情報処理学会論文誌　コンピューティングシステム　Vol.5　No.2　111–120 (Mar. 2012)

HPCクラウドの実現に向けた仮想化クラスタの性能評価

高野 了成[1,a)]　池上 努[1]　広渕 崇宏[1]　田中 良夫[1]

受付日 2011年7月19日, 採録日 2011年11月29日

概要：クラウド資源を活用した高性能計算（HPC）の可能性を探るために，HPC アプリケーションの性能測定を通じて，InfiniBand を PCI パススルー経由で利用した仮想化 HPC クラスタの性能を評価した．ハイブリッド並列アプリケーションを 16 ノード上で実行した結果，MPI 通信のスループットに対する PCI パススルーの大きな効果が確認でき，粗粒度を前提としたアルゴリズムであれば物理計算機に匹敵する並列化効率が得られることを確認した．これより HPC 向け Infrastructure as a Service（IaaS）を提供するのに十分に実用的な性能を得られるという見込みを得た．

キーワード：HPC クラウド，仮想化，InfiniBand，MPI

Performance Evaluation of a Virtualized Cluster toward HPC Cloud Computing

Ryousei Takano[1,a)]　Tsutomu Ikegami[1]　Takahiro Hirofuchi[1]
Yoshio Tanaka[1]

Received: July 19, 2011, Accepted: November 29, 2011

Abstract: The feasibility of the cloud computing in the field of high performance computing (HPC) is assessed by measuring the performance of HPC applications on a virtualized cluster system equipped with PCI passthrough InfiniBand devices. We evaluate some hybrid parallel applications on 16 compute nodes. The result shows PCI passthrough produces great improvement for MPI communication throughput, and the parallel efficiency of coarse-grained parallel applications is comparable to the real machines. This paper leads to a positive prospect that Infrastructure as a Service (IaaS) for HPC users is feasible.

Keywords: HPC cloud, virtualization, InfiniBand, MPI

1. はじめに

クラウドコンピューティングは，計算資源を抽象化して運用する手段として近年その利用が拡大している．なかでも，計算機のコモディティ化の進展を背景に，計算機ハードウェアそのものを仮想化する Infrastructure as a Service（IaaS）が実用的なサービスとして提供されるようになった．特に，クラウドコンピューティングに対する高性能計算（High Performance Computing, HPC）の高い需要を受け，Amazon EC2 [1] の Cluster Compute Instance や Open Cirrus [2] の HPC オンデマンドサービスなど，HPC を意識した IaaS が現れ始めたことは注目に値する．Amazon EC2 の Cluster Compute Instance では，仮想計算機イメージを物理サーバへ配備するために仮想化技術が用いられているが，サーバの集約化は行わず，1 台の物理サーバをそのまま仮想化した構成を採用している．

仮想化のオーバヘッドを考えると，HPC アプリケーションを実行するのに仮想計算機を用いるのは得策とはいえず，物理計算機をそのまま使用する方が効率は良い．しかし，HPC 向け計算資源の仮想化はユーザ側にも実行環境整備の省力化の点でメリットが大きい．物理的にはネットワークで接続された複数のスーパコンピュータにより構成されるインフラを，クラウドコンピューティングのように容

[1] 独立行政法人産業技術総合研究所情報技術研究部門
Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Ibaraki 305–8568, Japan
[a)] takano-ryousei@aist.go.jp

易に利用できる仕組みの実現が期待されているが，従来型のアプローチでは利用するサイトごとにHPCアプリケーションを整備していく必要がある．HPCアプリケーションは往々にして実験的なコードを含んでおり，親切なインストーラがいつも利用できるとは限らない．この場合，計算機環境の調査と設定・バイナリの構築・テストデータを用いたバイナリの検証といった一連の作業を，利用するサイトごとに繰り返していくことになる．これは特に巨大アプリケーションでは時間と労力を要する作業であり，利用サイトの追加を阻む大きな要因となっている [3], [4]．ここでもし各サイトで共通の仮想計算機を設定し，計算機環境の差異を仮想計算機のレベルで吸収できれば，アプリケーションの整備は1回で済む．その他の利点として，仮想計算機マイグレーションやチェックポイントによる耐障害性の向上などもあげられる．

以上の背景をふまえ，我々はユーザの利便性と性能の追求の両立を目指した，HPC向けIaaS「HPCクラウド」の構築を構想している．まず，ユーザが手元の計算機でアプリケーションを含む仮想計算機イメージを作成，テストする．これを任意のサイトに配備し，ユーザの要求に応じた規模の仮想化HPCクラスタをオンデマンドに構築する．サイト内のインターコネクトはInfiniBandや10 Gigabit Ethernet（10 GbE）など，HPC用途に耐えうる高速なデバイスを前提とする．仮想計算機イメージはインターコネクトへの依存性を排除し，サービスが提供されるサイトで利用可能な最善のインターコネクトを選択し実行できる，性能可搬性を保証する．

我々は先行研究 [5] において上記のようなクラウドコンピューティング環境を想定した予備評価を行った．計算ノード単体の計算性能に対する仮想化のオーバヘッドはアプリケーションに依存するものの，5〜15% 程度に収まった．一方，16ノードを用いたMPI通信性能は，最善でも実ハードウェアの半分程度にとどまり，インターコネクトに用いた10 GbEの仮想化オーバヘッドが実用上無視できないことが分かった．HPCアプリケーションの通信には，低レイテンシかつ高スループットが求められるので，IO処理の仮想化による性能低下は大きな問題である．この性能低下を極力回避するためには，仮想マシンモニタをバイパスし，ゲストOSから直接IO処理を実行できる，PCIパススルーを用いることが現実的な解であると考えている．そこで，先行研究 [6] では，InfiniBandをPCIパススルーで用いる仮想化クラスタを構築し，その効果を検証した．評価対象の仮想計算機モニタ（Virtual Machine Monitor, VMM）には，完全仮想化を採用するKVMと準仮想化を採用するXenを用いた．PCIパススルーによってスループットは物理計算機と遜色ない性能を得られたが，レイテンシに関しては割込みインジェクションの実装方法に起因するオーバヘッドが見られ，特に準仮想化版のXenではその影響によると考えられるアプリケーション性能の低下を観測した．また，準仮想化のXenと完全仮想化のKVMの比較は公平な評価といえないという懸念もあった．

本論文では，評価対象を完全仮想化に絞り，仮想化環境上での性能を最適化する手法としてPCIパススルーに加えて，NUMAアフィニティの効果を調査する．ローカルメモリとリモートメモリのアクセス時間が異なるNUMAアーキテクチャにおける性能最適化手法として，アフィニティの設定はよく知られているが，仮想化環境においても有効であるかは自明ではないからである．MPIとOpenMPのハイブリッド並列プログラムを実用的な性能で実行できる仮想化クラスタを構築できるか性能評価を行い，HPCクラウドの実現に向けて，今後解決すべき技術的課題を明らかにする．

以下，2章で評価の目的と対象とする仮想計算機技術の概要について述べる．実験環境，および使用したベンチマークについて3章で述べ，4章で実験結果と得られた考察を示す．5章で関連研究について簡単に言及する．最後に6章でまとめを行う．

2. 評価の目的と評価対象

2.1 仮想計算機のHPC利用における課題

既存のHPCユーザをHPCクラウドに引き込むには，ユーザの利便性や管理面の運用性だけでなく性能が最重要であることはいうまでもない．実用的なHPCクラウドを実現するには，(1) 仮想化環境上でも十分な性能が出るアプリケーションの抽出，(2) 仮想化環境上での性能最適化のための方法論，について検討が必要である．HPCを対象にした既存研究は主に項目 (1) に着目しており，項目 (2) にまで踏み込んで評価した研究は我々が知る限り行われていない．具体的には，NUMAアフィニティの設定やPCIパススルーの活用など，実計算機の構成情報を極力隠蔽せずゲストOSに開示することで，アプリケーション性能を高めることが可能と考える．

ここでは特にNUMAアフィニティについて取り上げる．Intel Nehalemなどのコモディティ CPUにおいても，NUMA（Non Uniform Memory Architecture）アーキテクチャの導入が進んでいる．NUMAではローカルメモリとリモートメモリへのアクセスに速度差が生じるので，高性能を追求するためにはスレッドをメモリの近くで実行する，アフィニティの考慮が不可欠である．たとえば，今回用いた実験環境（詳細な仕様は3.1節で記す）では，MPIによるローカルメモリアクセスのピーク性能は5.7 GB/s，リモートメモリアクセスのピーク性能は3.9 GB/sと1.5倍弱の性能差がある．したがって，OSはスレッドをどのCPUにスケジューリングし，どこからメモリを確保するか適切に決める必要がある．また，OS任せの設定が困難な場合に対して，CPUやメモリに対するアフィニティをランタ

イムおよびアプリケーションプログラムから明示的に指定し性能を最適化するための手段も提供している．具体的なコマンドとしてはプロセスの CPU アフィニティを設定する `taskset` や，プロセスやメモリの NUMA ポリシを制御する `numactl` が存在する．物理計算機で有効な NUMA アフィニティなどの最適化手法が仮想計算機でも通用するのかは自明ではない．

2.2　評価対象

仮想計算機の実現方式は，ゲスト OS に改変を必要とする準仮想化と，ゲスト OS への改変が不要な完全仮想化に分類できる．仮想計算機の歴史は完全仮想化とともに始まったが，IA32 アーキテクチャの仮想化に対する制約の回避，性能向上を狙って準仮想化が提案された．さらに近年は完全仮想化のオーバヘッドをハードウェアオフロードにより軽減するため，Intel VT などの仮想化支援機能が普及し，完全仮想化が今後の主流になると考える．

本論文では仮想計算機のオープンソース実装である Xen と KVM を用い，完全仮想化方式を用いた仮想化環境にて性能評価を行った．以降，本章では仮想 CPU のスケジューリング，メモリ，IO アーキテクチャについて，Xen と KVM の差異に着目して簡単に説明する．

2.3　仮想 CPU

仮想 CPU のスケジューリングに着目した Xen と KVM の違いを**図 1** に示す．Xen では，VMM 上で動作する仮想計算機のことをドメインと呼び，実計算機へのアクセスや他のドメインを管理する特権的な仮想計算機をドメイン 0 と呼ぶ．仮想 CPU は VMM によってスケジューリングされる．一方，KVM では，VMM はホスト OS である Linux のカーネルモジュールとして実装されており，実計算機へのアクセスはユーザランドの QEMU プロセスを経由して実行される．QEMU プロセス内では，仮想 CPU と 1 対 1 に対応するスレッドが生成され，Linux カーネルのスケジューラによって，通常プロセスと同様にアフィニティを考慮したスケジューリングがなされる．

本論文では，物理 CPU と仮想 CPU の紐付け，ゲスト OS 内のアフィニティ制御に着目する．仮想 CPU と物理 CPU を紐付けるために以下の手段を用いる．Xen では `xm vcpu-pin` コマンドを用いる．KVM では仮想 CPU スレッドのスレッド ID を QEMU コンソールの `info cpus` コマンドで調べ，`taskset` で物理 CPU と紐付ける．さらに，KVM は物理計算機の NUMA トポロジを`-numa` オプションを利用して設定できるので，ゲスト OS で `numactl` を利用できる．一方，Xen では物理計算機の NUMA トポロジは隠蔽され，1 ソケット構成に見えるので，ゲスト OS 内から `numactl` を利用できない．

2.4　MMU 仮想化

仮想化環境では，仮想計算機の仮想アドレス（Guest Virtual Addres, GVA）から物理アドレス（Guest Physical Address, GPA）へ，さらに物理計算機の物理アドレス（Host Physical Address, HPA）へと，2 段階のアドレス変換が必要となる．

完全仮想化ではゲスト OS を改変できないので，GVA から HPA へ変換するシャドウページテーブルが VMM に導入された．CPU（MMU）はゲスト OS のページテーブルの代わりにこれを参照する．VMM はページテーブルの更新を監視することで，シャドウページテーブルとの一貫性を保持する必要がある．2 段階のアドレス変換処理を CPU にオフロードする仕組みとして，拡張ページテーブル（Extended Page Table, EPT）方式が提案されている．CPU は，ページテーブルに加えて拡張ページテーブルも参照することで，ソフトウェアによるシャドウページテーブル方式よりも概してオーバヘッドが小さいアドレス変換を実現している [7]．本実験では，Xen，KVM ともに EPT 方式を用いる．

2.5　IO アーキテクチャ

仮想計算機における IO アーキテクチャの概要を**図 2** に示す．ゲスト OS から物理デバイスにアクセスする方式は，現時点で利用可能な技術として，IO エミュレーション方

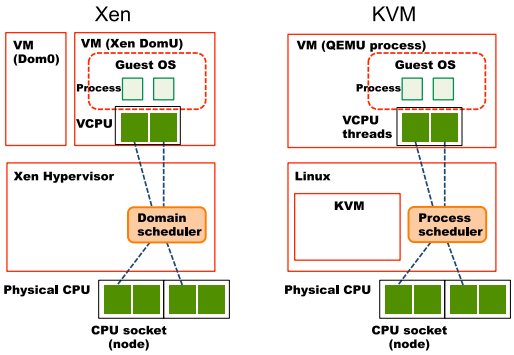


図 1　NUMA システム上の仮想 CPU
Fig. 1　Virtual CPUs on a NUMA system.

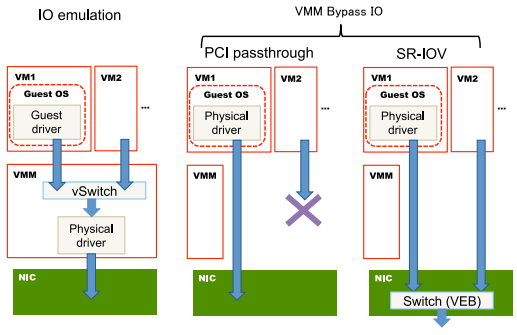


図 2　仮想計算機の IO アーキテクチャ
Fig. 2　IO architecture on a virtual machine.

式と VMM バイパス IO 方式に大別できる．なお，以降の説明ではネットワークデバイスを仮定する．

IO エミュレーション方式では，ゲスト OS 側のゲストドライバが VMM 側のホストドライバが一組で動作し，仮想スイッチを介してパケットを授受する．実装の詳細には違いがあるが，Xen のスプリットデバイスドライバや，KVM の virtio はこの方式である．複数の仮想計算機から物理デバイスを共有することに理論上の制限はなく，特殊なハードウェア支援も不要である．しかし，オーバヘッドが大きいという問題が存在する．

VMM バイパス IO 方式では，ゲスト OS から VMM を介さずにデバイスに直接アクセスできるので，その入出力性能は物理計算機に匹敵する．CPU やチップセットは，Intel VT-d などの仮想化支援機能に対応している必要があり，具体的には PCI パススルーや Single Root IO Virtualization（SR-IOV）などの方式が存在する．PCI パススルーでは，1 つの仮想計算機が該当する物理デバイスを排他的に占有するので，複数の仮想計算機からデバイスを共有できない．一方，SR-IOV では，デバイスの共有は可能であるが，その数は Virtual Function（VF）数に制限される．InfiniBand は元々 OS をバイパスして利用されるデバイスであり，VMM バイパス IO 方式との相性は良い．Mellanox の InfiniBand HCA は SR-IOV に対応しているが，現時点でデバイスドライバが公開されてないので，本論文では VMM バイパス方式として PCI パススルーを利用した．

データ転送に関しては，GVA から HPA へのアドレス変換をハードウェアにオフロードすることで VMM の介在を不要にできるが，割込みに関しては，IRQ が共有されたり，仮想 CPU が実行可能状態にない可能性もありうるので，まず VMM が割込みを受け取り，それを仮想計算機に通知する実装となっている．つまり，割込みごとに仮想計算機から VMM へ，さらに仮想計算機へとモード遷移が発生する．この処理は割込みインジェクションと呼ばれ，通信レイテンシの増加要因になりうる [8]．割込みインジェクションの実装は，次に示すとおり準仮想化と完全仮想化では異なり，後者の方が高速である．準仮想化の Xen では，割込みはイベントとして抽象化され，イベントチャネルを経由して VMM からゲスト OS に通知される．一方，完全仮想化の Xen や KVM では VT-d の仮想化支援機構を利用しており，VMM は仮想計算機のコンテキストなどを格納する VMCS（Virtual Machine Control Structure）領域に発生した割込み原因を示すビットを設定することで，ゲスト OS に対して割込みを挿入できる．

3. 実験

3.1 実験環境

実験には AIST Green Cloud（AGC）クラスタの一部の 16 ノードを使用した．AGC の計算ノードは，ブレードサーバ Dell PowerEdge M610 で構成されており，16 ノードが 1 エンクロージャに格納されている．各エンクロージャは InfiniBand QDR と 10 Gigabit Ethernet（10 GbE）のブレードスイッチを持ち，InfiniBand に関しては 16 ノードで 1 つのサブネットを構成している．**表 1** に AGC クラスタの諸元をまとめる．

表 1　AGC クラスタの諸元
Table 1　AGC Cluster specifications.

Node PC	
CPU	Intel Xeon E5540/2.53 GHz x2
Chipset	Intel 5520
Memory	48 GB DDR3-1066
InfiniBand	Mellanox ConnectX (MT26428)
Ethernet	Broadcom NetXtreme II (BCM57710)
Switch	
InfiniBand	Mellanox M3601 Q
Ethernet	Dell PowerConnect M8024

各ノードは，Quad-core Nehalem（E5540 2.53 GHz）を 2 基搭載し，各 CPU ソケットに 24 GB，計 48 GB のメモリが接続されている．L3 キャッシュサイズはコア共有で 8 MB である．ノード間は InfiniBand および 10 GbE で接続されているが，本実験では InfiniBand だけを利用する．InfiniBand QDR の理論最大性能は 4 GB/s である．PCI パススルーに対応するため，ConnectX のファームウェアを 2.6.00 から 2.7.80 へ更新した．Hyper Threading は無効化した．

OS は物理計算機（Bare Metal Machine, BMM），仮想計算機ともに 64 ビット版の Linux ディストリビューション Debian/GNU Linux 6.0.1 を使用した．Linux カーネルのバージョンは 2.6.32-5-amd64 である．

物理計算機には 1 台の仮想計算機を起動し，それぞれに 8 個の CPU コアをすべてと，45 GB のメモリを割り当てた．InfiniBand は PCI パススルーを用いて仮想計算機に割り当てた．仮想化環境は Xen 4.0.1 および QEMU-KVM 0.12.5 を用いて構築した．QEMU の実行に際し，SSE 拡張命令セットなど可能な限りの CPU 機能をゲスト OS から利用するために，CPU モデルには `host` を指定した．また，CPU とメモリのトポロジを物理計算機と揃えるために `-smp` および `-numa` オプションを設定した．Xen に関しては NUMA 対応は進められているが，ゲスト OS に対しては 1 ソケットにすべての CPU とメモリが接続されているように隠蔽されるので，`numactl` は利用できない．

3.2 ベンチマークプログラム

PCI パススルーの効果を確認するため，MPI の基本通信性能を Intel MPI Benchmarks 3.2 を用いて測定した．ノードあたりのメモリ搭載量が増加の傾向にあることから，特にメッセージサイズの大きなところまで（最大 1 GB）調

べた．時間測定には MPI_wtime を使用しており，その精度はマイクロ秒である．なお，仮想化環境下において，時間測定の精度が劣化する可能性を考慮し，各イテレーションの実行回数を 100 万回に増やして求めた平均値を結果とした．なお，イテレーション回数は小メッセージサイズでも安定した結果が得られるのに十分大きな値である．

PCI パススルーおよび NUMA アフィニティの効果を確認するための HPC アプリケーションでのベンチマークとして，NAS Parallel Benchmarks 3.3.1（NPB）および自作のアプリケーション Bloss を用いた．

NPB は MPI と OpenMP のハイブリッド並列性能を測定する Multi-Zone 版（NPB MZ）[9] を選択した．NPB MZ には LU，SP，BT の 3 種類のベンチマークが含まれるが，LU はプロセス数の上限が 16 に制限されることから省いた．SP と BT はともに 3 次元空間における非定常圧縮性 Navier-Stokes 方程式を ADI 法を用いて解くベンチマークであるが，BT の方がメッシュの分割が一様ではなく，負荷分散が難しい．問題サイズはクラス C を選択した．クラス C は全体で 800 MB 程度のメモリを使用する．隣接通信が主であり，平均メッセージサイズは SP が 48 KB，BT が 26 KB で，最大メッセージサイズは SP が 58 KB，BT が 82 KB と，BT の方がばらつきが大きい．予備実験の結果より，MPI 1 プロセスあたり 2 本の OpenMP スレッドを割り当てた．したがって 16 ノードで最大 64 プロセスとなる．

Bloss はブロック櫻井・杉浦法を用いた疎行列非線形固有値問題の内部固有値解法アプリケーションである [10], [11]．Bloss のプログラム構成と通信量を**図 3** に示す．プログラムは最大 10 GB のメモリを必要とする OpenMP 並列ジョブを MPI で束ねる構成をとっており，MPI 部分は比較的粗粒度の並列となっている．MPI 通信パターンが単純で最大 1 GB の集団通信 (MPI_Bcast，MPI_Reduce，MPI_Gather 関数）が主体であること，OpenMP 部分で大規模なメモリアクセスが発生することが特徴である．Bloss の実行では，主にメモリ要求量の観点から MPI 1 プロセスあたり 4 本の OpenMP スレッドを割り当てた．したがって 16 ノードで最大 32 プロセスとなる．

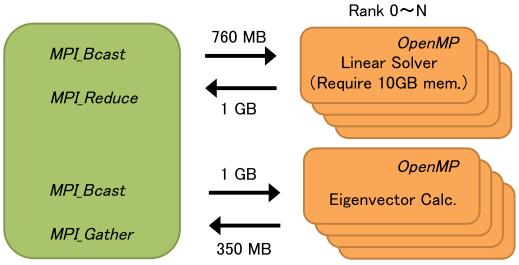


図 3　Bloss のプログラム構成と通信量

Fig. 3　Program structure of Bloss and the amount of communication.

コンパイラは gcc/gfortran 4.4.5 を用い，最適化オプションは`-O3 -fopenmp` を指定した．Bloss は並列数値計算ライブラリとして，Intel Math Kernel Library（MKL）11.1 を用いた．MPI 実装は OpenMPI 1.4.2 を用いた．InfiniBand 使用時に，MPI 集団通信の性能を向上させるために，実行時オプションとして`--mca mpi_leave_pinned 0` を付加した．実行バイナリは BMM 上で 1 回だけ生成し，これを全環境で流用した．

Nehalem 以降のアーキテクチャでは，動作していない CPU コアが存在するなど一定の条件を満たすと，CPU コアを自動的に定格の動作周波数よりも高速化する Turbo Boost が動作する．HPC アプリケーションの並列化効率を調べる際，Turbo Boost による影響を無視するため，ノード上の 8 CPU コアをすべて使用することにした．

3.3　NUMA アフィニティに関する予備評価

HPC アプリケーションの評価を行う前に，適切な NUMA アフィニティ設定を探るための予備評価を行った．実験には Bloss を用い，メモリアクセスの有無による挙動の違いを考慮するため，問題サイズには small と large の 2 種類を用意した．small のプロセスあたりのワーキングセットは 300〜400 kB で L3 キャッシュに載り，large は最大 7.5 GB と L3 キャッシュに載らない．スレッド数は 1 本と 8 本の場合で実行した．small は 10 回の実行時間の最小値を，large は 1 回の実行時間を結果とした．

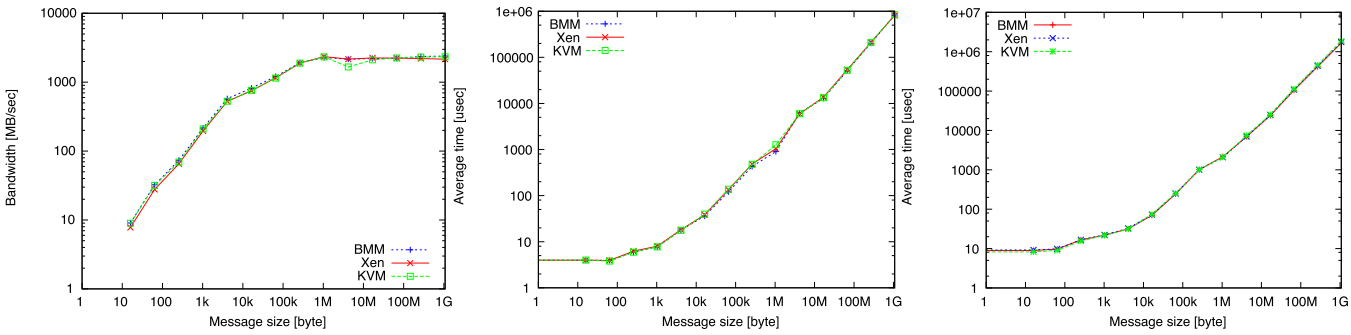
結果を**表 2** に示す．ここでは，物理 CPU と仮想 CPU の明示的に紐付けした場合を pin，これらの対応付けを VMM にまかせる場合を nopin と呼ぶ．さらに物理 CPU をスレッドに紐付けた場合を bind と呼ぶ．pin は VMM 側の設定，bind はベンチマークを実行する OS 内での設定となる．BMM の pin と Xen の bind はそれぞれ設定できないので空欄とする．

BMM の場合，「small/1 thread」を除いて small と large ともに，bind の効果が現れている．bind によりローカルメモリアクセスと CPU キャッシュの効果があると推測できるが，キャッシュに載らない large で効果が大きいことから前者が主要因であり，NUMA アフィニティの効果が

表 2　Bloss における CPU アフィニティ設定の効果 [sec]

Table 2　Effect of setting CPU affinity on Bloss [sec].

	nopin	pin	bind	nopin	pin	bind
	small/1 thread			large/1 thread		
BMM	4.067	-	4.250	525.8	-	518.2
Xen	4.302	4.349	-	780.9	568.3	-
KVM	4.157	4.296	4.293	562.8	547.2	526.7
	small/8 threads			large/8 threads		
BMM	2.567	-	2.437	343.1	-	327.7
Xen	2.496	2.482	-	366.5	367.4	-
KVM	2.561	2.644	2.498	351.5	355.9	334.9



(a) IMB PingPong　(b) IMB Broadcast　(c) IMB Reduce

図 4　MPI 通信性能

Fig. 4　MPI communication performance.

現れたと考える．Xen の場合，概して大きな効果は見られないが，「large/1 thread」では pin しないと性能が著しく悪化している．この現象には再現性があり，リモートメモリアクセスの発生が原因であると推測する．KVM の場合，「small/1 thread」を除いて bind 時の実行時間が最も短く，仮想計算機上でも NUMA トポロジを意識したアフィニティの設定が効果的であることを示している．また，bind しない場合の pin の効果は逆効果であり，プロセススケジューラの負荷分散に任せればよいことが分かる．pin が逆効果になる要因として，物理 CPU の負荷はゲスト OS のスケジューラからは見えないので，仮想 CPU スレッドとホスト OS 上のその他のスレッドと物理 CPU の使用が競合した場合に，適切な負荷分散を実行できない点があげられる．

この結果を受けて，以降の HPC アプリケーションの実験では，BMM に関しては bind，Xen に関しては pin，KVM に関しては nopin と bind 両方に関して調査することにする．

4. 結果と考察

4.1 通信性能

2 ノード間で IMB PingPong ベンチマークを実行し，片道レイテンシとスループットを測定した．片道レイテンシは 0 バイトメッセージの PingPong の平均実行時間から求めた．その結果，BMM が 1.61 マイクロ秒，Xen が 1.91 マイクロ秒，KVM が 1.62 マイクロ秒となった．Xen のレイテンシがやや長いが，準仮想化版の Xen 3.4.3 では，同じ実験に 3.30 マイクロ秒要しているので [6]，64%の改善になる．Xen のバージョンが異なるため厳密な比較ではないが，改善の主要因は割込みインジェクションにおけるオーバヘッドの削減と考える．

PingPong のスループットを図 4 (a) に示す．Xen と KVM ともに PCI パススルーを用いれば BMM と遜色がないスループット性能が得られ，ピーク性能は 2.4 GB/s に達している．ただしこれは InfiniBand QDR の理論最大性能 4 GB/s に対して 60%の性能にとどまっている．この原因として，PCI Express 接続のバスボトルネックと，Open MPI のオーバヘッドの 2 点が考えられる．InfiniBand verbs API を直接用いた qperf ベンチマークの実測値は 3.4 GB/s であった．したがって，性能低下のおおよその内訳はバスボトルネックで 0.6 GB/s，Open MPI で 1 GB/s となる．また，メッセージサイズが小さい場合の Xen において 15%程度の性能低下が見られる以外は，性能低下は数パーセントに収まっている．性能低下の原因はレイテンシと同様で割込みインジェクションに起因すると考える．

表 3　計算ノード単体での実行時間

Table 3　Execution time on a single node.

	SP-MZ [sec]	BT-MZ [sec]	Bloss [min]
BMM	94.41 (1.00)	138.01 (1.00)	21.02 (1.00)
Xen	102.42 (1.08)	142.46 (1.03)	22.55 (1.07)
KVM (nopin)	104.57 (1.11)	141.69 (1.03)	22.12 (1.05)
KVM (bind)	96.14 (1.02)	139.32 (1.01)	21.28 (1.01)

続いて MPI 集合通信性能を調査するために，16 ノードを用いた Broadcast，Reduce の実行時間を測定した．結果を図 4 (b)，(c) に示す．概して性能低下は数パーセントに収まっている．しかし，Reduce でメッセージサイズが 1 GB のとき，Xen は BMM に対して 5%の性能低下に対して，KVM は 11%も性能低下している．

4.2 HPC アプリケーション：ノード単体の性能

仮想化がノード単体の性能に及ぼす影響について調べる．本実験はノード間通信をともなわないので，NUMA アフィニティおよびメモリ仮想化が性能に影響を及ぼす可能性がある．NPB MZ ではプロセスあたり 2 スレッド，Bloss ではプロセスあたり 4 スレッドを立ち上げ，どちらも合計 8 スレッドでの実行時間を測定した．結果を**表 3** にまとめる．括弧内の数字は BMM に対する相対性能である．Xen と KVM はほぼ同じ性能を示し，BMM との比較から仮想化のオーバヘッドはアプリケーションに依存して 1〜

10%程度あることが分かる．また，KVM の場合は bind に著しい効果が見られ，いずれのベンチマークもオーバヘッドは 1〜2%に収まっている．

bind 時の KVM の結果より，NUMA トポロジを考慮することで，仮想化のオーバヘッドはほぼ無視できる．これは同時に，MMU 仮想化のオーバヘッドは無視できるほど小さいことを意味する．Xen と KVM の双方は，EPT 方式の MMU 仮想化を使用しているので，その部分のオーバヘッドは同じと考えると，Xen でも NUMA トポロジを考慮したメモリアクセスに対応することで，KVM と同程度まで性能が向上すると期待できる．

4.3　HPC アプリケーション：並列性能

プロセス数を 4 から 64 まで増やしながら NPB MZ を実行し，ノード数が増えたときの並列性能を調べた．結果を**図 5** および**図 6** に示す．X 軸のラベル「MxN」は MPI プロセス数が M，プロセスあたりの OpenMP スレッド数が N であることを意味する．棒グラフは絶対性能（Mop/s），折れ線グラフは並列化効率を示す．並列化効率は 1 ノード（4 プロセス）実行時の性能に対して算出した．

SP-MZ，BT-MZ ともにノード数にほぼ比例して性能が向上し，並列化効率は漸減した．16 ノード使用時の並列化効率に着目すると，SP-MZ では BMM が 91%，Xen が 85%，KVM が 81%であり，BT-MZ では BMM が 87%，Xen が 83%，KVM が 86%であった．また，仮想化のオーバヘッドは，SP-MZ では Xen が 14%，KVM が 12%であり，BT-MZ では Xen が 8%，KVM が 3%であった．BT-MZ の方が BMM での並列化効率の落ち込みが大きいので，仮想化のオーバヘッドが隠れる結果となった．

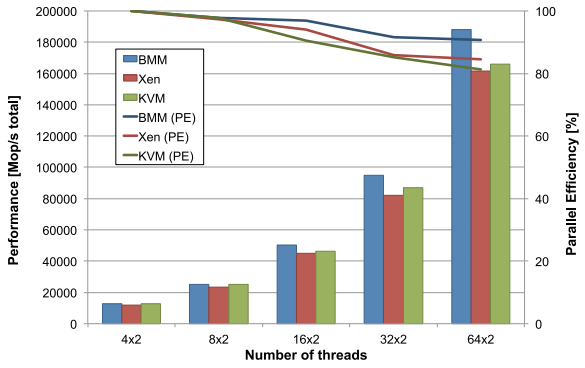


図 5　NPB SP-MZ の性能と並列化効率
Fig. 5　Performance and parallel efficiencies of NPB SP-MZ.

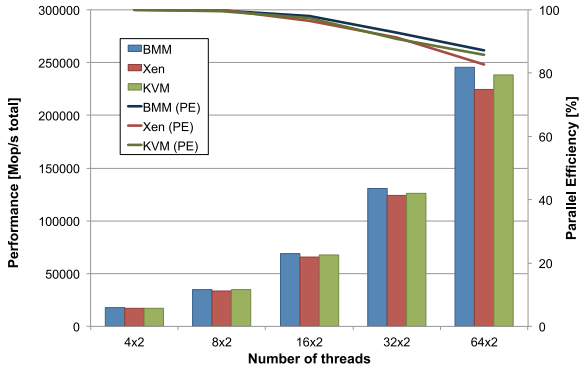


図 6　NPB BT-MZ の性能と並列化効率
Fig. 6　Performance and parallel efficiencies of NPB BT-MZ.

並列化効率低下の原因として，SP-MZ に関してはノードあたりのデータ量が一定なので，ノード間通信の増加が考えられる．SP-MZ に関しては，加えて動的負荷分散によるノードあたりのデータ量のばらつきによる影響が考えられる．また，仮想化技術の違いの視点で比較すると，絶対性能は KVM の方が高いが，並列化効率に着目すると，BT-MZ の 16 ノード時を例外として，概して Xen が KVM よりも高い傾向を示した．

Bloss ではプロセス数を 2 から 32 まで増やしながら並列化効率を測定した．結果を**図 7** に示す．並列化効率は 1 ノード（2 プロセス）時を基準に算出した．Bloss には各プロセスで処理が重複する箇所があり，本質的に並列化効率の低下が避けられない．このアルゴリズム由来の並列化効率（ideal）をグラフ中にあわせて示した．この曲線との差分が，通信由来の並列化効率の低下分となる．

ノード数が増えるに従い並列化効率は低下し，16 ノード使用時で BMM が 80%，Xen は 74%，KVM (bind) が 68%となった．なお，ideal の性能が 100%になるように正規化すると，BMM が 94%，Xen が 90%，KVM (bind) が 80%となる．また，16 ノード使用時の仮想化オーバヘッドは，Xen が 11%，KVM (bind) が 16%であった．KVM の性能低下の問題を追及するため，QEMU の`-numa`オプションを指定せず，pin も行わない設定で実行したところ（以下 w/o numa と記す），2 ノードまでは bind した方が性能は良いが，4 ノード以上で逆転した．16 ノード使用時の並列化効率は 74%，仮想化オーバヘッドは 14%であった．

Bloss における MPI 通信は NPB MZ と比較すると比較的粗粒度で，メッセージサイズの大きな集団通信が主体であり，レイテンシよりもスループットが性能に大きく貢献する．KVM (bind) による性能低下をさらに調査したとこ

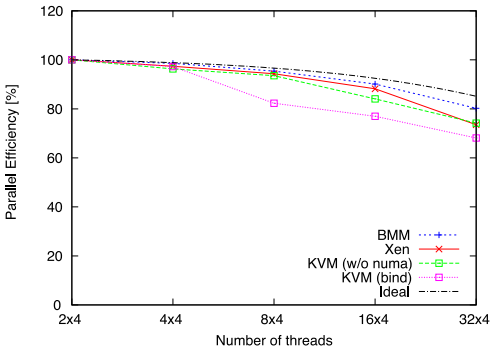


図 7　Bloss の並列化効率
Fig. 7　Parallel efficiencies of Bloss.

表 4　Bloss の各プロセスが線形方程式解法に要した実行時間 [秒]
Table 4　Execution time during the linear solver of bloss processes [sec].

	average	min.	max	deviation
BMM	78.11	77.12	79.45	0.28
Xen	84.69	81.60	89.43	5.07
KVM (w/o numa)	86.44	82.32	90.46	5.71
KVM (bind)	87.08	79.08	95.95	44.82

ろ，**表 4** に示すように線形方程式解法中の LU 分解の処理時間がノードごとに大きくばらついていることが分かった．MPI_Reduce で各ノードの計算結果を集めるときに，全プロセスが同期されるので，実行時間は一番遅いプロセスに律速される．この箇所は MPI 通信を含まない OpenMP プログラム部であり，全体の実行時間の 82%強を占る．また，bind の有無にかかわらず-numa オプションを指定すると再現した．問題の原因として，ホスト上 OS の負荷や割込み処理などのノイズが仮想計算機に与える影響や，動的負荷分散の失敗，意図どおりに NUMA アフィニィが機能していないなどの可能性が考えられる．

5. 関連研究

Web サービスやエンタープライズ用途が中心となるデータセンタでは，負荷量に応じて物理計算機上に複数の仮想計算機を集約することが有効であり，仮想化の影響に関する評価は数多く行われている．一方，HPC アプリケーションは計算機資源を排他的に占有するなど，負荷の傾向が異なるので，上記の評価結果をそのまま参考にはできない．HPC 用途における仮想化クラスタの性能評価に関しては，いくつかの研究 [12] は存在するものの十分ではなく，特に InfiniBand を PCI パススルーで用いた際に，HPC アプリケーションの実行に与える仮想化の影響を詳細に解析した研究はまだ少ない．Regola ら [13] は，Xen および KVM 環境において，InfiniBand を PCI パススルーで用いた評価も試みているが，4 ノードにおける NPB MPI の評価にとどまっている．また，使用されている KVM 環境に無視できない性能問題が存在するので，仮想化の影響を正確に知ることはできない．本論文では，MPI と OpenMP のハイブリッド並列を対象とした，より現実的な HPC アプリケーションでの評価を行い，その並列化効率を示した．また，実計算機と同様に NUMA アフィニティによるチューニングの効果を示した．

IO 処理を考慮した仮想計算機スケジューリングの改善に関していくつかの研究が行われている．これらは（1）ゲスト OS の処理の一部を VMM にオフロードする，（2）スループットとレイテンシのどちらを重視するかといった要求に応じてスケジューリングパラメータを調整するという提案に分類できる．

Wang ら [14] は Amazon EC2 上の通信性能を解析し，1 つの物理 CPU コア上で複数の仮想計算機が動作するスモールインスタンス環境における TCP 通信性能の低下について報告している．Kangarlou ら [15] は，この問題に対して，仮想計算機のスケジューリングに起因するラウンドトリップ時間の増加が主な原因であると指摘している．そこで，TCP コネクションを仮想スイッチ内で終端し，Ack パケット処理をオフロードすることで，TCP 通信性能を改善する仕組みを提案している．

本庄ら [16] は，Xen 環境で MPI プログラムを実行した場合，通信完了待ち時に実行される MPI_Waitall がビジーループする影響で，仮想計算機間のスケジューリングが滞り，結果として性能が大きく低下するという問題を報告し，タイムスライスを短くする提案を行っている．このように仮想 CPU がスピンロックなどでビジーループしている場合に，次の仮想 CPU がスケジューリングされず，実質的に物理 CPU が何も実行できない問題は Lock holder preemption と呼ばれる．この問題に対して Pause-loop exiting などの手法が提案されており，Linux カーネル 2.6.33 や Xen 4.0 以降で対応されている．一方，MapReduce のような IO インテンシブなアプリケーションに対して，Kang ら [17] は，IO 処理をまとめてバッチ的に処理することでコンテキストスイッチを削減し，IO の効率性と VM 間の公平性を改善するスケジューラを提案している．

ここであげた研究は物理 CPU を複数の仮想計算機から共有することを仮定しているが，本論文では物理計算機上で動作する仮想計算機は 1 台と仮定しているので，上記の問題は発生しない．また，仮想スイッチ自体のオーバヘッドも大きいので，VMM バイパス IO を積極的に活用することを考えている．

既存の VMM の代わりに，HPC 用途に特化した軽量 VMM を用いるアプローチも存在する．Palacious [18] では PCI パススルーのほか，ページング機構の選択，仮想計算機のプリエンプションの制御によって，仮想化による性能低下や性能のばらつきを削減している．また，割込みインジェクションによるレイテンシの増加を避けるため，ユーザレベルでのポーリングを採用している．これは Cray XT のハードウェア機構に依存しているが，Intel VT が selective interrupt exiting に対応すれば同程度のオーバヘッド削減が可能と指摘している．

実行環境整備の省力化の観点では，Bare metal cloud [19] のように，仮想計算機イメージ（Amazon Machine Image）を変換して，仮想計算機ではなく実計算機上に配備するアプローチも検討に値する．性能面は問題ないが，マイグレーションできないなどの制約がある．

6. まとめと今後の課題

本論文では，InfiniBand をインターコネクションに持つ

仮想化クラスタを構築し，仮想化が計算機性能に及ぼす影響を，HPC アプリケーションにおける計算性能および通信性能の観点から評価した．その結果，HPC 向け IaaS を提供するのに十分に実用的な性能を得られるという見込みを得た．

計算性能はアプリケーションの性質に依存するものの，おおよそ 1〜10%のオーバヘッドと見込まれる．特に KVM ではゲスト OS に対して，物理計算機と同じ NUMA トポロジを開示することが可能である．NUMA アフィニティを考慮した最適化は非常に効果的であり，仮想化のオーバヘッドを 1〜2%にまで削減できた．

通信性能に対する PCI パススルーの効果は大きく，レイテンシに関しては割込みインジェクションに起因する多少のオーバヘッドはあるものの，スループットに関しては物理計算機と遜色ない性能を得られた．したがって，通信量が多くても，粗粒度を前提としたアルゴリズムであれば十分，仮想化環境での実行に耐えると考える．さらに HPC クラウドの適用領域を広げるためには，よりレイテンシに対する要求が厳しい細粒度の通信を行うアプリケーションを対象とした評価も必要である．また，通信と計算が重畳させた場合は，割込みインジェクションごとにモード遷移が発生し CPU 実行が妨げられるので，計算性能にばらつきを及ぼす可能性がある．割込みインジェクションが及ぼす影響の解析や削減方法の検討は今後の課題とする．

ノード単体性能は十分な性能を示している一方で，ノード数が増加した場合の並列化効率の低下は残された課題であり，16 ノード実行時で 3〜18%の仮想化オーバヘッドを観測した．アプリケーションによっては十分許容できるものもあるが，クラスタの大規模化に向けて障害となる可能性がある．特に 4.3 節で示したように KVM で NUMA アフィニティを設定した場合に，ノードごとの性能のばらつきが大きく，全体の性能を低下させるという問題が明らかになった．したがって，より大規模なクラスタでは Xen の方が性能的に優位になる可能性もある．今後はこの問題をさらに詳細に解析し，その根本的な原因を追求する予定である．また，プロセスあたりのスレッド数は BMM の性能を基に決定したが，仮想化環境ではシステムのバランスが異なり，動的負荷分散の効果も異なる可能性がある．仮想化環境に応じた性能チューニング方法を確立することも今後の課題と考える．

本研究で得た知見を基に，NUMA トポロジを意識した仮想 CPU 割当てや VMM バイパス IO によるデバイスの占有も考慮した，クラスタ全体の資源スケジューリング機能を有するクラウドスタックを実現し，プライベートクラウドとして運用する予定である．

謝辞　実験環境の整備では株式会社創夢の大田晃彦氏にご尽力いただいた．謹んで感謝の意を表する．

参考文献

[1] Amazon Elastic Compute Cloud (Amazon EC2), available from ⟨http://aws.amazon.com/ec2/⟩.

[2] Campbell, R., Gupta, I., Heath, M., Ko, S.Y., Kozuch, M., Kunze, M., Kwan, T., Lai, K., Lee, H.Y., Lyons, M., Milojicic, D., O'Hallaron, D. and Soh, Y.C.: Open Cirrus Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research, *Proc. USENIX HotCloud* (2009).

[3] Takemiya, H., Tanaka, Y., Nakada, H., Sekiguchi, S., Ogata, S., Kalia, R.K., Nakano, A. and Vashishta, P.: Sustainable Adaptive Grid Supercomputing: Multiscale Simulation of Semiconductor Processing across the Pacific, *Proc. 2006 ACM/IEEE Conference on Supercomputing* (2006).

[4] Ikegami, T., Maki, J., Takami, T., Tanaka, Y., Yokokawa, M., Sekiguchi, S. and Aoyagi, M.: GridFMO – Quantum Chemistry of Proteins on the Grid, *Proc. 8th IEEE/ACM International Conference on Grid Computing*, pp.153–160 (2007).

[5] 池上　努，高野了成，田中良夫，中田秀基，関口智嗣：クラウドコンピューティングの性能評価，情報処理学会研究報告，Vol.2010-HPC-128, No.14, pp.1–6 (2010).

[6] 高野了成，池上　努，広渕崇宏，田中良夫：InfiniBand を PCI パススルーで用いる仮想化 HPC クラスタの性能評価，*SACSIS 2011*, pp.109–116 (2011).

[7] Wang, X., Zang, J., Wang, Z., Luo, Y. and Li, X.: Selective Hardware/Software Memory Virtualization, *Proc. 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp.217–226 (2011).

[8] 渡邉和樹，永島　力，茂田井寛隆，片山吉章，毛利公一：Xen における PCI Passthrough の性能評価，情報処理学会研究報告，Vol.2010-OS-113, No.3, pp.1–8 (2010).

[9] Jin, H. and Van der Wijngaart, R.F.: Performance characteristics of the multi-zone NAS parallel benchmarks, *Journal of Parallel and Distributed Computing*, Vol.66, pp.674–685 (2006).

[10] Ikegami, T., Sakurai, T. and Nagashima, U.: A filter diagonalization for generalized eigenvalue problems based on the Sakurai-Sugiura projection method, *J. Comp. Appl. Math.*, Vol.233, pp.1927–1936 (2010).

[11] Sakurai, T. and Sugiura, H.: A projection method for generalized eigenvalue problems using numerical integration, *J. Comp. Appl. Math.*, Vol.159, pp.119–128 (2003).

[12] Nussbaum, L., Anhalt, F., Mornard, O. and Gelas, J.-P.: Linux-based virtualization for HPC clusters, *Proc. Ottawa Linux Symposium*, pp.221–233 (2009).

[13] Regola, N. and Ducom, J.-C.: Recommendations for Virtualization Technologies in High Performance Computing, *Proc. 2nd IEEE International Conference on Cloud Computing Technology and Science*, pp.409–416 (2010).

[14] Wang, G. and Ng, T.S.E.: The Impact of Virtualization on Network Performance of Amazon EC2 Data Center, *Proc. IEEE INFOCOM* (2010).

[15] Kangarlou, A., Gamage, S., Kompella, R.R. and Xu, D.: vSnoop: Improving TCP Throughput in Virtualized Environment via Acknowledgement Offload, *Proc. 2010 ACM/IEEE Conference on Supercomputing* (2010).

[16] 本庄賢光，窪田昌史，北村俊明：VM 上の MPI プログラムの通信オーバヘッドの性能評価，コンピュータシステムシンポジウム 2010，情報処理学会，pp.91–100 (2010).

[17] Kang, H., Chen, Y., Wong, J.L., Sion, R. and Wu, J.: Enhancement of Xen's Scheduler for MapReduce Workloads, *Proc. 20th ACM Symposium on*

High-Performance Parallel and Distributed Computing (2011).
[18] Lange, J., Pedretti, K., Dinda, P., Bridges, P., Bae, C., Soltero, P. and Merritt, A.: Minimal-overhead Virtualization of a Large Scale Supercomputer, *The 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (*VEE 2011*), pp.169–180 (2011).
[19] 高宮安仁，田浦健次朗，安井雄一郎，藤澤克樹：“Bare Metal” Cloud：実マシンを提供するクラウドサービス，情報処理学会研究報告，Vol.2010-HPC-126, No.39, pp.1–8 (2010).

高野 了成　（正会員）

2005 年東京農工大学大学院電子情報工学専攻博士後期課程単位取得退学．2003 年（株）アックス入社．2008 年より独立行政法人産業技術総合研究所情報技術研究部門研究員．博士（工学）．オペレーティングシステム，分散環境における高性能計算に関する研究開発に従事．ACM，IEEE，USENIX 各会員．

池上 努　（正会員）

1989 年東京大学理学部化学科卒業．1991 年同大学大学院理学系研究科修士課程化学専攻修了．1992 年より慶応義塾大学理工学部化学科助手，岡崎国立共同研究機構分子科学研究所助手，Universidad de Puerto Rico ポスドク，独立行政法人産業技術総合研究所グリッド研究センター研究員を経て，現在，同所情報技術研究部門研究員．博士（理学）．計算化学を中心に，高性能科学技術計算の研究に従事．日本化学会，日本物理学会会員．

広渕 崇宏　（正会員）

2001 年京都大学理学部地球物理学教室卒業．2002 年同大学大学院理学研究科地球惑星科学専攻修士課程退学．2007 年奈良先端科学技術大学院大学情報科学研究科博士課程修了．同年独立行政法人産業技術総合研究所入所．現在，情報技術研究部門研究員．博士（工学）．グリッドコンピューティング，クラウドコンピューティング，Green IT の研究に従事．システムソフトウェア，ネットワーク，仮想化技術等に興味を持つ．

田中 良夫　（正会員）

1995 年慶応義塾大学大学院理工学研究科後期博士課程単位取得退学．1996 年技術研究組合新情報処理開発機構入所．2000 年通産省電子技術総合研究所入所．2001 年 4 月より独立行政法人産業技術総合研究所．現在，同所情報技術研究部門主幹研究員．博士（工学）．分散環境における高性能計算，ミドルウェアおよびセキュリティに関する研究に従事．平成 18 年度情報処理学会論文賞．平成 21 年度科学技術分野の文部科学大臣表彰科学技術賞（研究部門）．ACM，IEEE 各会員．