13-IS04

次世代スーパーコンピュータ向けの軽量な仮想計算機環境の実現に向けた研究開発

品川高廣（東京大学）

概要　我々が研究開発している軽量な仮想マシンモニタ「BitVisor」を応用して、次世代スーパーコンピュータのため軽量な仮想計算環境の実現に向けた研究開発をおこなう。近年のスパコンは多数ノードから構成されていることを活用し、ユーザに対してノード単位での割り当てをおこなうことにより、それぞれのノードのハードウェアはユーザが直接制御できるようにする。一方、各ノードのハードウェアへのアクセスは、仮想マシンモニタによって必要最小限の監視によりアクセス制御をおこなうことで、ノードのセキュリティ確保と多様な用途に応じたノード性能の最適化を両立することを目指す。今年度は、実際のスパコン環境において仮想マシンモニタのオーバーヘッドを最小限に抑えるための最適化を実現する研究開発をおこなった。その結果、オーバーヘッドの要因となっている箇所を 3 箇所突き止めて大幅に性能改善をおこなったほか、残存オーバーヘッドの要因に関する分析もおこなった。

1. 研究の目的と意義

我々が研究開発してきた軽量仮想マシンモニタ「BitVisor」を応用して、次世代スーパーコンピュータのための軽量な仮想計算環境の実現に向けた研究開発をおこなう。

近年のスパコンは、多数のノードから構成される集合体として実現されており、これら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、スパコンのユーザやその用途はますます多様化してきており、単一のシステムで全ての要望を満たすことは難しくなりつつある。

例えば、現在のスパコンでは OS として Linux のみならず各種 UNIX や Windows も使われており、ユーザや用途によって最適な OS を使い分けたいなどの要望が増加することが考えられる。また、次世代スパコンではノード数が更に増加すると予想され、既存の汎用 OS ではスケールさせることが難しくなりつつある。また、用途別に特化した専用の軽量 OS を使いたいといった要望も出てくることが予想される。

しかし、ユーザが OS を自由に選択できるようにすると、ユーザ間でのセキュリティが問題となる。一般に OS は全てのハードウェア資源にアクセスすることが可能であり、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施することが難しくなる。また、将来的に一般企業を含む様々なユーザにスパコンを開放して有効活用できるようにすることなどを想定すると、必ずしも信頼できるユーザだけを想定することは出来なくなるため、システムとして確実なセキュリティを実現することが重要な課題となってくる。

複数の OS・環境を安全に使い分ける手法としては、仮想化技術が近年注目されており、実際にスパコンで仮想マシンモニタを導入する動きもある。しかし、従来の仮想マシンモニタでは、仮想化によるオーバーヘッドが大きく、スパコンにおけるノード性能を最大限に引き出すという目的には必ずしも適しているとは言えない。一方で、マルチブートなど複数の OS を切り替えて利用する場合、特にユーザが専用 OS を用意する場合などにおいては、ノードの全権限がユーザに渡されることになり、セキュリティを確保することが難しくなってしまう懸念がある。

本研究では、代表者が研究開発してきた仮想マシンモニタ「BitVisor」を応用して、OS に対するオーバーヘッドを極めて低く抑えつつ、必要最小限のセキュリティを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。

近年のスパコンは多数のノードから構成されていることを活用して、一台のノードに複数の仮想マシンを構築するのではなく、ノードの集合の一部を切り出して仮想スパコンとしてユーザに割り当てることにより、それぞれのノードのハードウェアはユーザが直接制御できるようにする。これにより、目的・用途に応じたノードの最適化を可能にする。

一方で、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。これにより、性能とセキュリティの両立を実現する。また、ノードの集合である仮想スパコンの管理機能や独自 OS や大容量のデータを各ノードに高速にデプロイするための枠組みも提供することを目指す。

今回の課題では、上記の環境の実現に向けた最初のステップとして、実際のスパコン環境においてBitVisor を動作させ、ハードウェアの制御などを試みることで、提案方式の実現可能性を探る。本研究で目指す方式が実現されることにより、ユーザの用途に応じて様々な OS を選択することが安全に実現できるようになる。これにより、計算目的や性質に応じて最も適切な OS・システムソフトウェアを利用することが可能になり、スパコンの性能を最大限に引き出すことが可能になる。

例えば、MPI 通信のオーバーヘッドやジッタを最小限に抑えたい場合は、デーモンを可能な限り止めた OS や通信パラメータをチューニングしたカーネルなどを自由に使えるようになる。一方で、Hadoop 環境や IaaS/Paas 等のホスティングサービスを提供する場合には、既存の VMM と組み合わせて複数 OS を同時に稼働させることも可能になる。

どのノードをどの目的で用いるかといった設定も短時間で変更できるため、一時的に全ノードを使った大規模計算をゼロオーバーヘッドで実行したり、フラッシュクラウドが発生した際にクラウドサービスを提供するノード数を増加させるといった従来の VMM で提供される機能とを併存させたりすることが可能になる。

2. 当拠点公募型共同研究として実施した意義

(1) 共同研究を実施した拠点名および役割分担

東京大学

(2) 共同研究分野

超大規模情報システム関連研究分野

(3) 当公募型共同研究ならではという事項など

実際のスパコンを活用した研究開発や、スパコン利用者の視点が得られる点。

3. 研究成果の詳細と当初計画の達成状況

(1) 研究成果の詳細について

今回の課題では、まず実際のスパコン環境においてBitVisor を動作させ、基本機能が正しく動作することを確認した。その後、性能改善のために今までの BitVisor のオーバーヘッドの分析を詳細におこなって、スパコン環境での利用に耐えられるようにオーバーヘッドを極限まで削減するための詳細な分析と最適化をおこなった。

以下、3.1 節では、まずBitVisor の概要について説明し、BitVisor を動作させることで生じるオーバーヘッドの要因について説明する。また、オーバーヘッドの詳細なブレークダウンの分析をおこなって、タイマーアクセス、タイマースレッド機能、外部割り込みによってオーバーヘッドが発生することを特定したこと、及びそれを改善するための実装について説明する。また、これらの改善によって実際にオーバーヘッドが改善したことを示す実験結果を示す。

3.2 節では、実際に学際大規模情報基盤共同利用・共同研究拠点で用意されているマシンを利用してオーバーヘッドに関する実験をおこない、さらなる改善の余地を示すための分析結果について示す。

3.1 BitVisor 概要

BitVisorは準パススルー型という特徴的なアーキテクチャを持った仮想マシンモニタ（ハイパーバイザ）である。通常の仮想マシンモニタは、完全仮想化もしくは準仮想化というアーキテクチャになっており、ゲスト OS から見えるハードウェア

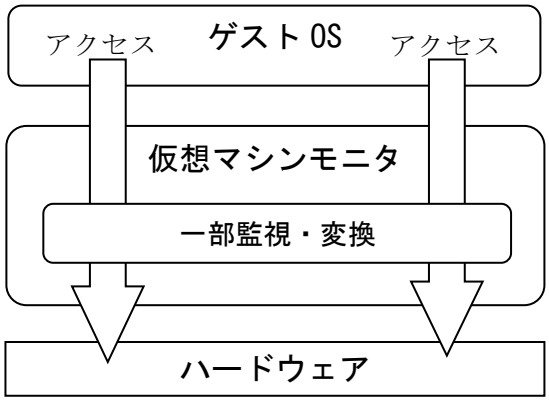

図１　準パススルー型アーキテクチャ

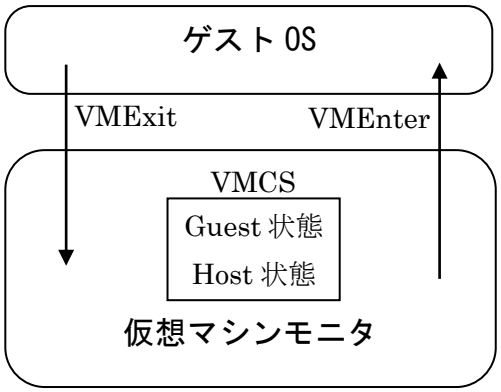

図２　VMExit と VMEnter

は原則として仮想マシンモニタによって仮想化された仮想デバイスになっている。準仮想化では、更にゲスト OS を改変することにより、現実には存在しない架空のデバイスを見せることによって、オーバーヘッドを削減することを実現している。一方、準パススルー型アーキテクチャでは、ゲスト OS から見えるハードウェアは原則として物理マシンと同じもの(物理デバイス)となっている。すなわち、ゲスト OS からハードウェアへのアクセスは原則としてパススルーとなっており、ゲスト OS は物理的なハードウェアを直接制御することが出来る。一方で、BitVisor で実現したい処理に最低限必要なアクセスだけは確実に捕捉して、必要に応じてアクセスをブロックしたり内容を変換したりすることで、セキュリティを始めとした様々な機能を実現する仕組みになっている。

図１に準パススルー型アーキテクチャの概要を示す。ゲスト OS からハードウェアへのアクセス（矢印）は、原則としてパススルーとし、仮想マシンモニタは何もしない。一方で最低限必要なアクセスは監視したり、捕捉・変換したりすることで必要な機能の実現をおこなう。例えば、ディスク I/O の一部を監視して特定の領域へのアクセスを禁止したり、ディスクの内容を暗号化したりするといった機能を実現することが可能である。

BitVisor では、CPU の仮想化を実現するために、ハードウェアの仮想化支援機構を活用している。具体的には、Intel 社の VT-x や AMD 社の AMD-V を活用している。これらのハードウェア仮想化支援機構では、CPU の状態のうちゲスト OS から仮想化してみせるべき内容にアクセスされた場合に、制御をゲスト OS から仮想マシンモニタに移行するためのメカニズムが備わっている。ここでは、このメカニズムを VMExit と呼ぶ。すなわち VM を Exit して仮想マシンモニタに制御が移るという意味である。一方、仮想マシンモニタから OS に制御が移行することを VMEnter と呼ぶ。

VMExit は仮想化を実現する上で必要不可欠なものであるが、ゲスト OS と仮想マシンモニタとの間でコンテキストスイッチが発生するため、VMExit が頻繁におこなわれると仮想化のオーバーヘッドが大きくなる。したがって、仮想マシンモニタのオーバーヘッドを低減するためには、なるべく VMExit が発生する頻度を減らすことが重要である。

また、VMExit が発生した際には、仮想マシンモニタは仮想化のために必要な様々な処理をおこなう。例えば、CPU のレジスタの値を実際の値とは違う値にゲスト OS に見せるための処理をおこなったり、仮想マシンモニタ内のスレッド処理を実行したりする。このような仮想マシンモニタ内の処理にかかる時間はオーバーヘッドに直結するため、これらの処理はできるだけ短い時間で実行することが重要である。

図２に VMExit と VMEnter の関係を図示したものを示す。VMExit はゲスト OS から仮想マシンモニタに制御が移ることであり、VMEnter は仮想マシ

ンモニタからゲスト OS に制御が移ることである。これらの制御移行時には CPU のコンテキストが切り替わるが、それぞれのコンテキストの情報は Intel VT-x の場合、VMCS(Virtual Machine Control Structure)と呼ばれるメモリ上のデータ領域に保存されている。図２の VMCS には、Guest 状態と Host 状態の 2 つの領域が示されている。ゲスト OS の CPU 状態は Guest 状態に保存され、仮想マシンモニタの CPU 状態は Host 状態に保存される。VMExit や VMEnter でコンテキストスイッチが切り替わる際には、CPU の内部状態がここに保存されたりここから復元されたりする。

3.1.1 タイマーアクセスのオーバーヘッド削減

BitVisor はタイマーにアクセスする機能を有している。これは、定期的に実行するスレッドなどを実現するために、前回実行時との時間の差分を取得する必要があるためである。

今までの BitVisor では、現在時刻を取得するために ACPI (Advanced Configuration and Power Interface)と呼ばれる規格で定められている手法を用いていた。ACPI とは、PC の電源制御や構成要素に関する情報を管理するために定められた統一規格であり、この規格に則ったマシンであれば統一されたやり方でハードウェアにアクセスすることが出来る。

ACPI で時刻取得をおこなうためには、PMT (Power Management Timer)というデバイスへアクセスする。ACPI PMT とは、ACPI 対応のマザーボードのチップセットに搭載されている 3,579,545Hz で動作する 32bit のタイマーである。このタイマーの値を読み出すことで、前回の読み出し時との時間の差分を比較的正確に取得することが出来る。

ACPI PMT の読み出しは、I/O ポートへのアクセスによっておこなう。I/O ポートのアドレスは、ACPI で定義されている FADT (Fixed ACPI Description Table)というメモリ上のテーブル状のデータ構造に記述されており、この値を用いることでタイマーの値にアクセスすることが出来る。

ACPI PMT の読み出しは、I/O ポートを経由しておこなうため、通常のメモリ読み出しに比べると読み出しにかかる時間が大きくなるという問題がある。今までの BitVisor では、VMExit するたびに ACPI PMT にアクセスしていたため、VMExit が頻繁に発生するようなワークロードをゲスト OS が実行していた場合、仮想マシンモニタによる累積のオーバーヘッドが大きくなって、ゲスト OS の性能に大きな影響を及ぼしていることが分かった。

ACPI PMT の代わりとなるタイマーとしては、いくつか選択肢がある。例えば、古典的な PIT (Programmable Interval Timer)や、Local APIC (Advanced Programmable Interrupt Controller)、HPET (High Precision Event Timer)、TSC (Time Stamp Counter)などがある。これらのうち、PIT や Local APIC, HPET は I/O ポートや MMIO (Memory Mapped I/O)でアクセスするタイマーであるため、タイマーへのアクセスに一定の時間がかかる。一方、TSC は CPU の内部のレジスタとして存在しており、非常に高速にアクセスすることが可能である。

TSC は CPU のクロックに同期してカウントアップされるタイマーである。そのため、CPU の動作周波数によってカウントアップされる間隔が変わってくることになる。一般的には、起動時に別のタイマーを利用することで、TSC がカウントアップされる周波数を測定して、それを元にカウンターの値を秒に変換して時間を取得する。

しかし、近年の CPU では、SpeedStep や TurboBoost など実行時にクロック周波数が動的に変動する技術が一般的に用いられている。これらの技術が導入された当初には、TSC の周波数が変動するかどうかは定義されておらず、CPU によっては CPU のクロック周波数が低下すると連動して TSC の周波数も低下するものが存在している。

この場合、TSC がカウントアップされる周期が実行中に変化してしまうため、正確な時間の測定には用いることが難しくなってしまう。特にノート PC などでは実際に頻繁に周波数が変化するため、TSC のサイクル数の差が同じであっても実際の時間の差が著しく大きくなる現象が確認されている。そのため、今までの BitVisor では TSC は使

用せず、ACPI PMT を利用してきた。

しかし、最新の CPU では、Invariant TSC という機能がサポートされるようになってきている。Invariant TSC では、CPU のクロック周波数の変動には影響されず、カウンターが常に一定の周期でカウントアップし続けられる機能である。この機能は CPUID の 80000007H:EDX[8]をチェックすることでサポートされているかどうかを確認できる。

そこで、最新の BitVisor では、CPUID の値を確認して、Invariant TSC が利用可能である場合には、TSC をタイマーの値として用いる実装をおこなった。TSC は ns 単位での読み出しが可能な非常に高速なタイマーであり、VMExit のたびに呼び出してもオーバーヘッドを非常に低く抑えられるようになった。

3.1.2 タイマースレッド機能のオーバーヘッド削減

BitVisor には、内部にスレッド機能が搭載されている。また、このスレッド機能を用いて定期的な作業を実施するためのタイマースレッド機能が搭載されている。これは、主にセキュア VM の機能を実現するために用意されている。

セキュア VM とは、ストレージやネットワークの暗号化をゲスト OS から透過的におこなう機能である。これを実現するためには、ストレージやネットワークへのアクセスをハードウェアのレベルで捕捉・変換する必要がある。例えば、ストレージについては、内蔵 HDD/SSD であれば ATA や AHCI (Advanced Host Controller Interface)、USB であれば UHCI (Universal Host Controller Interface)や EHCI (Enhanced Host Controller Interface)、ネットワークであれば Intel Pro1000 等のホストコントローラのインターフェイスのレベルで監視することで実現されている。

これらのインターフェイスの中には、ゲスト OS からハードウェアへのアクセスを仮想マシンモニタで直接的に捉えることが難しい仕様のものがある。具体的には、メモリ上にディスクリプタと呼ばれるデータ構造を配置し、この構造をゲスト OS のドライバとハードウェアのホストコントローラでやりとりする仕様の場合、メモリ構造へのアクセスを補足するためには、そのメモリ構造を含むページ全体へのアクセスを補足する必要があり、オーバーヘッドが著しく大きくなる。

そこで、仮想マシンモニタ内のタイマースレッドを活用することにより、定期的にポーリングをおこなって変更点を検出することにより、ハードウェアへのアクセスを捕捉する手法を実現している。

しかし、セキュア VM の機能を実現しない場合には、スレッド機能は必ずしも必要とはしない。また、本研究においてアクセス制御を実現する場合においても、ハードウェアのインターフェイスによっては、スレッド機能を使用しなくてもハードウェアへのアクセスを捕捉することが可能であり、その場合にはタイマースレッド機能は必ずしも必要ではない。

今までの BitVisor の実装では、タイマースレッドを使用するかどうかにかかわらず、常にタイマースレッドが動作し続けている状態であった。タイマースレッドの実装では、リスト構造とロック機構を使った比較的単純な実装であったため、何もしないタイマースレッドを一回動作させるだけでも比較的大きなオーバーヘッドが発生していることがわかった。

そこで、最新の BitVisor では、タイマースレッド機能は必要になるまでは起動しない状態になるように設定できるように実装をおこない、通常時にはオーバーヘッドが発生しないようにした。

今後の課題としては、タイマー機能のデータ構造とロック機構を改良して、低オーバーヘッドのタイマー機構を実装する必要があるとも思われる。

3.1.3 外部割り込みによるオーバーヘッドの削減

ゲスト OS から仮想マシンモニタに制御が移る VMExit の主な要因の一つに外部割り込みがある。外部割り込みとは、ハードウェアデバイスが発生させる割り込みのことで、ディスクの転送終了やネットワークのパケット到着などハードウェアデバイスに関連したイベントを通知するために使用されている。

従来の仮想マシンモニタでは、ハードウェアデバイスは通常は仮想マシンモニタ自身に内蔵されているデバイスドライバによって制御をおこなう。したがって、外部割り込みが発生した場合には仮想マシンモニタが制御に移るようにすることが通常である。特定のデバイスに対して仮想デバイスを用いずに、ゲスト OS から直接アクセス可能にするパススルー構成にした場合であっても、外部割り込み自体は仮想化する必要がある。これは、割り込みコントローラー自体は依然として仮想マシンモニタによって制御されているためであり、従来の仮想マシンモニタでは割り込みコントローラーをパススルーにすることは出来ない。そのため、外部割り込みは一旦仮想マシンモニタで受けたあと、仮想的に割り込みを発生させる割り込みインジェクションという操作をおこなうことで実現している。

一方、BitVisor では必ずしも外部割り込みで VMExit を発生させる必要はない。まず、BitVisor では割り込みコントローラーを仮想化していないため、ゲスト OS が直接割り込みコントローラーを制御している。そのため、外部割り込みを捕捉せずにそのままゲスト OS に送ったとしても、その割り込みはゲスト OS によって正しく処理させることが可能である。

しかし今までの BitVisor では、外部割り込みが発生した場合には必ず VMExit を発生させる実装になっていた。これは、前述のスレッド機能を実現するためには、定期的に仮想マシンモニタに制御がわたる必要があるためである。仮に外部割り込みで VMExit が発生しないようにすると、仮想マシンモニタに制御が戻るまでのタイミングは、ゲスト OS の挙動に著しく依存することになり、場合によっては秒単位で仮想マシンモニタに制御が戻ってこない可能性がある。

そこで、最新の BitVisor では、最新の CPU に搭載されている機能である Preemption Timer という機能を活用できる実装を追加した。Preemption Timer では、CPU に対して指定した時間で強制的に VMExit するよう指定することができるようになる機能である。これにより、外部割り込みのたびに割り込みを発生させる必要はなくして、VMExit の頻度を減らしつつ、なおかつ指定した一定の間隔で確実に VMExit を発生させることが可能になった。

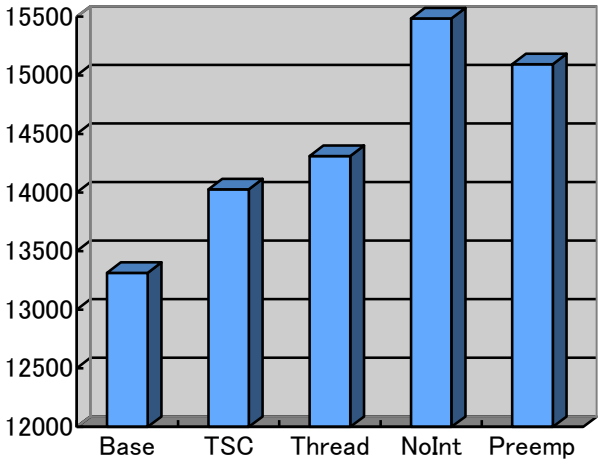

図３　Netperf の性能（Trans. per sec.）

3.1.4 オーバーヘッド削減効果

図３に、3.1.1 項から 3.1.3 項までのオーバーヘッド削減手法を適用した場合の性能実験の結果を示す。実験は、10Gbit Ethernet で接続された 2 台のマシンで”Netperf -t TCP_RR”を実行し、TCP における Transaction per second を測定した。グラフの値は左から、最適化前の BitVisor の場合（Base）、ACPI PMT タイマーの代わりに TSC を用いた場合（TSC）、タイマースレッドを停止した場合（Thread）、外部割り込みによる VMExit を停止した場合（NoInt）、NoInt に加えて Preemption Timer による定期的なスレッド実行を追加した場合を示す。

実験結果から、それぞれの最適化により、5%～16%まで性能が向上していることがわかる。これにより、3.1.1 項から 3.1.3 項までのオーバーヘッド削減手法が有効に機能していることが分かる。

3.2 残存オーバーヘッドの分析

3.1 節の最適化で BitVisor のオーバーヘッドは削減されたが、依然として BitVisor がない状態と比べるとオーバーヘッドが残っている。特に InfiniBand 等の広帯域ネットワークの性能を計測

すると、比較的大きなオーバーヘッドが発生することがわかった。そこで、このオーバーヘッドの要因を調べる分析をおこなった。

その結果、second level paging と呼ばれる CPU の機能により比較的大きなオーバーヘッドが生じていることが分かった。Second level paging とは、Intel では EPT (Extended Page Table)、AMD では NPT (Nested Page Table) と呼ばれている機能で、ゲスト OS のページングにより仮想アドレスからゲスト物理アドレスに変換されたものを、更に仮想マシンモニタが管理するページテーブルを利用して、CPU の機能によりゲスト物理アドレスをマシン物理アドレスに変換する機能である。

Second level paging は CPU のハードウェアによってアドレス変換がおこなわれるため、一般的にはソフトウェアのみで実現する shadow paging よりはオーバーヘッドを低く抑えられる。しかし、second level paging でも、CPU の性能によってアドレス変換のオーバーヘッドは異なる。特に CPU の世代によってオーバーヘッドが大きく異なっており、最新の CPU ではよりオーバーヘッドが少なくなっている。

また、BitVisor では、second level paging のページテーブルに用いるエントリの数を限定している。これは、second level paging に使用されるメモリ量を制限するためである。したがって、エントリ数が足りなくなると、古いエントリを再利用することで対応しているが、十分なエントリ数がないとページフォルトが頻繁に発生して、オーバーヘッドが大きくなる可能性がある。

これらの影響を調べるために、second level paging の構成を変更した時のネットワーク性能を調べる実験をおこなった。InfiniBand で接続された JHPCN の 2 台のマシンを使用して、netperf によるスループットを測定した。図 4 に実験結果を示す。グラフの値は左から通常の BitVisor (1K)、second level paging のエントリ数を 8K に増やしたもの (8K)、同じくエントリ数を 16K に増やしたもの (16K)、seond level paging を無効にしたもの (NoPG)、BitVisor を動作させないベアメタルマシン (Bare) である。

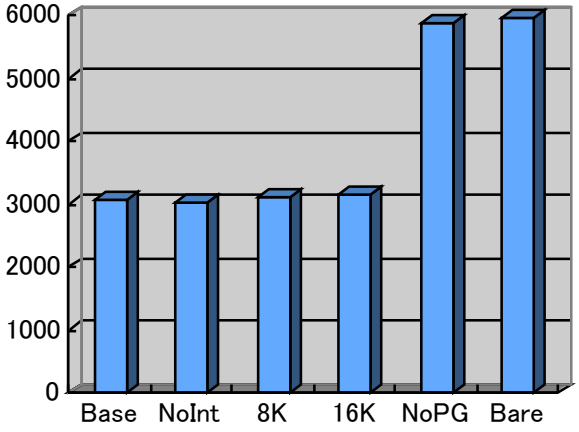

図 4　Second level paging の影響

グラフから、second level paging のエントリ数は、性能改善に若干の効果はあるものの、全体としてはあまり影響が大きくないことがわかる。したがって、エントリ数を制限することはオーバーヘッドの主要な要因ではないと考えられる。一方で、second level paging を完全に OFF にすると、性能は倍近くにまで向上する。これは、second level paging 自体のオーバーヘッドが依然として大きいことを示唆している。

この原因の一つとしては、CPU の世代が少し古いことがあげられる。今回の実験で使用した機材は、Intel Xeon X5680 (3.33GHz)を 2 台搭載したマシンであるが、これは Westmere と呼ばれる世代の CPU で、最新の CPU と比べると数世代前の CPU である。したがって、CPU のハードウェアによる second level paging による 2 段階の page walk の性能があまり高くないことが原因として考えられる。

これまでの研究開発により、今までの BitVisor をスパコン環境に適用する上でネックとなっていたオーバーヘッドの一部について改善することが出来た。具体的には 3.1.1 項から 3.3.3 項までの最適化により 5%〜16%まで性能が向上した。一方で、実際のスパコン環境に適用するためには、InfiniBand を始めとする 1Gbps を超えるネットワークが接続された環境にも絶える必要があると考

えられ、この点ではまだまだ改善の余地があることがわかった。具体的には、InfiniBand のスループットが半減してしまっており、Second level paging に関する仮想化によるオーバーヘッドが無視できないほど大きくなってしまっていることがわかった。

その後、新たに入手したマシン上での実験をおこなった結果、最新の CPU では Second level paging のオーバーヘッドが大幅に低減することが分かった。また、残存するオーバーヘッドについても分析を進めた結果、Second level paging 以外にもオーバーヘッドの要因が少し残っていることが判明した。具体的には、MTRR (Memory Type Range Registers や PAT(Page Attribute Type) MSR にアクセスした時に発生する VMExit の削減や、BitVisor 内部において CPUID を発行する回数の削減、BitVisor 内部におけるメインループ内における余分な分岐処理の削減や分岐予測の活用、switch 文の代わりに関数配列を使用するなどの最適化を施した結果、Unixbench などを用いたベンチマーク結果が若干改善されることを確認した。

また、当初計画でおこなう予定であった InfiniBand や SAS などのハードウェアのアクセス制御に向けた研究開発にも着手した。SAS に関しては、MegaRAID の仕様を調査するなどして、I/O アクセスの基本的な補足を行うことが出来るようになった。

(2) 当初計画の達成状況について

当初計画で第一段階として実施予定であった、Fujitsu PRIMARGY RX200 を用いた BitVisor のコア機能の動作確認や、Intel X5670 に搭載されている VT-x を始めとしたハードウェア仮想化支援機構を正しく利用できることの確認は予定通り実施して達成できた。また、性能評価をおこなってコア機能のオーバーヘッドの測定や性能向上に向けた最適化もおこなって、現時点では可能な限りの最適化を実施することできた。一方で、InfiniBand や SAS のアクセス制御については、状態遷移の把握までは実現したものの、実際にアクセス制御を実現するところにまでは至っていない。

4. 今後の展望

今回の研究課題によって、スパコン向け仮想計算環境の実現に向けた最初のステップはクリアしたため、今後は実際にカスタマイズした OS や軽量 OS を用いて、ユーザの用途に応じて様々な OS を選択することが安全に実現できるようになることの意義を明確にしていきたいと考えている。

5. 研究成果リスト

(1) 学術論文（投稿中のものは「投稿中」と明記）

なし

(2) 国際会議プロシーディングス

なし

(3) 国際会議発表

[1] Yushi Omote, Takahiro Shinagawa, Kazuhiko Kato. Towards agile and elastic bare-metal clouds. The 24th ACM Symposium on Operating Systems Principles (SOSP '13), Pennsylvania, Nov. 2013.

[2] Daichi Serikawa, Yushi Omote, Takahiro Shinagawa, Kazuhiko Kato. Towards Secure and Efficient Volunteer Computing. 4th Asia-Pacific Workshop on Systems (APSYS 2013), Singapore, Jul. 2013.

(4) 国内会議発表

[1] 深井 貴明, 表 祐志, 品川 高廣, 加藤 和彦. 物理マシン間のライブマイグレーション手法の提案. 第 127 回システムソフトウェアとオペレーティング・システム研究会.
情報処理学会研究報告, 第 2012-OS-127 巻, 情報処理学会, 2013 年 12 月.

[2] 石井 智也, 忠鉢 洋輔, 表 祐志, 品川 高廣, 加藤 和彦. VMM を用いた 2 段階認証の支援. 第 25 回コンピュータシステムシンポジウム, 豊洲, 2013 年 12 月.

[3] 深井 貴明, 表 祐志, 品川 高廣, 加藤 和彦. 物理マシン間のライブマイグレーション手法の提案. 第 25 回コンピュータシステムシンポジウム, 豊洲, 2013 年 12 月.

(5) その他（特許，プレス発表，著書等）

[1] BitVisor 1.4 公開. 2014 年 5 月.