Windows PCをグリッド環境で利用するための軽量Linuxバイナリ実行システム

上村佳史[†1] 中島佳宏[†1,†2] 佐藤三久[†1]

ネットワーク上にはWindowsやLinuxなどの異なるOSが用いられたPCが多く接続されている．これらのPCはグリッドコンピューティングの計算資源にすることができると予想される．このようなPCのOSの多くはWindowsが用いられているため，グリッドの計算資源として新たにWindows PCを利用することが考えられる．しかし，グリッドのミドルウェアなどは主にLinuxで開発されている．そのため，Windows PCを利用するためにグリッドミドルウェアをヘテロなOS環境に対応させる必要がある．本稿では，Windows PCをグリッドRPCワーカとして利用することを考え，WindowsとLinuxからなるヘテロOSグリッド環境において，Linuxプログラムを Windows上で直接実行するための軽量Linuxプログラム実行環境BEEの設計・開発を行った．そしてBEEをOmniRPCに組み込むことでヘテロOSグリッド環境においてWindows PCを計算資源として利用する．BEEを用いた性能評価の結果より，システムコールの実行性能はWindowsのネイティブプログラムと同程度の性能が得られた．また，OmniRPCを用いた実アプリケーションをLinuxとWindowsのヘテロOSグリッド環境での実験を行い，ヘテロOSグリッド環境上でWindowsをグリッドRPCのワーカとして利用できることを示した．

Light-weight Linux Binary Execution System to Utilize Windows PC in Grid Environment

Yoshifumi Uemura,[†1] Yoshihiro Nakajima[†1,†2] and Mitsuhisa Sato[†1]

Local area or campus-type networks consist of PCs using different operating systems such as Windows and Linux. These PCs are expected to have enormous potential computing power for grid computing. The majority of PCs in this type of environment run on Windows, while grid applications and middleware are often developed on Linux. The challenge is to absorb the heterogeneity of operating systems. Grid RPC is a promising programming model for the development of grid applications. We have designed and implemented an agent called BEE, which enables direct execution of Linux binary programs on Windows for a Grid RPC worker. We have integrated the BEE agent into an OmniRPC system in order to make use of Windows PCs as computing resources in a hybrid grid environment combining Windows PCs into grid computing resources. The BEE agent allows Linux binaries of the program of the OmniRPC worker to be exported and run under Windows without any modification of its Linux binaries. The results of our experiments show that the performance of a worker program using BEE is almost the same as that of Windows native binary. We have demonstrated a hybrid grid environment combining Window PCs in a conventional grid of Linux nodes.

1. はじめに

近年，ローカルエリアなどのネットワークにはWindowsやLinuxなどの様々なOSのPCが多く接続されており，これらは潜在的に非常に大きな計算資源である．また，広域ネットワークやそのインフラ技術の進歩により，大規模環境，高性能分散計算環境，計算グリッドなどを構築することが可能となってきている．

大規模グリッド環境を構築する際には，従来利用されているLinuxのPCクラスタに加えて，計算資源として新たに研究室やオフィスなどのPCを利用することが考えられる．このような研究室やオフィスなどのPCには，WindowsやLinuxといった様々なOSが用いらており，その中でもOSの多くにはWindows

†1 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 日本学術振興会特別研究員
Research Fellow of the Japan Society for the Promotion of Science

が用いられている．一方，グリッドのミドルウェアは主に Linux で開発されている．そのため，そのままでは Windows を計算資源として利用することができない．

この問題を解決する簡単な手法としては，Linux のミドルウェアをそのまま扱うために，PC を Windows と Linux のデュアルブート環境とするか，または既存の環境や構成を変更することなく簡便に Linux 環境にできる LiveCD [1] を用いることである．そしてグリッドの計算資源として提供できるようなとき，たとえば，夜間など長時間 PC が遊休状態となる場合に Windows 環境を一時的に Linux 環境とする．このような手法とするのは個人の PC を完全に Linux 環境とすることは難しいためである．なぜならば，Windows ユーザはオフィスで使用しているソフトウェアを利用するため，デュアルブートなどによる OS 環境によりソフトウェアが利用できなくなってはならない．他の手法として，Windows のバックグラウンドサービス，またはシステムトレイに常駐するようなソフトウェアにより Windows 環境にグリッドのための機能を提供し，計算資源として利用する．

本稿では，Windows PC の計算能力をグリッド RPC ワーカとして利用する．そのために，Linux プログラムを Windows 上で直接実行可能とする軽量なシステムとして，Linux プログラム実行環境 BEE の設計・開発を行った．グリッド RPC はグリッドのプログラミングモデルの 1 つであり，このプログラミングモデルでは非同期 RPC によるマスタ/ワーカモデルの並列プログラムをサポートする．我々は BEE をグリッド RPC 環境で扱うために，グリッド RPC のミドルウェアである OmniRPC [11),12),14)] に組み込む．これにより Windows をグリッド RPC の計算資源として利用することができる．

BEE は Windows 上で Linux プログラムをロードし直接実行する．仮想マシンなどと異なるのはハードウェア環境や Linux カーネルをエミュレートせず，ネットワーク通信などのグリッド RPC ワーカに必要とされるシステムコールのみをエミュレートすることである．BEE では入出力処理を Windows のネイティブ I/O で実行できるため，アプリケーションに軽量で性能の良い入出力処理を提供することができる．

本稿の構成は以下のようになっている．2 章では関連研究について述べる．3 章では我々が想定する Windows を計算資源として利用するためのヘテロ OS 環境について述べる．4 章では提案システムである BEE の設計と実装について述べる．5 章では BEE をグリッド RPC 環境で扱うために，BEE を組み込んだ OmniRPC について述べる．6 章では提案システムの性能評価について述べる．最後にまとめと今後の課題について述べる．

2. 関 連 研 究

ヘテロ OS グリッド環境において異なる OS を利用する手法はいくつか存在する．まず，Java などのようなバイトコードを生成するようなプログラミング言語を用いる手法である．Java の場合，バイトコードは Java 仮想マシンにより実行時に逐次ネイティブコードに変換され実行される．このため，各 OS に対応する Java 仮想マシンを導入しておくことで，同じプログラムを異なる OS 上で実行することができる．性能に関しては，近年ではネイティブプログラムと比較しても同程度の性能を得ることができる．しかし，科学技術計算など我々が想定するような高性能計算を行う場合では，ネイティブプログラムのほうが高い性能を得ることができる [2)]．

次に仮想マシンを利用する手法である．仮想マシンは仮想化したハードウェア環境をソフトウェアで提供する．これにより，異なる OS であっても，仮想マシン上に同じ OS 環境を構築することで，同じプログラムを用いることが可能となる．仮想化環境を提供するソフトウェアには VMware [15)]，VirtualBox [10)]，Xen [6)] coLinux [3)] などがある．

3 つ目として，他の OS 環境上でプログラムを直接実行する手法である．この手法を実現するシステムとして Wine [16)] がある．Wine は Linux 上で Windows プログラムを直接実行させることができる．そのために，Windows プログラムをロードするプログラムローダと，Windows API のためのライブラリを提供する．しかし，Linux プログラムを Windows 上で動作させるという研究はない．我々が提案する Windows 上で Linux プログラムを実行するというアプローチはこの手法に近い．

それ以外の手法として，それぞれの OS ごとにプログラムを作成する手法がある．本稿では Windows を計算資源として利用することを想定している．この場合，Cygwin 環境 [4)] のようなクロスコンパイラ環境を利用することができる．Cygwin は Linux システムコールの API をエミュレートするためのライブラリ群などにより Linux ライクな環境を提供する．これにより，Cygwin 環境上で Linux のソースコードを再コンパイルすることで，Windows バイナリを作成することができる．本手法では，Cygwin と比較して Linux

プログラムを Windows 上で直接実行することができるため，再コンパイルを行う必要がないという利点がある．

また，Linux システムコールを Windows 上で実行する手法としては，先に述べた仮想マシンなどで可能である．仮想マシンではハードウェアなどを仮想化をしており，システムコールなどの入出力処理は仮想化したハードウェアによりトラップされ処理されている．また，coLinux は仮想マシンとは異なり，Windows 上のアプリケーションとして Linux カーネルを動作させている．システムコールなどの入出力処理はこの Linux カーネルで実行される．我々の提案システムでは，ハードウェアの仮想化や Linux カーネルを動作させることはせず，より軽量な手法としてシステムコールのみトラップし，実行する．

3. Windows 上で Linux プログラムを実行するヘテロ OS グリッド環境

図 1 に，我々が想定する Windows PC を計算資源とするヘテロ OS グリッド環境を示す．従来の Linux クラスタや PC に加えて，グリッドの計算資源としてオフィスや研究室の Windows PC を利用する．我々の想定するグリッド環境では，マスタは Linux のみを用いる．これはグリッドのアプリケーションが主に Linux で開発されており，我々が BEE をグリッド環境で扱うために利用したグリッド RPC システムである OmniRPC も Linux 上で開発されているためである．Windows PC はグリッド RPC の計算資源であるワーカとして利用する．Windows PC をワーカとして利用するには，Linux 上で作成されたワーカプログラムを Windows 上で実行することで可能となる．本章では Windows 上で Linux ワーカプログラムを実行するための方法について述べる．

本稿では Windows をワーカとして利用する際に，以下の点を重要とする．

- Flexibility：Windows ユーザは Windows がワーカプログラムの実行中であっても普段どおり Windows 環境を利用できる．可能であれば PC が遊休の場合のみ計算資源として利用する．
- Performance：ワーカプログラムの実行性能はネイティブプログラムと同等の実行性能を得られる．
- Manageability：Windows ユーザはワーカ環境の構築，利用者はプログラムの作成などを容易にできる．

Cygwin のようなクロスコンパイラ環境を用いることで，Linux プログラムのソースコードを Windows

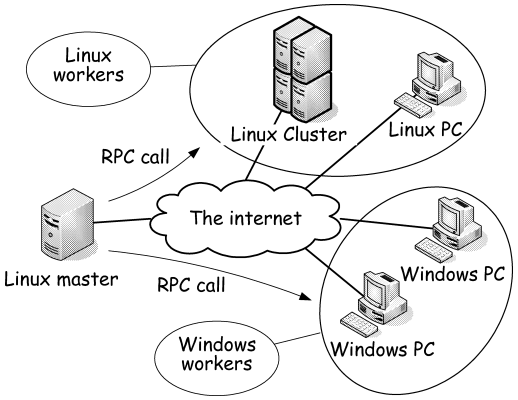

図 1 想定するヘテロ OS グリッド環境
Fig. 1 Overview of a hybrid grid environment.

のネイティブプログラムとして再コンパイルができる．ヘテロ OS グリッド環境上で Cygwin 環境を用いる場合，利用者は Linux と Windows のそれぞれのプログラムを用意する必要がある．そのため，ワーカにプログラムを配布する場合，ワーカプログラムを Linux と Windows を同様に配布することができず，グリッド RPC システムに新たにワーカ OS の識別機能の追加などの変更を必要とする．

仮想マシンは，仮想マシン上に Linux 環境を構築することができるため，Linux のプログラムでの互換性がある．仮想マシンはホスト上のアプリケーションとメモリ空間は分けられているため，仮想マシン上での障害によりホスト側に影響を与えることはない．しかし，仮想マシンはホスト上のシステム資源と多くのメモリやエミュレーションのための処理が必要となる．さらに，Linux 環境のインストールや設定，管理などが必要となる．Windows PC をグリッド RPC ワーカとして利用する観点から考えると，このような Linux 環境のすべての機能は必要ではない．

本稿では，Windows 上で Linux プログラムが実行可能な軽量システム BEE を提案する．これにより，Cygwin のように OS ごとのプログラムの作成やグリッド RPC システムにワーカ OS 識別機能などの新たな機能を追加する必要がなくなる．さらに，仮想マシンのように，Linux プログラムを実行するためだけに，ホスト側に多くのシステム資源を要求しない．

4. BEE の設計と実装

Windows と Linux からなるヘテロ OS グリッド環境において，Windows をグリッド RPC ワーカとして利用するために Linux プログラム実行環境 BEE を開発した．本章では，BEE の設計と実装について述

べる．

Linux は様々なアーキテクチャに対応しているが，BEE は Windows を対象とするため，対象とするアーキテクチャは IA-32 アーキテクチャのみである．これによりマシン命令は同じであるので，命令のエミュレーションを必要とせずにプログラム自体は実行できる．しかし，プログラムのフォーマットやシステムコールの実装は OS ごとに異なるため，単純に Linux プログラムを Windows 上で実行することはできない．そのために BEE ではプログラムローダとシステムコールエミュレータの機能を提供する．

次に，BEE で提供するプログラムローダとシステムコールエミュレータについて述べる．

4.1 プログラムローダ

Linux と Windows では実行可能なプログラムのフォーマットが異なるため，Linux プログラムを Windows 上で実行することはできない．Linux では ELF（Executable and Linking Format）などの様々なプログラムフォーマットをサポートするが，Windows では PE（Portable Executable）フォーマットなどが用いられている．このため，プログラムローダは Linux プログラムのフォーマットを解析し，Windows 上のプロセスとする機能である．

現在，BEE でサポートしているプログラムフォーマットは Linux で一般的な ELF のみである．また，ELF では静的リンクや動的リンク方式のプログラムがあるが，BEE では静的リンクのみのプログラムをサポートしている．なぜならば，動的リンクでは実行時に共有ライブラリを必要とするため，Windows 上にこれら共有ライブラリを保持しなければならなくなる．また，共有ライブラリのバージョンも問題となる．作成した Linux プログラムで必要な共有ライブラリのバージョンと Windows 上で保持している共有ライブラリのバージョンが異なっている場合に，実行ができなくなるといった不具合が生じるためである．

BEE では Linux プログラムを Windows 上のプロセスとするために，Linux プログラムは BEE の仮想メモリ上に展開する．**図 2** に Linux プログラムを BEE の仮想メモリ上に展開したときの状態を示す．このとき，BEE 自身と Linux プログラムが同じメモリ位置に展開されてはならない．ELF は通常，仮想メモリ上の *0x8048000* に展開される．そのため，BEE と Linux プログラムのアドレス衝突を避けるために，BEE を ELF が展開されるアドレスよりも低位のアドレスに置く．

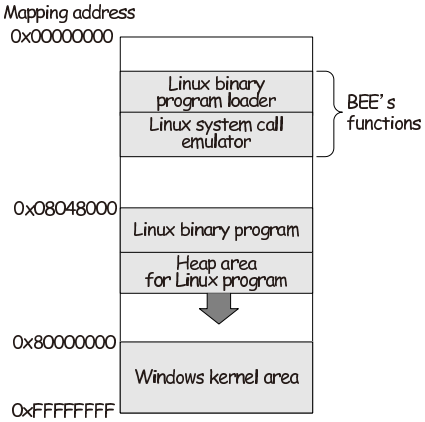

図 2 Windows 上の BEE のメモリレイアウト
Fig. 2 BEE memory map on Windows.

4.2 システムコールエミュレータ

Windows と Linux で同じ IA-32 アーキテクチャを用いるため，OS が異なっていてもマシン命令自体は実行することができる．しかし，入出力処理などはシステムコールを通して実行されており，このシステムコールの実装は OS ごとに異なる．そのため，Linux プログラムを Windows 上で実行することができない．

これを解決するために，BEE は Linux システムコールを Windows 上で実行するためのシステムコールエミュレータの機能を提供する．システムコールはソフトウェア割込みを発生させ，その割込みに対応する処理により実行される．Windows のシステムコールの割込みは NT/2000 では *int 0x2E*，XP 以降では *sysenter* 命令により行っている．一方，Linux では *int 0x80* により行われている．Windows では Linux で用いられている *int 0x80* に対応する割込み処理は使用されていない．そのため，BEE では *int 0x80* に対応する割込み処理を新たに追加することで，Linux システムコールを Window 上で実行させる．この割込み処理は，単純なトランポリンコードとなっており，*int 0x80* により状態を BEE のシステムコールエミュレータへ移行する．Linux システムコールのエミュレート後は Linux システムコールが発生した状態へと戻し，Linux プログラムの実行を継続する．割込み処理の追加にはデバイスドライバを用いる．このデバイスドライバは BEE の実行時に Windows に登録され，終了時に削除される．

しかし，この BEE により追加される割込み処理が Windows に影響を与えることがあってはならない．*int 0x80* により状態が割込み処理に移行するが，この割込み処理内ではシステムコールエミュレータに状態を移

行するためにレジスタ情報の書き換えを行うのみで，内部で特殊な処理は行わない．そのため，この割込み処理により他のプロセスや Windows に影響を与えることはないと考えられる．

現在，システムコールエミュレータは OmniRPC に対応するように実装している．OmniRPC ではネットワーク通信のシステムコールのみが必要であるため，send()，recv() などのシステムコールを実装している．また，Linux では write()，read() においてもネットワーク通信が可能であるため，これらのシステムコールによるネットワーク通信もサポートしている．ネットワーク通信に用いるシステムコール以外では，Linux プログラムを実行するうえで必要な exit() などのシステムコールを実装している．その他のシステムコールに関しては必要に応じて実装を行う．しかし，signal などの複雑なシステムコールやいくつかのシステムコールでは，Windows において対応しているシステムコールが存在しない，または実装や使用法が Linux と異なるといった場合がある．このようなシステムコールは Linux と完全互換を持つことは難しいため，用途に応じて目的の動作が得られるような実装とする．

4.3 Linux プログラムの実行の流れ

図 3 に Linux プログラムを実行したときのプログラムローダとシステムコールエミュレータの動作を示す．

(1) BEE の実行時に *0x80* に対応した割込み処理をデバイスドライバにより追加する．
(2) プログラムローダは Linux プログラムを BEE 自身のメモリ空間上に展開，実行する．
(3) Linux システムコールが発生した場合，デバイスドライバにより登録された割込み処理が実行される．
(4) 割込み処理は BEE のシステムコールエミュレータへと状態を移行する．
(5) システムコールエミュレータにより，Linux システムコールが実行され，その後，Linux システムコールの発生時に状態を戻し，処理を継続する．
(6) Linux プログラムからの exit() により BEE はデバイスドライバの削除を行い，終了する．

5. BEE を用いた OmniRPC の概要

BEE は Linux プログラムを実行するための機能しかサポートしていない．Windows をグリッド RPC ワーカとして利用するには，BEE をグリッド RPC のミドルウェアへ組み込む必要がある．我々は BEE を

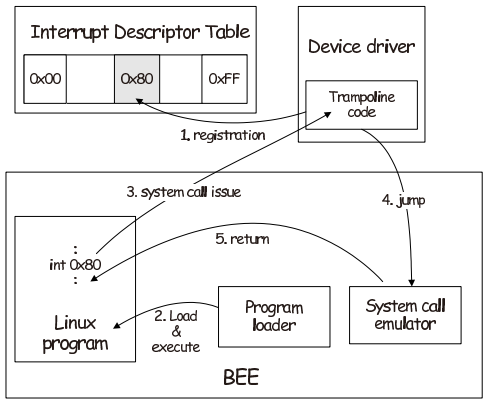

図 3 BEE による Linux プログラムの実行の流れ
Fig. 3 Program execution flow with BEE.

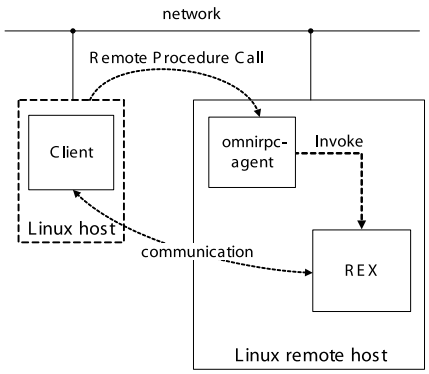

図 4 Linux 上の OmniRPC の動作
Fig. 4 Overview of original OmniRPC on Linux.

グリッド RPC システムの 1 つである OmniRPC に組み込んだ．本章では我々が用いた OmniRPC について述べる．OmniRPC はローカルなクラスタ環境から広域ネットワークで構成されたグリッド環境までのシームレスな並列プログラミングを可能にするマスタ/ワーカモデルのグリッド RPC システムである．

図 4 に Linux 環境での OmniRPC の RPC 処理の全体像を示す．OmniRPC は client，Remote executable module（REX），agent の 3 つから構成されている．client はマスタのプログラムでありワーカに RPC 処理を要求する．REX がワーカプログラムであり，マスタからの要求によりリモートノード上で実行される．agent はマスタとワーカ間の橋渡しを行い，マスタからの要求により REX の遠隔実行を行う．また，agent はクラスタ環境においてマスタのみが外部に公開されている環境において，マスタとワーカが通信を行う場合にプロキシとしても動作する．

OmniRPC では，マスタはリモートノード上のワーカプログラムを直接実行しない．マスタは最初にリ

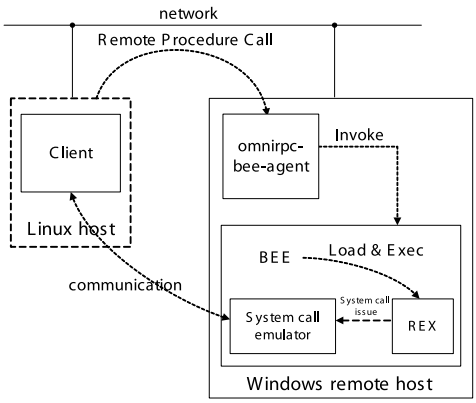

図 5 BEE を組み込んだ OmniRPC の Windows 上での動作
Fig. 5 ARPC system of integration of BEE into OmniRPC.

モートノード上の agent に対して ssh や Globus[8),9)] の GRAM[5)] などによる認証を行う．その後，agent を通してリモートノード上のワーカプログラムの遠隔実行を行う．

図 5 に Windows 上で BEE を用いた OmniRPC について示す．Windows はワーカとしてのみ利用し，マスタは Linux を用いる．Windows 上には ssh などの認証機構がないため，agent はあらかじめデーモンとして動作させる．しかし，agent にはデーモンとして動作する機能がないため，デーモンとして動作するように agent の機能拡張を行った．

OmniRPC の計算資源はグリッド環境上の PC クラスタを対象としている．このような環境では，一般的にワーカプログラムなどのファイルは各ノード間で共有されている．一方，我々が想定するヘテロ OS グリッド環境の場合，オフィスや研究室の PC を対象とするためワーカプログラムなどのファイル共有は難しい．よって，実行時に必要なワーカプログラムを配布する機能がグリッド RPC に必要となる．そこで，BEE を用いた OmniRPC に，マスタは実行時に必要な Linux ワーカプログラムの自動配布機能，agent にワーカプログラムを受信する機能の追加拡張を行った．

6. 性能評価

本章では，BEE の基本的な性能とヘテロ OS グリッド環境でのグリッド RPC アプリケーションの性能評価を行う．性能評価の比較としてクロスコンパイラ環境である Cygwin，仮想マシンである VirtualBox を用いる．性能評価は，基本的な性能評価としてシステムコールと通信性能の評価，また，実アプリケーションによる性能評価として OmniRPC を用いた並列固有値計算プログラムで行う．ヘテロ OS グリッド環境での評価は Windows と Linux の環境の混在環境により行う．

表 1 マスタ環境
Table 1 RPC master node configuration.

Linux	OS	Linux kernel 2.6.9
	CPU	Xeon 2.4 GHz
	Memory	1 GByte
	Network	Gigabit Ethernet

表 2 ワーカ環境
Table 2 RPC worker node configuration.

Windows	OS	WindowsServer2003 Enterprise Edition SP1
	CPU	Xeon 3.2 GHz
	Memory	2 GByte
	Network	Gigabit Ethernet
	Node	4
Linux	OS	Linux kernel 2.6.9
	CPU	Xeon 2.4 GHz
	Memory	1 GByte
	Network	Gigabit Ethernet
	Node	4
VirtualBox	Version	1.3.8
	OS	Linux kernel 2.6.11
	Memory	1 GByte
	Network	TAP ドライバを用いたブリッジ接続
	Node	4

仮想マシンとして VirtualBox を用いているが，これは，我々が想定する環境では Windows 上で扱える仮想マシンでなければならいためである．また，Windows 上で利用できる仮想マシンとして VMware といった商用仮想マシンがあるが，商用仮想マシンを利用しないのは，測定結果の使用に関して制約があるためである．VirtualBox のような測定結果などに制約のない仮想マシンでは，ブリッジ接続によりネットワーク通信を行う際に，そのほとんどには仮想 NIC として TAP ドライバを利用している．このため，他の仮想マシン環境においても同様の結果が得られ，仮想マシン環境により性能値に差異はないと考えられる．

6.1 評価環境

グリッド環境での性能評価は Windows と Linux のヘテロ OS グリッド環境で行う．マスタは Linux のみ，ワーカは Windows と Linux を用いる．評価環境は**表 1** にマスタ環境，**表 2** にワーカ環境とそれぞれ使用したノード数を示す．VirtualBox と Cygwin 環境はそれぞれワーカ環境の Window 上に構築する．

6.2 システムコールの性能評価

BEE の基本的な性能評価として，write() と read() システムコールの実行性能について測定した．測定は

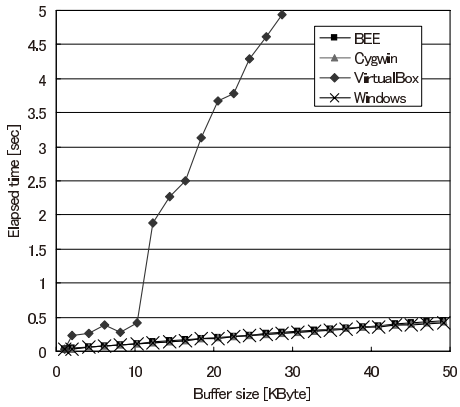

図 6 wirte() システムコールの実行時間
Fig. 6 Execution time of write system call.

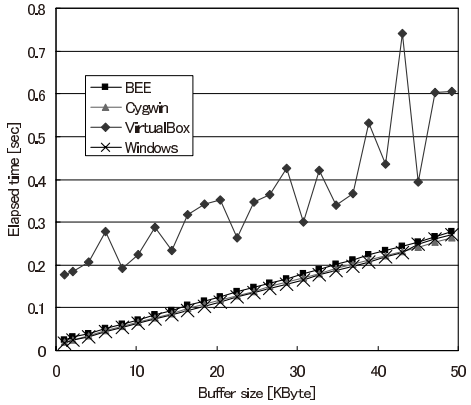

図 7 read() システムコールの実行時間
Fig. 7 Execution time of read system call.

バッファサイズを変化させ，それぞれのバッファサイズにおいて 1 万回のシステムコールを実行するのに要する時間を比較する．性能の基準として Windows のネイティブプログラムでの測定も行う．

図 6 と**図 7** にそれぞれの環境における wirte() と read() のシステムコールの測定結果を示す．この結果より，BEE におけるシステムコールの実行性能は Windows のネイティブプログラム，Cygwin とほぼ同程度の性能が得られた．しかし，BEE における 1 回のシステムコールを発行するのに要する時間は，平均で Windows のネイティブプログラムと比較して 0.6 μsec，Cygwin と比較して 0.5 μsec ほど遅い．この時間差は Linux システムコールのために新たに追加した割込み処理によるオーバヘッドであると考えられる．BEE では Linux システムコールが実行されるたびに，Linux システムコールと Windows API によるエミュレーションでの 2 回の割込み処理が要求されるためである．Cygwin と Windows ネイティブプログラムとの実行時間の差は，Cygwin での Linux システムコールをソフトウェアエミュレーションに要する処理時間である．VirtualBox のシステムコールの実行性能は，システム環境の仮想化を行うために，他の環境に比べて実行性能はそれほど得ることができないと考えられる．また実行性能は不安定である．特に，wirte() システムコールの場合では，バッファサイズが 10 KByte 以上では大きく性能が低下する．

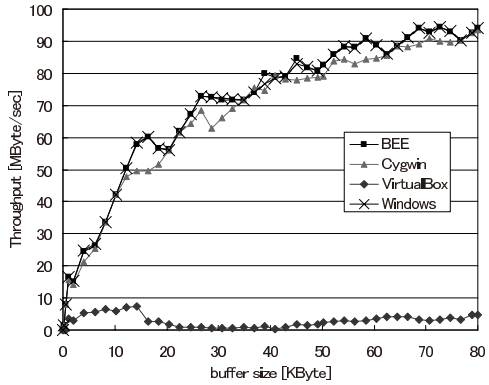

図 8 メッセージの ping-pong による TCP の通信性能
Fig. 8 Communication performance with ping-pong.

6.3 通信の性能評価

グリッド RPC の性能の重要な要素であるマスタ/ワーカ間の通信性能について評価した．我々が想定する環境は Linux をマスタとし，Windows はワーカとしてのみ利用する環境であるので，通信性能の評価は Linux と各評価環境間で行う．性能の基準として Windows のネイティブプログラムを用いての測定も行う．

図 8 に各環境の性能を示す．測定はノード間でメッセージの ping-pong を行い，バッファサイズを変化させた際の実行時間からからスループットを算出する．この結果から，BEE の通信性能は Windows のネイティブプログラムと同程度の性能が得られ，最大で 93.9 MByte/sec であった．Cygwin の最大スループットは 93.6 MByte/sec であり，メッセージサイズが大きい場合に BEE よりも若干性能が下がる結果となった．原因として，今回使用した測定プログラムでの通信処理は，Linux でのネットワーク通信処理で一般的に用いられている write() や read() システムコールにより行っている．これを Cygwin 上でコンパイルすると write() や read() システムコールは writev(), readv() システムコールに置き換えられる．これらのシステムコールの Cygwin での実装は内部でメモリコピーを行うため，このような性能低下が発生する

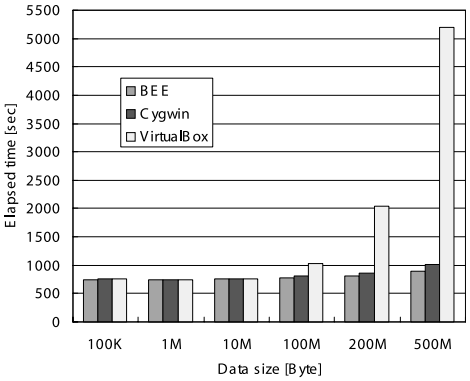

図 9 OmniRPC の基本性能評価プログラムによる実行時間
Fig. 9 Execution time of a synthetic program.

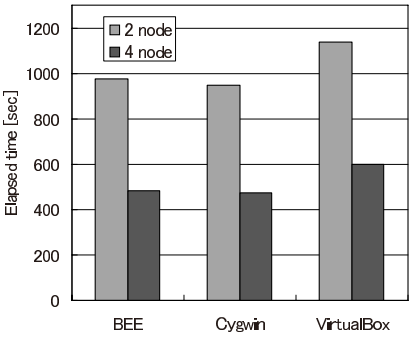

図 10 並列固有値計算プログラムの実行時間
Fig. 10 Execution time of parallel master-worker eigenvalue solver.

と考えられる．VirtualBox でのスループットは最大で 7.3 MByte/sec しか得ることができず，また，バッファサイズが 15 KByte 以上ではほとんど性能を得ることができなかった．

6.4 OmniRPC を用いた基本性能評価

BEE を用いたヘテロ OS グリッド環境において，基本的な RPC モデルのアプリケーションをシミュレーションにより性能評価を行う．この測定で用いるグリッド RPC の 1 ジョブは，ワーカはあるサイズの初期データを受け取り，処理を行うと仮定したジョブである．実際にはワーカ上での処理は sleep() により一定時間待機する．このプログラムを用いて初期データを変化させたときの実行時間を測定する．測定はワーカ上での待機時間を 30 秒で固定とし，ジョブ数を 100 とする．

図 9 に Windows を 4 ノード用いたときのそれぞれの環境での実行時間を示す．理想値ではデータサイズにより実行時間の変化はない．しかし，結果より初期データサイズが大きくなるにつれ実行時間も増加している．これはワーカへのデータ転送に多くの時間を費やしているためである．また，このようにデータ転送がボトルネックになるとすべてのノードを使用できなくなるという問題も発生する．OmniRPC のジョブスケジューラは処理を行っていないワーカに対してジョブを発行する．現在実行するジョブをあるノードに割り当て，そのジョブが次のジョブが発行する前に終了した場合，次のジョブは同じノードに対して割り当てられる．このため，同じノードに対してジョブを割り当て続けるため，すべてのノードが利用されなくなり，結果的に全体の実行性能が低下してしまう．このように，データ転送にかかる通信時間が全体の性能に関わるため，通信性能が重要となる．本測定においても 200 MByte 以下ではすべての環境において 4 ノードすべてが利用されていたが，それ以上のデータサイズでは BEE や Cygwin 環境では 3 ノードしか利用されなかった．また VirtualBox の場合，特に通信性能が他の環境に比べて低いため，200 MByte 以上では 2 ノードのみしか利用することができず性能は大きく低下している

6.5 実アプリケーションによる性能評価

実アプリケーションによる性能評価として，並列固有値計算プログラム[13] を用いて性能評価を行った．このプログラムは，大規模な一般固有値問題のための解法であり，複素空間上の円周上の点に対応する方程式を並列に解くことによりその円周内にある固有値を効率的に求めるというものである．この固有値計算プログラムのジョブ数は 40 である．性能評価は，それぞれの比較環境を 2 ノード，4 ノードを使用したときで行った．

図 10 に各々の環境での実行時間を示す．BEE の実行時間は Cygwin よりも若干遅い結果となった．この原因として，いくつかの要因がある．まず，BEE による Linux プログラムのロードにかかる処理である．この並列固有値プログラムの場合，BEE がワーカプログラムをロードするまでに約 0.5 msec の時間を要している．次に通信の際のオーバヘッドである．OmniRPC のデータ転送は，データサイズにかかわらず 1 KByte 単位に分割してデータ転送を行っている．1 KByte でのデータ転送では，通信性能は図 8 に示すように BEE と Cygwin は同程度の性能である．しかし，システムコールを実行する際に先に述べたようなソフトウェア割込みのオーバヘッドがある．この実アプリケーションにおいて 1 ジョブごとに初期データサイズとして約 55 MByte のデータ転送があるが，OmniRPC ではこのデータを 1 KByte 単位に分割する．そのため，多くのシステムコールを発行しなけれ

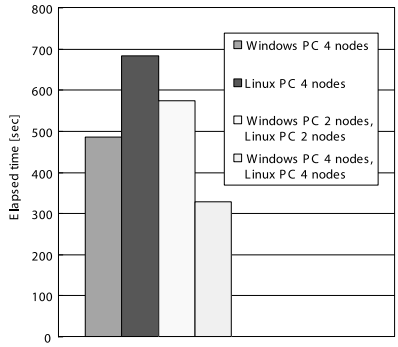

図 11 Linux と Windows のヘテロ OS 環境での並列固有値計算プログラムの実行時間
Fig. 11 Execution time of parallel eigenvalue solver in a hybrid environment which contains Windows and Linux machines.

ばならず，これが性能低下の要因になっていると考えられる．VirtualBox では，他の環境と比べ通信性能が低いため，マスタとの通信がボトルネックとなり，他の環境に比べ実行性能は得られないと思われる[⋆1]．

6.6 ヘテロ OS 環境での機能評価

並列固有値計算プログラムを用いて Linux と Windows のヘテロ OS グリッド環境での性能評価を行った．**図 11** に Linux と Windows をそれぞれ 2 ノード，4 ノードを使用したヘテロ OS 環境のときの実行時間を示す．Linux と Windows ではマシン構成が異なるため，同じノード数を用いても実行時間は異なる．この実験より重要な点は Windows と Linux を用いたヘテロ OS グリッド環境において，BEE を用いることで Windows をグリッド RPC のワーカとして利用することが可能であるということである．この結果より，ヘテロ OS 環境において Windows がワーカとして利用できていることが確認できる．

7. まとめと今後の課題

本稿では Linux と Windows のヘテロ OS グリッド環境において Windows をグリッドの計算資源とするために，グリッド RPC のワーカとして利用することを想定し，Linux プログラムをいっさい変更することなく Windows 上で実行することができる軽量な Linux プログラム実行システム BEE の設計・実装を行った．BEE をグリッド RPC 環境で扱うために OmniRPC に組み込むことで，Windows をグリッド RPC ワーカとして利用することが可能となった．BEE の性能評価の結果，システムコールや通信性能は Windows のネイティブプログラムと同程度の性能が得られた．また，Linux と Windows からなるヘテロ OS グリッド環境において Windows をグリッド RPC のワーカとして利用できることを確認した．

今後の課題として次のことを検討している．

- 現在，BEE は Linux カーネル 2.4 で作成されたプログラムをサポートしている．Linux カーネル 2.6 以降では NPTL（Native POSIX Threading Library）[7] という新しいスレッドモデルや TLS（Thread Local Storage）のサポートにより BEE 上で実行する際にいくつか問題がある．現在，カーネル 2.6 上で作成されたプログラムをサポートを行えるようにしている．
- Linux のシステムコールは *int 0x80* による割込みだけでなく *sysenter* も使用されている．*sysenter* によるシステムコールも *int 0x80* と同様にデバイスドライバを用いることでシステムコールをフックすることが可能である．しかし，Windows も *sysenter* によりシステムコールが実行されているため，単純に *int 0x80* による割込み処理と同様の手法で行うことができない．*sysenter* による Linux システムコールについて検討を行っている．
- 本稿ではオフィスや研究室の Windows PC を計算資源として利用することを目的としているが，このような PC を用いた計算環境では，ノードは動的に参加や離脱を繰り返すという環境になる．BEE では Linux と Windows 間で共通のワーカプログラムを用いることができる．そこで，BEE を用いてこのような動的な環境に対応する Windows と Linux 間でのプロセスマイグレーションを検討している．

謝辞 本研究の一部は，文部科学省科学研究費補助金基盤研究 A 課題番号 172002，特別研究員奨励費課題番号 177324，および，日仏共同研究プログラム（SAKURA）による．

参 考 文 献

1) Boehme, C., Ehlers, T., Engelhardt, J., Felix, A., Haan, O., Kalman, T., Neumair, B., Schwardmann, U. and Sommerfeld, D.: Instant-Grid: Fully Automated Middleware-Deployment Using a Live-CD, *ICNS '06: Proc. International conference on Networking and Services*, Washington, DC, USA, p.70, IEEE Computer Society (2006).
2) Bull, J.M., Smith, L.A., Pottage, L. and Freeman, R.: Benchmarking Java against C

⋆1 いくつかの商用 VM においても同様の結果となった．

and Fortran for scientific applications., *Java Grande*, pp.97–105 (2001).
3) cooperative Linux. http://www.colinux.org/
4) Cygwin. http://sourceware.org/cygwin/
5) Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. and Tuecke, S.: A Resource Management Architecture for Metacomputing Systems, Lecture Notes in Computer Science, Vol.1459, pp.62–82 (1998).
6) Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P. and Neugebauer, R.: Xen and the Art of Virtualization, *Proc. ACM Symposium on Operating Systems Principles* (2003).
7) Drepper, U. and Molnar, I.: The Native POSIX Thread Library for Linux. http://people.redhat.com/drepper/nptl-design.pdf
8) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *The International Journal of Supercomputer Applications and High Performance Computing*, Vol.11, No.2, pp.115–128 (1997).
9) globus alliance, T.. http://www.globus.org/
10) innotek VirtualBox. http://www.virtualbox.org/
11) Nakajima, Y., Sato, M., Boku, T., Takahashi, D. and Gotoh, H.: Performance Evaluation of OmniRPC in a Grid Environment, *SAINT Workshops*, pp.658–665 (2004).
12) OmniRPC. http://www.omni.hpcc.jp/OmniRPC/
13) Sakurai, T., Hayakawa, K., Sato, M. and Takahashi, D.: A Parallel Method for Large Sparse Generalized Eigenvalue Problems by OmniRPC in a Grid Environment, *PARA*, pp.1151–1158 (2004).
14) Sato, M., Boku, T. and Takahashi, D.: OmniRPC: a Grid RPC system for Parallel Programming in Cluster and Grid Environment, *CCGRID*, pp.206–213 (2003).
15) Sugerman, J., Venkitachalam, G. and Lim, B.-H.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, *Proc. General Track: 2002 USENIX Annual Technical Conference*, Berkeley, CA, USA, pp.1–14, USENIX Association (2001).
16) WineHQ. http://www.winehq.com/

(平成 19 年 7 月 23 日受付)
(平成 19 年 11 月 16 日採録)

上村 佳史

昭和 59 年生．平成 18 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科在学中．グリッドコンピューティング等に関する研究に従事．

中島 佳宏

昭和 55 年生．平成 15 年筑波大学第三学群情報学類卒業．平成 18 年日本学術振興会特別研究員．現在，同大学大学院システム情報工学研究科在学中．グリッドコンピューティング等に関する研究に従事．

佐藤 三久（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年，通産省電子技術総合研究所入所．平成 8 年，新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より，筑波大学システム情報工学研究科教授．同大学計算科学研究センター勤務．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．情報処理学会，IEEE，日本応用数理学会会員．