# A Virtual Machine Migration System Based on a CPU Emulator

Koichi Onoue
University of Tokyo
koichi@yl.is.s.u-tokyo.ac.jp

Yoshihiro Oyama
The University of Electro-Communications
oyama@cs.uec.ac.jp

Akinori Yonezawa
University of Tokyo
yonezawa@yl.is.s.u-tokyo.ac.jp

## Abstract

*Migration of virtual computing environments is a useful mechanism for advanced management of servers and utilization of a uniform computing environment on different machines. There have been a number of studies on migration of virtual computing environments based on virtual machine monitors (e.g., VMware) or language-level virtual machines (e.g., Java). However, migration systems based on a CPU emulator have not received much attention and their viability in a practical setting is not clear. In this paper, we describe Quasar, a virtual machine (VM) migration system implemented on top of the QEMU CPU emulator. Quasar can migrate a whole operating system between physical machines whose architectures are different (e.g., between an x86 machine and a PowerPC machine). Quasar provides a virtual networking facility, which allows migrating VMs to continue communication without disconnecting sockets for migration. Quasar also provides a staged migration function to reduce the downtime of migrating VMs. We have examined the viability of Quasar through experiments, in which Quasar was compared with Xen, SBUML, and UML. The experiments assessed the performance of virtual server hosting, the sizes of the files that represent VMs, and the amount of downtime for VM migration.*

## 1. Introduction

The technology of virtual machines has significantly developed, and the performance and functionalities of virtual machines have greatly improved. There are various kinds of virtual machine. One kind is the language VM, which provides an original instruction set (e.g., Java VM and .NET CLR). Another kind is the virtual machine monitor, which provides a virtualized and multiplexed form of underlying physical hardware (e.g., VMware, Xen [4], and Denali [19]). The other kind is the CPU emulator, which emulates real hardware as software (e.g., bochs and QEMU).

CPU emulators bring strong platform-independency and the ability to execute legacy binaries. They have a potential to be good building blocks of advanced software infrastructures. However, their viability for realistic purposes has not yet been fully studied. There are few case studies on applying CPU emulators to real-world problems. For example, as far as we know, no case studies exist on the performance of a server hosted on a CPU emulator.

In this paper, we propose Quasar, which is a virtual machine migration system implemented on top of a CPU emulator. It enables users to migrate a virtual computing environment to another physical machine while keeping its execution state intact. Since a CPU emulator strongly decouples software from hardware, its users can enjoy a uniform computing environment on a wide range of platforms. They can do their jobs anywhere in their virtual computing environment, which may be migrated between machines with different architectures, such as x86 and PowerPC. Users do not have to modify the code of the applications or the operating systems to execute them in a virtual computing environment.

Building virtual machine migration systems on top of CPU emulators provides several benefits. First, programs can run on various physical machines because CPU emulators can virtualize many CPU architectures and various peripherals. Most of the existing virtual machine monitors, e.g., VMware, require the CPU of the underlying physical machine to have the x86 architecture. On the other hand, Quasar enables programs developed for one architecture to run on a machine that has another architecture. Second, native code applications that usually runs directly on physical machines can be executed in a virtual computing environment without modifying the code. Unlike language VMs, Quasar can execute native code applications such as Apache and Firefox, and even commodity operating systems such as Windows and Linux. In contrast, it is not straightforward to execute an operating system on language VMs. As a result, language VMs provide only application-level "run-anywhere". They are not suitable for migrating a complete execution environment, which contains a wide range of resources such as file systems. Quasar can migrate a complete computing environment, and hence, it provides OS-level "run-anywhere" capability.

The contribution of this work is as follows:

- This work presents the design and implementation of a virtual machine migration system based on a CPU emulator. There have been many studies on virtual machine systems based on language VMs and virtual machine monitors. However, as far as we know, there have not been many studies on a methodology of developing a virtual machine migration system using a CPU emulator as a building block. Our system enables us to migrate a virtual computing environment between an x86/Linux and a PowerPC/Linux.

- This work shows evaluation results on the viability of a virtual machine migration system based on a CPU emulator. Up to now, a number of CPU emulators have been proposed. However, their viability in realistic situations has not yet been fully studied. We expect that the experimental results discussed in this paper will be useful for researchers, developers, and machine administrators when they consider the potential of CPU emulators.

The rest of this paper is organized as follows. Section 2 presents an outline of how our system works. Section 3 describes the implementation of functions that our system provides. Section 4 evaluates the viability of our system. Section 5 discusses related work. We conclude the paper in Section 6.

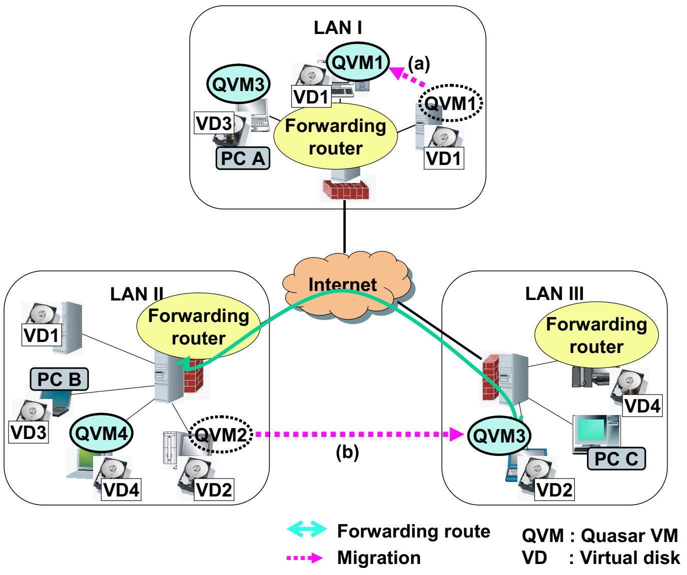

**Figure 1. VM migration environment in our system**

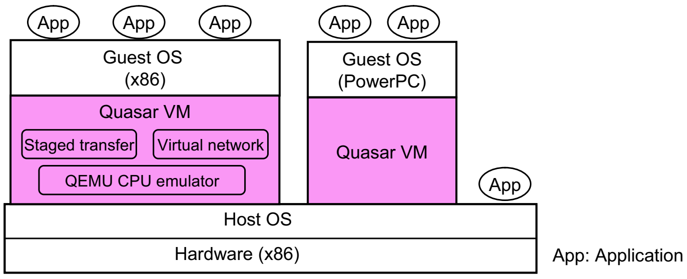

**Figure 2. Physical machine running Quasar VMs**

## 2. Overview of Quasar

### 2.1. Architecture

A VM migration environment provided by Quasar is illustrated in Fig. 1. Quasar is composed of the *Quasar VM* and the *forwarding router*. Quasar assumes there is one forwarding router in one local area network. In addition, The forwarding router needs to be running on a physical machine that is accessible from the external network. We assume that users has prepared a virtual disk on all the physical machines on which users intend to use a uniform virtual computing environment. For example, to use the virtual computing environment created by using VD3 on PC C deployed in LAN III, a user needs to copy VD3 from PC A deployed in LAN I or PC B deployed in LAN II (in Fig. 1). Quasar not only provides migration between physical machines on a local area network such as Fig. 1 (a), but also between ones on a wide area network such as Fig. 1 (b).

The Quasar VM provides a virtual computing environment. Fig. 2 is shown a physical machine on which two Quasar VM instances and one application run. The forwarding router handles migration and network routing for Quasar VM instances. The forwarding router maintains a list of all the Quasar VM instances running in the local area network. The forwarding router enables users to migrate their virtual computing environment to or from the physical machine which is not accessible from the external network.

## 2.2. Provided Functions

**Automation of migration procedures** When Quasar receives a migration request from a user, it automatically saves, transmits, and restores the user's virtual computing environment.

**Virtual networking facility** Quasar appropriately handles network routing before and after migration. A user can use the same IP address before and after migration. Moreover, communications are maintained during and after migration. As a result, programs running on the Quasar VM can continue to communicate with their communication peers without making the peers aware of migration. For example, Quasar allows users to keep an SSH session when a server and/or client program of SSH is migrated.

**Staged migration** To reduce the service downtime during migration, we take the *pre-copy* approach presented in the work of Amoeba [16]. Quasar divides the migration operation into two phases. In the first phase, the actions of saving and transmission of the execution state overlaps the provision of services in the virtual computing environment. In the second phase, the execution stops and the execution state that was modified during the first phase is transmitted. This staged migration scheme has been used in the work of VMotion [17] and Xen [5].

In the absence of a reliable and secure data transmission, the transfer data in Quasar might be lost or stolen, especially across wide area networks. Thus, Quasar uses a virtual private network to transfer data.

## 3. Implementation

### 3.1. Automatic Migration Mechanism

To migrate a virtual computing environment between physical machines, it is necessary to provide functions for saving and restoring virtual machine states and a virtual disk.

The virtual machine states include CPU, memory, programmable interrupt controller (PIC), and so on. QEMU already provides the functions for saving and restoring virtual machine states. We modified the function not to save and restore the clock ticks and real time clock states because there are problems with emulations related to them after migration.

To reduce the transfer size of a virtual disk, Quasar runs the Quasar VM in the *snapshot mode*. The snapshot mode is the function provided by QEMU. When the Quasar VM is running in this mode, all the writes to the virtual disk from the start of the Quasar VM change into the writes to a temporary file. Only the data that were written to the temporary file are transmitted to the destination machine.

### 3.2. Virtual Networking Facility

We implemented the network functions that include the virtual networking facility for supporting transparent connection mobility.

Quasar handles the network data received by the data link layer. We called the network data *raw packets*. We implemented the *bridged network*, which is supported in existing VM systems such as VMware and Xen. The guest OS which runs on the Quasar VM is identified by the MAC address of the virtual network interface card (NIC). The networking facility provides two benefits. One is that Quasar enables the guest OS to run on the Quasar VM to be assigned dynamic global IP addresses from the DHCP server in the local area network. The other is that the user does not need to add the Quasar VM specific network configuration (e.g., NAT) to the physical machine.

To enable users to use the same IP addresses before and after migration, Quasar forwards the raw packets destined for the Quasar VM. The virtual networking facility does not need to modify a host OS. The basic mechanism of our virtual networking are the same as that of Mobile IP [10] and VNET [14, 15]. Mobile IP is for the transport and network layers. VNET supports network-transparent mobility at the level of the data link layer. Unlike Mobile IP, Quasar enables users to use IP addresses obtained from the DHCP server running on the local area network when the guest OS starts, without adding a specific mechanism for supporting DHCP facility.

After receiving a migration request from the Quasar VM, the forwarding router identifies the migration route. If network addresses of the destination Quasar VM and the source forwarding router are same, the destination Quasar VM handles the raw packets. In this case, the raw packets are not forwarded by the forwarding router. If network addresses of the destination Quasar VM and the source forwarding router are different, the destination Quasar VM handles the raw packets via the forwarding router.

### 3.3. Reducing Service Downtime

Fig. 3 shows two migration mechanisms: the *bundle migration* and the *staged migration*. At the start of migration, the Quasar VM on a destination machine sends a migration request to the forwarding router.

In the bundle migration, we save and restore the virtual computing environment after stopping the Quasar VM on the source machine. This approach is taken in [7, 12]. The service downtime for the bundle migration is proportional

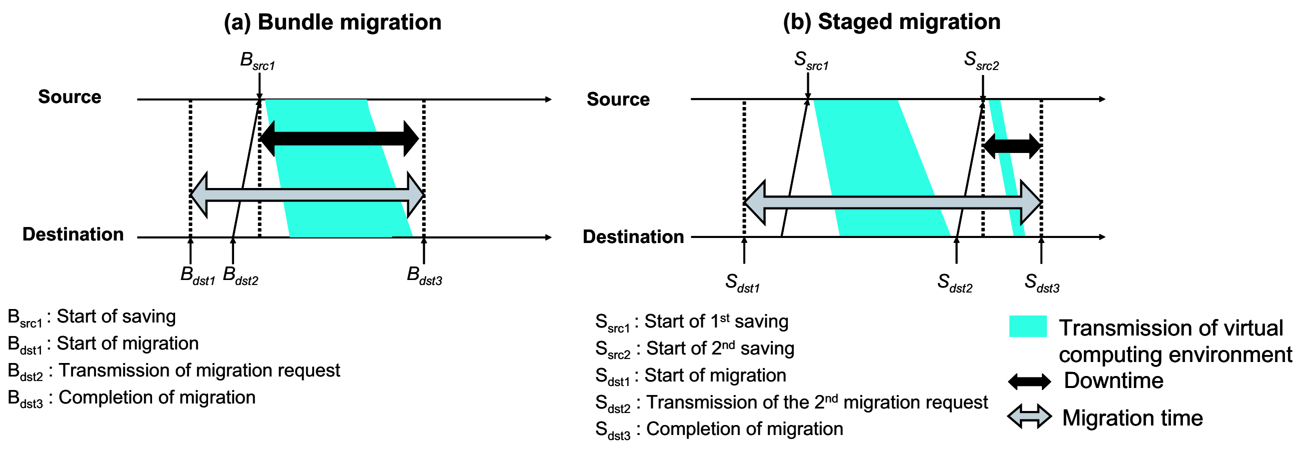
(a) Bundle migration
$B_{src1}$
Source
Destination
$B_{dst1}$ $B_{dst2}$
$B_{dst3}$
$B_{src1}$ : Start of saving
$B_{dst1}$ : Start of migration
$B_{dst2}$ : Transmission of migration request
$B_{dst3}$ : Completion of migration
(b) Staged migration
$S_{src1}$
$S_{src2}$
Source
Destination
$S_{dst1}$
$S_{dst2}$
$S_{dst3}$
$S_{src1}$ : Start of 1st saving
$S_{src2}$ : Start of 2nd saving
$S_{dst1}$ : Start of migration
$S_{dst2}$ : Transmission of the 2nd migration request
$S_{dst3}$ : Completion of migration
Transmission of virtual computing environment
Downtime
Migration time

**Figure 3. Migration transition**

to workloads on the guest and host OSes during migration and the amount of time and size required for saving and restoring the virtual computing environment. Thus, it might be significantly long.

To reduce the service downtime, Quasar provides the staged migration, which makes the service overlap migration operations. The staged migration has two phases.

In the first phase, Quasar saves, transfers, and restores the memory and the virtual disk states. At this point, the Quasar VM on a source machine are still running. We do not ensure the consistency of the memory and the virtual disk states that are transmitted in the first phase because the transmitted data do not necessarily correspond with their states at a certain moment. Instead, we ensure the consistency by using the transmitted data in the second phase.

In the second phase, Quasar stops the guest OS on the source machine. Then Quasar saves, transfers, and restores the virtual machine states including CPU, PIC, and NIC states and the remaining raw packets. Quasar also sends all the data written to the memory and the virtual disk from the start of the first phase until the start of the second phase. The start points of the first and the second phase are $S_{src1}$ and $S_{src2}$ in Fig. 3 (b). Finally, Quasar restarts the guest OS on the destination machine. The staged migration reduces the service downtime because the amount of time and transmitted data in the second phase are typically smaller than ones in the first phase.

To recognize the writes to the memory and the virtual disk during the first phase (between $S_{src1}$ and $S_{src2}$), Quasar prepares the bitmaps for recognizing the dirty pages and dirty sectors at the start of the first phase. Currently, we use the software MMU, which is supported by QEMU. We modified the write operations of the software MMU and the virtual disk. For every write operation to a page, Quasar sets the bit which represents the page in the memory bitmap. For every write operation to a sector, Quasar also sets the bit which represents the sector in the virtual disk bitmap. In the second phase, Quasar transmits only the pages and the sectors for which a bitmap has been set.

## 4. Experiments

We conducted three experiments to evaluate the viability of Quasar. For all the experiments, we used the Quasar VM which emulates the x86 architecture.

We used two physical machines in these experiments. One was an HP workstation xw4100 which had an Intel Pentium 4 3.0 GHz with HyperThreading enabled, 1 GB memory and a Linux 2.6.13 kernel ($PM_{hp}$). The other was a Thinkpad R51, which had an Intel Pentium M 1.5 GHz, 512 MB memory and a Linux 2.6.14 kernel ($PM_{tp}$). In the last two experiments, the two physical machines were connected via a gigabit network in a local area network. Both machines had a gigabit network controller and there were five gigabit switches between them.

### 4.1. Size of Data for Saving and Restoring

To migrate a uniform computing environment in Quasar, it is necessary to save and restore virtual machine states such as CPU and VM memory, and virtual disk. To save and restore the virtual disk state, Quasar ran in the snapshot mode and transferred only the dirty sectors.

To compare with existing systems with functions for saving and restoring a virtual computing environment, we also conducted the same experiment on Xen 2.0-testing and ScrapBook for User-Mode Linux (SBUML) [13]. Xen does not save or restore the virtual disk state. SBUML uses the copy-on-write file function provided by User-Mode Linux (UML) as the virtual disk. We configured all the

VMs to have a 1.25 GB virtual disk and installed Debian GNU/Linux 3.1 in the virtual computing environment.

We saved and restored the following four states (two CUI and two GUI states) for the 128 MB and 256 MB VM memories.

**login** We started the guest OS and got the CUI login prompt. This has few write accesses to the VM memory and the virtual disk.

**x-window** After login, we downloaded, installed and configured the *x-window-system* package. This command can automatically get packages on which the x-window-system package depends. The x-window-system package requires 63 new packages. This state has many write accesses to the VM memory and the virtual disk.

**xterm** We started the X window system and ran the window manager *twm* and one *xterm*.

**firefox** We started the X window system, ran the twm, one xterm and the web browser *Firefox*. We created five new tabs in Firefox and browsed the web page *www.google.co.jp* on the tab.

The results in Table 1 indicate the amount of the size required for saving and restoring the virtual computing environment. The saving and restoring size in our system is small. Almost all the size in our system consists of the VM memory state . The size of the other virtual machine states, such as CPU and PIC, is about 52 KB. QEMU provides the simple compression for saving and restoring the VM memory state. The compression effect is indicated the case of login, xterm (for the 128 and 256 MB VM memories) and firefox (for the 256 MB VM memory) in Table 1.

## 4.2. Networking and Service Hosting Throughput

To evaluate Quasar networking and the cost of hosting services on Quasar, we used Netperf 2.4.1 and ApacheBench 2.0.41.

The server and client machines are $PM_{hp}$ and $PM_{tp}$ respectively. We configured the Quasar VM on the server machine to have a 256 MB VM memory. We also ran these benchmarks on Xen and UML for comparison.

At first, we measured the request and response throughput on a TCP and UDP network connection by using Netperf.

We ran the *netserver* process on the server machine and the *netperf* process on the client machine. The transaction in this experiment was a single communication with a data size of 1 byte. The results in Table 2 indicate that The performance in our system is about 4 and 4.5 times worse than that of Xen and 2.5 and 3 times worse than that of UML for TCP or UDP.

| | Quasar | UML | Xen |
|---|---|---|---|
| TCP | 1023 | 2668 | 4002 |
| UDP | 1294 | 3969 | 5395 |

**Table 2. Request and response throughput (transactions/second)**

To examine hosting web services on Quasar, we ran the web server, Apache 2.0.54, on $PM_{hp}$. We configured the Quasar VM to have a 256 MB VM memory. By using ApacheBench on $PM_{tp}$, we sent 1024 requests to get a static content to Apache on the Quasar VM. We used 1 KB and 100 KB files as the static content and we varied the concurrency of the request from 1 to 128 exponentially. We also ran these benchmarks on Xen and UML for comparison.

Table 3 indicates that the number of the processed requests per second were about 120 for 1 KB file and 24 for 100 KB file. The results indicate that the overhead incurred by hosting web services on current our system is large. In Quasar, the hosting throughput is virtually constant regardless of the increase of the concurrency. In contrast, the throughput in Xen and UML was increased. Thus, we will have to analyze the factors of the overhead in detail and enhance our system in the future.

## 4.3. Migration between Physical Machines

Finally, we measured the migration time and the service downtime due to the bundle migration and the staged migration from $PM_{hp}$ to $PM_{tp}$.

We configured the Quasar VM to have a 256 MB VM memory and 1.25 GB virtual disk. The forwarding router ran on the source machine $PM_{hp}$, and we sent a migration request from the destination machine $PM_{tp}$.

To examine the overhead of the migration time and the service downtime incurred by using SSL, we measured the migration time and the service downtime with SSL enabled and with it disabled. Fig. 3 indicates the measuring points of the migration time and the service downtime for each migration. We defined the pseudo-service downtime for the bundle migration and the staged migration as the time between $B_{dst2}$ and $B_{dst3}$ and the time between $S_{dst2}$ and $S_{dst3}$, respectively. We regarded the pseudo-service downtime as the actual service downtime because $B_{src1}$ ($S_{src2}$) and $B_{dst3}$ ($S_{dst3}$) were measured on the different physical machines.

We migrated the following active execution states of the Quasar VM.

VM memory: 128 MB

| | CUI | | GUI | |
|---|---|---|---|---|
| | login | x-window | xterm | firefox |
| Quasar (MS / DS) | 54 / 3 | 126 / 210 | 85 / 4 | 120 / 5 |
| Xen | 129 | 129 | 129 | 129 |
| SBUML | 15 | 258 | 34 | 69 |

VM memory: 256 MB

| | CUI | | GUI | |
|---|---|---|---|---|
| | login | x-window | xterm | firefox |
| Quasar (MS/DS) | 56 / 3 | 254 / 210 | 87 / 4 | 124 / 5 |
| Xen | 257 | 257 | 257 | 257 |
| SBUML | 17 | 353 | 37 | 71 |

MS: Machine states DS: Dirty sectors

**Table 1. Saving and restoring size (MB)**

(a) 1 KB

| | Concurrency | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
| Quasar | 115 | 113 | 115 | 122 | 121 | 118 | 117 | 116 |
| UML | 444 | 470 | 474 | 492 | 513 | 560 | 579 | 598 |
| Xen | 950 | 1042 | 1118 | 1189 | 1204 | 1274 | 1305 | 1381 |

(b) 100 KB

| | Concurrency | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
| Quasar | 24 | 23 | 23 | 23 | 22 | 22 | 22 | 21 |
| UML | 111 | 116 | 117 | 118 | 121 | 120 | 119 | 120 |
| Xen | 42 | 60 | 100 | 220 | 491 | 564 | 577 | 628 |

**Table 3. Web service throughput (requests/second)**

**top** After CUI login, we started running the *top* command. In this case, the write accesses to the VM memory and the virtual disk were small, and the CPU workload was low during migration.

**x-top** After CUI login, we started running the X window system. Then we ran the twm, one xterm, and the top.

**kernel** After CUI login, we started creating the bzImage of the Linux kernel 2.4.23. In this case, the write accesses to the VM memory were large and the CPU workload was high.

**em-kernel** After login, we downloaded, installed and configured the *emacs21* package. To get this package, it is necessary to get 6 new packages. After the configuration was completed, we started creating the bzImage of the kernel 2.4.23. In this case, the amount of write accesses to the VM memory and the virtual disk were large.

The migration request was sent one minute after the last command, such as top and creating the bzImage, was executed on the source machine $PM_{hp}$. The last command was in progress before and after migration.

The results in Tables 4 and 5 indicate the migration time and the service downtime for each execution state. In all the cases, although the migration time for the staged migration is lager than the one for the bundle migration, the service downtime for the staged migration is smaller than one for the bundle migration. Compared with the SSL-disabled case, the SSL-enabled migration took longer, but the difference of the service downtime due to the staged migration was small.

For the staged migration, we also measured the amount of the transmitted VM memory and virtual disk in the first and second phases. The results are shown in Table 6. In the second phase, the virtual execution states were transmitted and the size was about 52 KB. The table indicates that more data was transferred in the first phase than in the second phase. This is one of the factors that led to the shorter downtime of the staged migration.

## 5. Related Work

There has been a lot of work on applying virtual computing environments to migration and server hosting. [2, 3, 5,

(a) Migration time

| | top | x-top | kernel | em-kernel |
|---|---|---|---|---|
| Bundle | 3.5 | 3.6 | 3.9 | 13.4 |
| Staged | 3.6 | 3.6 | 4.5 | 16.7 |

(b) Service downtime

| | top | x-top | kernel | em-kernel |
|---|---|---|---|---|
| Bundle | 2.86 | 3.43 | 3.70 | 12.80 |
| Staged | 0.03 | 0.05 | 0.42 | 0.59 |

**Table 4. Migration time and service downtime with SSL-enabled (seconds)**

(a) Migration time

| | top | x-top | kernel | em-kernel |
|---|---|---|---|---|
| Bundle | 2.0 | 2.0 | 2.0 | 5.0 |
| Staged | 2.0 | 2.0 | 2.1 | 5.5 |

(b) Service downtime

| | top | x-top | kernel | em-kernel |
|---|---|---|---|---|
| Bundle | 1.87 | 1.88 | 1.89 | 4.31 |
| Staged | 0.01 | 0.02 | 0.06 | 0.60 |

**Table 5. Migration time and service downtime with SSL-disabled (seconds)**

(a) SSL-enabled

| Phase | | | top | x-top | kernel | em-kernel |
|---|---|---|---|---|---|---|
| First | VM Memory | (MB) | 39.17 | 57.97 | 64.70 | 216.41 |
| | Virtual Disk | (KB) | 2573 | 1724 | 3919 | 95571 |
| Second | VM Memory | (MB) | 0.76 | 1.47 | 13.60 | 18.20 |
| | Virtual Disk | (KB) | 3 | 3 | 0 | 461 |

(b) SSL-disabled

| Phase | | | top | x-top | kernel | em-kernel |
|---|---|---|---|---|---|---|
| First | VM Memory | (MB) | 34.65 | 57.26 | 64.71 | 216.42 |
| | Virtual Disk | (KB) | 2565 | 1736 | 3865 | 95531 |
| Second | VM Memory | (MB) | 0.63 | 1.87 | 10.19 | 12.22 |
| | Virtual Disk | (KB) | 0 | 3 | 0 | 100 |

**Table 6. Transfer size for staged migration**

8, 9, 12, 17, 20].

Many migration systems based on virtual machine monitors have been proposed. VMware and Xen have the mechanism to reduce the service downtime [5, 17]. They assume a disk storage shared among different virtual machines, which is typically provided by storage area network (SAN), network attached storage (NAS), or a distributed storage system such as Parallax [18]. Collective [12] has proposed some optimizations, e.g., the compression of the virtual disks and reducing the amount of transmitted data by sending only differences of the virtual disks. Internet Suspend/Resume [7] enables migration by combining virtual machine technology (VMware) and distributed file systems. However, these systems do not have a mechanism to reduce the service downtime due to the migration. The above migration systems restrict the available host CPU because they are built on top of the x86 architecture.

There are many hosting and migration systems based on virtualization of system call execution and resource views. Examples are Zap [9] and SoftwarePot [6]. Many systems following this approach have the advantage over our system in that virtualization overhead is smaller than when using a CPU emulator. However, they are more dependent on an underlying real computing environment.

One.world [1] is a Java-based framework that supports mobility of applications in ubiquitous environments. Since it is implemented on Java, it cannot execute native code applications such as Apache.

Mobidesk [3], VNC [11], and Remote Desktop in Windows enable a user to provide a uniform computing environment on various physical machines. These systems make applications run on servers, and exchange user input and application output between server and client machines. These systems cannot be used in disconnected environments. Furthermore, these systems depend on the network performance and/or the distance between server and client machines because of the interaction between servers and clients. Unlike these thin-client approaches, all process executions and computations run on the local physical machine in our system.

## 6. Conclusion

Quasar is a virtual machine migration system that is built on top of the QEMU CPU emulator. By integrating migration functions with a CPU emulator, it enables a user to use a uniform computing environment. It provides three functions: an automatic migration mechanism, a virtual networking facility, and a mechanism to reduce the service downtime. We implemented a prototype on Linux and evaluated its viability. The prototype had reasonable size required for saving and resuming a virtual computing environment and the service downtime due to migration. The experimental results of the web server hosting on the current prototype were that the number of the processed requests per second were about 120 and 24 for 1 KB and 100 KB files respectively. and the overhead was large.

We plan to analyze and enhance our system so it can provide hosting services with a smaller overhead. Moreover, we are going to use a multi-staged migration to provide better support for reducing the downtime during migration. We are also interested in applying to security systems.

## References

[1] L. Arnstein, R. Grimm, C. Hung, J. H. Kang, A. LaMarca, G. Look, S. B. Sigurdsson, J. Su, and G. Borriello. Systems Support for Ubiquitous Computing: A Case Study of Two Implementations of Labscape. In *Proceedings of the First International Conference on Pervasive Computing*, pages 30–44, Zurich, August 2002.

[2] A. A. Awadallah and M. Rosenblum. The vMatrix: Server Switching. In *Proceedings of the 10th IEEE International Workshop on Future Trends in Distributed Computing Systems (FTDCS '04)*, Suzhou, May 2004.

[3] R. A. Baratto, S. Potter, G. Su, and J. Nieth. MobiDesk: Mobile Virtual Desktop Computing. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MOBICOM 2004)*, pages 1–15, Philadelphia, September 2004.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 164–177, New York, October 2003.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Waarfield. Live Migration of Virtual Machines. In *Proceedings of 2nd Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, May 2005.

[6] K. Kato and Y. Oyama. SoftwarePot: An Encapsulated Transferable File System for Secure Software Circulation. Technical Report ISE-TR-02-185, Institute of Information Sciences and Electronics, University of Tsukuba, January 2002.

[7] M. Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications*, pages 40–46, June 2002.

[8] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04)*, Pittsburgh, November 2004.

[9] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, pages 361–376, Boston, December 2002.

[10] C. E. Perkins and A. Myles. Mobile IP. In *Proceedings of International Telecommunications Symposium*, pages 415–419, 1997.

[11] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, January 1998.

[12] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rsenblum. Optimizing the Migration of Virtual Computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, pages 377–390, Boston, December 2002.

[13] O. Sato, R. Potter, M. Yamamoto, and M. Hagiya. UML Scrapbook and Realization of Snapshot Programming Environment. In *Proceedings of the Second Mext-NSF-JSPS International Symposium on Software Security (ISSS 2003)*, volume 3233, pages 281–295, Tokyo, 2003. Springer.

[14] A. Sundararaj, A. Gupta, and P. Dinda. Increasing Application Performance In Virtual Environments Trough Run-time Inference and Adaptation. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, Research Triangle Park, July 2005.

[15] A. I. Sundararaj and P. A. Dinda. Towards Virtual Networks for Virtual Machine Grid Computing. In *Proceedings of the 3th Virtual Machine Research and Technology Symposium (VM '04)*, pages 177–190, San Jose, 2004.

[16] M. Theimer, K. A. Lantz, and D. R. Cheriton. Preemptable Remote Execution Facilities for the V-System. In *Proceedings of the 10th ACM Symposium on Operating System Principles*, pages 2–12, Orcas Island, Washington, December 1985.

[17] VMotion. `http://www.vmware.com/products/vc/vmotion.html`.

[18] A. Warfield, R. Ross, K. Fraser, C. Limpach, and S. Hand. Parallax: Managing Storage for a Million Machines. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS X)*, Santa Fe, NM, June 2005.

[19] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, pages 195–209, Boston, December 2002.

[20] M. Zhao, J. Zhang, and R. Figueiredo. Distributed File System Support for Virtual Machines in Grid Computing. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC '04)*, pages 202–211, Honolulu, June 2004.