E. J. Samberg. Devirtualizable Virtual Machines Enabling General, Single-node, Online Maintenance. In Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XI, pages 211-223, Oct. 2004.
[18] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes. In Proceedings of the 21st ACM Symposium on Operating Systems Principles, SOSP ’07, pages 335-350, Oct. 2007.

[19] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo, and Kazuhiko Kato. BitVisor: A Thin Hypervisor for Enforcing I/O Device Security. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '09, pages 121-130, Mar. 2009.