Master's Thesis Academic Year 2010

A Strategy for Virtual Machine Migration Based on Resource Utilization

Keio University
Graduate School of Media and Governance

Keisuke Muda

Abstract of Master's Thesis

Academic Year 2010

A Strategy for Virtual Machine Migration Based on Resource Utilization

People use computers for different purposes, which require varying computing resources, such as CPU power, memory space, disk I/O, or network throughput. Sometimes users demand computing resources greater than what is available, requiring either the physical or virtual addition of resources. Even when users demand the addition of computing resources, there are physical constraints on the computer architecture. To dynamically change resources, there must be a way for computers to adapt to the available resources without physical interaction.

In this thesis, virtualization technology is used to allow computers to dynamically adapt to the pressure for more resources. When the applications running on the virtual machine demand certain computing resources, the virtual machine is migrated to a hypervisor that provides a better match. The system chooses to migrate a virtual machine when a higher-performance system or an idle system is available, and the workload exceeds a threshold for a certain period of time.

Live migration of a virtual machine has some overhead cost, as the contents of allocated memory space and the virtual machine state must be copied from the source hypervisor to the destination hypervisor. Because of the synchronization of memory spaces, live migration of a virtual machine will not occur instantly. Sometimes the performance may not improve when migration does not complete before it reaches a point where the performance improvement will become apparent.

Experiments were conducted to observe the effects of available computing resources on virtual machine performance. The relationships between processes running inside of the virtual machine and their impact on the live migration were measured. Broadly, the tests have shown that the live migration of a virtual machine in a CPU-bound state may be beneficial but a virtual machine in an I/O-bound state should not be migrated.

Keywords :
1. Virtualization, 2. Computer Resources, 3. Live Migration, 4. Boundaries

Keio University
Graduate School of Media and Governance

Keisuke Muda

修士論文要旨

2010 年度（平成 22 年度）

資源の利用状況に基づいた 仮想マシンのマイグレーション指標の検討

今日，計算機は様々な用途に利用されている．計算機の処理は，CPU 処理能力・メモリ空間・ディスク I/O・ネットワークスループット等，多様な計算資源を利用することで実現されている．そのため，単一の計算機で利用可能な計算資源には限りがあり，より多くの計算資源を必要とする際には物理的・論理的に計算資源を追加する必要がある．しかし，そのような計算資源の追加や削除では，物理的に計算資源に触れることが必要となる他，一度処理を停止しなくてはならないなど，いくつかの制約が発生する．計算資源の人々の要求に合わせた動的な変更の実現には，利用可能な計算資源に対して計算機が適応する手法が必要となる．

本研究では，アプリケーションの計算資源の追加・削除要求に対して計算機が動的に利用するため，仮想化技術を用いる．その中で，仮想マシン内部における処理が特定の計算資源を要求した際に，仮想マシンを目的に適したハイパーバイザへとライブマイグレーションさせる．本論文においては，閾値以上の資源への負荷が一定時間継続した際に，より適切な処理性能を持つ計算機へのライブマイグレーションを実行する．

仮想計算機のライブマイグレーション時には，ハイパーバイザ間において一時記憶の内容を同期させ，仮想マシンの実行状態を旧ハイパーバイザから新ハイパーバイザへ移転する動作が発生する．この同期処理のため，ライブマイグレーションは要求時に即座に実行されない．処理の内容や環境により，ライブマイグレーションが実行中の処理全体の内で性能を向上させる時点までに完了しなかった場合などに，処理性能が向上しない場合がある．

それらの条件を踏まえ，仮想マシンに対する計算資源の変化の効果を測定するための実験を実施した．実験においては，仮想計算機内で実行された処理の内容とライブマイグレーション完了までに要した時間・ネットワーク資源との関係を測定した．実験より，CPU 処理能力が必要な状態においては仮想計算機のライブマイグレーションにより性能の向上が見込め，一方で入出力が多発する状況下ではライブマイグレーションを実行すべきではないとの結果が得られた．

キーワード

1．仮想化，2．計算資源，3．ライブマイグレーション，4．境界

慶應義塾大学大学院 政策・メディア研究科

六田 佳祐

Table of Contents

Figures

Tables

1. Introduction

This chapter gives introduction of the thesis, and also some background information.

1.1 Introduction

Nowadays, computers are used everywhere around humans, regardless of whether they are recognized or not. When people hear the word "computer", a great portion of them imagine the computers in the form of desktop or laptop computers. However, there are many other "computers" around us, for example cell phones or digital television recorders. Computers reside everywhere around humans, and the computers are used for fulfilling users' demands.

When users use the computers, they have different purposes in mind and make different demands on the computers. Some users use computers for "light" purposes, such as web browsing, e-mail, and on-line chat. On the other hand, some users use computers for "heavy" purposes, such as video or graphics encoding, running video games with numerous three-dimensional objects, scientific calculation, or other similar purposes. Often, users purchase a computer that satisfies their average usage; users who use computers primarily for light purposes will buy cheap, small computers such as netbooks or laptops with less capability, and users who use computers primarily for heavy purposes will buy computers with great computational power and better central and graphic processing units. Each user makes different demands on the computers, and each user obtains a computer that fulfills his or her demands.

However, there are certain situations where users would demand different computational capabilities, regardless of whether they are using computers for light purposes or heavy purposes. For example, users with light computers may suddenly wish for more computing power to carry out some heavy numerical computation or to try video editing for their families. Even if there is another computer with better capabilities, the user needs to terminate the running job, bring files and application environments to the other computer, and start running the job again on the better computer. Even though computing capability of a single computer is limited, there are times where users demand more than what is available on their computers.

There are several ways to add computing capabilities to existing computers. A simple method would be to add or exchange the computer parts that compose the computer.

For example, users may exchange the CPU on the computer for a faster one or one with more processor cores, or add more RAM to the computer. However, users face several problems when exchanging computer parts. A prominent problem is the fact that parts need to be attached physically to computers. If the CPU socket or bus interface is different, or if no expansion slots for adding modules are available, it is difficult to extend the capabilities of the computer with additional hardware. Also, since changing the hardware configuration involves actual physical operations such as opening the case, it is bothersome for experts and intimidating for non-technical users, and cannot be done repeatedly on a short time scale. Another solution would be to change software or the operating system (OS) running on the computer. Even for the same purpose, a different implementation may perform better. This might add some capability to the computer, but the ultimate hardware bound remains the same.

Therefore, it has become common for users to export their tasks to other computers in some manner. One example is to own a shared computing server in the network system, and remotely control the server using secure shell (SSH), or remote desktop applications such as Microsoft Remote Desktop Protocol (RDP) or Virtual Network Computing (VNC)[1]. The recent trend is "cloud computing", where both user data and applications are saved on Internet servers, and users access the "services" over web browsers. In the remote control applications and cloud computing services, the user's own computer doesn't have to be an expensive, high-powered machine; since computations are done on the computers at remote sites, local computers only need to have input and output devices. Such terminals, called thin clients, have come and gone over the years; a "netbook" might be the current equivalent. This is one way to efficiently add computing resources to users' computers, which can be done frequently and easily compared to adding computational resources to computers physically.

Computing resources are elements that include both computer hardware components and user data that are computed on the computer. Computer hardware components are hardware such as: computational power on CPU, memory space, storage, user interfaces, and peripheral devices that add capabilities to the computer. User data are files which users save on the computers, such as documents, spreadsheets, applications and their settings, e-mails, browser bookmarks, and others. All elements that compose a computer can be considered as computer resources, and developing an interconnect that allows elements to be connected and managed to each other is necessary in this architecture.

However, even though the ideas of using the computational resources of other computers over an Internet connection have become popular, there are still several problems. One of the problems is synchronization of data. When users have their data on their local computers, and when they wish to use the data on the remote services, users have to synchronize the data between the local and the remote computers. The solution may be to save all of a user's data online, and access the data from both local

and remote computers. However, this solution requires users' consoles to be always connected on the Internet to access the data. Even using mobile wireless connections, there are many places where users cannot access the Internet while they are moving, such as underground or rural areas, and the assumption of "always connected to the Internet" cannot be made on the existing computational environment. Moreover, since the computer running the processes will change when using remote control or accessing remote services, the continuity of their tasks is likely to be terminated when exporting the tasks to remote services. Therefore, when users are working on the tasks, and they wish to have more computational resources, the in-progress work needs to be saved or checkpointed, and processes have to be terminated in order to export the tasks. This sometimes may be a bottleneck as a user service, as sometimes certain applications need to be restarted from the beginning instead of resuming from where they were terminated.

In this thesis, a system architecture based on virtualization technology with dynamic migration will be introduced. In this architecture, the entire user environment, including OS, applications, and user data, is always running on a hypervisor as a virtual machine. When a user wishes to obtain additional computational resources, the virtual machine migrates from the user's local computer to a computer with more capabilities. The migration decision is made based on the status of the user's computer, such as CPU usage, available memory space, and device inputs and outputs (I/O). The migration takes place either with or without user interaction, and the user consoles and disk drives remain unchanged even after migration; the virtual machine is the only element in the system that migrates from a hypervisor to another. A virtual machine is a virtual "environment" of a single computing system, allowing the application status and user data to remain unchanged even at migration. In addition, as long as a virtual machine is running on the local hypervisor, it can be running even when the hypervisor is not connected to the Internet.

1.2 Purpose

Nowadays, the extension of computational powers using external resources has become common. In order to improve performance of applications running inside a computer, an architecture that allows computers to utilize computational resources to the maximum extent and minimizes the resource consumption is necessary.

Thus, an architecture based on virtual machine migration is expected to improve performance of virtual machines by adapting the virtual machine onto a computing resource that is the most comfortable for the computational work running on the virtual machine. In order to adapt virtual machines onto available computing resources, a methodology for selecting a proper hypervisor with necessary computing resources

becomes necessary. As was mentioned previously, several resources that affect computing performance are CPU, memory, and device I/O. There are various I/O devices in a computer, but the two prominent examples are disk I/O and network I/O. By observing the utilization of these resources and migrating virtual machines to a hypervisor that satisfies resource requirements, the performance of virtual machines is expected to improve.

The architecture utilizes live migration of virtual machines for dynamically changing the hosting hypervisor to an appropriate one. Hypervisors are computers that host virtual machines, and since they are also computers running hypervisor OSes, each hypervisor would have different computing resources. Moreover, since a single hypervisor can host multiple virtual machines, a hypervisor with the greatest computing resources may not always be the most effective hypervisor to run the virtual machine on. Thus, in order to introduce the architecture proposed in the thesis, a method for evaluating available computing resources is necessary.

In addition to evaluating computing resources, the cost of performing live migrations must also be considered. Performing live migration of virtual machines will transfer memory contents of the virtual machine to a remote hypervisor, consuming resources and network bandwidth. The migration procedure typically takes a few seconds up to a few minutes on modern PC-class hardware, depending on available network bandwidth. Even if other hypervisors have better computing resources, if the cost of migration is too high, sometimes it may be better not to migrate the virtual machine. Therefore, a strategy is necessary for maximizing the use of computing resources available to virtual machines.

1.3 Contributions of the Work

The contributions of this thesis are as follows:

- Propose an architecture in which virtual machines can adapt to resource demands and utilize network-available resources by performing a migration.
- Propose a multi-variable method for selecting the best environment for a given virtual machine, taking migration cost into account.
- Evaluation of the above by triggering migration based on workloads on a computer, demonstrating the crossover point at which migration pays off.

1.4 Thesis Outline

This thesis consists of 8 chapters. Chapter 2 describes problems that the existing computer architecture has, and introduce a computer architecture that utilize IP

networks as its common bus. Chapter 3 describes advantages of using virtualization technology, which is a key technology used in this thesis. Chapter 4 describes virtual machine migration, as well as comparing it with process migration. Chapter 5 gives an architectural view of the system, and Chapter 6 describes prototype implementation. The migration threshold is evaluated in Chapter 7, and Chapter 8 concludes the thesis.

2. Computer Architecture

A computer is a set of parts that are connected to each other inside of a hardware case. This chapter gives an overview of physical constraints that exist in the computer architecture, and how boundaries between computers have changed.

2.1 Boundaries of Computer Hardware

The philosophy of computer architecture has remained the same in the past decades of computing history. Since hardware resources primarily communicate with each other using local buses, the resources are expected to communicate with each other using specialized protocols in hardware and software, which often come with limits such as maximum response time. Therefore, it can be considered that the existing computer architecture is stable in terms of communications among resources, but this architecture has limits on its scalability.

2.1.1 Computer as a "Box"

The computer today is primarily handled in units of "boxes". One computer, or one box, is composed of parts, such as central processing unit (CPU), random access memory (RAM), motherboard, disk storage, input and output peripherals, and other components. Users may select to add peripherals to the box by either inserting expansion card to motherboard or by connecting peripherals externally, but the computer is still boxed as a set of computer parts, with a full set of computing resources (Figure 2.1).

The boxed architecture has been common since the first appearance of computers. Even though the materials used for computers have changed from tubes to transistors, and even though computers are now able to communicate with other computers through the Internet, computers are still sets of components that reside in a single location, in one or a few boxes. This is advantageous for certain reasons. First, since all the components are stored in a single box, it can be used standalone. For example, even if the computer cannot access the Internet, it still can operate independently with the user environment and data that are installed on the computer. Next, since the computer is enclosed in a single box, the communications between computer components can

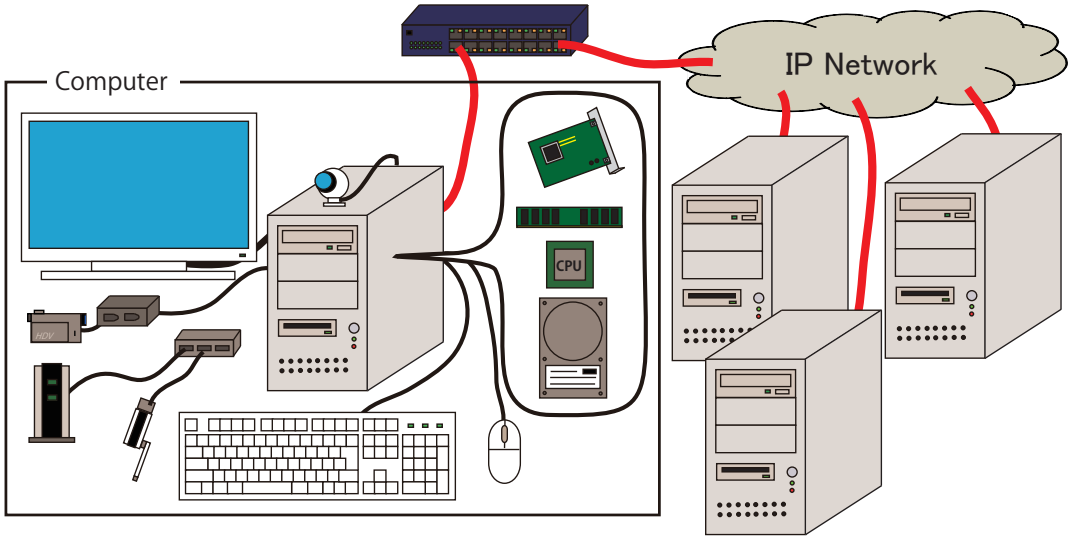

Figure 2.1: Traditional computers are defined by the hardware boxes

be done at a high rate, without the interference of external phenomena. This allows computer parts to function at a defined bandwidth and expected response time.

Each computer component has different requirements, even if the components use the same device bus. For example, a keyboard and a storage device can be connected to the same USB bus. However, when users type keys with keyboards, they may wish to see a better response to the actions rather than seeing keys they have typed a couple of seconds ago. On the other hand, when users use storage devices, even though performance is important, writing to and reading from storage without losing data is more important. Even if the device bus is the same, the aspects that devices require on the bus are different. Since the devices are enclosed in a box, the device bus that transports data are expected to respond within the time period defined in its specification, and to cover the errors that occur on the transport bus.

2.1.2 Networks of Computers

In the existing computer architecture, computers can be described as boxes that contain computational resources. When computers are connected to each other on IP networks, the box itself can be considered to be the boundary of resources between those computers. In other words, the resources that reside outside of the boundary are not for the particular computer to use without transferring the resource within the computer's boundary.

Even though boundaries exist, some applications today override the boundaries. One example of running a single set of computational work on multiple computers at very large scale is volunteer computing, epitomized by programs such as SETI@home[2] and

Folding@home[3], which asks users to install client applications on their computers and run the applications while the computers are idle.

Not only the home-scale computers, but there are large-scale computers that are operated as a set of multiple computers. An example of such a supercomputer is IBM's BlueGene Supercomputer[4] family, which consists of multiple chips on a single compute card, multiple compute cards on a single node board, multiple node boards in a single cabinet, and a system constructed of multiple cabinets. In a supercomputer architecture, the boundary of computational power is quite different from ordinal computers; there are multiple computers inside of a single box physically, and there are multiple boxes in a single set of system, turning the system into a set of computers by hundreds and thousands of computational nodes. Thus, the boundaries of computational power is blur on the supercomputing systems compared to those of the traditional computer architecture.

As the examples of volunteer computing and supercomputers show, although computers have physical hardware boundaries between them, there are methods for connecting the computing resources useful for accomplishing certain computations that can be divided among the multiple nodes that construct a computing system. However, there are still problems on the system: even though computational work can be divided into multiple computers, the computers are still "boxed" into a single unit and have limitations on scalability; the issues described in Section 2.2.4 are not resolved in the supercomputing systems. Therefore, it can be said that supercomputing systems may blur the boundaries of computational "power", but they don't remove the boundaries of computational "resources" including peripherals or user environments.

2.2 Adding and Removing Resources

Since each computer has limited space with a limited number of device buses, the ways of adding and removing computing resources from the "box" is somewhat limited. This section gives brief description of methods used by computers to expand their resources.

2.2.1 Physical Interaction

One of the direct ways of adding computing resources is to add, exchange, or remove computer devices physically from computers. For example, when users need to have more computational power or memory space to expand a large set of data, users may wish to add or exchange CPU or RAM to the computers. Or, for another example, when users need to have more storage space, users may add disk drives to the computers. This method of adding computing resources produces a direct resource on

computers, but generally the number of slots available for users to add these devices is limited. For instance, when the motherboard of a computer has only a single CPU slot, users can only exchange the CPU rather than adding another CPU. Moreover, since there are different types of CPU sockets available for different generations or different vendors of CPUs, not all CPU can be added to the CPU slot available on that specific computer. In addition, physical addition of processing units to computers requires physical interventions with computers. The added resource cannot be shared over multiple computers, and it becomes difficult to alter the resources frequently since they are connected to computers internally.

Adding computing resources can be done at outside of computers for several resource types. For example, buses such as PS/2, IEEE1284, RS-232C, SCSI, USB, and IEEE1394 are frequently used to add computing resources (generally, peripherals) outside of the computer box. Adding computer resources over these buses doesn't require users to open the computer case, which makes it easier for users to add computing resources. An example of simplifying the resource additions through external buses would be adding disk drives, which can be added both internally and externally to the computers. To users, adding computing resources externally will make the addition of resources easier as it doesn't require users to physically interact with the inside of computer cases, which is generally considered an obstacle to users without knowledge of computers. In addition, many external buses support using hubs or making cascaded connections of devices up to several levels, expanding scalability on device buses. The drawbacks of external addition of computing resources is that the components that constitute the central portion of computation, such as CPU or RAM, cannot be added externally in computer architectures that are generally used in today's environment. Also, even though the resources can be added externally, there are limitations on physical distance between the computer hardware and additional resources. The resources need to be connected to computers themselves, and the cables used for adding the resources have limits on distances that are determined in specifications.

2.2.2 Sharing Resources at the Application Layer

Physical addition of resources is a direct way of adding hardware resources to computers, but the limitation placed on this method is also the fact that the resources need to be added physically. Because of the physical connection, users must interact with resources when adding or removing resources to computers. In addition, the resources can be used on a single host at a time; resources connected to a computer cannot be used on other computers while they are occupied on a single computer.

To resolve the issues on physical distance between resources and computers, and the issues on sharing resources among multiple computers, various ways of using computer resources over IP networks have been developed. One of the usages that is used

frequently is software-based sharing of storage resources. The examples of sharing storage resources with software includes NFS and Samba (CIFS)[5]. On these protocols, resources are handled as "files" rather than raw "blocks" used for storage devices.

Another example of software-based sharing of resources is sharing of printers, which is frequently done with protocols such as LPD, IPP, or Samba. These are supported either on additional software installed on print servers, or implemented within the printer firmware to accept the print requests.

There are various software used for sharing resources on multiple computers over IP networks. These software allow easy sharing of devices among multiple computers. However, since these resources are shared on application-basis, there are also drawbacks on using the applications. For example, users cannot format the storage space shared on application-based methods as they would do on the real hardware, not allowing to split partitions or change file system types. The benefits of using application-based sharing of computing resources is the fact that these resources can be shared easily, without requiring physical interaction. However, because they are abstracted in different ways, not all functions provided by the corresponding resources may be used on application-based approaches.

2.2.3 Sharing Resources as Real Resources over IP Networks

A method that is used frequently to share computing resources is, as explained in the previous section, to share them at application layer. This method is generally easier for users to configure services, and this method abstracts real hardware against computers that access the resource over IP networks. However, this method doesn't allow the use of computing resources as if they are locally-connected, real hardware.

Another method of sharing computing resources over IP networks is to share the resources in a unit of "device" rather than "service". In other words, when a USB peripheral device is shared over IP networks, the remote computer that uses the device will recognize it as a device that is attached to the computer, which is similar to the other devices that are connected to the computer itself. The examples of this method includes use of dedicated hardware such as "device servers" or running software such as USB/IP[6][7] on individual hosts. The other example, which is standardized as Request For Comment (RFC), is iSCSI[8]. Hardware resources that are added to a computer with this method are recognized as real hardware on the computer. In fact, in USB/IP, USB protocols are encapsulated in IP packets, and adds virtual host controller in the architecture to construct the system.

Encapsulating the device protocol into IP packets allows the entire protocol to be transported over IP networks. However, there are several obstacles. For example, bus protocols assume that the requests and responses made in between devices will take on the order of nanoseconds, with limits specified in the protocol standards. The bus

protocols generally assume that all requests and responses will be handled correctly, and not being lost during bus communications. And, bus protocols take up a much broader bandwidth compared to IP networks; the most recent USB protocol, USB 3.0 SuperSpeed, can take up to 5Gbps[9], which is not the bandwidth that can be consumed by average IP networks. Encapsulating device protocols into IP packets will allow users to handle computer resources as these are connected to computers directly. However, there are obstacles on IP networks that need to be overcome on encapsulating the device protocols.

2.2.4 Scalability Issues of Computer Architecture

Even though the boxed architecture of computers is beneficial in terms of stability of communications among resources, there are several drawbacks to the architecture, especially on physical scalability.

First of all, because computers need buses that allow communications between their components, the physical distance between computer hardware and devices must be kept short. If the device is an expansion card, the device needs to be inserted to the appropriate slot on the motherboard; if the device is an external device, the device and the computer needs to be connected with cables that are limited to a certain length, depending on device bus specifications.

Next, since computers have a limited number of motherboard expansion bus slots, the number of devices that can be attached is limited. Some device buses, such as SCSI, USB and IEEE1394, can be chained to certain levels, connecting a device to another device that is connected to the computer motherboard. However, this still limits the number of devices that can be attached to a computer, as the maximum level of chains is specified on each device. Even more, there are computers that don't have certain device buses on their motherboards. Some computers don't have what is called "legacy" device buses, and some portable computers only have a minimum set of device buses. Naturally, this limits the physical connectability of devices. For example, AGP video boards cannot be used on a motherboard without AGP buses, and PCI peripheral cards cannot be added when all PCI bus slots are occupied on the motherboard.

The existing computing architecture has issues on scalability, and these issues need to be resolved in order to allow users to user their computers freely without considering the physical limits placed on a boxed computer.

2.3 All-IP Computer Architecture

Physical scalability is a problem in the existing computer architecture. To resolve the scalability issues, the idea of the "All-IP Computer" is used in this thesis. The All-IP

Computer architecture uses the Internet Protocol (IP) as a single bus that transports both computer device communications and user data (Figure 2.2).

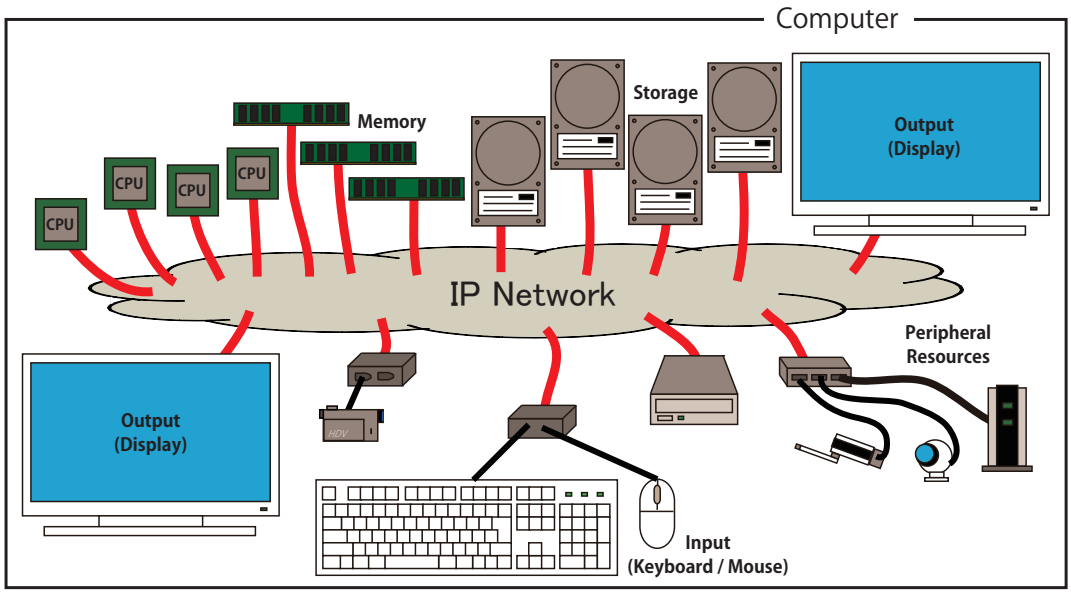

Figure 2.2: All-IP Computer Architecture

2.3.1 Using IP as a Common Bus

In the All-IP architecture, an IP network is the bus the computer components use to communicate with other components. An IP network is a shared bus that transports various kinds of data on the same network. The data that is transported over IP networks includes, but is not limited to: HTTP web requests, mails, remote control applications, multimedia, and so on, as well as control packets such as ARP, ICMP, TCP acknowledgments, and others. These communications are made between multiple computers, and thus it can be said that the IP network is the bus that carries communications with computers outside of the "box". All-IP Computer architecture uses IP network as a heterogeneous bus for transporting all communications that both users and computers use for any types of data, both in terms of software and hardware communications. The hardware communications are done at device level by encapsulating the instructions into IP packets.

To use an IP network as a common bus for computing resources, there are several properties of IP networks that affect the communications in some ways. First, IP networks have greater latencies than computer buses. Device buses sometimes require device requests to be processed in order of nanoseconds, while IP packets are transported over IP networks in milliseconds to sometimes as much as a few seconds. Next,

IP networks sustain chances of losing IP packets on the communication. When TCP is used, the reliability of packet transfer is improved by re-transmission of packets. However, this will increase the delay that occurs on the communications. If UDP is used as a transport protocol, the packet may arrive faster compared to TCP, but the packets may be lost or arrive out of order; because devices are communicating with specified protocols, the messages should not be lost or reordered while being transported over IP networks. In addition, the bandwidth that can be used on IP network is lower than the bandwidth of device buses. Certain device buses, such as Serial ATA 6.0Gbps may use bandwidth up to 6Gbps, while the bandwidth of many IP networks is less than 100Mbps. Since the bandwidths are limited, using IP networks as device buses requires actions such as compression or caching.

The benefit of using IP networks as a common transport of resource communications is the heterogeneous use of a single bus, without considering differences of bus protocols that are specific to each bus type. While most device buses are independent from other device bus types, IP networks can transport any information that can be encapsulated into IP packets. In addition, not only the data that is transported over IP networks, but the types of networks connected to the Internet are also different among different networks; homogeneity is not guaranteed on IP networks, and tweaks on device communications may become necessary on transporting resource communications over IP networks that have narrower bandwidth and greater latency compared to peripheral buses. Thus, methods such as compression of data, utilization of cache, or reducing device protocols become necessary.

There also must be security infrastructure in the system. In other words, if device communications are wiretapped while being passed over IP networks, the device functions may be duplicated by unauthorized third parties. Thus, in addition to user and resource authorizations and authentications, encryption of the device communication should be considered in the system.

2.4 All-IP Computer-Related Work

Connecting computer resources over networks has been done in research projects prior to All-IP Computing. This section describes work done prior to the All-IP Computer architecture.

2.4.1 Desk Area Network

Desk Area Network (DAN)[10] connects computer devices over Asynchronous Transfer Mode (ATM) networks. The idea of connecting devices that reside the outside of

computer hardware is similar to All-IP Computer architecture. An example implementation of DAN is ViewStation[11], which was implemented at the Massachusetts Institute of Technology.

An important difference between DAN and All-IP Computer architecture is that boundaries are defined in DAN while the goal of All-IP Computer architecture is to remove the boundaries between computer hardware. In DAN, devices that exist within the same ATM network can be used by the main computer, but not the devices on other networks. This approach is effective for resource management, such as managing resource existence and address management, and security as the system only needs to consider communications within the network. Since all resources exist within the administrative network, the entire set of resources can be managed by the system administrator.

The drawback of DAN is the limitations on distances between computers and devices. The goal of All-IP Computer architecture is to use computer resources on IP networks without considering the physical and logical locations of computing resources. On the other hand, since DAN uses devices within the same ATM network, both physical and logical locations are limited in the network.

2.4.2 Netstation

Netstation[12][13] introduced the idea of Network-Attached Peripherals (NAPs)[14], which are connected over network protocols. The goal of the project was to heterogeneously unify all the devices on IP regardless of dedicated physical buses for the NAPs. Unlike DAN, which was limited to use on ATM networks, NAPs introduced in Netstation were designed to be used with IP networks.

2.4.3 Plan 9

Plan 9[15] was a project at Bell Labs, and many UNIX kernel developers participated in the project. A goal of Plan 9 project was to re-design the computer resource management for a multi-user computing environment. Plan 9 implemented management of resources that allows mapping between namespace within a specific scope and global IP networks, and also stores all files and user information on a central server. Thus, the system can use resources, which includes all of data, files and users, in integrated manner with IP networks.

The idea of Plan 9 project is similar to All-IP Computer architecture as all computing resources reside on the Plan 9 system are connected to each other over IP networks. One of the descriptive characteristics of the project is that management of entities in the system is centralized in a single management server, which could become a single point of failure in the system. Although resource discovery and administration are

not fully defined in All-IP Computer architecture yet, it is designed to have resource manager and rendezvous manager on the Internet, and the they are responsible for the resources within the same network; managers communicate with each other to share resource information.

3. Virtualization

The computer environment proposed in this work involves virtualization of computers. This chapter briefly describes the virtualization technologies used in common computing environments.

3.1 Virtualizations Used in Application Software

The first examples of virtualization technology involve virtualization used on user applications. These are different from virtualization architectures used in this work, but the requirements for server virtualization have relevance in explaining the architecture proposed in the work.

There are several cases where users may wish to run multiple servers on a single server hardware. If the server hardware has sufficient resources to run virtual machines on the computer, the server administrator may choose to run multiple instances of virtual machines on the hardware. For example, an Apache web server can host multiple websites with the `VirtualHost` directive in its configuration file. This is done by running a user application on top of a single instance of an OS.

This virtualization by application software allows to run multiple instances of servers on a single instance of an OS, without requiring any additional hardware to support the hosting of multiple sites on a single server hardware. The VirtualHost allows server administrators to host multiple sites on a single hardware without any additional hardware support, reducing the number of server machines necessary for hosting multiple sites. On the other hand, VirtualHosts are isolated at the level of web sites only, and all other entities are shared on a single machine. For example, sites defined as VirtualHost share the same application libraries, OS, network links, and hardware.

3.2 Emulation of Computer Hardware

Virtualization of server daemons allows server administrators to host virtual sites on a single hardware, but there are several problems with using virtual sites to construct the virtual servers. Virtual sites differ from real hardware in that they reside on a single OS environment, and the fact that these virtualizations rely on applications. Since these server applications reside on the single OS environment, an incident that occurs on

a single site may affect other sites on the same server. Since these site virtualizations are done on application-basis, whether they can be virtualized or not and how virtualization is done depends on each application. Therefore, even though sites virtualization can be done if the server application supports it, this is not a full virtualization of the whole computing environment. The full virtualization of a computing environment means to virtualize the whole environment, including hardware, OS, and application software.

Computer hardware virtualization has been used since the 1960s. The method used for virtualizing, or "emulating" a computing environment is the translation of instructions issued by CPU, translating physical memory space into the virtual machines' memory areas, using a file as a virtual disk drive, and translating or bridging their network connection into the network which host computer is connected to. This method of computer hardware virtualization has been around as both open source software and proprietary products. Examples of open source computer virtualization include Bochs[16] and QEMU[17][18], and examples of proprietary products include Virtual PC[19] and VMware[20] Workstation, which both later support hardware-assisted virtualization that is described later in this chapter.

Computer hardware emulation in software doesn't require specialized hardware; any computer architecture can run the emulation software regardless of the CPU or chipsets placed on the computer. Emulation of computer environments can emulate other CPUs, such as running ARM-based system on Intel x86 system, even though there is a disadvantage on speed of execution. Since computer emulation requires translation of CPU instructions and memory spaces, the execution speed of the emulated computer environment may be an order of magnitude or more slower.

3.3 Hardware-Assisted Virtualization

The attention shown toward virtualization has grown stronger as use of Information and Communication Technologies (ICT) became popular in real world. As the situations where using computers and servers have increased, the issues of space for server machines, power consumption, and emission of heat have become major interests. Since the numbers of server and client computers have increased, they have taken up more space in the datacenters, and the weights of server racks have become heavier and started to consume more power, which results in producing heat. One method for reducing the numbers of computers is to run multiple computers on a single hardware.

3.3.1 Hypervisors

Recent virtualization technologies add an additional layer called the "hypervisor" to the existing OS architecture. The hypervisor monitors the resources of the host OS,

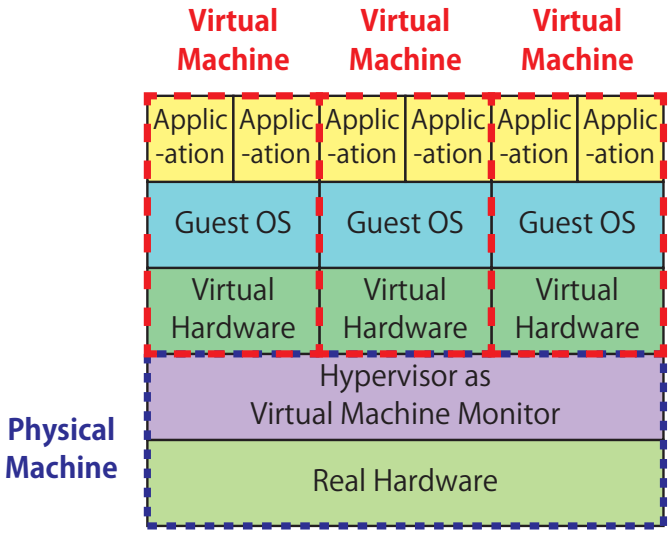

Figure 3.1: A bare-metal hypervisor

which is an OS that manages virtual machines, as well as the guest OS, which is an OS running inside of a virtual machine.

The hardware-assisted virtualization can be classified into two types: bare-metal hypervisor and hosted hypervisor. A bare-metal hypervisor runs directly on top of the real hardware, and the guest OS and applications are always executed on top of the hypervisor (Figure 3.1). On the other hand, hosted hypervisor runs virtual machine monitor as one of the applications or daemons on the host OS (Figure 3.2). Since hosted hypervisors turn existing OS into hypervisors using kernel modules or application programs, virtual machine subsystem and other applications can reside on the same computer system simultaneously.

Examples of bare-metal hypervisors include Xen[21][22], VMware ESX (Figure 3.3), Linux Kernel-Based Virtual Machine (KVM)[23][24] (Figure 3.4) and Microsoft Hyper-V[25], and examples of hosted hypervisors include Parallels Desktop[26], Oracle VM VirtualBox[27], and VMware Workstation. These are the major virtualization products used in the markets today, for both desktop-use and server-use. In desktop-style virtualization, users execute instances of virtual machines on their desktops; in server-style virtualization, virtual machine servers are running within the network, and users use client software to access the server.

3.3.2 Para-Virtualization and Full-Virtualization

The techniques used for virtualizing computers can be divided into three categories: emulation using binary translation, para-virtualization, and full-virtualization[28]. Full-virtualization of a computer is to virtualize or emulate real hardware on virtual ma-

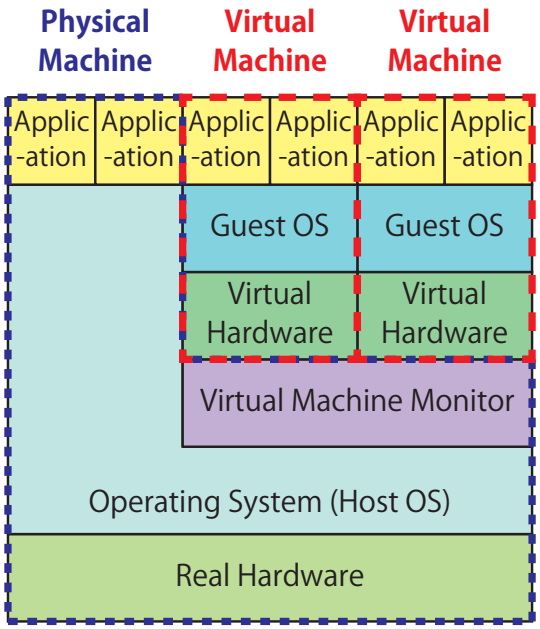

Figure 3.2: A hosted hypervisor

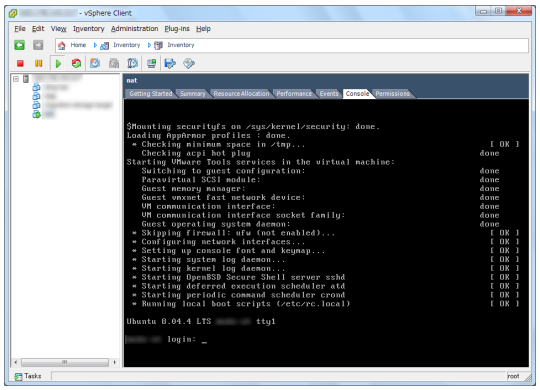

Figure 3.3: VMware ESX vSphere Client

chines, whereas para-virtualization "abstract" some of the hardware in the ways suitable for virtualization system to handle. For example, Xen and VMware support both para-virtualization and full-virtualization of virtual machines. And, binary translation of virtualized machines are to emulate hardware on virtualization software, that translates instructions between virtual machines and real hardware.

In Xen, para-virtualization of a virtual machine is done by modifying Linux kernel to use with Xen. When the guest OS on virtual machine calls system calls, the instructions are translated into "hypervisor calls", which are instructions designed for para-virtualization hypervisor architecture. Then, the hypervisor calls are issued to CPU to handle the instructions; when interrupts were made from real hardware, the

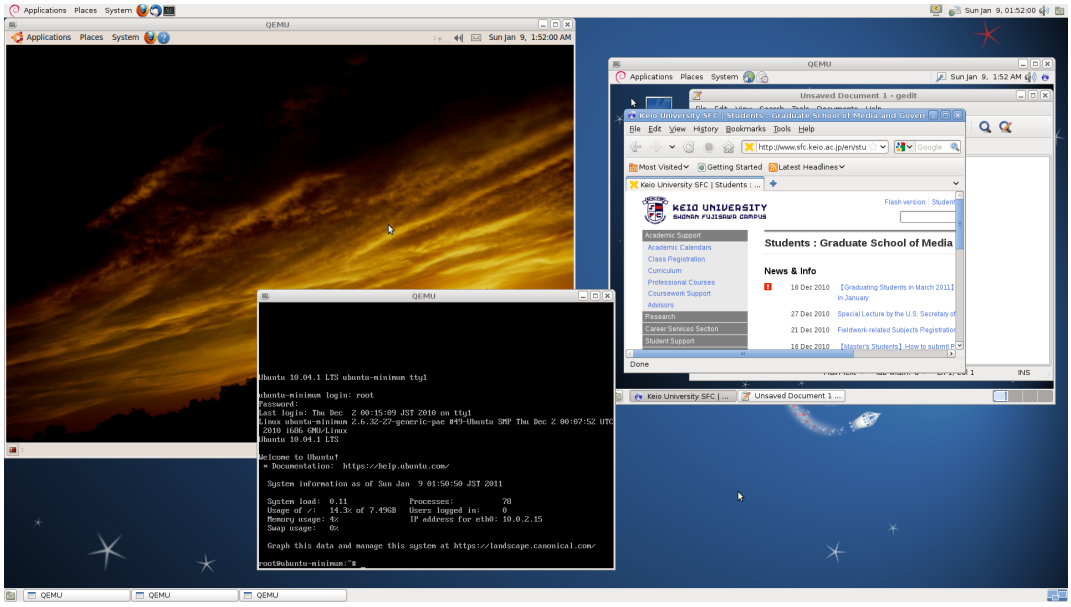

Figure 3.4: Screenshot of QEMU-KVM

hypervisor translates the interrupts into software interrupts that are notified to the guest OS.

Para-virtualization of virtual machine reduces overhead on virtual machines as functions of the OS are translated for virtualization-specific instructions. The drawbacks of para-virtualization is the fact that the OS kernels must be modified to support hypervisor calls. On the other hand, full-virtualization is useful as no OS modifications are necessary, while virtual machine performances are affected for making translations of instructions between virtual machine and real hardware. Thus, recent CPU and chipsets have added supporting mechanisms for full-virtualization.

3.3.3 Principles of Hardware-Supported Virtualization

As virtualization technology has evolved, hardware-based assistance of virtualization has become common (Figure 3.5). The hardware-assistance of virtualization is done on CPU and chipsets, allowing translation of CPU instructions and memory addresses between the real hardware and virtual machines. Support of instructions on CPU are so called Intel VT-x[29] and AMD-V, that allow CPUs to have an additional privilege level to run the virtual machines (Figure 3.5).

Hardware-assisted support of virtualization became necessary due to the ring protection architecture of Intel x86 CPU, which is one of the most commonly used CPUs in the market today. In Intel x86 CPUs, CPU have four different levels called “rings” for executing codes, where ring 0 is the highest privilege and ring 3 is the lowest. Kernels

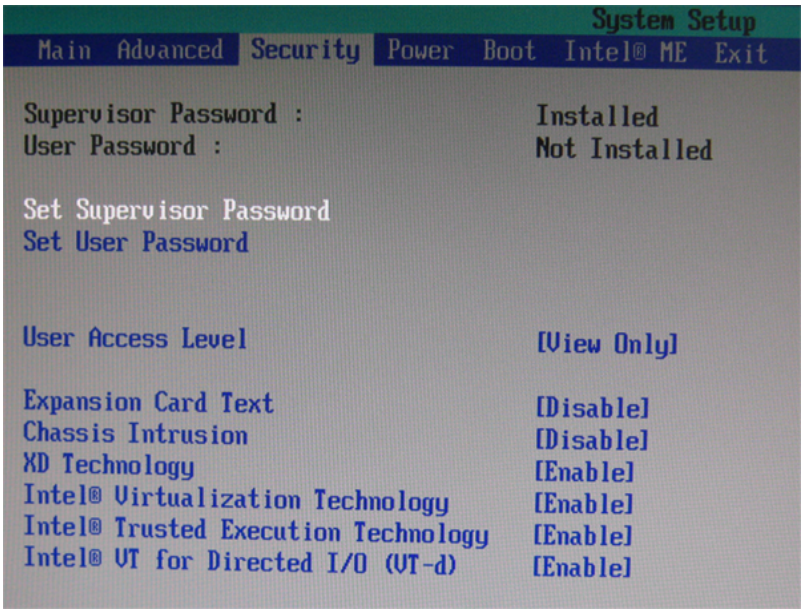

Figure 3.5: Screenshot of BIOS setup for motherboard supporting Intel Virtualization Technology

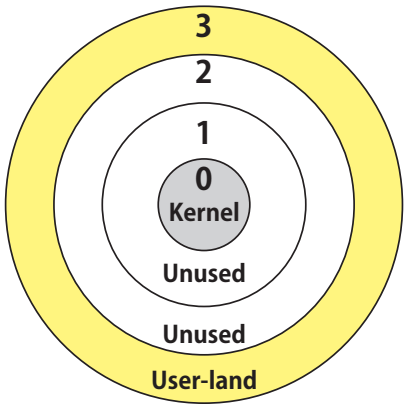

Figure 3.6: Rings in Intel x86 CPU

are executed in ring 0, and applications are executed in ring 3 (Figure 3.6).

Since rings 1 and 2 had not been used, AMD x86_64 architecture has removed rings 1 and 2 in its specifications. When adding hypervisors to run kernels in virtualized environments, the hypervisor layer should be executed at a layer with higher privilege compared to the kernel, which is already occupying the highest privileged layer in the ring protection system. Thus, in CPU virtualization support, new privilege modes were added to CPUs. This support is called "VMX" in Intel CPU products, and "Guest Mode" in AMD CPU products[30]. VMX works independently from ring protection, and has two modes: "VMX root" where virtual machine monitors are executed, and

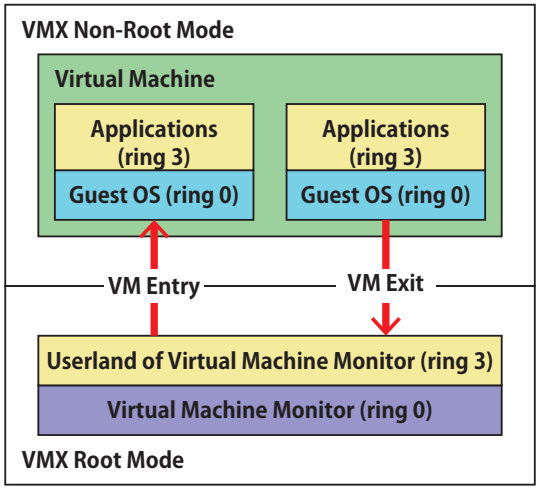

Figure 3.7: VMX modes in Intel x86 CPU

"VMX non-root" where virtual machines are executed (Figure 3.7).

Another necessary hardware feature is the translation of memory space between virtual device addresses and real memory addresses, handled by the Input/Output Memory Management Unit (IOMMU). AMD has IOMMU supports in its chipset products and Intel calls it VT-d, and their functions are primarily to remap DMA memory space to allow use of real devices connected to computers from virtual machines.

In addition to the virtualization applications supporting virtualization features, Linux kernel has features to assist virtual machines, such as virtio[31]. Virtio provides I/O support for storage and network accesses, virtual machine memory space allocation, and use of PCI emulation. The purpose of virtio is to develop a device driver that is specifically designed for virtual device I/O, and to provide a common driver interface for multiple virtualization implementations at the kernel level.

3.4 Cloud Computing

Cloud computing is a buzzword that has become common in recent years. The idea of cloud computing is to export subscribers' work to systems that reside somewhere on the Internet, and subscribers access the service with client software such as web browsers. Since the hardware and basic software installations are done by service providers as a service, cloud computing allows subscribers to install the services relatively easier than installing the entire system at the subscribers' sites[32].

Some examples of cloud services are Amazon Elastic Compute Cloud (Amazon EC2)[33], Google App Engine[34], and Force.com service by Salesforce.com[35].

Table 3.1: Types of cloud computing services

Entity	**Service types and who install the services**		
	Infrastructure as a Service (IaaS)	**Platform as a Service (PaaS)**	**Software as a Service (SaaS)**
Applications	Users		Service Providers
Middleware	Users	Service Providers	
Server Hardware	Service Providers		
Infrastructures	Service Providers		

3.4.1 Types of Cloud Computing Services

When describing cloud computing services, the services can be categorized into several types depending on the services that are provided and the services which subscribers install themselves. The types of cloud computing services are shown in table 3.1.

Under the cloud computing service types, subscribers use provided applications on the Software as a Service (SaaS) subscription model. This approach has been used for several years, especially for web-based e-mail services such as Hotmail[36] or Gmail[37], and it has been common to access the cloud services using web browsers as client software.

3.4.2 Role of Virtualization in Cloud Computing

In contrast to the SaaS subscription model, Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) subscription models allow users to install their own applications on the provided spaces. In the IaaS subscription model, each subscriber can even obtain his or her own OS environment. When allowing subscribers to provide their own OS environment, virtualization technology becomes a key technology in the service.

Providing a real hardware system for each subscriber is difficult, as it requires physical space in server racks for placing hardware, providing sufficient power supply for running the hardware, providing air circulation system to cool the hardware, and the cost of installing and operating the real hardware. Thus, in cloud services, use of virtualization have gotten attention; instead of installing real hardware for each subscriber, creating virtual machines for each subscriber resolves the requirements described above. That kind of server is sometimes called Virtual Private Server (VPS), and it is a service that has become popular in recent years.

It can be said that use of virtual machines have become common in real-world services.

3.5 Benefits of Virtualization

There are several advantages to using virtualization technologies. Because of those advantages, virtualization is being used in various situations at various locations.

3.5.1 Virtual Disk Images

The first advantage is the use of virtual disk images. A virtual disk image can be copied as a file, and it can be backed up, restored, or duplicated when users demand; for example, when a user wishes to create ten computers with same user environments. On physical hardware, the user needs to take the same steps of installing OS and user applications, changing OS and application settings, applying necessary software patches, and other activities before the environment becomes usable. The user could choose to use a HDD duplicator, but that still involves physical attachment and detachment of HDD, which is time consuming and requires the user to have physical access to the HDD.

Virtual disk images are treated as a "file" on the hypervisor computer. Thus, the virtual disk images can be copied or moved freely, just as transferring a file on a computer. Thus, the user can duplicate the same virtual disk image for the number of computers the user wants, change a few settings such as host names and network configurations, then start running multiple machines simultaneously on the same or different hypervisors.

This handling of virtual disk image files cannot be done on physical hardware. Therefore, it can be said that treating virtual disk images is an advantage of using virtualization.

3.5.2 Checkpointing Virtual Machines

The next advantage is the checkpointing of virtual machines. Virtual machines can be paused and resumed when necessary, and their states can be saved or restored when necessary. This applies to both contents of memory spaces allocated to the virtual machines, as well as disk drives.

For memory contents of virtual machines, virtual machines can "save" the state, and "pause" the machine. The pause does not involve any shutdown of applications nor the OS itself. If for some reasons the hypervisor needs to be shut down or rebooted, the virtual machines can be paused, the hypervisor restarted, then the execution of virtual machines resumed.

Virtual disks can also be checkpointed. The simplest way of checkpointing a virtual disk is to make a copy, as described in Section 3.5.1. In addition to making a full copy, in many hypervisors, it is possible to take diffs of the virtual disks. For example, a

"snapshot" of the virtual disk is taken at a certain point. Since the snapshot is being taken, changes are written to a separate virtual disk space instead of the original disk space, keeping the original disk unchanged. When the user wants to roll back the virtual disk image to the time where the snapshot was taken, the differential part of the virtual disk will be removed. When the differentials were removed, the virtual disk is restored to the state where the snapshot was taken.

Checkpointing of virtual machine memory spaces and virtual disk images are another benefits of using the virtualization, which is not available on using the physical hardware.

3.5.3 Live Migration

Another benefit of virtualization is the live migration of virtual machines. With live migration, the virtual machines can move from one hypervisor to another without terminating the execution of the virtual machine. This allows virtual machines to be temporarily executed on another hypervisor while the hypervisor that normally hosts the virtual machine is undergoing maintenance. Or, when certain virtual machine is occupying hardware resources available on the hypervisor, either that specific virtual machine or other virtual machines at idle state may migrate from the hypervisor to reduce load on the original hypervisor.

The details of migration will be described in the next chapter, but live migration of virtual machines is one of the major benefits of using virtualization technology.

4. Migration of Processes and Virtual Machines

The concept of migration has been explored at several levels, which have different units of entities to be migrated over the networks. The most thoroughly researched is process migration, but migration of objects at different granularity has also been studied.

Recently, migration at the virtual machine level has become both viable and attractive. Migration at the virtual machine level is also supported by hardware, as already discussed in Section 3.3.3.

4.1 Process Migration

Migration of processes has been expected to increase the computer performance when the process requires more computing resources; migrating processes to the computers with more resources may improve computer performance. Various process migration mechanisms were introduced in Nuttall[38].

4.1.1 A Process and Its Resources

A process is an instance of a program, which could be either a user application or a system-related program, running on an OS environment. Numerous processes are running on top of an OS, and they share the resources provided by computer hardware and managed by the OS.

When a process is executed on an OS, a memory space for storing the process image is allocated. Then, the process runs on the computer, storing necessary data on the segments of main memory for handling the necessary instructions to a CPU. Many of the processes use numerous resources other than just the memory space. Some of the common resources include, for example, file descriptors for reading from and writing to files, and sockets for communicating other hosts on IP networks. Running programs depend on shareable libraries and other version-dependent features of the OS. They also depend on namespaces, including file systems and even process IDs. Some processes storing a great chunk of data may consume a noticeable number of memory pages.

Nowadays, a process has grown larger in its image size, and relationships between multiple processes running on either a single or multiple computers have become common. Thus, it can be said that the management of resources consumed by a process has become complex in modern OSes.

In addition to the visible resources, processes consume CPU time, which is one of the major resources on a computer system. Even though multiple processes seem to be running on a single computer, each core of a CPU can be utilized by only a single process at a time. The parallel execution of numerous processes have become possible with time sharing system (TSS), and the processes that are currently not being executed on the CPU are put into a "sleep" state. This scheduling is another important role of an OS, and being scheduled on the OS means the process will consume some CPU resources while it is running.

4.1.2 Ideas of Process Migration

The concept of process migration has been explored since the earlier years of computing age. Process migration is the movement of a running process from a computer to another, continuing execution of the process and sustaining resources utilized by it.

Process migration becomes useful when resources on a computer that the process is running have become short, or at a case where the process must be kept running while the computer being required to restart. If a process could be migrated from a computer to another, the running process could claim additional resources if necessary and if available.

4.1.3 openMosix: an Example of a Process Migration Facility

openMosix[39] is an example of process migration implementation. It had been developed on Linux kernel 2.4, but the project was discontinued on March 1, 2008. Mosix[40] is still being developed as a commercial product, and it works on Linux kernel 2.6.

openMosix and Mosix use a patched Linux kernel to allow migration of processes. Once a host is added to the cluster of computers, the resources available on those computers are treated as a single resource pool on the cluster. When a process requires more resources on the computers, the process is migrated automatically to one of the members of the cluster. The process may migrate back and forth, depending on the availability of resources.

Sometimes process migration facilities require the program to be modified to support the migration, such as management of file descriptors. However, openMosix and Mosix can run the programs without modifying them.

4.1.4 Issues of Process Migration

Process migration may be useful for purposes such as load balancing. On the other hand, there are difficulties for sustaining process states, differences of file descriptors on each hosts, and management of memory space becomes necessary.

The issues of process migration resides on the following elements. A computer that had process running will be called "source computer", and the computer that the process will resume on after process migration will be called "destination computer" in the list.

- **Both source and destination computers need to have a CPU with the same architecture.** A process is built dependent to the CPU architecture of which the process was designed for. If the CPU architecture changes, the process cannot be continued after it is migrated.
- **Both source and destination computers need to have the same shared libraries.** If a process migrates, there is no guarantee that the elements external to the process itself would be similar on the destination computer. If the process was running on the source computer referring to a shared library, the same version of the same shared library must exist on the destination computer.
- **File descriptors must be managed independent of the source computer.** The file descriptor is locally managed on the computer running the processes. If a process is migrated, the file descriptors may become different on the destination computer.
- **Other entities in the OS must reside as similar to the source computer.** For example, if a process was writing to and reading from a file, the file used by the process must reside on the destination computer. If the process was communicating with other processes running on the same computer, those other processes must also exist on the destination computer. If any of the entities in the OS change after the process migrated to the destination computer, it is likely that the process will face problem on the destination computer.

In addition, each process has its own state, and the state must be synchronized when the process migrates. Process migration is considered as finished when transfer of states completes from one node to another[41]. Several approaches were considered in the process migration, but most of the approaches are unused even after years of research.

Thus, migration of processes require numerous management of their resource utilization, and are considered to be obstacles for achieving the process migration to be used in the real world environment.

4.2 Live Migration of Virtual Machines

Process migration handles migrating objects in units of "processes". This may sound efficient as the memory size of processes is generally limited to the order of several megabytes to a couple of hundred megabytes, which is a reasonable size to transfer over networks from one node to another. However, as was described in Section 4.1, resource management is considered to be a major obstacle in achieving process migrations.

4.2.1 Ideas of Live Migration

Virtual machine migration is the movement of environment in units of "machines". Live migration[42] of virtual machines is used for moving a virtual machine running on a hypervisor to another hosting hypervisor without terminating the virtual machine execution. In general, to change the hypervisor executing the virtual machine, steps such as:

- Terminate or pause (hibernate) the virtual machine
- Copy the virtual machine configuration to a new hypervisor
- Copy virtual disk image (or have a network-shared disk)
- Start or resume the virtual machine on the new hypervisor

becomes necessary. Live migration of virtual machines achieves this procedure without involving termination of the virtual machines.

Live migration of virtual machines is useful when the hypervisor needs to be terminated for hardware maintenance or other reasons, or the hosting hypervisor is short on resources. In these cases, virtual machines can be migrated to another hypervisor, and continue to run on the new hypervisor. In this thesis, the hypervisor that has the original virtual machine is called the "source hypervisor", and the hypervisor that accepts the migrated virtual machine is called the "destination hypervisor".

To migrate a virtual machine from one hypervisor to another, there are several requirements that should be met. One of the requirements is that both source and destination hypervisors need to have a shared storage space, for example shared over NFS or iSCSI, unless synchronization of disks is to be done at the time of migration. Synchronization of disks consumes enormous amounts of bandwidth and time, so generally hypervisors are configured to access shared disk space. Several research projects have investigated duplicating the necessary parts of disks first, then continue the rest after virtual machine is migrated or when that portion of the disk is requested[43].

Live migration of a virtual machine takes several steps to accomplish. First, when migration is triggered, a "blank" virtual machine with the same configuration as the

original virtual machine is made on the destination hypervisor. Second, the contents of the memory space allocated to the migrating virtual machine are copied to the destination hypervisor. During the synchronization, the virtual machine continues to run. Therefore, it is possible that the contents of the memory pages will be altered. When the first pass of the synchronization is done, the virtual machine is paused, and the memory pages that were modified after the synchronization are copied again. Finally, when the contents of memory pages are synchronized, the state of virtual machine execution is moved to the new virtual machine on the destination hypervisor; the virtual machine on the destination hypervisor is resumed, and the virtual machine on the source hypervisor is terminated.

Live migration is useful for improving the availability of the virtual machines. Migrating virtual machines to other hypervisors will allow them to run even when the hypervisor is terminated, by running them on other hypervisors. If the migration is done live, without terminating virtual machine execution, it becomes possible for virtual machines to continuously provide their services even at the time of executing migrations.

4.2.2 Migration Procedure

When live migration of a virtual machine is executed, several steps are taken before the migration actually takes place. The virtual machine migration is accomplished in the following steps:

- **Create receiver virtual machine:** on the destination hypervisor, create a virtual machine with the same settings as the source hypervisor to accept the migrating virtual machine.
- **Transfer memory contents:** transfer memory contents from the source hypervisor to the destination hypervisor to synchronize the memory space of the virtual machine. At this time, the virtual machine is still running on the source hypervisor. Applications can still make changes on the memory space allocated to the virtual machine.
- **Synchronize changes made on memory space:** if changes are made on memory space after they were copied for live migration, the changed portion of the memory space is synchronized again. The synchronization will iterate until the diffs of memory space on source and destination hypervisors become small enough. The allowance of diffs are defined by hypervisors.
- **Pause virtual machine, and copy remaining memory contents and machine state information:** when memory contents are transferred, the virtual machine is paused. Then, the remaining memory space is synchronized, and

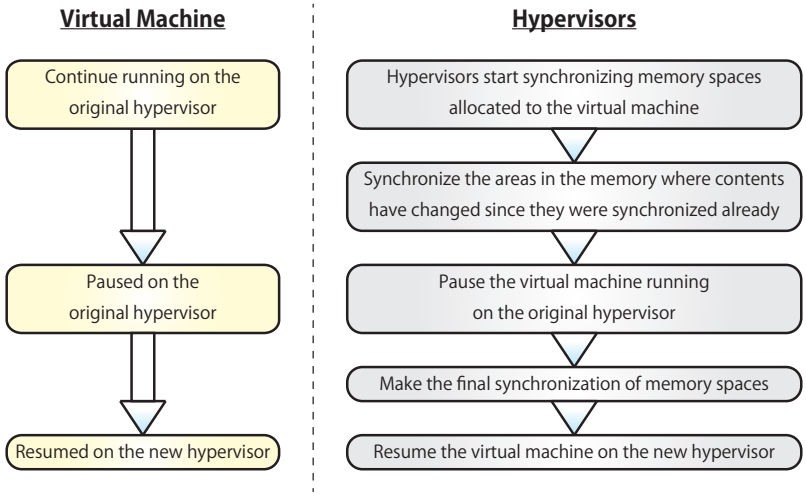

Figure 4.1: Migration sequence

hardware state information, for example contents of CPU registers and states of device I/Os, is transferred to destination hypervisor.

- **Resume virtual machine on destination hypervisor:** when the state is transferred to destination hypervisor, execution of virtual machine is resumed. Once the virtual machine is resumed on the new environment, resources on source hypervisor can be freed.

Because of this change, memory contents will be synchronized again when it has been synchronized once. When the differentials of memory spaces between the original and destination hypervisors become small enough, virtual machine will be paused on the original hypervisor, make the final synchronization, and resume it on the new hypervisor. The sequence is shown in Figure 4.1.

The architecture proposed in this thesis utilizes live migration of virtual machines to improve computer performance. The next chapter, Chapter 5, describes the architecture and strategies.

4.3 Comparison of Process and Virtual Machine Migrations

Even though both process and virtual machine migrations are called "migration", there are many differences between these two styles of migration, as comparison is shown in Table 4.1.

One of the major differences is the entity that is being transferred during the migration procedure. Process migration transfers only the memory image of the running

Table 4.1: Comparison of process and virtual machine migrations

Migration Entity	**Process**	**Virtual Machine**
Synchronization of memory contents	Smaller	*Larger*
File descriptor management	*Complicated*	Easier
Network connections continuity	*More difficult*	Easier
Thread management	*Difficult*	Easy
User environment	*Each computer*	Continue

application, while virtual machine migration transfers the memory image of the entire virtual machine. On the Mosix website, the authors argue that virtual machine migration is expensive in time and required memory[44]. It is true that on virtual machine migration, when larger memory space is allocated to the virtual machine, it will take longer for the live migration to accomplish. In terms of memory synchronization, process migration may seem to work better, depending on the workload that cause the migration.

However, the unit of transfer also affects the types of processes that can be executed on computers. On virtual machines, any applications can be executed as long as they are supported by the guest OS; applications can be executed without being modified, and types of applications being executed are not limited. Shared libraries and other files, which may be referred by from the process, will also migrate with the virtual machine. Thus, there is no need to be concerned about whether or not the required libraries are available on the destination computer.

In contrast, many limitations are placed on process migrations. The examples of limitations, from Mosix, are shared memory[45] and threaded applications[46]. Using shared memory on process migration is an obstacle as it becomes difficult to change contents of memory on one node while running the process on the other node; the same memory space used on the application-running node may or may not be available on the other node. When applications become complicated regarding resource management, process migration may not be as effective as it is expected to be.

As the couplings between entities become common, process migrations become more and more difficult. Some of these issues are alleviated by choosing to migrate the entire virtual machine, rather than the process. There is also a weakness on virtual machine migration, which is the amount of memory space that needs to be transferred during the migration procedure. However, as networks have evolved and bandwidth have become broader, the issue of transferring a large size of data have become easier than the past. Thus, even with this drawback of memory transfer size, it can be said that the virtual machine migration is easier to manage than the process migration.

5. Systems Design

As was explained in Chapter 2, computers today have physical limitations on device buses, and have boundaries that separate computers from each other. This thesis proposes a virtualization-based computer to utilize computer resources that are demanded by applications running inside of the virtual machines. This chapter describes the architecture of the system.

5.1 Virtualization-Based Computer Architecture

This thesis introduces a virtual machine architecture with adaptation to resource requirements. This architecture is expected to take part in an IP bus-based computer architecture, which was explained in Chapter 2. Since IP networks have less bandwidth, and more latency and effects that affect computer performance compared to peripheral buses, it becomes necessary to reduce the communications between the OS and hardware. Each computing resource has different requirements on bandwidth and latency. If the processes which users are running can migrate to resources that meet their requirements, the performance may be improved.

In this architecture, "computers" are running hypervisors, and "OS"es that users use as their computing environments are running in forms of virtual machines. Since each hardware system is different from the others in terms of CPU clocks, cores, cache, memory space, chipsets, and other resources, some hypervisors are stronger than others in terms of computing resources. When there are multiple candidates to host a given virtual machine, the system should automatically select a hypervisor on which to run the virtual machine.

5.1.1 Migration Based on Resource Utilization

In this architecture, migration of virtual machines is triggered based on the utilization of computing resources. Different processes have certain behavior patterns, and their performance is dependent on different resources. For example, video encoding often requires computational power on CPUs, while parsing and finding the right file among a large stack of text files requires disk I/Os to accomplish the requested tasks. When using computers for daily uses, there are certain cases where users may wish to have

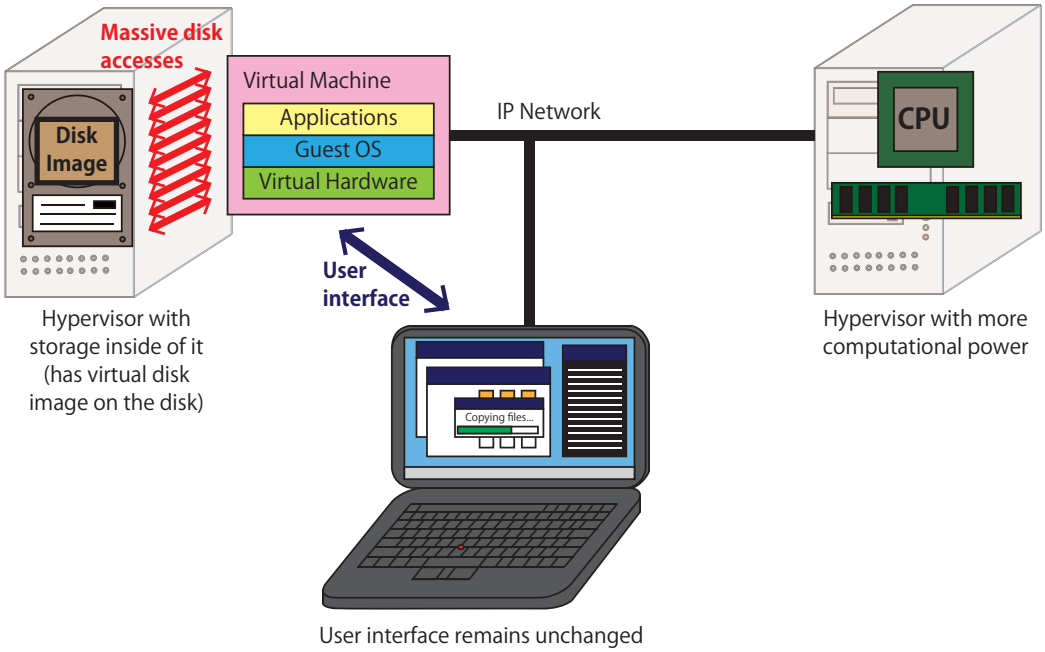

Figure 5.1: An I/O-bound virtual machine migrates as close as possible to the storage location

access to better computational resources for a certain period of time; if a user can "borrow" the resources for a certain period of time, that would be sufficient for meeting the user's requests.

Thus, when users request certain computing resources, virtual machines migrate from one hypervisor to another; the requests are determined by monitoring the running applications on the virtual machine. To illustrate, assume that there are two hardware systems running cooperating hypervisors, one with better computational power and the other holding the disk image which the virtual machine is accessing. The user interface runs on a terminal, such as a laptop, in front of the user, and remains unchanged. When the user executes a process that performs a lot of disk I/O, the virtual machine is migrated to a hypervisor closer to the disk that is used by the virtual machine (Figure 5.1). On the other hand, when the virtual machine is running applications that require more computational power, the virtual machine is migrated to the hypervisor with better computational power (Figure 5.2). When the virtual machine requires both disk I/Os and computational powers, the migration is balanced to one of the hypervisors by observing the trends of resource usage.

In this design, the user interface (a keyboard, a mouse and a display) remains unchanged. The virtual machine migrates transparently, allowing the user to continuously use the environment.

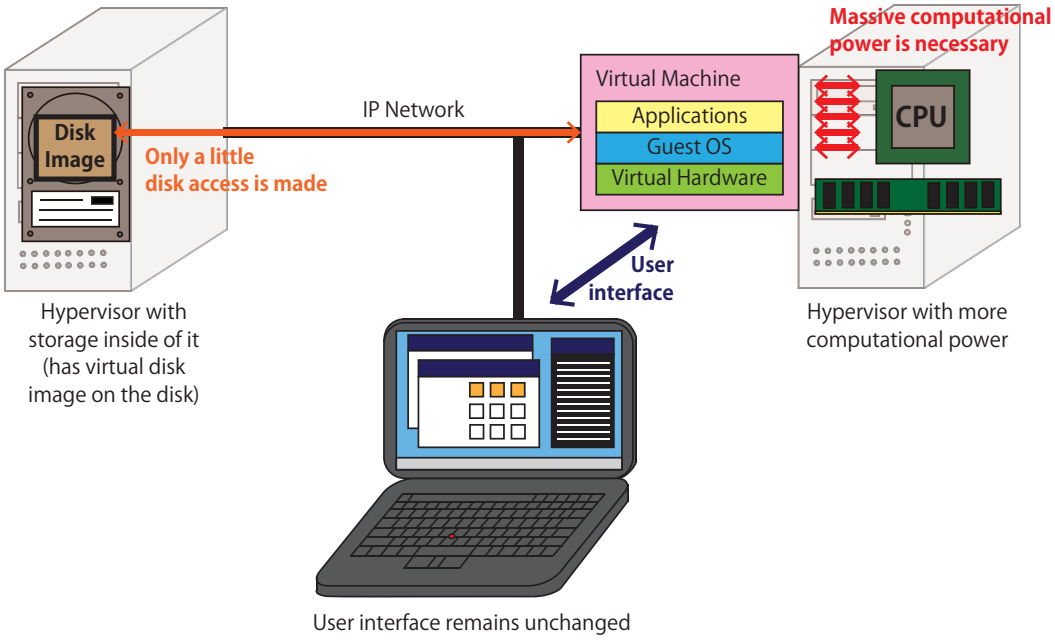

Figure 5.2: A compute-bound virtual machine migrates to a hardware environment with good CPU power

5.1.2 Benefits of the Architecture

As mentioned in Figures 5.1 and 5.2, sometimes the relationships between hypervisors, virtual machines, and available computing resources may not be balanced to maximize resource utilization. Thus, a method for reducing unnecessary I/Os becomes necessary, especially when resources are accessed over IP networks that have narrower bandwidth and higher latency.

This reduction of I/Os has been studied before, in the field of process migration. Intelligent Disks (IDISKs)[47] aimed to move data-intensive processing closer to the data themselves. IDISKs are hard disks with embedded processors, RAM space, and Gigabit network links, and the links are connected to a switch, allowing full link bandwidth between disks. The architecture has several benefits, such as off-loading computation from expensive desktop processors and reducing data movement through the I/O subsystem. Since the demand against disk storage has increased, the architecture may be efficient in certain cases.

The ideas of moving computation closer to the necessary computing resources has been considered for many years, but management of the resources that the processes use is an obstacle in process migration architecture. Computer architecture based on virtual machine architecture resolves issues on resource managements. Since live migration takes place in units of “machines”, issues of managing memory addresses and file descriptors become unnecessary; virtual machine monitors translate them into the

real hardware with hardware assistance or by binary translations. Since the achievement of process migrations were considered difficult for those management reasons, the architecture should reduce the cost of management and therefore allows architecture to be more realistic than that of the process migration.

5.1.3 Difficulties in Achieving the System

The architecture provides benefits as described in Section 5.1.2, but there are additional issues that need to be resolved for achieving the system. First of all, since virtual machine migration takes place by monitoring the resource requirements, an architecture for monitoring and gathering the statistics, as well as a system for making migration decisions out of the statistics become necessary.

Next, even if resource usage statistics were gathered from hypervisors and virtual machines, making decisions of migration is an obstacle in the architecture. Since it is difficult to determine how long the resource would continuously be used in the future, the decisions must be made based on multiple parameters and information. If the process ends while migration is taking place, the migration procedure even becomes an overhead in the system. Making a best decision from given parameters is difficult, and thus a method for making decisions is necessary.

The other issue that is considered an obstacle in this architecture is utilization of remote resources. For example, extending computer resources over IP networks, in means of running software on remote sites, connecting to peripherals over IP networks, and exporting works to computers on the Internet, is one of the required features for computer architecture that is to be developed in the next generations. When using remote resources, connections to IP networks are necessary, but IP networks have much narrower bandwidth with higher latency and jitter compared to the computer buses. If the entire virtual machine migrates to the hypervisor that has necessary resources, the issues on IP networks become less critical. In detail, a virtual machine migrates to use the resource on the other hypervisors, and when it is done with using the resource, migrating back to the original hypervisor. It does require time and network bandwidth for migrating virtual machines, but the cost of migration is less critical compared to the bandwidth and latency required on using IP networks as a bus to communicate with remote resources.

In addition, even though parameters can be collected from hypervisors and virtual machines, the values that are gathered from the nodes are much more complicated than just the numbers. For example, even with just the CPUs, they have several parameters that relate to each other in effect on performance. Those parameters include:

- **Clock speed**: speed at which processors can handle instructions.

- **Cache size**: cache size, and availability of level three cache, may affect CPU performance.
- **Memory access speed**: affects performance of applications and OS itself when a massive access to memory occurs.
- **Number of cores and availability of Hyper-Threading**: number of cores which CPU has, and the number of threads supported by CPU.
- **Additional instruction sets**: instructions that are supported by CPUs for specific purposes. An example of instructions include Streaming SIMD Extensions (SSE).

Even if the clock speeds are the same, if the number of cores is different, the performance may change depending on the level of parallelism achievable for the given workload. Or, even if the clock speeds are the same, if the instruction sets supported by CPUs are different, the performances would differ. Even with just the CPU, several variables must be evaluated for determining the execution of migration.

In this system, virtual machines migrate from a hypervisor to another. The migration decision is made dynamically based on the resource requirements which the virtual machines need. Therefore, in addition to hypervisors and virtual machines, a mechanism for collecting and managing virtual machine statistics, as well as a mechanism for making decisions of migration are necessary.

5.2 Dynamic Migration Decisions

A focus of the architecture is to determine migration based on the resource utilization of hypervisors and virtual machines. Dynamic migration decisions are made based on available resources and their utilization, as well as estimated duration for accomplishing the live migration.

5.2.1 Parameters That Should be Considered

Computer workloads are often either CPU-bound or I/O-bound. In past work on load balancing, indices such as CPU queue length, CPU utilization, normalized response time, remaining processing time, the sum of processing time by active processes, and accumulation of processing time were often brought up[48]. However, in addition to the CPU workloads, it is obvious that memory accesses, disk accesses, and network performance should be taken into consideration when measuring loads on a computer system[49].

In addition to the computing resource utilization, since live migration of a virtual machine requires time and network bandwidth, and reduces virtual machine performance while the live migration takes place, the cost of executing the migration should be taken into account in this system. Thus, the parameters that are considered to be necessary for making migration decisions are information such as:

- **CPU consumption**: the time and clocks occupied by running applications.
- **Memory consumption**: the amount of memory space occupied by applications, as well as the speed and amounts of I/O accessing to memory.
- **Disk I/Os**: the amount of I/Os issued to disks by the OS.
- **Network I/Os, bandwidth, and latency**: network accesses that virtual machines use.
- **Migration cost**: time and bandwidth necessary for performing live migration of virtual machines.

As was described in Section 5.1.3, to begin the discussion of the architecture, this thesis focuses primarily on CPU and disk I/O consumptions. However, when considering the computer resource utilization, the parameters listed above may also be targeted; taking all the resources above will be an extended work to this thesis.

5.2.2 Migration Cost

The cost of performing live migration can be calculated from the memory size allocated to the virtual machine and available bandwidth. The cost is determined in units of seconds, which is the duration to accomplish the migration.

Out of the migration procedure steps, the transfer of memory contents from source hypervisor to destination hypervisor takes up the bulk of the time. Since the memory space allocated for the virtual machine can be obtained from virtual machine configurations, the time in seconds necessary for transporting memory contents can be estimated in the following equation:

$$T_M = \frac{MemorySize(MB) \times 8}{NetworkBandwidth(Mbps)} + TimeforStateSynchronization(s) \qquad (5.1)$$

Since there will be network activity from other processes and there may be a bottleneck somewhere other than the network itself, there is no guarantee that the full network bandwidth can be used to transport memory contents.

5.3 Pre-Experiment

To figure out the consideration points for designing the architecture, pre-experiments were done.

5.3.1 Encoding Videos While Migrating

The first pre-experiment involves encoding of a ten-minute Digital Video (DV) file into MPEG video format, using the `ffmpeg` application. The facilities used in the experiment are shown in Table 5.1, and the topology is shown in Figure 5.3.

Table 5.1: Pre-experiment facilities

	Hypervisor 1	**Hypervisor 2**	**Virtual Machine**
CPU	Intel Core i5 660 3.33GHz (2 cores, 2 threads)	Intel Core 2 Duo E8400 3.0GHz (2 cores)	1 core assigned
Memory	4GB DDR3	4GB DDR2	1024MB allocated
Chipset	Intel P55 Express	Intel Q45 Express	Simulated Intel 440FX
Network (NFS)	Realtek 8168d/8111d	Realtek 8110s	-
Network (Migration)	Realtek 8169	Intel PRO/1000	-
OS	Debian GNU/Linux 6.0 squeeze x86_64		Debian GNU/Linux 5.0 lenny i386

The disk image of the virtual machine is stored on a NFS server. The hypervisors and NFS servers were connected to a switching hub with Gigabit Ethernet connection. Networks for NFS accesses and migrating were two separate networks.

Figures 5.4 and 5.5 show the performance of ffmpeg video encoding, with migration from hypervisor 2 to hypervisor 1. The migration was executed while running ffmpeg encoding. From the figures, it can be inferred that the CPU clock affects the performance of virtual machines, and the change will take effect even though the virtual machine migrated while executing the application.

The first pre-experiment was drawn to see the effects of CPU changes while running an application. The next pre-experiment involves disk I/Os.

In the second experiment, a FreeBSD dummynet bridge was installed to limit the network bandwidth between hypervisor 1 and NFS server. The performance was measured, without migrating but staying on hypervisor 1, to observe the effects of disk I/Os on a NFS server on performance.

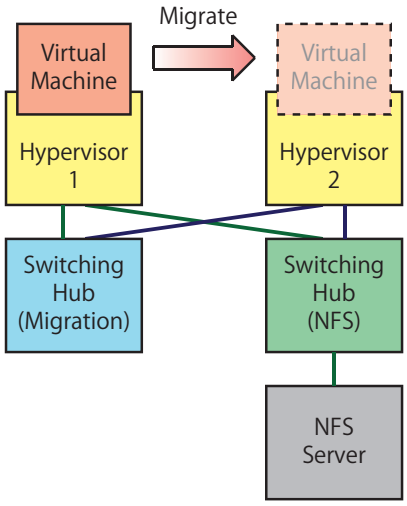

Figure 5.3: Video encoding pre-experiment map

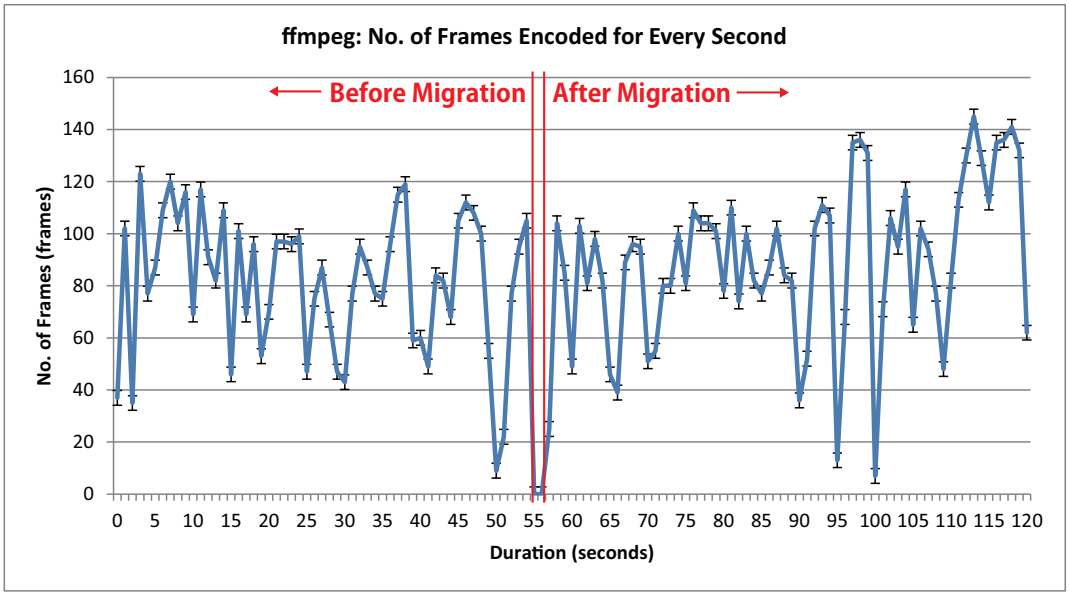

Figure 5.4: Number of frames processed every second on ffmpeg, migrated from a hypervisor to another

The number of frames encoded, with bandwidth limitation to the NFS server, is shown in Figures 5.6 and 5.7. It is obvious that disk I/O waits affect the application performance.

The pre-experiments show that both CPU performance and disk I/O performance affect application performance, and thus the balance between those values must be taken into account.

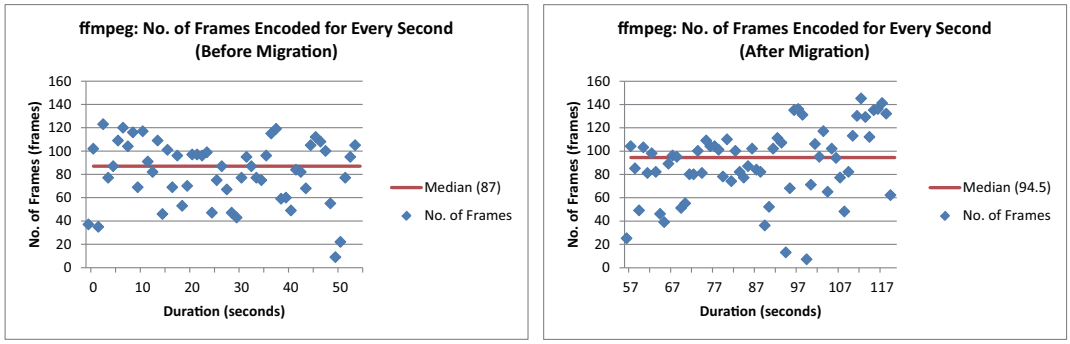

Figure 5.5: Number of frames processed every second on ffmpeg, migrated from a hypervisor to another in two graphs

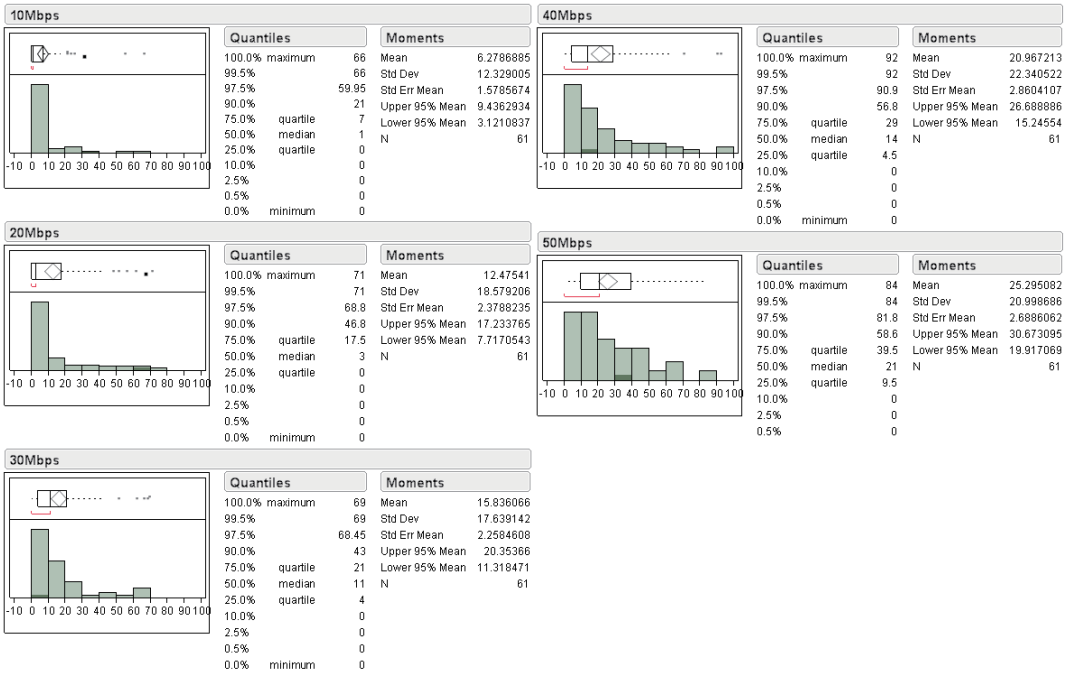

Figure 5.6: Number of frames processed every second on ffmpeg, with NFS bandwidth limited to 10-50Mbps

5.3.2 Bandwidth and Migration Time

The other consideration for improving application performance is the time required for performing live migrations. The live migration is done to improve the performance, shortening the time period necessary for accomplishing a task. Thus, there must be a method of estimating the time period required for migrating to ensure that making the migration will improve performance.

The experiment was done using the same hardware on Table 5.1, except that CPU

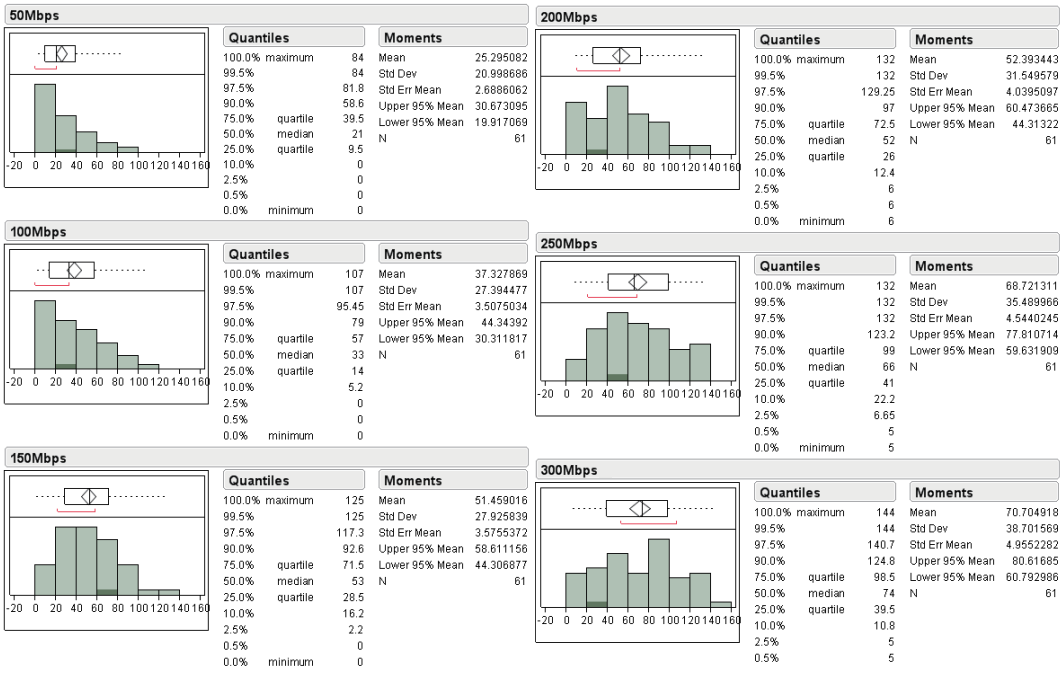

Figure 5.7: Number of frames processed every second on ffmpeg, with NFS bandwidth limit (by 50Mbps)

clock on hypervisor 1 was limited to 1.6GHz (133MHz base clock multiplied by 12.0). A FreeBSD dummynet bridge was placed between the hypervisors, to limit the bandwidth available for making live migration. The hardware specification of the dummynet bridge is shown in Table 5.2, and the topology is shown in Figure 5.8.

Table 5.2: Dummynet bridge hardware specification

	Dummynet Bridge
CPU	AMD Phenom II X4 810 2.6GHz
Memory	2GB DDR2
Chipset	AMD 780G
Network	Intel 82574L Gigabit Ethernet (PCI-Express) Intel 82541PI Gigabit Ethernet (PCI)
OS	FreeBSD 8.1 i386

The ffmpeg encoding was executed again to observe the relationships between migration and improving application performances. The results of the experiment are shown in Table 5.3.

Experiment (1) is simply running ffmpeg video encoding within the virtual machine without making any live migrations. Experiments (2), (4), and (5) runs the video

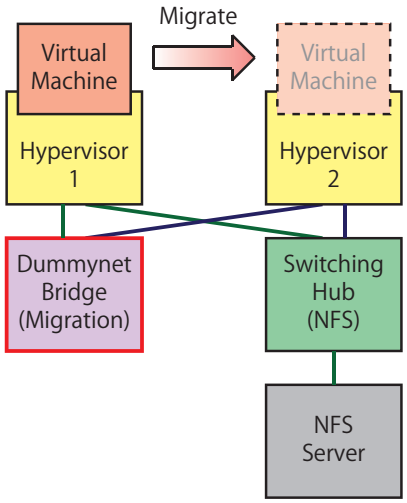

Figure 5.8: Migration bandwidth pre-experiment map

Table 5.3: ffmpeg migration pre-experiment

Migrate at	**Band -width** (Mbps)	**Hypervisor 1** (1.6GHz)			**Hypervisor 2** (3GHz)		
		ffmpeg (min)	**Mig to 2** (min)	**iperf** (Mbps)	**ffmpeg** (min)	**Mig to 1** (min)	**iperf** (Mbps)
(1) None	-	4:55	-	-	3:22	-	-
(2) 30 sec	1000	3:46	0:35	751	4:37	0:38	667
(3) 60 sec		3:38	0:30		3:38	0:41	
(4) 30 sec	200	4:03	1:33	179	4:33	1:55	173
(5) 30 sec	100	5:04	3:25	93.1	3:45	4:43	87

encoding, and start executing live migration manually after 30 seconds of wall-clock time. Experiment (3) is the same, except that the live migration was executed after 60 seconds. The duration for completing video encoding is shown in column "ffmpeg", the duration between executing migration and actual migration occurred is shown in column "Mig to", and the result of `iperf`, network throughput benchmark, is shown in column "iperf" to indicate the actual network bandwidth available.

In the table, experiment (1) shows that video encoding performance will be affected by CPU clock speed available on the hypervisor. Next, experiment (2) shows that the performance will be affected when migration is made to another hypervisor. When the virtual machine migrated from hypervisor 1 to 2, the performance improved; when the virtual machine migrated from hypervisor 2 to 1, whose CPU clock is lower than the original hypervisor, the performance declined.

In experiment (3), performance on hypervisor 2, which performed a migration from a hypervisor with the faster CPU to the slower, has improved since the virtual machine

stayed on the faster hypervisor longer than experiment (2). The performance was expected to decline compared to start migration after 30 seconds on migration from hypervisor 1 to hypervisor 2, but the performance turned out to be better on this case.

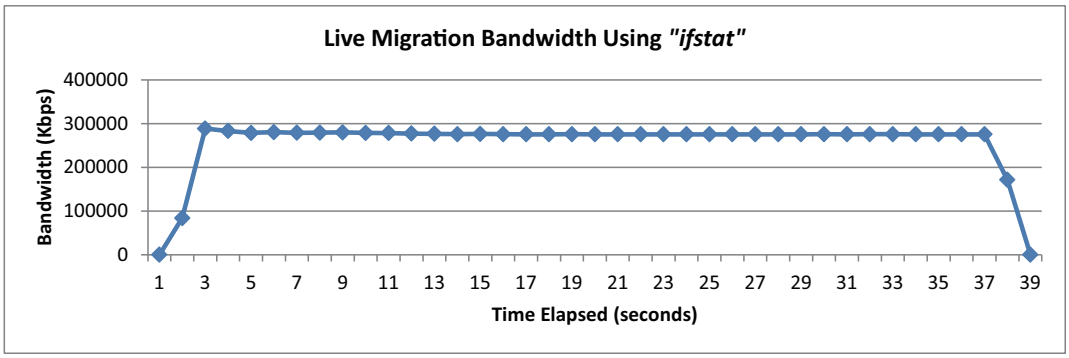

Figure 5.9: Bandwidth consumed by migration, monitored using `ifstat`

When a live migration is performed, network bandwidth is consumed as shown in Figure 5.9. From the data, it can be informed that no more than 300Mbps of bandwidth was consumed for performing migration. Thus, in experiments (4) and (5), network bandwidth available for performing live migration was limited to 200Mbps and 100Mbps.

When network bandwidth was limited to 200Mbps (experiment 4), the migration time took three times longer than without having network bandwidth limitation. The experiment has shown that the migration is effective even when network bandwidth is limited to 200Mbps, but the performance improvement compared to without performing migrations has declined compared to without having limits by approximately 7%. This is the case where migration was still effective even when the network bandwidth is limited.

On the other hand, when the network bandwidth was limited to 100Mbps in experiment (5), the performance did not improve. Especially when the test was run on hypervisor 2 for migrating to hypervisor 1, the video encoding had ended before the live migration has completed. This is the case where migration has become a waste: the synchronization of memory contents wasn't necessary, and the performance may have been affected while performing the migration. This particular virtual machine had 1024MB of memory space allocated, and average of approximately 75.50Mbps of network bandwidth was consumed during migration. If Equation 5.1 is appropriately stated, simply copying the memory contents would take approximately 108 seconds, shown in Equation 5.2.

$$T_M = \frac{1024(MB) \times 8}{75.50(Mbps)} \approx 108(seconds) \tag{5.2}$$

However, migration took 283 seconds until it completes. The test was running a video encoding application, which may have changed memory contents while they were being synchronized. Even during the migration process, applications can make changes on memory spaces regardless of whether the space has already been synchronized or not; the change will be synchronized again when it has been synchronized once.

By default, KVM continues to synchronize memory contents until the duration between pause and resume of virtual machine for performing actual migration falls less than 30 milliseconds. We can infer that since synchronization was too slow to complete, a greater portion of memory contents has been modified since the first synchronization. Because the final stage of migration will not take place until memory changes are settled, the synchronization has iterated multiple times, not allowing the final migration to complete. This phenomena did not appear when the same test was conducted with no limitation on migration network bandwidth. From this experiment, it can be said that network bandwidth available for migrating virtual machines will affect migration duration and effectiveness of the migration.

This pre-experiment has shown that live migration will not always improve the performance of the virtual machine. The duration of the migration, which is to be estimated from available network bandwidth and allocated memory space, is possible. However, additional overhead of synchronizing modified memory space must be taken into account, and this duration may not be easily estimated without knowing the memory space being modified by applications running inside the virtual machine.

5.4 System Architecture

From the requirements and pre-experiments described in the prior sections, a system architecture for making dynamic migration decisions is designed.

5.4.1 Components

To collect the information described in Section 5.2, a management system is necessary. The management system could be organized in peer-to-peer architecture, or a management server may exist to gather all information and make migration decisions.

In this architecture, a virtual machine is "homed" to a specific hypervisor. A virtual machine is an environment which the users use as a daily-use environment. Since each user "owns" a virtual machine, the hypervisor that is closest to the user would be a **home hypervisor**; virtual machines migrates to other hypervisor when necessary, but would come back to the home hypervisor at idle status.

Therefore, in this system, a client-server architecture is used to manage hypervisors and virtual machines. The home hypervisor also runs **Migration Management**

Server, which collects statistics information from virtual machines. Each hypervisor and virtual machine runs a **Statistics Client**, which reports resource utilization statistics to Migration Management Server. When the Statistics Client is started on a hypervisor, the hypervisor is registered to Migration Management Server; when it is started on a virtual machine, the virtual machine is registered to Migration Management Server with information of which hypervisor the virtual machine is hosted on. Thus, it can be said that management server becomes a central host that gathers all information from both hypervisors and virtual machines (Figure 5.10).

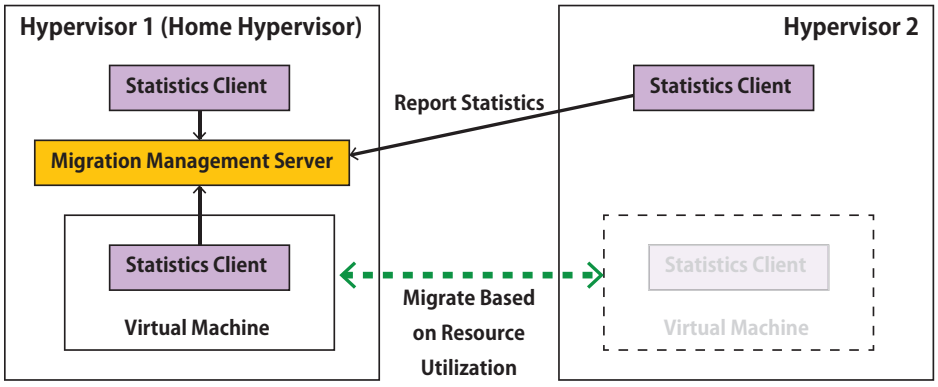

Figure 5.10: Migration Management Server and Statistics Client

The Migration Management Server is the entity that is responsible for making migration decisions. Migration Management Server uses collected information, and if migrating the virtual machine to other available hypervisors is expected to improve the performance, migration instruction is issued to the hypervisor that currently hosts the virtual machine.

5.4.2 Features Necessary on the System

To achieve the system designed in the previous sections, the following mechanisms are necessary:

- **Managing list of hypervisors and virtual machines**: Migration Management Server must keep track of the hypervisors and virtual machines that the servers can obtain information from.
- **Collecting statistics in hypervisors and virtual machines**: to report statistics information to a management server, hypervisors and virtual machines must collect their statistics information in the way which management servers can interpret.

- **Reporting statistics to management server**: then, hypervisors and virtual machines report their statistics to management servers. Management servers need to receive those information and store them for a certain period of time.
- **Making migration decisions at management server**: in accordance with collected information, the management server determines the migration decisions. If a migration should take place, the management server tells the source hypervisor, which hosts the virtual machine, to issue migration request to the destination hypervisor. The management server should also tell the destination hypervisor that the virtual machine will be migrated from the other hypervisors.
- **Negotiation between hypervisors**: when running virtual machines, hypervisors need to negotiate the parameters that are used for executing virtual machine for receiving the migrating virtual machine. Then, migration should be executed when both hypervisors are ready for migration.
- **Updating information on management server**: management server holds information of which hypervisor runs the virtual machine at that time. This information needs to be updated on the management server.

The implementation must have the features listed above for allowing virtual machines to migrate in accordance with available computing resources.

6. Implementation

The implementation of the system is described in this chapter. The system obtains performance statistics from hypervisors and virtual machines to determine whether or not to migrate the virtual machine.

6.1 Entities

The system described in the thesis is designed around Linux systems. The basic principles of the system architecture could be applied to other virtual machine monitor systems, but this implementation is specifically designed to run on QEMU-KVM virtual machine monitors. This implementation uses QEMU-KVM version 0.12.5 on Linux Kernel 2.6.32.5 (x86_64 64-bit architecture). QEMU-KVM already has live migration feature, thus the system for determining and executing the live migration is designed in the implementation.

As was explained in Section 5.4.1, the system is composed of the following entities:

- **Migration Management Server**: collects information from hypervisor and virtual machine clients. The Management Server can be running on any node, and it is also responsible for making migration decisions.
- **Statistics Client**: runs on each hypervisor and virtual machine. It collects information of hypervisors and virtual machines, and reports resource utilization status to Management Server.

The entities communicate with each other using a client-server model using TCP connections.

6.2 Implementation Design

In this implementation design, the Management Server is responsible for gathering all statistics, making migration decisions, and managing the migration status of virtual machines. This could be implemented in other ways, such as having each virtual machine and hypervisor report resource usage statistics individually to the other hypervisors, but for simplicity of implementation, this particular implementation is centrally managed on the Management Server.

6.2.1 Migration Management Server

Migration Management Server receives information from both hypervisor and virtual machine statistics clients, and organizes information to make migration decisions. The two primary types of information handled on Migration Management Server are statistics from each client, and statuses related to migration.

In the system, clients send statistics reports to Migration Management Server, and statistics from all clients are stored in Migration Management Server. Migration Management Server checks statistics of clients when statistics were updated, and when Migration Management Server discovers the necessity of migrating a virtual machine, the statuses of each virtual machine, destination and source hypervisors are marked as "should be migrating".

After each time Hypervisor Statistics Clients send statistics reports to Migration Management Server, the clients also poll for the statuses stored on Migration Management Server. When clients are marked to get involved in migrations, clients send information related to migrations, which will be described in Section 6.2.2. The Migration Management Server is responsible for managing the progress of live migration procedures. Then, the statuses of those clients are reset on the server, which continues to monitor clients for their resource utilization statistics.

Sequence of the Migration Management Server is shown in Figure 6.1.

6.2.2 Hypervisor and Virtual Machine Statistics Client

Statistics Clients are executed on hypervisor and virtual machine hosts. They collect information from the hosts and report the statistics to the Migration Management Server. The information collected from the host is as follows:

- **CPU:** clock speed in MHz, number of cores, model name, model number, cache size, and utilization rate
- **Memory:** free and total real memory space, free and total swap space, memory access speed
- **Disk:** CPU I/O wait at the moment, and benchmark score
- **Network:** traffic sent and received, and bandwidth available
- **Processes:** list of processes running on the host, and their loads against system

CPU I/O wait is the time of which CPU has been idle for waiting the I/O to complete. This value can be used for determining if I/O has been a bottleneck on the system or not. The information is taken from the kernel statistics and from making benchmarks.

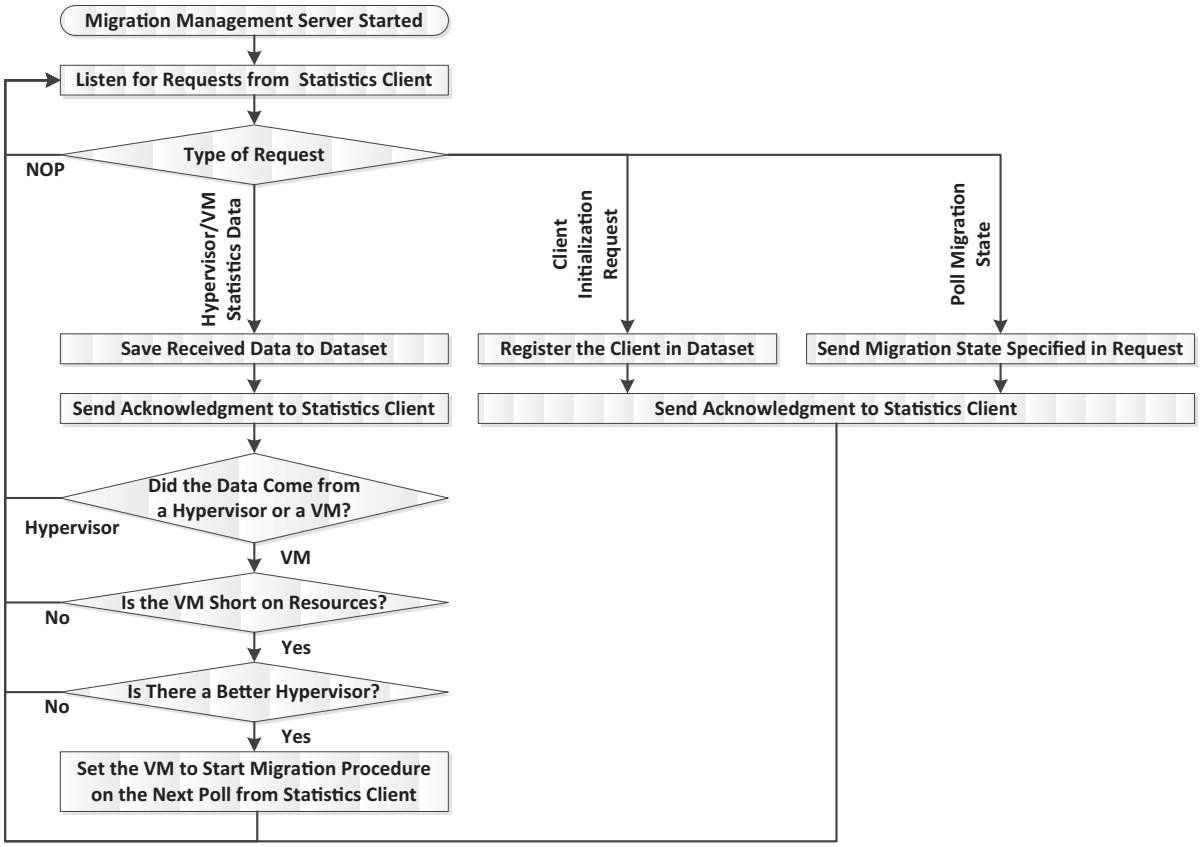

Figure 6.1: Function sequence of Migration Management Server program

Access speed to memory and disk, and network bandwidth are not available as kernel parameters. Thus, when the Statistics Client starts on a hypervisor, benchmark tests are executed for making comparison of hypervisor capabilities. For measuring memory access speed, `memcpy()` is used to measure a rough performance for copying memory space; for measuring disk access speed, `iozone` benchmark is executed to the disk where virtual disk image for the virtual machine resides. And, for measuring network bandwidth, `iperf` is executed to another hypervisor within the same migration network, and to the NFS server. The benchmarks are only rough measurements of resources, but they are usable for obtaining base scores for making comparisons.

The collected information can be classified into dynamic statistics, which change among the hosts running different processes, and static statistics that will not change while the host is running. Classification of static and dynamic statistics are shown in Table 6.1. In the table, measured scores are written in *italic*. These scores may change over time as they are affected by other processes running on the same machine. However, to make a baseline for comparing of hypervisors, these scores will be considered as static in this implementation.

The functions of Statistics Clients are shown in Figure 6.2.

Table 6.1: Classification of statistics into static and dynamic

	CPU	Memory	Disk	Network	Processes
Static	Clock Speed, Model Info., Cache Size	Total Space, *Access Speed*	*Base Benchmark Score*	*Bandwidth*	-
Dynamic	Total Utilization in Percent	Total Utilization in Percent	CPU I/O Wait Percentage	Traffic Sent and Received	Process Name, CPU/Memory Utilization per Process

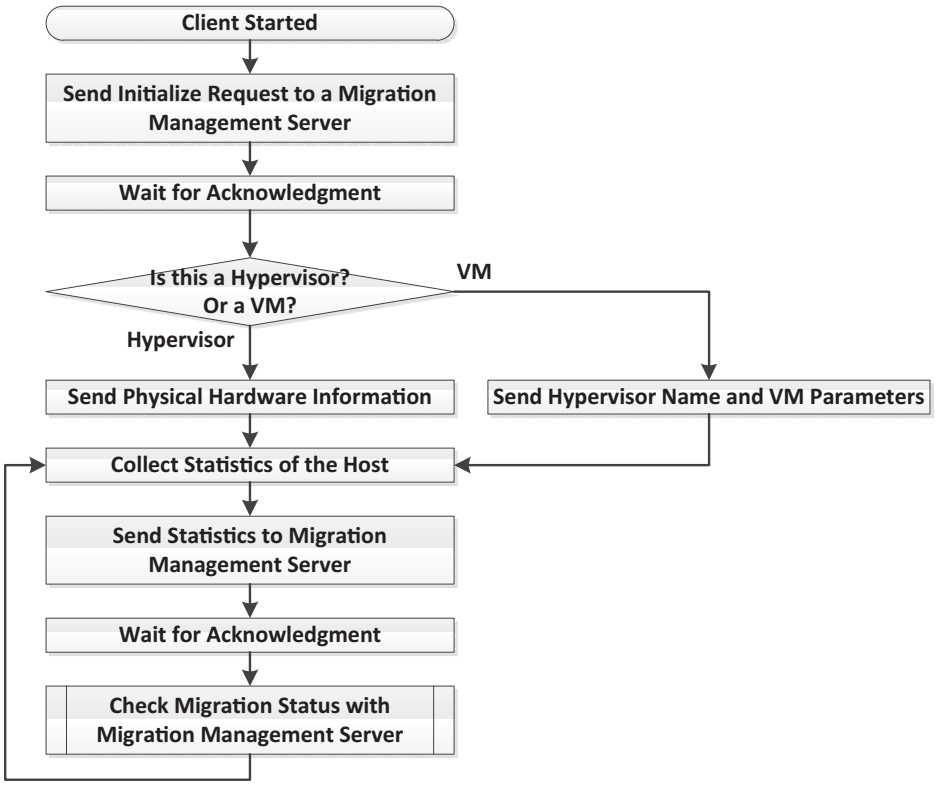

Figure 6.2: Function sequence of Statistics Client program

6.3 Determining Virtual Machine State

Dynamic migration of the virtual machines is determined from the parameters collected from hypervisors and virtual machines. Migration decisions are made by the Migration Management Server as it is the only entity in the design that has information about all hypervisors and virtual machines.

6.3.1 Determining Load on Virtual Machine

Whether or not to migrate, and where, is determined in accordance with the resource requirements of the virtual machine. As a simple indicator of the virtual machine state, load average can be used.

Load average indicates the average number of processes that are waiting for a CPU to become available or are waiting on I/O. When this value is high, the system is loaded. The load average is calculated by the kernel, and on Linux systems, it can be obtained from `/proc/loadavg` file easily. An output of load average will look like: `0.04 0.22 0.23 1/331 13990`. The kernel reports five numbers: the number of jobs in the run queue or waiting for disk I/O, averaged over 1, 5, and 15 minutes, the number of processes or threads in kernel scheduling entities currently executing and existing on the system, and the process ID of the process that was most recently executed on the system.

According to the author[50], on a CPU with single core, system administrators should be concerned with the resource consumption if the load average has been greater than two for more than fifteen minutes. The article also suggests that in a multi-core or a multi-CPU system, dividing the load average by number of CPUs will give an indication of whether the system is loaded or not.

Hypervisor and Virtual Machine Statistics Clients report the load average and number of CPU cores on the system. Thus, the load average can be used to determine whether the system is loaded for some reason or not.

6.3.2 Determining Whether a Virtual Machine is CPU or I/O-Bound

If the virtual machine is loaded, migrating it to a hypervisor with better resource capability may improve the application and virtual machine performance. To determine the most suitable hypervisor for executing the virtual machine, virtual machine statistics are analyzed to determine which resource is demanded by the virtual machine.

The primary distinction of the virtual machine state is whether the running processes are CPU-bound or I/O-bound. The resource which the process relies on is determined by CPU utilization rate and CPU I/O wait time. CPU utilization rate indicates the amount of CPU processing time occupied by running processes or the OS itself, and I/O wait time indicates the percentage of CPU being idle due to waiting for I/O to complete before the next computation can be continued. The CPU utilization and I/O wait time can be obtained from the `sar` command (Figure 6.3).

Since the CPU utilization rate and I/O wait time share the same total of 100% CPU time, comparing these two parameters will give a rough view of whether the virtual machine is waiting for computation to complete or for I/O to be finished, or is idle. If

```
$ sar -u 1 5
Linux 2.6.32-5-amd64 (muda-desktop)     01/09/11        _x86_64_        (4 CPU)

13:44:22        CPU     %user     %nice   %system   %iowait    %steal     %idle
13:44:23        all      0.53      0.00      6.61     42.06      0.00     50.79
13:44:24        all      0.46      0.00      6.21     40.00      0.00     53.33
13:44:25        all      0.25      0.00      2.76     44.61      0.00     52.38
13:44:26        all      0.00      0.00      0.26     52.08      0.00     47.66
13:44:27        all      0.94      0.00     12.74     33.25      0.00     53.07
Average:        all      0.45      0.00      5.84     42.18      0.00     51.53
$
```

Figure 6.3: Example output of `sar` command (at I/O wait state)

either the CPU utilization or I/O wait time rate is excessively high, the determination will be simple: if CPU utilization rate is high, the virtual machine is running CPU-bound jobs; if I/O wait time rate is high, the virtual machine is running I/O-bound jobs. Where it becomes difficult is when the CPU utilization rate and I/O wait time rate are similar. There must be a way to distinguish whether the migration will improve the total performance or not.

In this thesis, the change in resource capability is calculated among available hypervisors. If the change in resource capability is greater than the differences between CPU utilization rate and I/O wait rate, the migration may improve performance. Otherwise, the migration may not bring enough improvement on performance, resulting the migration to become waste; there may even be a risk of performance declining after the migration.

6.3.3 Additional Considerations for Determining Virtual Machine State

The CPU utilization rate and I/O wait rate give a rough idea of whether the virtual machine is CPU-bound or I/O-bound. However, there are other additional considerations that could be taken into account to improve the accuracy of migration execution. Some such additional parameters are memory utilization rate and network traffic.

One of the additional considerations is the amount of memory space occupied within the virtual machine. Sometimes virtual machines are not allocated all of the memory space available at a particular hypervisor. When the free memory space becomes short, a virtual machine may start paging the memory space to swap space on the disk, causing a lot of disk I/O to the virtual disk image. Thus, even if the virtual machine is at the state where it is CPU-bound, if memory consumption seems high, I/O should be considered, even more than CPU power; when swap events occur, CPU has to wait for disk I/O, reducing the total computational power.

The other consideration is network traffic. If the virtual machine is consuming

network bandwidth, the virtual machine may be migrated to a hypervisor with better network bandwidth.

6.4 Determining Live Migration

From the parameters described in the previous sections, conditions for executing live migration were defined. In this thesis, when the virtual machine falls into all of the following conditions, the live migration is triggered:

1. **Load average is increasing, and the value of load average is above the number of CPU cores assigned to the virtual machine.** As described in Section 6.3.1, load average indicates whether or not the computer is loaded, and the computer is considered to be loaded when the value of load average is above the number of CPU cores. However, because load average could be above the number of CPU cores right after a heavy workload ended and before the computer gets settled, the implementation should check the trend to see whether the load average is decreasing or increasing.

2. **For duration of 20 seconds or longer, either a single process utilizes 70% or more of the CPU, or CPU I/O wait of the entire OS is 10% or above.** These parameters are used for determining if the virtual machine is in a CPU-bound, I/O-bound, or idle state. The load average gives a rough idea of whether the virtual machine is loaded or not, but the reason for the load may not be obtained from the value of load average. These parameters are used for determining the real condition of the virtual machine.

3. **The resource should be utilized longer than the expected migration completion time.** Expected migration completion time is calculated by adding twice the base migration time for synchronizing the whole memory space and the time for synchronizing memory space allocated to the active processes. The base migration time was doubled to consider the case where the whole memory space had to be iterated; this may not be enough if massive accesses to memory space were made, but is used for weighing to ignore short-term workloads.

The values were picked informally by observing trends while testing the implementation. When the resource is utilized at the threshold value or greater for threshold duration, the resource is considered as utilized.

Since a CPU assigned to a virtual machine is a thread on the hypervisor, sometimes the values may be reported smaller than the actual utilization. Thus, to avoid errors, the implementation was designed to ignore three consecutive errors. That is, if the value is decreasing for four entries in the dataset, the load is considered to be decreasing; if

the value has been increasing, and the value decreased for past two entries and then increase again, the load is still considered to be increasing.

6.5 Hypervisor Selection

Once the virtual machine state is determined, the most suitable hypervisor for accomplishing the job is selected from the available hypervisors. From the observations made in Sections 6.3.2 and 6.3.3, the criteria for selecting a hypervisor are determined.

Upon selecting a suitable hypervisor, the hypervisors are scored on their capabilities and availabilities. Capabilities are "static" information on Table 6.1, and availabilities are calculated from dynamic statistics. Hypervisor selection is based on the calculated scores.

6.5.1 Scoring CPU Performance

Our algorithm for scoring the raw hardware performance of a CPU has several parameters including: clock speed, number of cores, cache size, or additional instructions available on CPU, such as MMX, Streaming SIMD Extensions (SSE), Extended Page Table (EPT), and so on. In addition to these static parameters, CPU utilization must also be taken into account.

Out of the parameters, the number of cores can be evaluated easily as assignment of virtual CPUs to virtual machines on QEMU-KVM is done in units of threads; if one virtual CPU is assigned to a virtual machine, it will appear as a single-threaded process on the hypervisor[51]. Thus, it can be estimated that if any one of the cores on the hypervisor is close to the idle state, that is enough to execute the virtual machine on top of the hypervisor. The same estimation can be made even when multiple virtual CPUs are assigned to a single virtual machine; if the number of CPU cores available on a candidate hypervisor is greater or equal to the number of virtual CPUs assigned to the virtual machine, the candidate should be able to execute the virtual machine.

The other CPU-related parameter that can be evaluated easier is the instructions on the real CPU. Even if the physical CPU has newer instructions, the virtual machines don't get to see the additional instructions. This is because when starting QEMU-KVM, CPU type is specified as one of the following:

- **QEMU-KVM Specific:** qemu32, qemu64, kvm64
- **Intel CPU:** pentium, pentium2, pentium3, coreduo, core2duo, (Atom) n270
- **AMD CPU:** athlon, phenom

In order to execute QEMU-KVM with hardware-assisted virtualization, Intel VT or AMD-V is required. Out of the processors listed above, Intel VT is supported on Core Duo and Core 2 Duo CPUs only, and AMD-V is supported on relatively newer Athlon and Phenom CPUs. Since these CPUs are the same or newer than the CPUs available in the selection, it can be assumed that the instructions supported on QEMU-KVM are also supported on the hypervisor. Thus, even though the system collects CPU flags, they can be taken out from consideration at this time; this consideration may become necessary in the future, but this thesis will not take the instruction sets into account.

The remaining CPU parameters are: clock speed, cache size, and utilization rate. Clock speed obviously affects the computation speed when the process is CPU-bound. Cache size may also improve performance when it increases, but a web article[52] indicates that when cache size changed from 1MB to 4MB, the performance increment was approximately one-level of CPU clock increasement on video encoding and file compression while the performance did not improve on the CPU-intensive benchmark applications. From the article, it can be inferred that having more CPU cache will improve the performance, but is not as important as clock speed.

6.5.2 Selecting a Hypervisor

With the variables collected from hypervisors and the virtual machine, a suitable hypervisor is selected. If either CPU or I/O consumption is dramatically higher than the other, the hypervisor with the highest score will be selected. The scores are calculated in the following manner:

- **CPU:** clock speed multiplied by (1 - CPU utilization rate)
- **I/O:** result of the disk benchmark executed at the initialization

The scores are measured under a unified method, which is obtained by calculation for CPU and benchmark for I/O. The other variables, such as the current load on the hypervisors, memory access speed, and network bandwidth, are used if the scores above cannot select a single hypervisor that is appropriate for executing the virtual machine.

6.5.3 Estimating Migration Time

If a hypervisor is selected, and if that selection is different from the hypervisor that currently executes the virtual machine, migration may seem to be effective for improving the performance. However, one other step is added for determining migration, which is to estimate the migration duration. As observed in Section 5.3.2, if migration duration is too long, the migration may not complete before the computation ends in the virtual machine.

The base migration time can be estimated by Equation 6.1, which was introduced in Section 5.2.2.

$$T_M = \frac{MemorySize(MB) \times 8}{NetworkBandwidth(Mbps)} + TimeforStateSynchronization(s) \quad (6.1)$$

Out of the variables, memory size can be obtained from the virtual machine parameter, and network bandwidth was calculated as a benchmark at the startup. The fraction portion of the equation can be calculated, but the actual time for synchronizing the modified portion of memory space can only be estimated from the memory space consumed by the process in the virtual machine. Equation 6.2 will be used to estimate the time for additional synchronization, assuming that the all memory spaces used by processes in the virtual machine get modified while making the first round of synchronization.

$$T_S = \frac{MemorySize(MB) \times 8 \times MemorySpaceOccupied(\%)}{NetworkBandwidth(Mbps)} \quad (6.2)$$

However, even with the Equation 6.2, the time for accomplishing the state synchronization may not be estimated as the memory space will be continuously changed if the processes are writing to memory while the synchronization is running. As described in Section 4.2.2, the synchronization is iterated until the diffs of memory space on source and destination hypervisors become small enough. By default, QEMU-KVM uses 30 milliseconds as the allowance to pause the virtual machine execution while executing migration, which is the sum of: pausing the virtual machine, synchronizing the last portion of memory space, transferring CPU and I/O states, and resuming the virtual machine on the new hypervisor.

This additional synchronization is not always easy to estimate as the number of iterations are not predictable at the state where the processes are running. Thus, this additional synchronization will be used with a parameter rather than making a simple calculation, which will be configured in the system with several different values.

From the estimation and the parameter, the migration duration is predicted. If the migration seems to end in a reasonable duration, which is another threshold parameter, the migration will be executed.

6.6 Executing Migration

Even though the Statistics Clients are named “Statistics”, they will also be responsible for making actual live migrations. As mentioned in Section 6.2.1, Statistics Clients poll the Migration Management Server for the necessity of migration. When migration is initiated, first the source hypervisor sends parameter information for executing

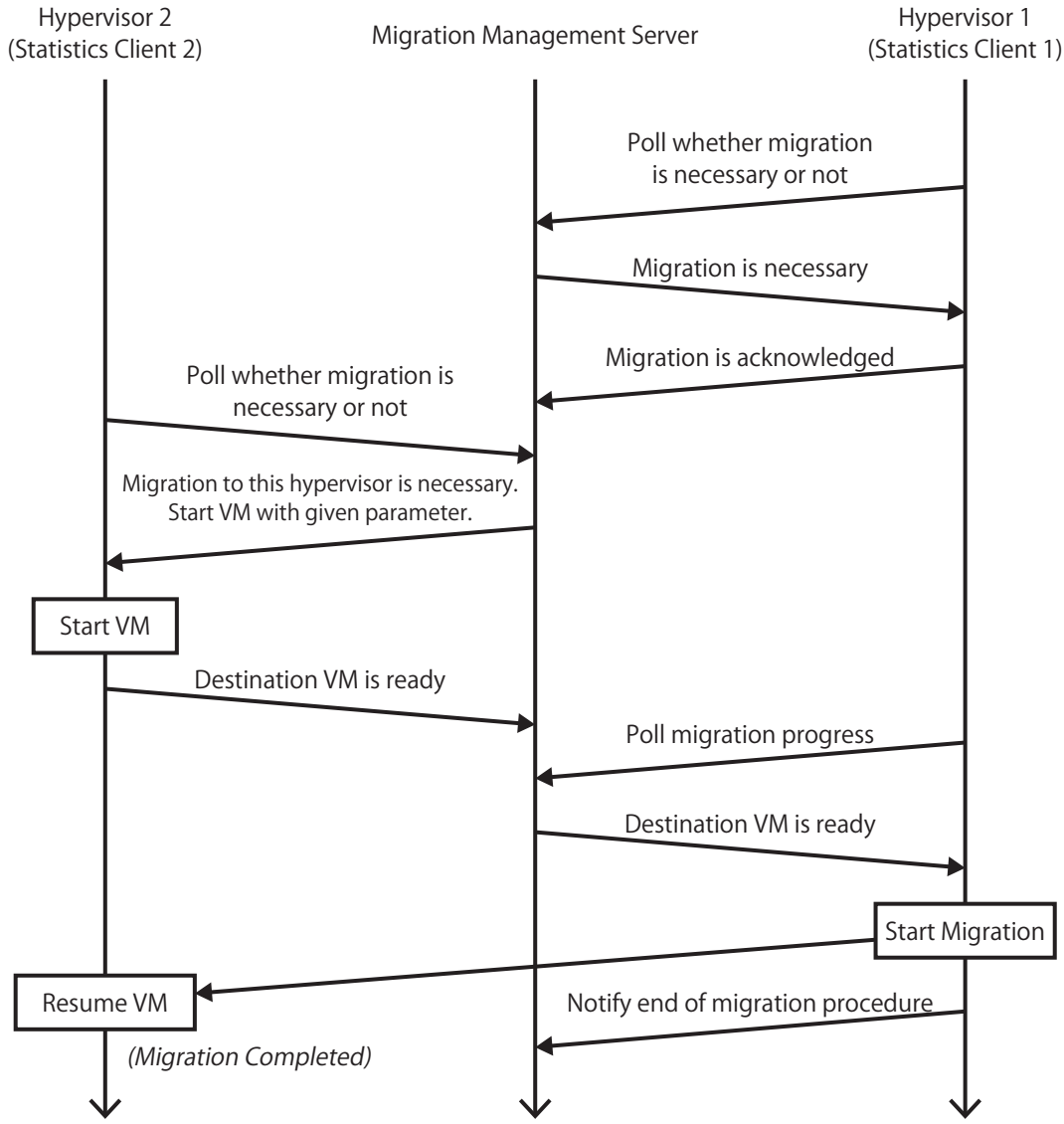

Figure 6.4: Migration procedure on the system

an equivalent virtual machine to the Migration Management Server; the parameters are passed as arguments to QEMU-KVM for specifying memory space allocated for the virtual machine, disk drive information, network and other peripheral information, etc. Until the parameter information becomes available on the Migration Management Server, the destination hypervisor will wait, and continue to update host statistics and poll for status updates (migration sequence shown in Figure 6.4; status update is shown in Figure 6.2).

Then, when the parameter information becomes available on the server, the destination hypervisor receives the parameters, and creates a receiver virtual machine. Once the receiver virtual machine becomes available, the destination hypervisor reports to Migration Management Server that it is ready to accept the migration. When the

source hypervisor polls the server in the following sequence, the source hypervisor will be notified that migration is ready, and the migration is executed.

7. Evaluation

To evaluate the effects of migration determination and appropriateness of hypervisor selections, several tests were conducted. The tests incorporated several applications with several different sets of thresholds and conditions, which are used for determining a suitable condition for executing live migrations.

7.1 Scenario and the Goal

As explained in Section 5.1.1, the purpose of proposing the architecture is to select a hypervisor that fulfills the resource needs for virtual machines running on the hypervisors. Numerous applications are running on a computer system utilizing various resources on it, but the computations can be roughly categorized into CPU-bound or I/O-bound. The goal is to move computations closer to data when I/O activities occupy a large portion of CPU time, and to move data closer to CPU when computation occupy a large portion of CPU time. By achieving this movement, the ultimate goal of executing the migration is to shorten the duration necessary for accomplishing the running processes.

7.1.1 Evaluation Scenario

This system may become useful in cases where more computational power becomes necessary. For instance, assume that a user is running a virtual machine on his or her laptop computer. When the user starts a CPU-bound application on the laptop, the virtual machine migrates to a hypervisor with higher-performance CPU. And, when the user is leaving and needs to take the laptop with him or her, the virtual machine has to be brought back to the source hypervisor in order for the user to carry it around again.

In this scenario, the laptop executes the source hypervisor. The laptop can be carried around, and a user may use the laptop outside of his or her home or office. Thus, the laptop must contain both user data and the OS environment; otherwise, the laptop won't be usable if access to either data or OS environment is not available. To be able to run the virtual machines without an Internet connection, the disk image of the virtual machine must reside on the laptop's hard drive. When the user arrives

at his or her home or office, where a hypervisor with better computational power can be accessed over a network connection, the virtual machine may migrate to another hypervisor. The hypervisor will be chosen because it has a faster CPU or more memory, but because the user's data remains on the laptop, disk access is done over the network, probably lowering disk I/O performance.

7.1.2 Reasons Not to Make Immediate Migration

It may seem that when a computer with better computational power is available, immediately executing migration would improve computation speed. However, there are several reasons for not immediately migrating:

- **Migration may not be necessary depending on the computation duration:** Migration takes from a few seconds up to a couple of minutes until the whole procedure completes. Even though the virtual machine continues to run during the migration procedures, there will be a massive access to allocated memory space, and produces a massive traffic on network for memory synchronization. Thus, there will be side-effects on, at least, memory and network accesses. If the computation does not require a fair amount of resources, the migration could even become a waste, or net loss of time and the virtual machine performance.

- **Migration may result in reduction of performance if resource requirements weren't selected appropriately:** Some computational loads spend more time on I/O wait than actual computation. In that case, reducing I/O wait time would improve performance.

- **In this scenario, laptops may be carried around:** If the virtual machine is hosted on the hypervisors in a data center, the hypervisors are always connected to networks, and the virtual machine can freely migrate around. However, in this scenario, the laptop, which is the hypervisor that carries a virtual disk image, may be disconnected from the network, either manually or accidentally. Thus, when migration to other hypervisors is not necessary, it would be better to keep the virtual machine on the source laptop to reduce the accidental termination of virtual machine or the burden of bringing the virtual machine back before removing the laptop from the hypervisor network.

Migration needs to be determined by examining the resource utilization and availabilities. This scenario is used to evaluate the developed system.

Table 7.1: Computers used for evaluation

	Laptop Hypervisor	Desktop Hypervisor 1	Virtual Machine
CPU Model	Intel Core 2 Duo T7200 2.0GHz	Intel Core i5 660 3.33GHz	QEMU-KVM Virtual CPU
CPU Cores	2 cores	2 cores, 2 threads	1 core assigned
Chipset	Mobile Intel 945GM Express	Intel P55 Express	Intel 440FX Emulated
Memory	3GB DDR2	4GB DDR3	1GB assigned
HDD	HTS723212L9A360 7200rpm SATA-II	HDT725032VLA360 7200rpm SATA-II	RAW Format QEMU Disk Image
Network	Intel 82573L Onboard Gigabit	Marvell 88E8053 PCI-Express Gigabit	Bridged Interface
OS	Debian GNU/Linux 6.0 squeeze x86_64		
Hypervisor	QEMU-KVM 0.12.5 (from APT Package)		-

7.2 Evaluation Environment

Having the scenario and with the goal in mind, the evaluation environment and test sets were selected.

7.2.1 Evaluation Hardware and Software

In the evaluation, a laptop hypervisor and a desktop hypervisor are used to run a single virtual machine. The hardware used for the evaluation is listed in Table 7.1. As it can be informed from the table, laptop hypervisor has relatively limited hardware resources than the desktop hypervisor. The table indicates that this desktop hypervisor is number "1"; desktop hypervisor 2 will appear later in this chapter.

Both hypervisors run on CPUs having Intel VT-x virtualization support. The desktop hypervisor also has Intel VT-d I/O virtualization support, while the laptop hypervisor doesn't. The desktop CPU has Intel SpeedStep technology, which can change the CPU clock dynamically according to the load on the host. However, it is disabled as CPU clocks are reported lower when the machine is at idle state, which leads to an error when selecting a hypervisor based on CPU clocks.

The relationships between entities are shown in Figure 7.1. Migration Management Server is executed on the laptop, which will be the "home hypervisor" for the virtual machine. Both the laptop and desktop hypervisors will be running Statistics Clients, and so as in the virtual machine.

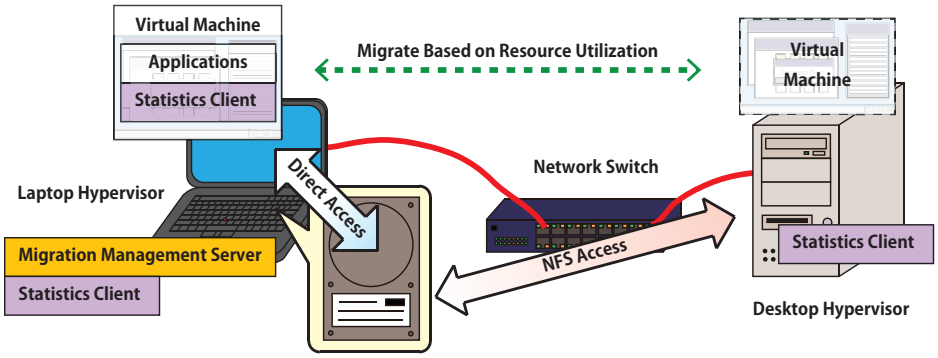

Figure 7.1: Relationships between laptop and desktop hypervisors

7.2.2 Test Set and Baseline Performance

To test the effectiveness of the system, several test sets were conducted. Some of the tests involve CPU-intense computations while the others involve more disk I/Os. For disk I/Os, there are tests with reading and writing relatively small files continuously, and reading and writing a single large file. Some of the tests involve both CPU and I/O-intense operations, which needs to be balanced out upon determining load and selecting a hypervisor by the system.

The test sets used for measuring performance on the laptop hypervisor and the desktop hypervisor are as follows:

- **Video encoding:** This test involves encoding of a video file using `ffmpeg` application. A five-minute DV format video created from the `kino` video editing application was converted to MPEG video. Its file size was 1,080,288,000 bytes (approximately 1.1GB). The encoder reads several frames of video and audio from the source file, encodes it, and writes out to the disk. Thus, a greater computation with some non-continuous disk I/Os are expected from the test. Later, a ten-minute video is also added for extending the test duration.

- **Copying a single large file:** This test involves duplication of a single large file on the same disk. The file used was a DVD ISO image for Fedora 14 Linux distribution, `Fedora-14-x86_64-DVD.iso` with file size of 3,520,802,816 bytes (approximately 3.3GB). This is a simple copy of a file within the same virtual disk space. Thus, more of the disk I/O is expected than the CPU computation power.

- **Copying smaller files continuously:** This test involves duplication of multiple files sequentially. The DVD ISO image for Fedora 14 Linux distribution, which resides on the same virtual disk as the copy destination, is `mount`ed with `-o loop` option. Then, the files under the mounted directory were copied to another

```
Command being timed:  "ffmpeg -i testvideo-5min.dv -v 4 -b 4096k -f
mpegvideo /tmp/testvideo-20110111001558.mpg"
User time (seconds):  49.07
System time (seconds):  2.51
Percent of CPU this job got:  86%
Elapsed (wall clock) time (h:mm:ss or m:ss):  0:59.76
Average shared text size (kbytes):  0
Average unshared data size (kbytes):  0
Average stack size (kbytes):  0
Average total size (kbytes):  0
Maximum resident set size (kbytes):  43456
Average resident set size (kbytes):  0
Major (requiring I/O) page faults:  35
Minor (reclaiming a frame) page faults:  4636
Voluntary context switches:  292
Involuntary context switches:  1343
Swaps:  0
File system inputs:  2113712
File system outputs:  306304
Socket messages sent:  0
Socket messages received:  0
Signals delivered:  0
Page size (bytes):  4096
Exit status:  0
```

Figure 7.2: Sample output of `time` command

directory with `-rp` options, which recursively copies files under the same directory tree while preserving file information. This test also involves disk I/Os, and it is expected to produce more CPU I/O wait time as the numerous files are copied sequentially from one place to another within the same virtual disk.

The performance was measured by the wall-clock duration for achieving the given task. The duration was measured using the `time` command (Figure 7.2).

To have a rough idea of how the test sets were performed on the virtual machine, the test sets were first tested on the virtual machine without involving live migration. The same test was conducted three times to avoid errors. The results are shown in Table 7.2.

Desktop hypervisor performed better on CPU-bound video encoding, even if all ac-

Table 7.2: Output of `time` command without migration

Statistics	**Desktop (NFS Disk)**			**Laptop (Local Disk)**		
	1	**2**	**3**	**1**	**2**	**3**
Encoding a Five-Minute Video						
User time	49.07	46.51	45.63	96.95	174.81	122.43
System time	2.51	1.95	1.83	4.95	7.28	5.92
% of CPU used by this job	86	87	94	96	98	97
Time (seconds)	59.76	55.24	50.49	105.67	185	131.85
Copying a 3.3GB ISO File						
User time	0.06	0.03	0.06	0.11	0.14	0.11
System time	7.90	7.80	7.82	22.33	19.63	19.05
% of CPU used by this job	3	3	3	10	11	10
Time (seconds)	208.40	209.53	215.92	206.32	237.99	234.78
Copying Contents of a ISO File						
User time	0.06	0.04	0.09	0.22	0.18	0.22
System time	7.66	7.56	7.56	21.62	21.68	21.68
% of CPU used by this job	3	3	3	12	12	12
Time (seconds)	222.28	223.69	221.45	181.71	179.55	180.87

cesses to the virtual disk were made through NFS protocol. The desktop hypervisor had 3.33GHz CPU while the laptop hypervisor had only 2.00GHz. Although the numbers are compared among only two hypervisors, and even though the CPU architecture of two hypervisors are at different CPU generations, the change in CPU clock speed has increased in the similar ratio to the change duration for encoding videos.

On the other hand, two disk access tests had different results on the desktop hypervisor and the laptop hypervisor. The virtual disk image was stored on the laptop's local hard disk drive, and desktop hypervisor was accessing the disk image over NFS connection while the laptop hypervisor was accessing it directly as a file on the local file system. From the configuration, the laptop hypervisor would seem to perform better on disk access tests as there is no overhead of transporting disk I/O over IP networks. However, the test has reported that the sequential access to multiple files performs better on laptop hypervisor with direct access to the disk file, while the desktop hypervisor that accessed the disk image over NFS connection performed better on copying a large chunk of file.

This data is used as a baseline for making comparison with the initial tests conducted in the next section, Section 7.3.

Table 7.3: Initial experiments with laptop and desktop hypervisors

Test	Run	Migration Duration (sec)	Network Bandwidth (Mbps)	Duration of Test (sec)	Performance Improvement (%)
ffmpeg (5 min)	**1**	38.084561	937	109.29	**17.11**
	2	35.312317	938	128.60	**2.46**
	3	38.012777	937	119.31	**9.51**
Copy ISO File within the Disk	**1**	84.715026	930	221.90	-5.90
	2	83.211531	938	251.77	-20.16
	3	138.726779	938	256.69	-22.51
Copy Contents of ISO File	**1**	107.713945	926	356.38	-60.33
	2	239.648646	939	281.21	-26.51
	3	181.278174	939	293.53	-32.05

7.3 Initial Triggered Migration Tests

The test was conducted with the designed system. The migration duration was also measured during the test using the `Wireshark` packet capturing utility. It was executed on the source hypervisor, and when traffic passes through the port specified for performing migration, the migration is in progress. The conditions for determining live migration execution were as described in Section 6.4.

Upon running the tests, virtual machines are first started on the laptop hypervisor, which is resource-poor. That is, for testing video encoding, it is expected to consume more CPU computational power than the disk I/Os. Thus, the virtual machine is started on a hypervisor with the weaker CPU (laptop hypervisor in this scenario), and later migrates to a hypervisor with a better CPU, even if that ends up in slowing down the disk I/Os. When running the disk I/O-related tests, the same thing applies; the virtual machine starts on a hypervisor accessing the virtual disk over NFS connection, and later migrates to a hypervisor that has the virtual machine's disk image. This limitation is set to make sure that the virtual machine will migrate during the test; if the virtual machine doesn't migrate, the test is not a success. Test results are shown in Table 7.3.

The table has four columns of data:

- **Migration Duration:** duration of migration from the start of memory synchronization to the end of the state transfer in seconds. This was measured by monitoring the TCP/IP port specified for using with QEMU-KVM live migration.
- **Network Bandwidth:** network bandwidth between the laptop and desktop hypervisors measured with `iperf` tool. This is here to show the network bandwidth

that was available for accessing NFS disks and executing live migrations.

- **Duration of Test:** shows duration in seconds till the specified application completed in seconds. The duration was measured using `time` command. The purpose of executing the live migration is to shorten this duration, compared to running them without executing live migration (Table 7.2).
- **Change on Performance:** The duration of test, divided by the median of the test without live migration (Table 7.2).

The same test was conducted three times.

The table, compared with Table 7.2, gives the following information:

- **In this particular case, CPU-bound jobs will perform better when migrated to a hypervisor with better CPU during the execution of the job.** Although migration took twice as long as it was estimated, the overall performance has improved as the video can be encoded faster on the better CPU.
- **Performance of the I/O-bound jobs will not improve even when migration was executed during the execution of the job. In this particular case, performance has even declined when migration was executed.** When disk I/Os were made on the virtual machines, the migration didn't finish for a significant duration of time. Thus, the migration didn't finish before it reaches the point where performance will be improved by executing the migration.

There are several possible reasons for the results, as I/O of the virtual machines are affected by numerous factors. One of the primal possibilities is the network I/O, which was used for both live migration and NFS access. In an enterprise network, often these networks are designed separately. However, this experiment scenario involved a laptop computer, which typically has a single Ethernet interface. Therefore, both NFS accesses and memory synchronization for live migration were passing through the same network interface. This may have saturated the network bandwidth, and additional tests are necessary to determine if the migration for I/O-bound jobs will not be accomplished or not.

The other possible reason is the side-effects of tracking statistics using external tools. For example, `Wireshark`, which was used for monitoring migration port usage, have consumed a significant quanta of the hypervisor's memory space. To prove the effects of the external applications, additional tests must be conducted with fewer external applications for taking statistics.

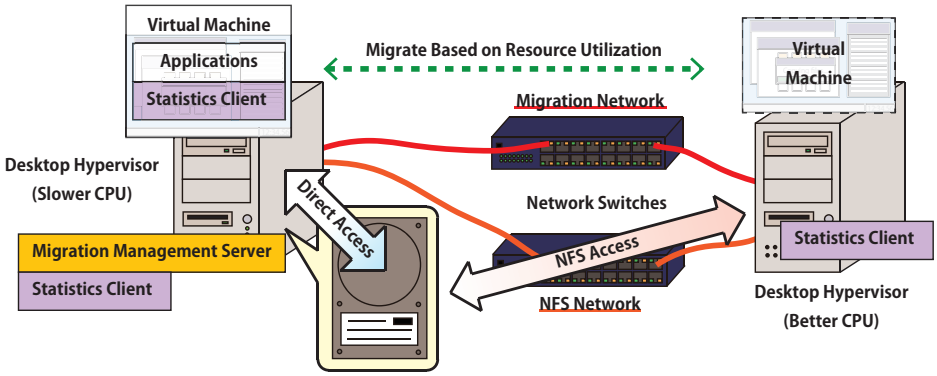

Figure 7.3: Evaluation hardware with separate networks for NFS and live migration

Table 7.4: Additional hypervisor hardware

	Desktop Hypervisor 2
CPU Model	Intel Core 2 Duo E6750 2.66GHz, 2 cores
Chipset	Intel Q45 Express (with Intel VT-x/d support)
Memory	4GB DDR2
HDD	HDT725032VLA360 7200rpm SATA-II
Network	Intel 82567LM-3 Onboard Gigabit Network Connection (Migration) Intel 82574L PCI-Express Gigabit Network Connection (NFS)
OS	Debian GNU/Linux 6.0 squeeze x86_64
Hypervisor	QEMU-KVM 0.12.5 (from APT Package)

7.4 Testing with Separate Networks for NFS and Migration

Section 7.3 gave a basis for discussing execution of live migration while the processes are running inside of the virtual machines. However, because there was only a single network, the bottleneck was unclear, even though the scenario involved a laptop that typically has a single network interface.

To more closely observe the performance of I/O-bound processes, another hypervisor is introduced. The relationship of the entities is shown in Figure 7.3, and the specifications of the added hypervisor hardware is shown in Table 7.4. The newly added hypervisor is substituting for the laptop hypervisor; this hypervisor will have the virtual disk image for the virtual machine on the local file system, and the other desktop hypervisor accesses the virtual disk image through NFS connection.

First, similar to the prior test, the performance was measured without involving live migration to obtain a baseline for making performance comparisons. For the rest

Table 7.5: Output of `time` command without migration

Statistics	Desktop 1 (NFS Disk)			Desktop 2 (Local Disk)		
	1	**2**	**3**	**1**	**2**	**3**
Encoding a Ten-Minute Video						
User time	175.55	176.23	176.42	230.06	230.07	229.99
System time	4.61	4.42	4.33	6.46	5.98	5.92
% of CPU used by this job	95	97	96	98	97	98
Time (seconds)	188.59	186.11	186.86	241.1	241.17	239.75
Copying a 3.3GB ISO File						
User time	0.06	0.10	0.04	0.02	0.04	0.04
System time	8.26	8.09	8.12	8.81	7.98	8.14
% of CPU used by this job	5	5	5	7	6	7
Time (seconds)	159.20	162.10	159.91	115.49	127.98	116.88
Copying Contents of a ISO File						
User time	8.26	7.97	8.19	9.05	8.78	8.79
System time	0.12	0.08	0.12	0.07	0.05	0.08
% of CPU used by this job	4	4	4	7	7	7
Time (seconds)	168.58	166.56	166.78	120.78	120.72	119.13

of this chapter, the desktop hypervisor that was also used in the initial test will be referred as "Desktop 1"; the desktop hypervisor added in this section will be referred as "Desktop 2". The results of the tests are shown in Table 7.5. The test was conducted on hypervisor desktop 1 again as the NFS server holding the virtual disk image has changed since the previous sections.

Then, the test was conducted on the new environment with the same test set and threshold as the initial test. This time, a ten-minute video file was used for `ffmpeg` test because encoding of a five-minute video completed within a minute or so; since migration while encoding a video file took around 40 seconds in the initial test, an encoding duration of a minute can be considered as too short.

This time, the migration duration was measured with the `ifstat` command instead of the Wireshark program as the Wireshark may cause a large impact on the hypervisor's memory usage. The ifstat command monitors the bandwidth consumed by each network interface on the system. Since there is one network dedicated for executing live migration and another for accessing NFS storage, monitoring the bandwidth consumed by each interface will give an idea of when the NFS traffic or migration traffic occurred.

Table 7.6: Encoding ten-minute DV file with ffmpeg with migration between two desktop hypervisors

Run	Time Elapsed Since the Beginning (seconds)						App. Duration (sec)	Performance Improvement (%)
	NFS			Migration				
	Start	End	Elapsed	Start	End	Elapsed		
1	106	219	113	67	106	39	216.89	**10.42**
2	90	220	130	54	90	36	217.98	**9.97**
3	94	221	127	57	94	37	217.53	**10.16**
4	96	221	125	58	96	38	214.56	**11.38**
5	97	221	124	60	97	37	218.06	**9.94**
6	98	223	125	61	98	37	219.87	**9.19**
7	93	219	126	53	93	40	216.31	**10.66**
8	99	220	121	60	99	39	217.32	**10.24**
9	99	221	122	60	99	39	217.74	**10.07**
10	99	221	122	61	99	38	216.90	**10.42**

7.4.1 Video Encoding

First, the test result for video encoding with ffmpeg software is shown in Table 7.6. This table has a different format compared to Table 7.3 in the initial test. There are eight values in the table: test count, start and end of accesses to NFS network, start and end of accesses to migration network, duration of each network access, duration of the process, and performance improvement. The start and end times in the table are when the accesses to the corresponding networks started and ended, relative to the start of the application. The performance improvement is calculated by comparing to the median of the test without executing migration (Table 7.5).

The table shows that the migration improves performance of CPU-bound processes. Each run of the experiment showed similar behavior, which is to start memory synchronization for migration approximately 60 seconds since beginning of the workload, and to take approximately 30 to 40 seconds for live migration to complete. The median of workload duration is shown in Figure 7.4.

In this test scenario, the performance had improved for approximately 10 percent while the difference of CPU clocks between two hypervisors were approximately 20 percent. In the test, the computation has lasted for 210 to 220 seconds, and the live migration has completed about 100 seconds since the computation has started, which is near half of the total workload duration. Thus, the first half of the computation took place on the first hypervisor, and then the latter half of the computation took place on the second hypervisor. If the migration of a CPU-bound job takes the same duration as this experiment, the workload will finish sooner when the live migration takes place

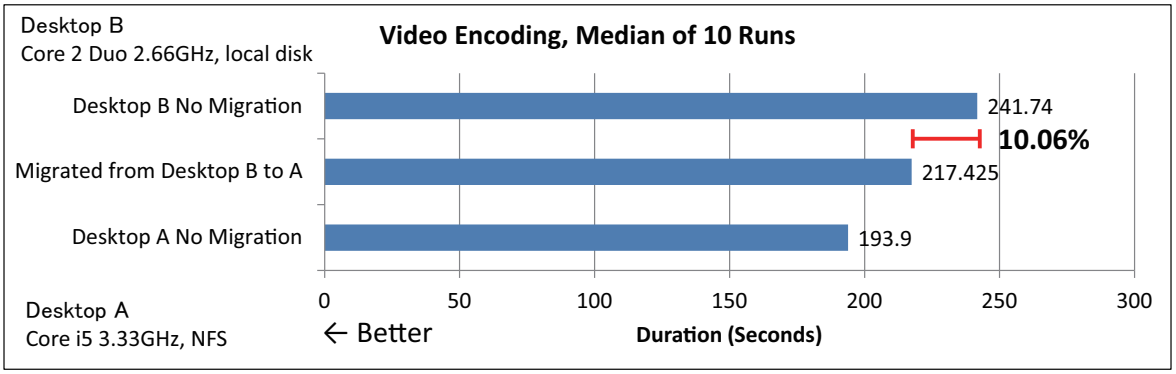

Figure 7.4: Median of workload duration for ffmpeg test

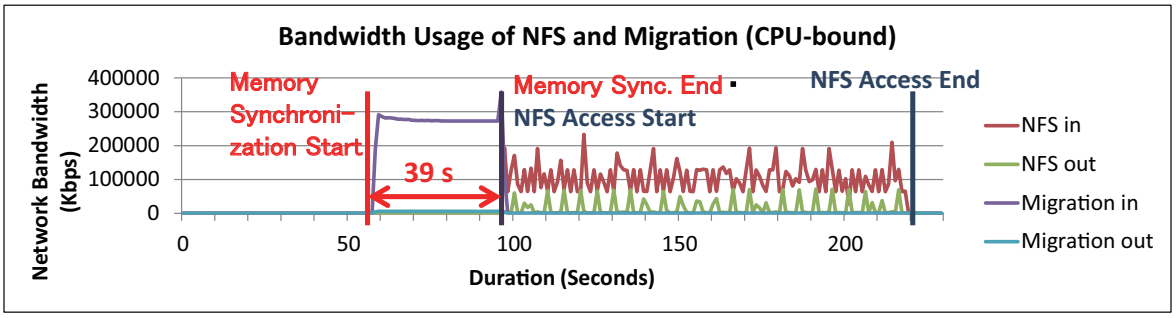

Figure 7.5: Median of network utilization for ffmpeg test

as soon as the workload has started. The performance improvement will depend on duration of the workload.

Figure 7.5 shows utilization of NFS and migration networks. The figure indicates that NFS accesses start right after the live migration completes; this is where disk I/Os have changed from local access to network access.

7.4.2 Copying a Single Large File

The next test was conducted to observe disk I/O performance. In the test with a laptop hypervisor, live migration did not finish when a disk I/O-intensive process was running on the virtual machine; now each hypervisor has two network interfaces, to avoid filling available network capacity with combination of NFS accesses and memory synchronization traffic.

The result is shown in Table 7.7. From the table, it can be inferred that the live migration didn't complete even when networks for NFS accesses and migration were separated. The performance was close to without performing live migration, or decreased for a little, as shown in Figure 7.6.

Table 7.7: Copying a single large file with migration between two desktop hypervisors

Run	Time Since the Beginning (seconds)						Diff. End Mig. & End NFS (sec)	App. Duration (s)	Performance Improvement (%)
	NFS			Migration					
	Start	End	Elps.	Start	End	Elps.			
1	0	167	167	52	180	128	13	164.36	*-1.14*
2	0	147	147	53	158	105	11	142.03	**12.60**
3	0	169	169	55	182	127	13	165.19	*-1.65*
4	0	167	167	57	180	123	13	163.74	*-0.76*
5	0	167	167	55	180	125	13	163.55	*-0.64*
6	0	167	167	59	180	121	13	163.62	*-0.68*
7	0	167	167	61	179	118	12	164.05	*-0.95*
8	0	168	168	64	180	116	12	164.82	*-1.42*
9	0	167	167	69	179	110	12	164.06	*-0.95*
10	0	167	167	48	179	131	12	164.55	*-1.26*

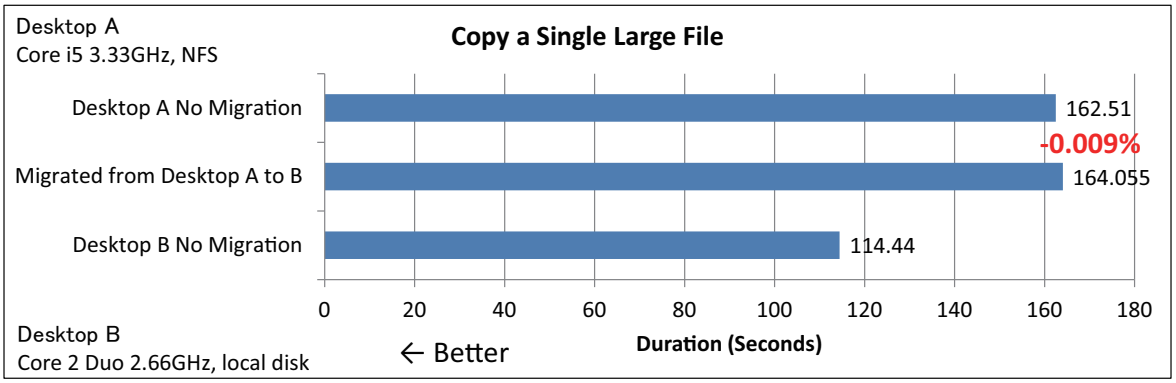

Figure 7.6: Performance comparison when copying a single large file, median of ten trials

Network traffic while running the test is shown in Figure 7.7. The network traffic had the same trend as running the test on a laptop hypervisor: memory synchronization iterates while NFS network has traffic, and the iteration finishes after 11 to 13 seconds since the end of NFS traffic.

7.4.3 Copying Multiple Files

Copying a single large file produced the same result whether running the test with a single network interface or with two. The next test involves duplication of multiple files. The result is shown in Table 7.8.

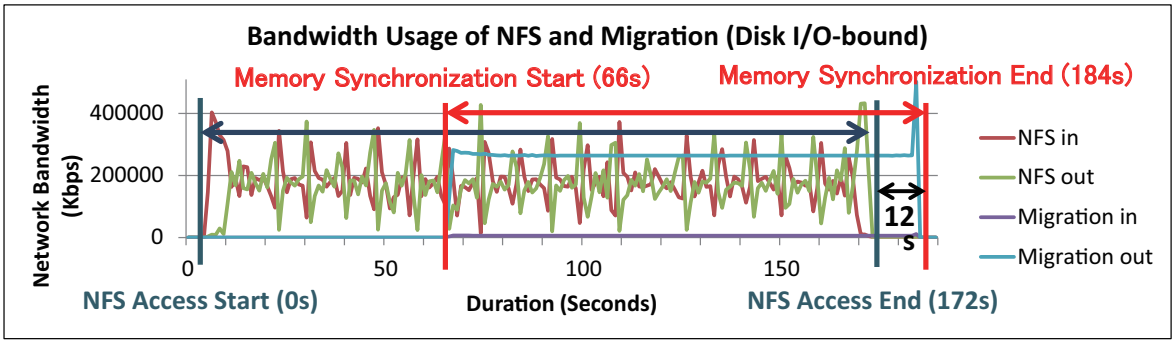

Figure 7.7: Network utilization when copying a single large file, median of ten trials

Table 7.8: Copying multiple files with migration between two desktop hypervisors

Run	Time Since the Beginning (seconds)						Diff. End Mig. & End NFS (sec)	App. Durat-ion (s)	Performance Improve-ment (%)
	NFS			Migration					
	Start	End	Elps.	Start	End	Elps.			
1	3	180	177	56	195	139	15	172.47	*-3.30*
2	4	182	178	67	195	128	13	175.24	*-4.96*
3	4	180	176	59	195	136	15	173.60	*-3.98*
4	4	181	177	59	196	137	15	174.86	*-4.73*
5	3	192	189	56	207	151	15	187.03	*-12.02*
6	3	181	178	55	195	140	14	174.14	*-4.30*
7	4	180	176	65	194	129	14	173.68	*-4.03*
8	4	185	181	58	196	138	11	175.67	*-5.22*
9	4	183	179	55	195	140	12	174.28	*-4.39*
10	5	184	179	58	197	139	13	174.92	*-4.77*

In the median of ten trials, performance for copying multiple files has declined by 4.56 percent; the result indicates that the performance will decline when live migration was attempted while copying multiple smaller files. The performance comparison is shown in Figure 7.8.

7.4.4 Examining the Results

The result indicated that executing live migration will improve performance of video encoding, but not on file copy tests, which was similar on the test conducted with the laptop hypervisor. On file copy tests, migration traffic ended eleven to fifteen seconds after NFS traffic ended. The throughput for the migration network, measured by the `iperf` utility, was approximately 700Mbps. From the Equation 5.1 introduced

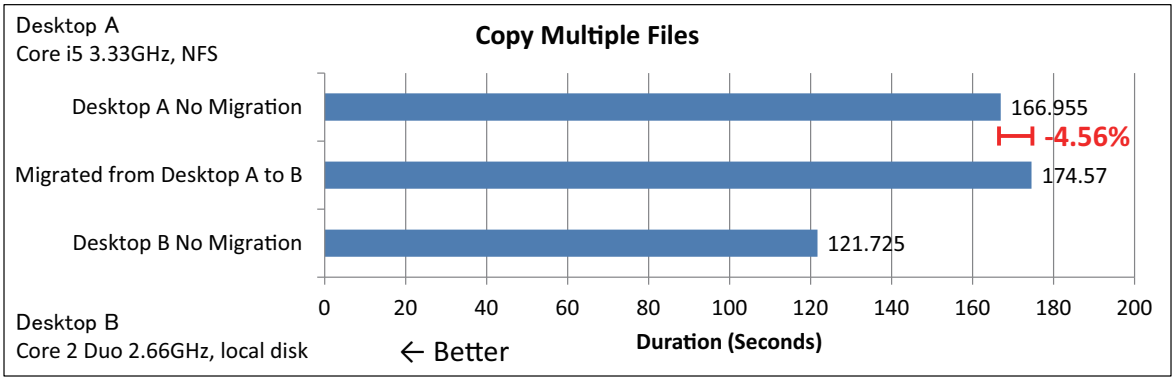

Figure 7.8: Performance difference with migration between two desktop hypervisors, while copying multiple files

in Section 5.2.2, it can be estimated that a virtual machine with 1GB of memory allocated at an idle state will take approximately 11 to 12 seconds, which is similar to the time difference between the end of NFS traffic and migration traffic. From the network traffic, it can be estimated that the last set of migration took place after the file operation has ended.

To investigate why the migration didn't finish during the file copy test, dirty pages on memory was examined. On a Linux system, information related to memory can be found in `/proc/meminfo`. In this file, a row "Dirty" indicates the memory space that may need to be written to disk or swap space. Documentation for RedHat Linux[53] gives a log file as an example, but files being written are first stored in cache, and written out to the disk when it gets to a certain size. This documentation was written for Linux kernel 2.4, but the idea remains the same on kernel 2.6.

When the row was `grep`ped every second while video encoding was in progress, dirty pages changed as shown in Figure 7.9; and they have changed as shown in Figure 7.10 during file copy. During `ffmpeg`, changes were ranged from several kilobytes to several megabytes per second. On the other hand, during file copy operations, sometimes changes were as big as ten to twenty megabytes in one second, or more. During the file copy, a massive area of the memory space may be predicted as being changed by file copy operations.

As explained in Section 4.2.2, live migration of a virtual machine requires synchronization of memory space on source and destination hypervisors. If massive changes are made to the memory space while executing a live migration, the migration procedure may not reach the point where it can proceed to the final step of transferring the machine state from the source hypervisor to the destination hypervisor. This may be the reason why migration did not complete during file copy operations.

Since the migration was done from a hypervisor accessing the virtual disk image over NFS connection on the destination hypervisor, the bandwidth of the NFS network

```
Dirty:               388 kB
Dirty:                24 kB
Dirty:               916 kB
Dirty:              2048 kB
Dirty:              3320 kB
Dirty:              4664 kB
Dirty:              5980 kB
...
```

Figure 7.9: Dirty row from `/proc/meminfo` file while running `ffmpeg`, showing the amount of memory modified per second

```
Dirty:            119548 kB
Dirty:            135452 kB
Dirty:            143628 kB
Dirty:            145432 kB
Dirty:            104940 kB
Dirty:            121096 kB
Dirty:            112904 kB
...
```

Figure 7.10: Dirty row from `/proc/meminfo` file while copying an ISO image

is also visible in the output. Figure 7.7 indicated that the network bandwidth for migration network has been constant since the beginning of the migration until the end (the graph ends at the point where migration was actually executed). This network bandwidth consumption was the result of iteration, which didn't end until the I/O-bound job running on the virtual machine was accomplished.

This is due to the behavior of QEMU, the emulation software running on top of KVM for utilizing virtualization technology. On November 20, 2010, there was a discussion on QEMU developer's mailing list on whether to add the limits for iterating memory space while executing migration[54]. According to the discussion, Xen hypervisor has a limit of three times the size of the guest OS's memory of iteration before migration executes. At the time this thesis was written, QEMU doesn't have this behavior, continuing to iterate memory space until the time remaining to complete the state transfer is estimated to fall below the threshold.

In the evaluation, the threshold of I/O wait was set to 10% for 20 seconds, which is

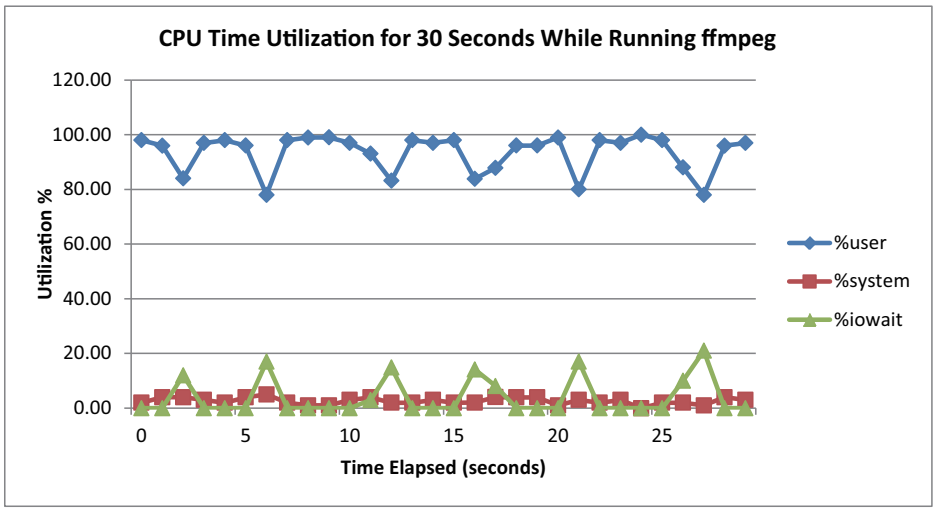

Figure 7.11: CPU utilization of virtual machine for 30 seconds while running `ffmpeg`

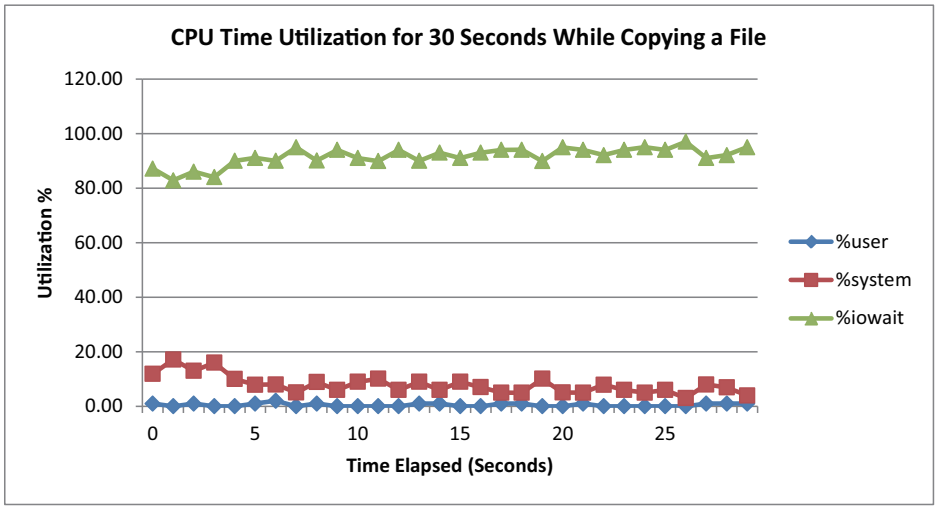

Figure 7.12: CPU utilization of virtual machine for 30 seconds while copying a file

not as great as the threshold of CPU computation time. The CPU time utilization for `%user`, `%system`, and `%iowait` were recorded using the `sar` command with `-u` option. 30 seconds were sampled from the output, and graphed in Figures 7.11 and 7.12. On `ffmpeg` test, CPU utilization was high, and some I/O waits occurred in interval of three to five seconds. On the file copy test, `%iowait` was high for the whole 30-second range of sampling. The interval on video encoding could be explained as: encoder reading some portion of source file, and writing out to disk after a certain chunk of video and audio was encoded. The threshold proposed in the thesis used both utilization and duration, both of which has turned out to be significant on determining the virtual machine state from the evaluation.

Table 7.9: Building an application with migration between two desktop hypervisors

Run	Time Elapsed Since the Beginning (seconds)						App. Duration (sec)	Performance Improvement (%)
	NFS			Migration				
	Start	End	Elapsed	Start	End	Elapsed		
1	167	233	66	116	167	51	231.53	**6.50**
2	153	231	78	109	150	41	226.69	**8.46**
3	146	228	82	107	144	37	225.4	**8.98**
4	150	232	82	112	150	38	227.18	**8.26**
5	169	240	71	110	169	59	233.78	**5.59**
6	153	230	77	111	150	39	226.21	**8.65**
7	169	236	67	131	166	35	232.07	**6.28**
8	172	238	66	129	168	39	230.78	**6.80**
9	169	235	66	135	168	33	230.79	**6.80**
10	234	254	20	195	233	38	249.81	*-0.88*

7.4.5 Combination of Computation and File Accesses

The previous tests has indicated that migrating a virtual machine while CPU-bound jobs are running would improve performance, and performing the migration while disk I/O-bound jobs are running would not. The experiments were strictly bound to either CPU or disk I/O. However, in the real world, there are applications that access the disk quite frequently while utilizing CPU at the same time. To illustrate an example, additional tests were conducted by building an application from its source code.

The sample application used for this experiment is ffmpeg, which is the video encoder used for the CPU-bound test. The source code for ffmpeg version 0.6.1[55] was compiled with the `make` command. C source code files are fetched from the disk, compiled with `gcc`, and output files are written to the disk. Then, when all the required source codes are compiled, files are linked to produce binary files. Thus, building an application from the source code is expected to cause both CPU and disk I/O utilization.

The test result is shown in Table 7.9. Although disk accesses have occurred during the workload, the migration has completed successfully in this case. However, the performance improvement was not as great as it was on video encoding, with improvement of 6.8 percent in median of ten trials (Figure 7.13). In fact, there was even a case where the performance has declined.

The difference between video encoding and application building is the type of CPU utilization. Figure 7.14 shows CPU utilization of video encoding and application building. In the graph, there are two lines for each workload: CPU utilization of user space (%user), and CPU utilization of kernel space (%system). The figure indicates that %user is constantly high on video encoding while %system sometimes becomes high on

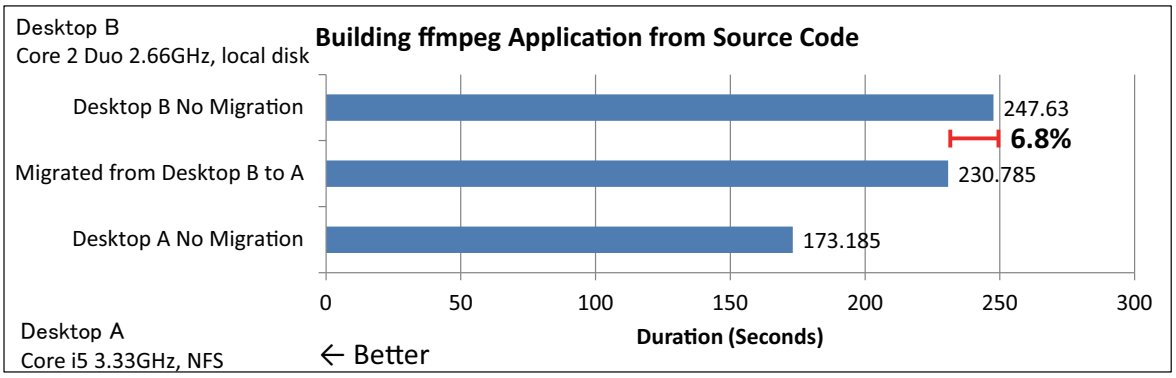

Figure 7.13: Performance improvement with migration when running application build test

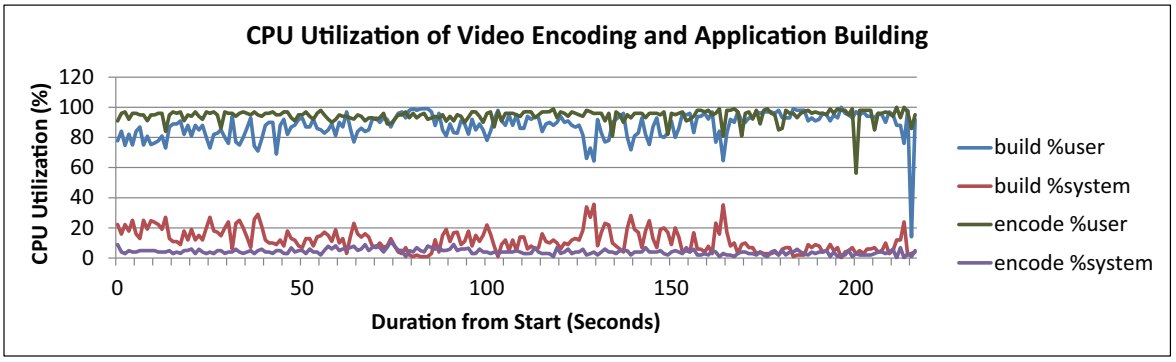

Figure 7.14: CPU utilization of virtual machines while encoding a video file and building an application

application building. The CPU utilization of %system for application building comes from disk I/Os; even though disk I/Os for application building were not great enough to affect the live migration, large disk I/Os occurred more frequent compared to the video encoding.

The system used for evaluation determines migration based on applications' CPU utilizations, which is %user in the figure. When %system, which is caused by disk I/Os, is great, the CPU utilization is not counted against application CPU utilization. In this case, because CPU utilization for the particular application is not massive enough, live migration will not occur; the computation is done on the relatively weak hypervisor. And, when the CPU utilization for user space became great for quite a duration of workload, finally the live migration was determined. This delay in determining live migration is likely to be a cause for declining performance. Even though this test did not, CPU utilization of the kernel space should also be incorporated for determining the live migration.

7.5 Migration Strategies from the Evaluation

From the evaluations, it can be inferred that:

- **It is effective to migrate CPU-bound jobs.** If a process seems to occupy a massive amount of CPU time, migrating to a hypervisor with better CPU is expected to improve performance.
- **Migration should not be executed when there is one or more I/O-bound jobs currently running in the virtual machine.** Because file I/O may change a big chunk of memory space allocated to the virtual machine, memory synchronization may repeat on the hypervisor until the I/O-bound job completes. Since the migration may even take place on idle state after the job is completed, the migration may not bring a positive outcome on resource utilization.
- **However, I/O performance itself will improve if the I/O-bound job is done locally close to data.** When file copies were tested in Table 7.5, file operations completed faster when the disk was accessed locally than accessing the disk over NFS connection. If migration was not executed **while** I/O-bound jobs are running, the virtual machine would be benefitted from migration.
- **And, when migrating a virtual machine with CPU-bound workload, the migration should be determined based on CPU utilizations of both user processes and the whole system.** Although the goal of the system is to shorten the workload duration, observing only the workload will not always successfully improve the performance; other activities on the system should also be taken into account.

Thus, a strategy for executing live migration based on resource utilization, which derived from the evaluation results, involves automatic migration based on CPU utilization, and avoiding migration when I/O-bound jobs are executed. In order to improve I/O performance, either: manual migration by user before running I/O-bound jobs, or detecting execution of I/O-bound applications inside the virtual machines and execute migration before the job starts would be a possible selection. Pausing the I/O-bound virtual machine allows iteration to complete, but the pause may affect other processes especially if they involve interactions with other entities, such as a file or other hosts on networks.

8. Conclusion

To conclude this thesis, this chapter summarizes the work that was done and the future work.

8.1 Summary

In this thesis, a computer architecture with dynamic adaptation to computer resource requirements was proposed. As new technologies, such as device and machine virtualization, are being used commonly in computers today, a new computer architecture based on the virtualization mechanisms has become possible.

8.1.1 Applications and Requirements

Applications running on computers have different requirements, depending on the tasks those applications are working on. Certain applications require CPU power and memory space, while other applications may take storage I/O time. As computers and applications have become virtualized and distributed, tasks that computers have to handle have become more complicated; these tasks cannot be divided simply into CPU-bound or I/O-bound processes, but have more complicated relationships.

8.1.2 Virtualization Architecture

Computer virtualization has gained attention since the era of IBM System/360. Computing resources were expensive at that time, and applications were more dependent on the computer hardware and the OS running on top of the hardware. Thus, virtualization has helped retaining compatibility of the software among different hardware.

Computing resource allocation over an IP network is another way to extend computing resources, in forms of applications or I/O transport over IP networks. Use of remote storage and remote controlling of a distant computer have been a major focus on this area. Even further, as computer networks gained more bandwidth and link quality, it has become common to transport device I/O over IP networks. Nowadays computer

resources can be accessed over device-level encapsulation in addition to application-level accesses, allowing transparent use of remote devices on operating systems and applications just as similar as locally available devices and applications.

In computer virtualization, there is an advantage which allows computers to migrate over IP networks, even though there are several requirements and restrictions. Hypervisors supporting virtual machines can absorb the differences of devices, which abstract different models of a same type of device into a single virtual device to virtual machines.

8.1.3 Computing Architecture Based on Virtualization

With the combination of virtualization and device I/O extension over IP networks, a new architecture for migrating virtual machines to different hypervisors is proposed. Virtual machines have advantages over ordinary computers as virtual machines can: abstract different models of devices into a single virtual device, bundle user data and OS environment into a single virtual machine, and migrate from a hypervisor to another without terminating the virtual machine. Even when a virtual machine migrates, applications and network communications running within the virtual machine continue to run. As network communications within the virtual machine continues even after migration, devices that are connected over IP networks can retain their communication sessions when migration occurs on the host virtual machine.

In this architecture, a virtual machine can migrate depending on the resource utilization of applications running inside of the virtual machine. Nowadays, computers no longer reside within a single "box". Computers are capable of migrating around different hypervisors, and computers can request additional resources when necessary. This will allow any types of users, not only the enterprise users but including the general consumers, to gain computing capability upon their requirements. As computers are widely used and distributed in the society, the system should be able to handle the requirements users expect on the computers.

8.2 Accomplishments of This Work

In this work, a new architecture of computer migration was proposed. The architecture was designed to allow virtual machines to migrate by observing the resource requirements of running applications. The migration of virtual machines based on application resource requirements is expected to improve computer performance with minimum duration of time and network bandwidth consumed for migration.

The goal of this architecture is to run virtual machines on hypervisors with suitable resources. When computational power is necessary, a virtual machine should migrate

to a hypervisor with better CPU and memory allocation; when disk I/O is critical, a virtual machine should migrate to the hypervisor that can access disk images with better access speed. In theory, since many applications which users execute on the computers require a certain amount of computational power with a certain amount of disk I/O, the requirement cannot be determined simply by looking at a single parameter. Thresholds for different resource utilizations were set, and when the resource utilization above the threshold continued for a certain period of time, migration was triggered.

When executing virtual machine migrations, not only the resource utilization but the cost for performing migration must be considered. Experiments showed that CPU-bound jobs perform better if the virtual machine was migrated to a hypervisor with better CPU. The migration duration may become longer than executing the migration at idle state, but the migration will accomplish at some point. On the other hand, migration of virtual machines that are executing I/O-bound jobs must be determined carefully. When I/O-bound jobs are currently running on the virtual machine, in some cases migration does not complete until the job running inside of the virtual machine has ended. The evaluation has shown that migrating the virtual machine to a hypervisor that is closer to virtual disk image will improve the virtual machine performance. Therefore, "when" to execute the migration becomes critical on a virtual machine with I/O-bound jobs being executed.

In conclusion, this work has proposed a new architecture of computer migration. The architecture allows migration of virtual machines by focusing on resource utilization of applications that are running inside of virtual machines.

8.3 Future Work

This work has introduced a migration architecture focusing on device I/O status. There are various possible additions to the proposed architecture.

First of all, this thesis has used several resource utilizations as criteria for determining whether or not to migrate the virtual machine. Since user applications would require much more variety of resources, using only several resource utilizations for determining migration may not be enough. In addition to the resources mentioned in the thesis, availability of other specific hardware resources, such as video capture devices or sound devices, may also be considered as a parameter for accomplishing the required application tasks to be completed. Additional availabilities should also be considered in addition to the basic computing resources.

Next, the tests conducted in the thesis used a program where execution duration was known beforehand. However, on most of the applications that are used in the real world, expected runtime durations of the applications are unknown when they are started. For certain applications, such as video encoding, the duration could be

estimated from the files which the applications are accessing to. However, since it is difficult to know the execution time of most of the applications that are used in the real world today, a method such as prediction or learning trends of "well-known" applications becomes necessary. However, evidence from process migration studies suggests that systems that are currently compute-bound are likely to remain so[56].

In addition, the experiments conducted in the discussion already had a NFS server running on one of the hypervisors. To improve the disk access efficiency, other mechanisms, such as iSCSI, could be used. Use of iSCSI may not be difficult in an enterprise-class networks, but the goal of introducing this architecture is to be used with various different computers, including from home-use laptops to an enterprise-class servers. Thus, the presence of iSCSI or other network-shared storage server may not always be a good assumption for all the targets focused in the proposal; even having NFS services running on the hypervisors may not be common if end-users become a major part of the users of this system. To make the live migration less complicated, a mechanism for switching the access method to disk drives, such as having a disk image within a laptop, and when migration is necessary, automatically export the disk image using NFS or iSCSI accesses, may become necessary.

Lastly, in this thesis, live migration did not complete while disk I/O-bound processes were running on the virtual machine. This phenomena was caused by design of QEMU-KVM, but on other hypervisor implementations, for example Xen, sometimes force the virtual machine to suspend for few seconds to accomplish the migration. This approach is effective when the virtual machine has to be removed from hypervisors, such as the case where the hypervisor needs to be shut down for maintenance or other reasons. However, because the virtual machine will pause for few seconds, there is a greater chance of running processes getting affected by the pause. For example, if the processes are creating network sockets, the connection may be terminated as the virtual machine stops responding to the network requests. Nowadays, many applications create network connections for various purposes. Thus, the termination of network connection may even affect the running processes. To avoid running processes from getting affected from network disconnection, introducing a new mechanism for accomplishing live migration without pausing the virtual machine may become a solution. This introduction of a new mechanism would be a future work for improving the system effectiveness.

The proposed system is only a step toward the computing architecture with virtualization and device I/O transport over IP networks. Various additional works can be made to extend the computer resources to be used widely without boundaries.

Acknowledgments

In writing this thesis, I got numerous suggestions from advisors in Jun Murai Laboratory at Keio University. I would like to acknowledge advisors, Professors Jun Murai and Osamu Nakamura, Associate Processors Hiroyuki Kusumoto, Jin Mitsugi, Keisuke Uehara, Rodney D. Van Meter III, Hitoshi Asaeda, Assistant Professors Noriyuki Shigechika, Kenji Saito, and Hideaki Yoshifuji for supporting my work. I particularly would like to thank Associate Professor Rodney Van Meter for making suggestions to the experiments and making comments on thesis.

Next, I would like to thank Takeshi Matsuya and Kouji Okada, doctoral course students at Jun Murai Laboratory. They gave me numerous advices and things to think about for accomplishing the thesis. I would also like to thank Masaaki Sato, Katsuhiro Horiba, Noriatsu Kudo, Tsuyoshi Hisamatsu, Kazuhisa Matsuzono, Hajime Tazaki, Masayoshi Mizutani, Michio Honda, Yuichi Nakamura, Takashi Tomine, Yohei Kuga, Tetsuro Tamura, Keigo Emura, Toshiaki Hatano, Yuki Uehara, Shota Nagayama, Kunihiko Shigematsu, Munehiro Minegishi, Takayoshi Mibe, Yu Ukai, Dan Sawada, Ryo Nakamura, Miyuki Ozawa, Shigeki Murakami, Hisaki Nakamura, Takuya Shibuta, Yudai Yamagishi, Daiki Fujimoto, Asuka Nakajima, Hiroaki Kono and other members of Jun Murai Laboratory for supporting me in writing the thesis. I would like to thank secretaries of Murai Laboratory for managing schedules with Professor Murai. And, I would like to acknowledge members of Jun Murai Laboratory for supporting my works.

I also would like to acknowledge friends, who've spent time with me chatting face-to-face, on phone, or on the Internet. I would like to thank Yusuke Kuromiya, Gen Mineki and Shiori Suzuki who've worked together on each one's master's thesis on Fall Semester 2010. And lastly, I would like to acknowledge my family for supporting me to work on the thesis.

February 14, 2011
Keisuke Muda

References

[1] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, Vol. 2(No. 1), January/February 1998.

[2] University of California. SETI@home. http://setiathome.berkeley.edu/.

[3] Vijay Pande and Stanford University. Folding@home. http://folding.stanford.edu/.

[4] N.R. Adiga, G. Almasi, G.S. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich, A.A. Bright, J. Brunheroto, C. Cascaval, J. Castanos, W. Chan, L. Ceze, P. Coteus, S. Chatterjee, D. Chen, G. Chiu, T.M. Cipolla, P. Crumley, K.M. Desai, A. Deutsch, T. Domany, M.B. Dombrowa, W. Donath, M. Eleftheriou, C. Erway, J. Esch, B. Fitch, J. Gagliano, A. Gara, R. Garg, R. Germain, M.E. Giampapa, B. Gopalsamy, J. Gunnels, M. Gupta, F. Gustavson, S. Hall, R.A. Haring, D. Heidel, P. Heidelberger, L.M. Herger, D. Hoenicke, R.D. Jackson, T. Jamal-Eddine, G.V. Kopcsay, E. Krevat, M.P. Kurhekar, A.P. Lanzetta, D. Lieber, L.K. Liu, M. Lu, M. Mendell, A. Misra, Y. Moatti, L. Mok, J.E. Moreira, B.J. Nathanson, M. Newton, M. Ohmacht, A. Oliner, V. Pandit, R.B. Pudota, R. Rand, R. Regan, B. Rubin, A. Ruehli, S. Rus, R.K. Sahoo, A. Sanomiya, E. Schenfeld, M. Sharma, E. Shmueli, S. Singh, P. Song, V. Srinivasan, B.D. Steinmacher-Burow, K. Strauss, C. Surovic, R. Swetz, T. Takken, R.B. Tremaine, M. Tsao, A.R. Umamaheshwaran, P. Verma, P. Vranas, T.J.C. Ward, M. Wazlowski, W. Barrett, C. Engel, B. Drehmel, B. Hilgart, D. Hill, F. Kasemkhani, D. Krolak, C.T. Li, T. Liebsch, J. Marcella, A. Muff, A. Okomo, M. Rouse, A. Schram, M. Tubbs, G. Ulsh, C. Wait, J. Wittrup, M. Bae, K. Dockser, L. Kissel, M.K. Seager, J.S. Vetter, and K. Yates. An Overview of the BlueGene/L Supercomputer. *Supercomputing, ACM/IEEE 2002 Conference*, November 2002.

[5] Samba Team. Samba. http://www.samba.org/.

[6] Takahiro Hirofuchi, Eiji Kawai, Kazutoshi Fujikawa, and Hideki Sunahara. USB/IP - a Peripheral Bus Extension for Device Sharing over IP Network.

Proceedings of the FREENIX Track: USENIX Annual Technical Conference, pages 47–60, April 2005.

[7] Takahiro Hirofuchi, Eiji Kawai, Kazutoshi Fujikawa, and Hideki Sunahara. USB/IP: A Transparent Device Sharing Technology over IP Network. *IPSJ Transactions on Advanced Computing Systems*, Vol. 46(No. SIG11(ACS11)):pages 349–361, August 2005.

[8] J. Satran, K. Meth, C.Sapuntzakis, M.Chadalapaka, and E.Zeidner. Internet Small Computer Systems Interface (iSCSI), RFC 3720. http://www.ietf.org/rfc/rfc3720.txt, April 2004.

[9] USB Implementers Forum, Inc. Universal Serial Bus 3.0 Specification. pages 3–3, November 2008.

[10] Mark Hayter and Derek McAuley. The Desk Area Network. *ACM SIGOPS Operating Systems Review*, Vol. 25(No. 4):pages 14–21, May 1991.

[11] Henry H. Houh, Joel F. Adam, Michael Ismert, Christopher J. Lindblad, and David L. Tennenhouse. The VuNet Desk Area Network: Architecture, Implementation, and Experience. *IEEE Journal of Selected Areas in Communications*, Vol. 13(No. 4):pages 710–721, May 1995.

[12] Gregory G. Finn. An Integration of Network Communication with Workstation Architecture. *ACM Computer Communication Review*, 21(5):18–29, October 1991.

[13] Rodney Van Meter, Gregory G. Finn, and Steve Hotz. VISA: Netstation's Virtual Internet Adapter. *ASPLOS VIII*, pages 71–80, October 1998.

[14] Rodney Van Meter. A Brief Survey of Current Work on Network-Attached Peripherals. *ACM Operating Systems Review*, January 1996.

[15] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, Vol. 8(No. 3):pages 221–254, Summer 1995.

[16] bochs: Open Source IA-32 Emulator Project. http://bochs.sourceforge.net/.

[17] QEMU. http://www.qemu.org/.

[18] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[19] Microsoft Windows Virtual PC.
http://www.microsoft.com/windows/virtual-pc/.

[20] VMware. http://www.vmware.com/.

[21] Xen. http://www.xen.org/.

[22] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, 2003.

[23] Kernel Based Virtual Machine. http://www.linux-kvm.org/.

[24] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the Linux Virtual Machine Monitor. In *Proceedings of the Linux Symposium*, volume One, pages 225–230, Ottawa, Ontario, Canada, June 2007.

[25] Microsoft. Windows Server 2008 R2 Virtualization with Hyper-V.
http://www.microsoft.com/windowsserver2008/en/us/hyperv-main.aspx.

[26] Parallels. http://www.parallels.com/.

[27] VirtualBox. http://www.virtualbox.org/.

[28] Yuji Nakagawa, Shigeyoshi Takahashi, Masakazu Kishima, Shinetsu Isawa, Yoshitami Fujisawa, Hitoshi Kobayashi, Tomohiro Yamashita, and Megumi Iwata. 2010: A virtualization odyssey (in japanese: 2010年仮想化の旅新世代を迎えたユーティリティコンピューティング). *Monthly ASCII .technolgoies, May 2010 Issue*, Vol. 15:24–85, March 2010.

[29] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, Vol. 10, August 2006.

[30] Detailed Comparisons of Hardware Virtualization: Intel VT vs. AMD AMD-V (in Japanese: Intel VT vs AMD AMD-V ハードウェア仮想化 徹底比較).
http://begi.net/files/virt6/HVML.pdf.

[31] Rusty Russell. virtio: towards a de-facto standard for virtual I/O devices. *SIGOPS Operating Systems Review*, Vol. 42(No. 5):95–103, 2008.

[32] Takao Kawazoe and Nobuo Miwa. Selection Called Private Cloud (in Japanese: プライベートクラウドという選択). *Monthly ASCII .technologies, August 2010 Issue*, Vol. 15:pages 83–103, June 2010.

[33] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[34] Google, Inc. Google App Engline. http://code.google.com/appengine/.

[35] Salesforce.com, Inc. force.com. http://www.salesforce.com/.

[36] Microsoft. Hotmail. http://www.hotmail.com/.

[37] Google. Gmail. http://www.gmail.com/.

[38] Mark Nuttall. A brief survey of systems providing process or object migration facilities. *SIGOPS Operating Systems Review*, Vol. 28(No. 4):64–80, June 1994.

[39] Moshe Bar. openMosix. http://openmosix.sourceforge.net/.

[40] A. Barak and A. Shiloh. Mosix. http://www.mosix.com/.

[41] Jonathan M. Smith. A Survey of Process Migration Mechanisms. *SIGOPS Operating Systems Review*, 22(3):28–40, 1988.

[42] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI '05: Proceedings of the 2nd Symposium on Networked Systems Design & Implementation*, pages 273–286. USENIX Association, 2005.

[43] Takahiro Hirofuchi, Hirotaka Ogawa, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. A Live Storage Migration Mechanism over WAN for Relocatable Virtual Machine Services on Clouds. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 460–465, 2009.

[44] A. Barak and A. Shiloh. Question: Why migrate processes when one can move a whole vm with a process inside. http://www.mosix.org/faq/output/faq_q0028.html.

[45] A. Barak and A. Shiloh. Question: Why shared-memory is not supported. http://www.mosix.org/faq/output/faq_q0071.html.

[46] A. Barak and A. Shiloh. Question: How to run a threaded application. http://www.mosix.org/faq/output/faq_q0072.html.

[47] Kimberly Keeton, David A. Patterson, and Joseph M. Hellerstein. A Case for Intelligent Disks (IDISKs). *SIGMOD Rec.*, 27(3):42–52, 1998.

[48] Domenico Ferrari and Songnian Zhou. An empirical investigation of load indices for load balancing applications. In *Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*, pages 515–528, Amsterdam, The Netherlands, The Netherlands, 1988. North-Holland Publishing Co.

[49] Kalinka Regina Lucas Jaquie Castelo Branco and Edward David Moreno Ordonez. Load indices - past, present and future. In *ICHIT Proceedings of the 2006 International Conference on Hybrid Information Technology*, pages 206–214, Washington, DC, USA, 2006. IEEE Computer Society.

[50] Martin Brown. System administration toolkit: Monitoring a slow system. http://www-128.ibm.com/developerworks/aix/library/au-satslowsys.html, June 2006.

[51] Hajime Taira, Kazuo Moriwaka, Ryoichirou Tsuruno, and Kouhei Maeda. *Thoroughgoing Introduction to KVM (in Japanese: KVM徹底入門).* Shoeisha, July 2010.

[52] Patrick Schmid. Does cache size really boost performance? http://www.tomshardware.com/reviews/cache-size-matter,1709.html, October 2007.

[53] Tips & tricks - featured article: /proc/meminfo explained. http://www.redhat.com/advice/tips/meminfo.html, Match 2003.

[54] QEMU-DEVEL Mailing List. Re: [qemu-devel] [patch] stop the iteration when too many pages is transferred. http://www.mail-archive.com/qemu-devel@nongnu.org/msg47184.html, November 2010.

[55] Ffmpeg downloads. http://www.ffmpeg.org/download.html, October 2010.

[56] Mor Harchol-Balter and Allen B. Downey. Exploiting Process Lifetime Distributions for Dynamic Load Balancing. Technical Report UCB/CSD-95-887, EECS Department, University of California, Berkeley, November 1995.