Direct Execution of Linux Binary on Windows for Grid RPC Workers

Yoshifumi Uemura, Yoshihiro Nakajima and Mitsuhisa Sato
Graduate School of Systems and Information Engineering
University of Tsukuba
{uemura, ynaka, msato}@hpcs.cs.tsukuba.ac.jp

Abstract— **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 and Cygwin and is better that that of using VMware. We have demonstrated a hybrid grid environment combining Window PCs in a conventional grid of Linux nodes.**

I. Introduction

Nowadays, local area or campus-type networks consist of PCs using different operating systems, including Windows and Linux. With 100 to 300 PCs in an average LAN, this presents enormous potential computing power. On the other hand, advances in wide-area networking technology and infrastructure have made it possible to construct large-scale, high-performance distributed computing environments, or computational grids, that provide dependable, consistent and pervasive access to enormous computational resources. To build a large-scale grid environment, PCs in offices and laboratories are expected to be used as computational resources in addition to dedicated conventional PC clusters.

In this type of environment, the challenge is to absorb the heterogeneity of operating systems. PCs run under various OSs, including Windows and Linux. The majority of PCs in offices and laboratories are run on Windows, while grid applications and middleware are often developed on Linux. Furthermore, in most cases, individual PCs cannot be completely dedicated to computation. One of the simplest solutions is the use of dual-boot installation or Linux LiveCD. The concept here is to temporarily convert the PC into a Linux system in order to join the grid when the PC is idle for a long time, for example, overnight. This full migration to Linux may not be allowed because Windows users do not want to change the system configuration by dual boot and want to continue working in a familiar environment, using office software or performing other tasks. Another solution is to install an agent or a small application on a Windows PC, which sits in the system tray or as a Windows service working for the grid.

In the present paper, we introduce an agent called BEE that enables direct execution of Linux binary programs on Windows for Grid RPC workers. The Grid enabled Remote Procedure Call (Grid RPC) has been proposed as a programming model for grid computing. The RPC-style system is particularly useful in that it provides an easy-to-use, intuitive programming interface, allowing users of a grid system to easily make grid-enabled applications. This model supports a typical master-worker type of parallelism, dispatching jobs to each worker by asynchronous RPC calls. We have integrated BEE into the OmniRPC system [1] in order to make use of Windows PCs as computing resources in a hybrid grid environment combining Windows PCs into conventional grid resources. The BEE agent allows Linux binaries of the OmniRPC worker program to be exported and run under Windows without any modification of its Linux binaries. The BEE agent is controlled by the master running on Linux (the master node).

BEE loads a Linux binary program in Windows and executes it directly, emulating a few system calls required for a Grid RPC worker such as network I/O operations. This differs from the virtual machine technology in that it does not emulate an entire Linux kernel. BEE provides lightweight and efficient I/O operations because the I/O operations are performed by Windows native I/O. Although a software driver is required to emulate Linux system calls, BEE is easy to install and remove, which can be accomplished in minutes.

The remainder of the present paper is organized as follows: Section 2 describes the design choices used to construct a hybrid grid environment combining Windows PCs as computing resources. Section 3 describes the design and implementation of BEE. Section 4 describes an overview of the OmniRPC system that we have used as a Grid RPC system and the integration of BEE into the OmniRPC system. The performance evaluation is presented in Section 5. Finally, conclusions and future areas for consideration are presented in Section 6.

1-4244-0910-1/07/$20.00 ©2007 IEEE

II. Hybrid Grid Environment: Executing Linux programs on Windows

Figure 1 shows an overview of a hybrid grid environment. Our objective is to utilize PCs running Windows in offices and laboratories as potential grid computational resources (workers) in addition to Linux clusters and PCs. The master uses Linux only because several grid applications are developed on Linux. The proposed Grid RPC system, OmniRPC, has also been developed in a Linux environment. Therefore, in order to make use of Windows PCs as computational resources in a Grid RPC system, we need to run Grid RPC worker programs, which are usually generated in a Linux node, on Windows PCs. In this section, we discuss how to run the worker programs in Windows.

For the present purpose, the following are important:

- Flexibility: Windows users (PC owners) can work as usual while the worker program runs. If possible, the program should run with low priority, or only when the PC is idle.
- Efficiency and Performance: When running the worker program, the execution should be nearly as fast as native ported binary.
- Manageability: The user can setup the environment and prepare the program running in Windows with minimal cost.

The following are existing solutions for running programs imported from Linux in Windows:

- Using cross-platform compilers such as Cygwin[2].
- Using virtual machine software such as VMware[3].

Cygwin is a collection of free software that consists of a library-implemented Linux system call APIs and a large number of application programs, including compilers, which provide a Linux-like environment equivalent to common programs on Linux. Since Cygwin provides an environment that is compatible with Linux, the source code of the Linux program can be executed by recompiling the code. When using Cygwin for a hybrid grid environment, the user has to prepare binaries of a worker program for both Linux and Windows. The program should be recompiled by Cygwin to make its Windows binary separately. In addition, in order to deploy remote worker programs for several Windows PCs, some deployment mechanism of Grid RPC workers is required for Windows as well as the Cygwin runtime system.

A virtual machine is software that provides a virtualized environment on one computer platform. In the present paper, we use the term "virtual machine" as a hardware virtual machine. This virtualization environment enables an OS environment to be run under different OSs. When running Linux on the Windows virtual machine, this software can act as a bridge on the Linux kernel and can run on top of Windows. Therefore, by installing the Linux OS on the virtual machine, the machine can execute the worker programs imported from Linux directly in Windows.

There are several kinds of virtual machine software such as VMware [3] [4], QEMU [5] [6] and Xen [7] [8]. Consider

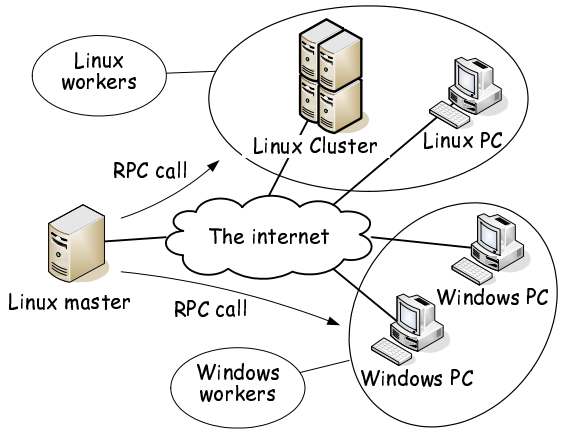

Fig. 1. Overview of a hybrid grid environment

VMware as an example. VMware is a full system emulator, so that it can emulate a complete system with full compatibility. Most systems designed for Linux can be executed in a virtual machine on the binary level. Since the emulator runs in a memory address space separate from the other host applications, any crash of the guest system will not affect the host. On the other hand, the VMware drains much of system resources and requires a huge amount of memory and intercepts several cycles. In addition, VMware requires full installation and configuration of a Linux kernel, which required significant management and installation costs. To run a PC as a Grid RPC worker requires a few system calls such as network I/O operations. In addition, VMware is a commercial product, and all Windows PCs will need licenses, which will require additional cost.

In the present paper, we propose an agent, called BEE, which enables direct execution of a Linux binary program exported from Linux on Windows. This does not require any recompilation for Windows, as in Cygwin, because the Linux binary is executed directly on Windows. In addition, a full installation of the Linux kernel, as in VMware, is not required. Only a few system calls required to execute a Grid RPC worker are emulated in BEE with a small Windows kernel driver.

III. Design and Implementation of the BEE agent

For a hybrid grid environment with Windows and Linux, we have implemented an agent called BEE for the Linux binary program execution environment to utilize Windows PCs as workers of the Grid RPC system. In this section, we describe the design and implementation of the BEE agent.

Although there are several types of system call in Linux, a sufficient number of system calls are supported to run a Grid RPC worker. Therefore, the target architecture of BEE is limited to Intel IA-32 architecture because most versions of Windows run on this architecture. Since both operating systems are run on the same instruction architecture, the binary program should run natively on the machine, except for the system calls of each operating system. We have implemented

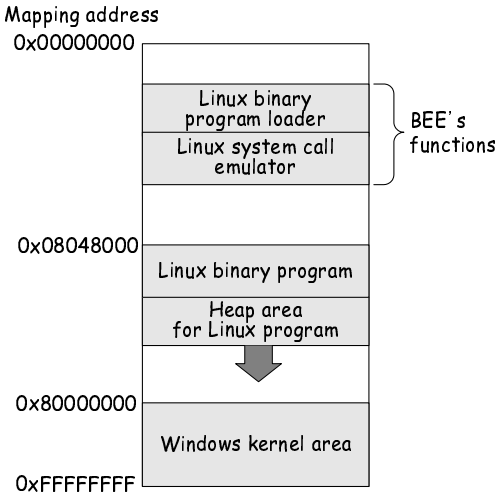

Fig. 2. BEE memory map on Windows

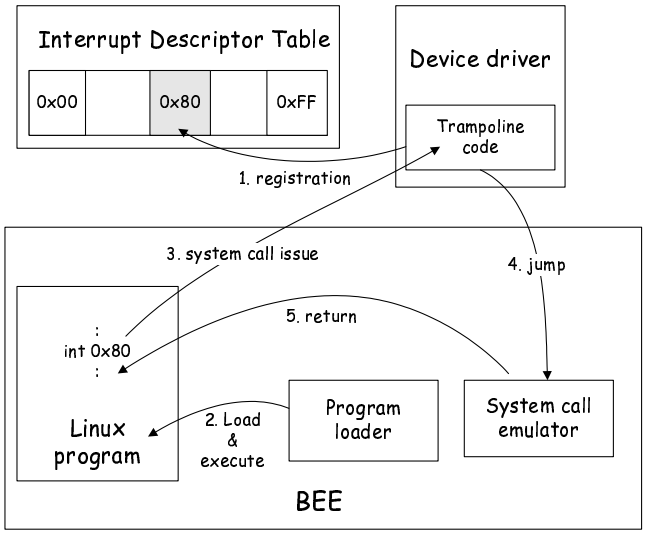

Fig. 3. Execution mechanism with BEE agent

a system call emulator to absorb the difference in operating systems.

The BEE has two functions: the binary program loader and the system call emulator.

A. Binary Program Loader

The binary program loader is a function by which to load a Linux binary program into the Windows memory address space. While Linux uses various program formats, such as the Executable and Linking Format (ELF) [9], Windows uses program formats, such as the Portable Executable (PE) format. By this function, the Linux program is executed as a process of Windows.

This function analyzes the Linux program format to load on Windows. The BEE supports the ELF binary, which is a standard program format in Linux. We assume that the Linux binary is statically linked because we do not support the shared libraries in the current implementation. Since Grid RPC workers are executed across the network, the shared libraries would be required to be transferred over the network. The version of libraries is occasionally different when binaries are compiled in different Linux nodes. This may be troublesome in several cases.

To run a Linux program under Windows, the binary program loader loads a Linux binary program into its own memory space. Figure 2 shows the memory map of BEE. The program loader maps the ELF executable at the address starting from the standard address 0x8048000. To avoid address conflicts between the program loader and the ELF binary program, the program loader is mapped at a lower address than that in which the binary is loaded.

Once the binary is loaded, control is transferred into the Linux program entry point to start the execution. During execution, the Linux heap area is allocated by the brk() system call. The Linux system calls are emulated by BEE, so that its heap area is allocated just after the address to which the ELF executable code is loaded.

B. System Call Emulator

As long as we use the same IA-32 architecture, with the exception of the system calls, the Linux program should run natively even on Windows. The system call emulator is a function by which to emulate Linux system calls. The BEE supports only I/O system calls required for Grid RPC workers.

The system call emulator function is implemented using a software interrupt, which transfers control of the kernel code corresponding to the issued system call. The implementation of system calls is different in Windows and Linux. While Windows uses the software interrupt of *int 0x2E* at NT/2000 and *sysenter* at XP to implement the system call, Linux uses the software interrupt of *int 0x80*.

Each software interrupt is associated with interrupt handlers, which are routines that take control when an interrupt occurs. These interrupt handlers are registered in the table supported by the IA-32 architecture, called the Interrupt Descriptor Table (IDT) [10]. Fortunately, Windows does not use the software interrupt associated *int 0x80* used for Linux. Therefore, BEE registers an interrupt handler associated *int 0x80* to emulate Linux system calls in Windows. The BEE uses a simple device driver to add the interrupt handler for the software interrupt because the handler code is executed in kernel space. The device driver is installed in advance before the BEE is executed.

The device driver for the software interrupt of the Linux system calls executes a simple trampoline code, which catches a software interrupt from the Linux system calls and jumps back to the system call emulation code in the user address space in BEE. The emulation of the Linux system calls is implemented using Windows APIs. Although there are several types of system calls in Linux, a sufficient number of system calls are supported in order to run a Grid RPC worker.

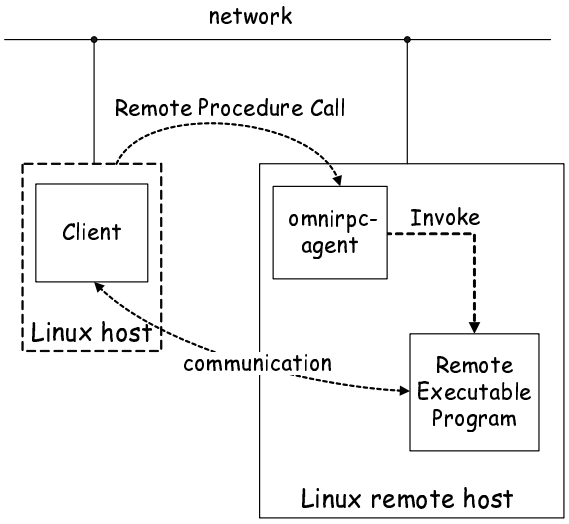

Fig. 4. Original OmniRPC in Linux

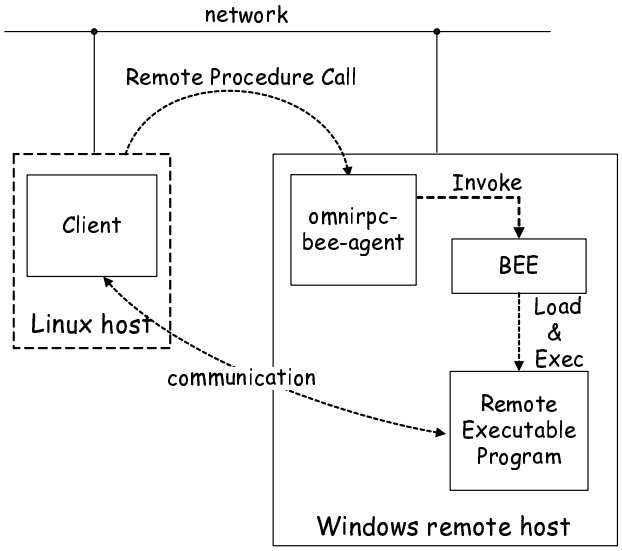

Fig. 5. RPC system of integration of BEE into OmniRPC

Currently, we have implemented file I/O, such as write() and read(), and network I/O, such as send() and recv(). After executing the called system call, control transfers back directly to the point at which the software interrupt occurred.

C. Executing the Linux Binary Program

Figure 3 shows the flow of execution of the binary program loader and the system call emulator in BEE. BEE executes the Linux binary program in following steps:

1) Before executing the Linux binary program, the interrupt handler for the software interrupt *0x80* of the Linux system is called by using the device driver in the IDT.
2) The binary program loader loads the specified binary into its own address space and runs as a Windows process.
3) When a system call occurs during execution, the registered interrupt handler is invoked.
4) The interrupt handler returns the control to the system call emulator in BEE.
5) BEE emulates the system call and returns control to the caller of the system call to continue the execution.
6) When the exit() system call is called, BEE terminates.

IV. Integration of BEE into the OmniRPC system

We have integrated BEE into the proposed Grid RPC system, OmniRPC, to execute OmniRPC workers in Windows PCs. The BEE agent enables the OmniRPC application to use Windows PCs as computational resources in a hybrid grid environment.

The OmniRPC system [1] [11] [12] provides seamless parallel programming for a local cluster to multi-cluster in a grid environment and supports typical master/worker grid applications such as parametric execution.

Figure 4 shows an overview of the RPC mechanism of OmniRPC in conventional Linux nodes. The OmniRPC system consists of three types of components: the client, the remote executable program, and the OmniRPC agent (omnirpc-agent). The client is a master program and dispatches calculations to worker. The remote executable program is a worker program and executes calculation on a remote host when the master program dispatches the calculations to it. The agent bridges between master and workers and invokes the remote executable programs as a worker. The agent may also work as a proxy of the communication between client and remote executable programs.

In the OmniRPC system, the master does not execute a remote executable program on remote host directly. The master invokes the agent with appropriate authentication, such as "ssh" or globus GRAM, at the beginning of Grid RPC applications and asks the agent to invoke a remote executable program as a worker in a remote host.

Figure 5 shows the configuration of the OmniRPC system with BEE on Windows. We make use of a Windows PC as a worker controlled by a master running on Linux. In the case of the Windows PC, the agent of Windows is invoked manually in advance as a daemon. We have extended the OmniRPC to support such a configuration.

The target resources of the original OmniRPC system are PC clusters in a grid environment. Within each cluster, files such as remote executable programs are generally shared, so that deployment is needed for the master node of each cluster. In a large hybrid grid environment, however, it is assumed that files are not shared between PCs in offices and laboratories. Therefore, the Grid RPC system is required to provide a deployment mechanism of remote executable programs for each worker. For OmniRPC worker integrated with BEE, we have extended the agent to receive executable files from the master at run-time. The master (client) automatically sends the executable files generated in Linux to the agent in Windows at the beginning of execution of grid applications.

TABLE I
RPC MASTER NODE

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

TABLE II
RPC WORKER NODE

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
VMware	Version	5.0.0
	OS	Linux kernel 2.6.11
	Memory	1 GByte
	Network	Bridge
	Node	4

V. PERFORMANCE OF HYBRID GRID ENVIRONMENT USING BEE

In this section, we report the basic performance of BEE and Grid RPC applications of a hybrid grid environment using BEE. To demonstrate its efficiency, we have compared the present approach with VMware as the virtual machine software and Cygwin as a cross-compiler environment. As a measure of basic performance, we have examined the performances of system calls and communication, and a synthetic program, which models a typical Grid RPC application. We have also examined the performance of a realistic application in a hybrid grid environment.

A. Experimental Settings

For the performance evaluation, we have a hybrid grid environment that consists of Windows and Linux PCs. Table I and Table II, respectively, show the master and worker configurations and the number of nodes used. For the measurement of the Windows environment, we used BEE and the Linux binary program. In addition, we set up both VMware and Cygwin environments in each Windows PC for comparison.

B. Performance of system calls

The performance of system calls is examined by measuring the time of write() and read() system calls. We changed the buffer size, and then measured the execution time required to execute the system calls 10,000 times. We also measured the time of Windows native system calls.

Figure 6 and Figure 7 show the execution times of write() and read() system calls, respectively, in each configuration.

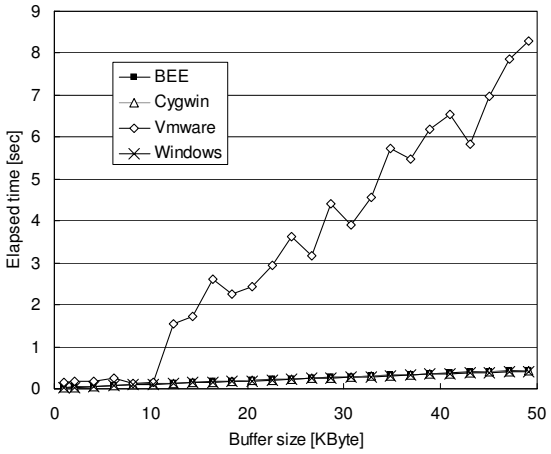

Fig. 6. Execution time of write system calls

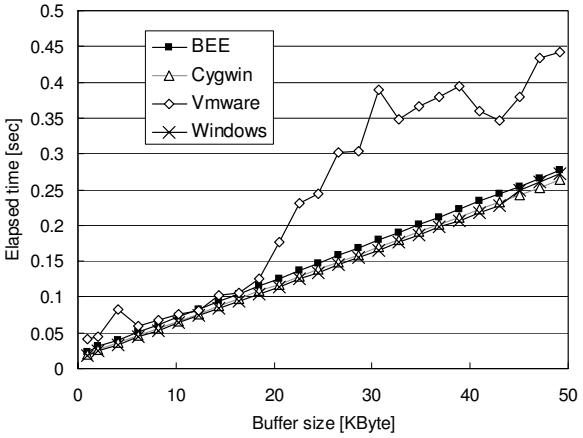

Fig. 7. Execution time of read system call

The result shows that the execution time of system calls in BEE is approximately the same as that of Windows native system calls and Cygwin. The execution time of one system call of BEE is approximately 0.6 μ sec slower than that of the Windows native system call and approximately 0.5 μ sec slower than that of Cygwin. This overhead is a result of the time for the software interrupt because the BEE requires twice the number of software interrupts in the system call emulator to execute Linux system calls. The execution time of Cygwin is approximately 0.1 μsec slower than that of Windows native system calls, because Cygwin requires extra time to emulate Linux for system calls. The performance of system calls of VMware is approximately the same as those of other configurations when the buffer size is small. However, the VMware performance decreases dramatically when the buffer size exceeds 10 KBytes for the write() system call and 20 KBytes for the read() system call.

C. Performance of communication

The communication performance between master and workers is one of the most important performance measures in Grid

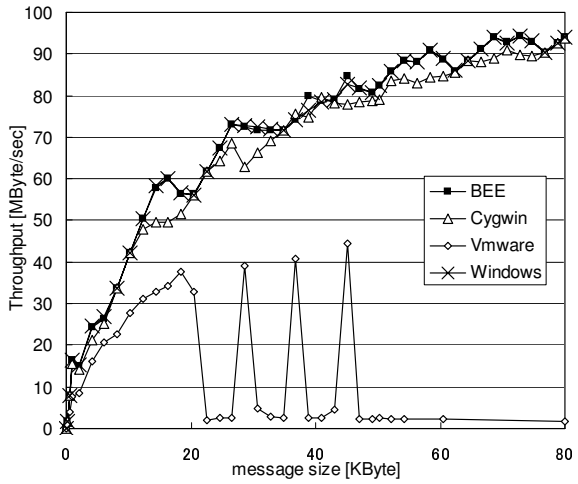

Fig. 8. Communication performance

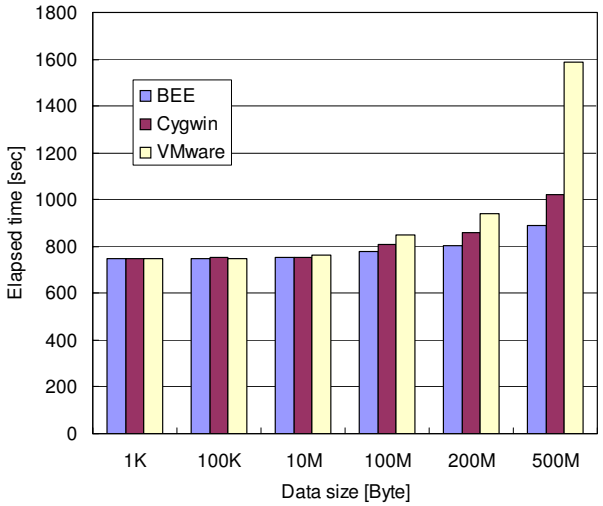

Fig. 9. Execution time of the synthetic program

RPC. In this experiment, the destination of communication is the master node of Linux, because the proposed Grid RPC applications run the master program in a Linux node, and Windows PCs are used only for the workers. We measured the communication performance between master and workers in each configuration. We also measured the Windows native communication performance for the comparison.

Figure 8 shows the TCP throughput in each configuration. We changed the message size and then measured the execution time of ping-pong for the message size. The throughput is calculated by the execution time.

The result shows that the communication performance of BEE is as good as that of Windows native program, and the maximum communication performance is 93.9 MByte/sec. The communication performance of Cygwin is 93.6 MByte/sec and becomes slightly worse when the message size becomes large because the communication of Cygwin requires memory copy. The communication routine in this experiment uses write() and read() system calls, as are generally used in Linux. When the program is compiled with Cygwin, write() and read() system calls are executed by writev() and readv() system calls, respectively. These system calls in Cygwin perform memory copy.

The communication performance of VMware is not so bad when the message size is small. However, when the message size exceeds 20 KBytes, its performance decreases dramatically and finally freezes due to unexpected errors.

D. Performance using a synthetic workload model

To measure the basic performance of a hybrid grid using BEE, we use a synthetic program that models RPC applications. This program receives a certain amount of initial data and executes several RPCs. We changed the size of the initial data from 1 Kbytes to 500 Mbytes and used the sleep API to simulate the RPC jobs. In the experiment, we set the computation time of workers on remote hosts to 30 seconds in the synthetic program. The master issues RPC calls a total of 100 times.

Figure 9 shows the execution time of the synthetic program when using four nodes in each configuration. The execution time of each configuration increases when the data size becomes larger, for up to 500 MBytes, because data transfer requires a certain amount of time. In the OmniRPC system, the RPC requests are scheduled on demand. If the previous job is finished before the next RPC is issued, the same worker is used for the next RPC job. If the data transfer becomes a bottleneck, then occasionally not all of the nodes are used. We observed that when the data size is less than 200 MBytes, all four nodes were used. When the data size is 500 MBytes, only three nodes were used in BEE and Cygwin, and two nodes were used in the VMware configuration because the communication performance of VMware is much lower than those of other configurations.

E. Performance for a realistic application

To examine the performance of a realistic application, we measured the performance of a parallel master-worker eigenvalue solver [13].

This program can solve a large-scale eigenvalue problem in parallel by solving the equation corresponding to the point on the circumference in the complex space. Small matrices for these points are transferred to workers in order to solve a small subprogram independently, and then these solutions of subproblems are gathered to calculate the final solution. The program finds all of the eigenvalues that lie inside a given domain. In this experiment, we compare the performance of two nodes and four nodes of Windows PCs using BEE. The number of the eigenvalue computation jobs is 40.

Figure 10 shows the execution time in each configuration. The result shows that the execution time of BEE is slightly slower than that of Cygwin. In the communication of OmniRPC system, the data transfer is performed by dividing 1-

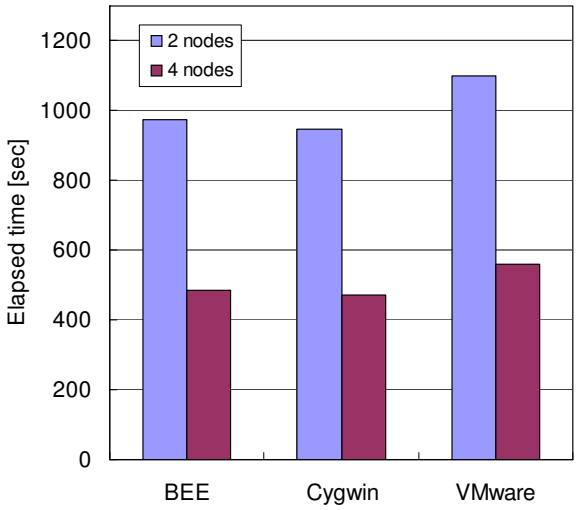

Fig. 10. Execution time of parallel master-worker eigenvalue solver

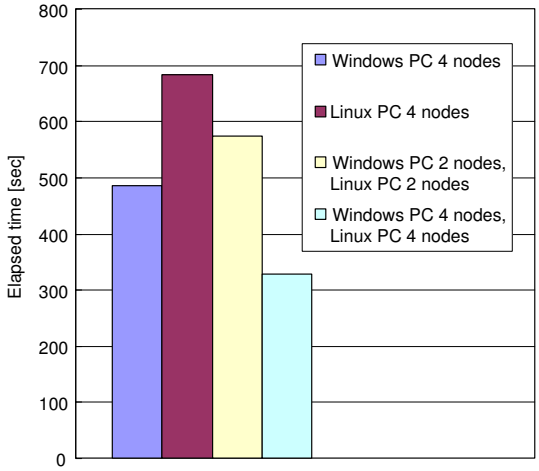

Fig. 11. Execution time of parallel eigenvalue calculation in a hybrid environment that contains Windows and Linux machines

Kbyte packets. As shown in Figure 8, the communication performance of BEE is approximately the same as that of Cygwin. Since BEE has the extra overhead of twice the number of software interrupts at the system call, its overhead is a big factor when the data size is so small. We found that in such realistic applications, many mmap() system calls are used to reserve memory area during the execution of worker programs. The mmap() system call requires the overhead of software interrupts for system calls. In BEE, loading the binary program so that the mmap() system call is also used for loading the program also takes some time. In this experiment, memory allocation for loading requires approximately 0.5 msec.

The execution time of VMware is slower than those of BEE and Cygwin due to its slow communication performance.

F. Performance of a hybrid grid environment

We examined the performance of a hybrid grid environment when using Linux and Windows PCs at the same time. As in the previous subsection, we used a parallel eigenvalue solver. Figure 11 shows the execution time when workers used two nodes and used four nodes respectively. For the comparison, Fig. 11 also shows the execution time when using either four Windows PCs or four Linux PCs.

Note that the performance of Windows PCs and Linux PCs are different. The CPU of the Windows PC is Xeon 3.2 GHz, and the CPU of Linux PC is 2.4 GHz. Therefore, the configuration using two nodes of Windows PCs and two nodes of Linux PCs is slower that of four nodes of Linux PCs. If these PCs have the same performance, then the performance of these configurations will be approximately the same.

This experiment demonstrates that we can combine Windows PCs and Linux PCs to realize a hybrid grid environment.

VI. Conclusion and Future works

In the present paper, we proposed a mechanism that utilizes Windows PCs as workers of a Grid RPC system in a hybrid grid environment that contains Windows and Linux PCs. We have designed and implemented an agent called BEE that consists of a binary program loader and a system call emulator to enable a worker program imported from Linux to run under Windows. The BEE enables a Linux binary program to be executed directly and efficiently under Windows without any recompilation. We have integrated BEE with the proposed Grid RPC system, OmniRPC, to realize a Grid RPC system for a hybrid grid environment.

We evaluated the basic performance of BEE, including system calls and communication, and the performance of realistic applications in a hybrid gird environment. We found that BEE provides nearly the same performance as Windows native and Cygwin because of direct execution of binary and the efficient implementation of system calls. The performance of BEE is more efficient that that of solution using virtual machine technology such as VMware. We have demonstrated that we can utilize Windows PCs as computing resources in a hybrid grid environment.

In the future, we intend to examine the following areas:

- BEE currently supports a program of Linux kernel 2.4. We discovered a number of problems in executing a binary program generated in Linux kernel 2.6. The new Linux kernel 2.6 implements a new thread model called Native POSIX Threading Library (NPTL) [14] and supports Thread Local Storage (TLS) [15]. Linux kernel 2.6 also introduces the TLS system calls for TLS. These system calls allow the allocation of a Global Descriptor Table (GDT), which is a CPU data structure entry that can be used to access memory, and this memory is accessed via a register such as *gs*. We are currently working on supporting BEE for a binary program generated in Linux kernel 2.6.
- The goal of the present research is the utilization of Windows PCs in offices and laboratories as computing

resources in a hybrid grid environment. These PCs are often volatile, so that we should investigate process migration between Windows and Linux on BEE with respect to such volatility.

ACKNOWLEDGMENT

The present study was supported in part by Grants-in-Aid from MEXT of Japan (No. 172002 and No. 177324) and by the Japan-France collaboration research program (SAKURA) of the JSPS.

REFERENCES

[1] OmniRPC, *http://www.omni.hpcc.jp/OmniRPC/*.
[2] Cygwin, *http://sourceware.org/cygwin/*.
[3] VMware, *http://www.vmware.com/*.
[4] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2001, pp. 1–14.
[5] QEMU, *http://www.qemu.com/*.
[6] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator ," in *Proceedings of USENIX 2005 Annual Technical Conference*, 2005, pp. 41–46.
[7] Xen, *http://www.xensource.com/*.
[8] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the Art of Virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003. [Online]. Available: citeseer.ist.psu.edu/dragovic03xen.html
[9] *TIS Committee: Tool Interface Standard(TIS) Executable and Linking Format(ELF) Specification version 1.2*, 1995. [Online]. Available: http://www.x86.org/ftp/manuals/tools/elf.pdf
[10] Intel(R), *IA-32 Intel(R) Architecture Software Developer's Manuals*, 2004.
[11] M. Sato, T. Boku, and D. Takahashi, "OmniRPC: a Grid RPC ystem for Parallel Programming in Cluster and Grid Environment," in *CCGRID*, 2003, pp. 206–.
[12] Y. Nakajima, M. Sato, T. Boku, D. Takahashi, and H. Gotoh, "Performance Evaluation of OmniRPC in a Grid Environment," in *SAINT Workshops*, 2004, pp. 658–665.
[13] T. Sakurai, K. Hayakawa, M. Sato, and D. Takahashi, "A parallel method for large sparse generalized eigenvalue problems by omnirpc in a grid environment." in *PARA*, 2004, pp. 1151–1158.
[14] U. Drepper and I. Molnar, "The Native POSIX Thread Library for Linux," *http://people.redhat.com/ drepper/nptl-design.pdf*, 2005.
[15] U. Drepper, "ELF handling for Thread-Local Storage," *http://people.redhat.com/drepper/tls.pdf*, 2005.