國立交通大學

資訊科學與工程研究所

碩 士 論 文

考慮虛擬機器間傳輸量的虛擬機器搬移機制

Virtual Machine Migration with Consideration of

Inter-Virtual-Machine Communication

研 究 生：梁昱雄

指導教授：張瑞川　教授

中 華 民 國　九 十 八 年 六 月

考慮虛擬機器間傳輸量的虛擬機器搬移機制

Virtual Machine Migration with Consideration of Inter-Virtual Machine Communication

研 究 生：梁昱雄　　Student：Yu-Hsiung Liang

指導教授：張瑞川　　Advisor：Ruei-Chuan Chang

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

考慮虛擬機器間傳輸量的虛擬機器搬移機制

研究生 ： 梁昱雄　　指導教授 ： 張瑞川 教授

國立交通大學資訊科學與工程研究所

論文摘要

在叢集式伺服器上提供穩定的服務及有效利用資源是很重要的一個課題。以往在伺服器上發生資源負荷過載的處理方法為行程中斷或是行程搬移，這兩個方法皆不夠有彈性及透明化。前者直接中斷行程，讓使用者感受到不便；後者雖可有效解決資源負荷過載的問題，但是必須先克服一些技術上的問題。

隨著虛擬機器的普及化，有人研究利用虛擬機器的搬移機制來解決資源負荷過載且擺脫以往不透明及沒有彈性的問題。但是之前的研究全都是把負載最重的虛擬機器搬移到負載最輕的機器上。

在此論文中，我們觀察出虛擬機器間內部傳輸較佳的特性，並根據此特性設計出一套可以增進叢集式伺服器內虛擬機器整體效能的搬移機制。在資源負荷過載時，此搬移機制可以把傳輸量大的虛擬機器聚在一起，藉以求得較佳的效能。

Virtual Machine Migration with Consideration of Inter-Virtual-Machine Communication

Student : Yu-Hsiung Liang Adviser : Prof. Ruei-Chuan Chang

Department of Computer and Information Science

National Chiao Tung University

Abstract

Using resource effectively and providing reliable services are both significant in the cluster. The previous researches to solve system overload are process suspend and process migration, but they are all not flexible and transparent. The former make user be aware of termination of service. And the latter can solve system overload successfully but it need some technique supports.

Some researches solve system overload by VM migration, it can successfully solve system overload and get more transparent and flexible. But the main idea of previous researches focus on that migrate the VM with highest load to lowest load destination.

In this thesis, we know that the performance of network traffic via Inter-Virtual-Machine Communication is better than via network. According to this feature, we present a Inter-Virtual-Machine Communication aware migration policy. When system overload occurs, the policy will union the VMs that have high traffic to each other in the same physical machine to get better performance.

致謝

首先要感謝我敬愛的指導老師　張瑞川教授、學長　張大緯教授和　張軒彬教授，感謝各位口委給予我最中肯的建議以讓我的碩士論文趨於完善。這兩年在張瑞川教授及　張大緯教授費心的指導下，學生方能順利完成此論文。在碩士班修業期間，老師們耐心地教導我正確的研究態度及指導我不同的研究方法，不時的導正學生的研究方向，是學生迷失研究方向時的兩盞明燈。在撰寫論文上，感謝張大緯學長不辭辛勞地給予我衷心的建議，教導我如何寫出一篇好的論文。

感謝作業系統實驗室碩士班學長們，宗恆、旻儒、函昱、子榮及彥百，同學智文及守為，有你們的陪伴讓我的實驗室生活更多采多姿。感謝博士班學長國政在實作上面給我的建議與指導。感謝明絜學長給予我許多 Xen 上面的知識，在我遇到 Xen 方面的問題時有人可以請教。感謝夢麟學長，在碩士求學後期一直給我正面的鼓勵。感謝亭彰學長，不論是在研究上還是生活上，一直以來都有你的幫忙和支持。

感謝阿弟、小民、神龍、機機、威爺、英奎、阿胖、賽門、凱嶸、老搞等好友們一路以來的相伴與支持，你們總是在我需要你們的時候出現，謝謝你們。感謝游池教練團們，強哥，家吭，阿忠，阿翔及吳明忠，感謝阿胖公司的同事，哲民、辣妹等，你們豐富了我的休閒生活，讓我在苦悶的研究生活注入了一股活水。

感謝一直以來支持我的家人，爸爸、媽媽、三位哥哥、大嫂及可愛的侄子姪女，你們無私的奉獻及純真的笑容是這篇論文完成最大的主因。最後感謝我的女朋友曉怡，感謝你在我研究生活最後的半年能體諒我的壓力，關心著我，陪伴著我度過這些日子，讓我可以專心完成我的碩士論文。最後僅以這篇論文，獻給我最摯愛的你們。

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

1.1 Motivation

Using resource efficiently and providing reliable services are both critical issues for server clusters. However, excessive utilizations of resources such as CPU, memory, network and disk bandwidths would lead to system overload, which decreases the service reliability and performance. A sudden load surge could cause a significant deterioration of service quality, even leads to service denial.

A number of load management mechanisms have been proposed for balancing the loads of the servers in a network service. In those mechanisms, the load controller dispatches the requests to the servers according to the server load. If an overload occurs, the controller can dispatch requests to the other servers or move some requests from the overloaded server (i.e., the hotspot) to the other servers.

Another way to solve the problem of system overload is through process migration, which can eliminate system overload effectively by moving process from the overloaded machine to another one with a lighter load. Process migration is not limited to load management within a network service. However, there are some challenges for process migration, such as full transparency, dependence of other processes, fast transferring for process state, proper migration algorithm and etc [1][5][9][10][11][12]. High implementation cost is needed if the operating systems themselves do not support process migration.

In virtualization environments, which rapidly gain in popularity in recent years, system overloads can easily be solved by using virtual machine (VM) migration [VM migration, sandpiper], which is user-transparent and hence avoids the above implementation costs. However, existing migration policies[22][23][24][25], which

determines the VM to be migrated and the destination host, focus mainly on the load balancing and hotspot elimination, but ignore the fact that different VMs in the same physical machine may communicate with each other.

In this thesis, we assume the communication performance in a physical machine is superior to that among physical machines. This assumption holds for low-cost clusters, which do not use specialized high speed links (such as Myrinet) for intra-cluster communication. Moreover, several techniques such as XenLoop and XenSocket have been proposed to improve the communication performance in a physical machine. As a result, it is common that the performance of optimized shared-memory based communication would outperform the NIC based communication.

We refer the communication between different guest VMs in the same physical machine as Inter-Domain Communication (IDC). Instead of passing the network interface card (NIC), IDC is usually implemented in shared memory in modern virtualization environments so that its bandwidth is not limited by the NIC. It is limited by the spend of CPU and Memory. Moreover, traffic via IDC can get better performance than original network because the speed of CPU is often faster than network interface card, especially in the workload of higher network I/O, such ftp, data center and web cluster.

According to our experimental results, grouping the virtual machines that communicate with one another, which are referred to as a *logical group* in this thesis, on the same machine helps to achieve a better performance. However, existing migration policies focus mainly on eliminating the hotspots and may separate the members (i.e., the VMs) of a logical group on different physical machines, leading to a performance degradation.

Table 1.1: Performance Results of Netperf via NIC and IDC

	Throughput (Mb/sec)	CPU Utilizations (%)
NIC	283.17	61.708725
IDC	1113.755	82.61254

As an example, Table 1.1 compares the performance results of netperf[30], a network benchmark, via NIC and IDC under a four-domain environment. Two netperf instances, with each of which contains a pair of netperf client and server programs, run in the environment, and each domain executes one of the programs. The IDC values show the results when all the netperf client programs communicate with their servers via IDC while the NIC show the results when all the netperf client programs communicate with their servers via NIC. As shown in the table, the throughput of IDC outperforms that of NIC by four times with the cost of about 21% CPU utilization. As another example, we run the Support test of SPECweb2005[31] with 90 sessions in the same four-domain environment. As shown in Table 1.2 and 1.3, communication via IDC can achieve a shorter response time without the extra cost of CPU resources. Specifically, 24.6% of the response time can be reduced in average.

Table 1.2 : Response time of each request type in SPECweb via NIC and IDC

	Request Types							
	home	search	catalog	product	fileCatalog	file	download	Average
NIC results(ms)	223	229.5	227	413.75	331.5	330.75	1757.75	415.25
IDC results(ms)	156	158.5	156	296.5	230.75	233	1547.75	313

Table 1.3 : CPU Utilizations of SPECweb via NIC and IDC

	CPU Utilizations (%)
NIC	70.2125
IDC	68.9375

To take advantage of the superior performance of IDC, we propose a new VM migration policy that considers IDC when making migration decisions. The policy solve system overload while trying to keep a logical group in the same physical machine (i.e., group union). Moreover, we also design and implement an automatic load management system and integrate the proposed policy into the system. According to the performance results, we demonstrate that the system is capable of solving multiple system overloads, and the IDC aware migration policy can achieve a superior performance by group union, when compared to the migration policy proposed by Sandpiper[24], an existing VM migration system. Specifically, the proposed policy can reduce the response time by up to 24% under the support test of the SPECweb2005 benchmark and improve the network communication performance by up to 102% under the netperf benchmark.

1.2 Structure of the Thesis

The rest of this thesis in structured as follows. Section 2 presents the related work. Section 3 presents the design and implementation of the load management system and the IDC aware migration policy. The performance evaluation is presented in Section 4. Finally, Section 5 gives the conclusions and the future work.

Chapter 2 Related Work

In this chapter, first, we introduce some researches about process migration task because it is the foundation of migration. Then, we introduce some researches about migration of virtual machine. Finally, we introduce some researches about Inter-Virtual-Machine that can improve the performance of our mechanism.

2.1 Process Migration

Process migration is an act of transferring an executing process between two machines. This idea was first presented by Finkel *et al.*[6], and also, Rashid and Robertson[7] in the 80's. Powell and Miller[5] firstly added feature of process migration on DEMOS/MP operating system. There are some challenges for process migration, such as full transparency, dependence of other processes, fast transferring for process state, proper migration algorithm and etc. The past researches[1][5][9][10][11][12] have provided several mechanisms to overcome those challenges in different operating systems, but it spent the expensive implementation cost if the operating systems do not support process migration.

Generally speaking, process migration for reaching load-balance in distributed systems has three major tasks: migration algorithm, load information management and distributed scheduling [13]. The algorithms of process migration are quite similar, and they can be summarized as following. Firstly, a migration request is issued to a remote node, and then the process is detached from source node. Secondly, we need to do some preparations before state transferring, such as redirecting the communication, extracting the process state and creating the destination process instance. Then, process state is transferred and imported into a new instance. Finally, resume the new

instance from destination node. Load information management is also an important component of process migration because we need to get the information that we want and normalize it. In addition, transferring the information as small as possible to improve the performance is also important. Distributed scheduling plays the major role of load balancing or overloads eliminating. The main goal is to determine *when* to migrate *which* process to *where*. Distributed scheduling can not only achieve load-balance but also eliminate overload; moreover, improve overall performance if we migrate right process to right destination.

Milojicic *et al.* [14] first suggested that move the client to destination where the server is located to improve performance because the client/server communication takes place in parallel. To get the better performance, the operating system needs to provide parallel programming environment and parallel run-time support system, such as PVM system [16].

2.2 Migration in Virtual Machine

Different from process migration, virtual machine migration can avoid many difficulties faced by process migration, such as process state saved and transferred, name space of process, and transparency of process migration. Moreover, the virtual machine environment provides a clean platform between operating system and hardware to solve the dependence issue.

Self-migration [19] used the mechanism, *resend-on-write* and transferred the remained dirty pages after checkpoint has been suspended to reduce the overall migration time, but it can not keep the service alive. Clark *et al.* [3] presented a transparent and fast virtual machine migration, named live-migration on Xen [4]. To achieve live-migration, there are three phases of transferring memory, *pre-copy*, *stop-and-copy* and *demand-copy*. When migration starting, all pages are transferred in

pre-copy phase. If the rate of dirty pages is lower than limited, it enters the stop-and-copy phase to transfer the remained dirty pages. After being finished of transferring dirty pages, the destination virtual machine starts and provides the service as before. If the page fault occurs, it will copy the page from source machine. They successfully minimized both downtime and total migration time and keep the service alive.

There are two types of storage in the common virtual machine environment, one is local disk and the other is using Network Attached Storage (NAS). NAS is a better choice for modern clusters because it can provide strong availability of data by building in RAID and the front-end can reduce the expensive disk I/O. Doing migration is unreasonable in the previous researches if only using local disk because the migrated domain on the destination machine will still access the local disk on the source machine. Nevertheless, Bradford *et al.* [20] has mentioned that transferring memory state as well as local persistent state in WAN, it will make the virtual machine environment friendlier. Therefore, doing migration is suitable for academy, industry, as well as common user.

With the maturity of virtual machine migration, some people try to solve system overload or making load-balance of clusters by using domain migration. Menasce and Bennani presented an autonomic virtualized environment [22], they consider dynamically CPU priority allocation and allocation of CPU shares in virtual machine to achieve load-balance. However, it just considers the factors of CPU and do not use the technique of virtual machine migration. Ruth *et al.*[23] builds autonomic virtual machine with consideration of CPU resource and memory resource, and it keeps each virtual machine's resource utilization within a specific range.

The above researches make the idea of load-balance and autonomic in the virtual machine become more practical, but they still lack consideration. T. Wood *et al.*[24]

show black-box and gray-box strategies to observe the utilization of resources and eliminate hotspots. They present Sandpiper, as a complete solution to support load-balance and autonomic virtualized environment with consideration of CPU, memory resource and network bandwidth in the virtual machine environment, and it consists of profiling engine, hotspot detector and migration manager. They integrate those resources into a formula to reduce the time of decision and use the swap mechanism to make migration more practical. Hyser *et al.*[25] presented overview of a virtual machine placement system and used a mean-value migration policy.

2.3 Inter-Domain Communication

Although the virtual machine environment can provide several advantages, the performance of virtual machine is still lower than the native machine. Some researches [26][27][28] introduced that I/O virtualization is the significant performance overhead of virtual machine environment because all of the I/O operations will go through domain-0 and VMM, and it caused the expensive domain switch and longer path of I/O operation. The main idea of solving the problem is bypassing domain-0 and VMM to reduce the overhead of communication between two domains on the same machine.

XenSocket [28] is a socket-based solution for increasing inter-domain throughput in xen. XenSocket avoid the TCP/IP overhead and bypass the domain-0 by providing a socket-based interface to shared memory buffers for inter-domain communication, but it can not support the general socket interface. Kim *et al.*[26] presented XWAY channel to deal with the socket between two guest domains. XWAY achieves high performance by bypassing TCP/IP stacks, avoiding page exchanging overhead, and create a directed and shorter communication path between two different guest domains. It also supports the general socket interface and live migration easily.

Chapter 3 Design and Implementation

In this chapter, we describe the design and implementation of the load management system and the IDC aware migration policy. Section 3.1 describes the design goal. Section 3.2 introduces the system design, which is followed by the description of the proposed IDC aware migration policy in Section 3.3. Finally, the implementation details are described in Section 3.4.

3.1 Design Goal

The design goals of the load management system are as follows.

- Maximize Resources Utilizations

  In the proposed load management system, the available resources of a physical machine should be used as a first step to eliminate the overload condition. VM migration is used when the available resources of a hotspot machine is not enough to solve the overload problem.

- Little Overhead

  The load management system requires gathering domain load information. For performance consideration, the load gathering job should not incur noticeable overhead to the system.

- Performance Improvement

  The migration policy should consider IDC so as to obtain a larger performance gain after VM migration.

3.2 System Overview

In this section, we give a basic description of the proposed load management

system.

3.2.1 Background

Xen is an open source virtual machine monitor (VMM) developed by University of Cambridge[4], and it has high performance due to the use of para-virtualization. Xen provides a split device driver model [8], in which a dedicated domain (i.e., domain 0) acts as the driver domain that serves all the device access requests from the guest domains. When an access request is generated from the guest operating system, the front-end driver of the guest domain passes the request to back-end driver of domain 0 through the shared-memory based I/O channel. Moreover, Xen supports live migration [3] and provides a *grant table* mechanism, called the *grant table mechanism*[8], to share memory pages between domains for improving the IDC performance.

3.2.2 System Architecture

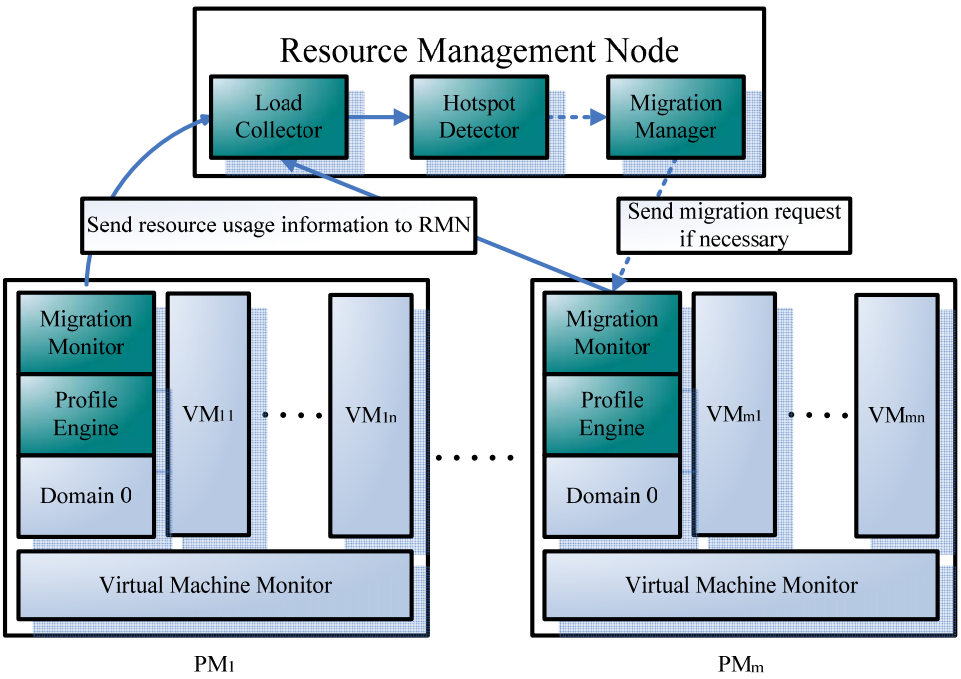

Figure 3.1 : The System Architecture.

The proposed load management system is based on Xen. Figure 3.1 shows the architecture of the load management system, which consists of the load collector, hotspot detector, migration manager, migration monitor and profiling engine. The former three components reside in a dedicated machine called Resources Management Node (RMN) while the latter two run in the domain 0 of each cluster node. The profiling engine collects the resource usage information of each domain on a cluster node and sends the information to the load collector periodically. The per-domain resource usage information consists of the CPU utilization, the size of the allocated memory, the frequency of swap writes and utilized network bandwidth. After receiving the resource usage information, RMN records the information into a history table. Then, the hotspot detector analyzes the information and detects if is there is a hotspot. If a hotspot machine is detected, the migration manager selects a victim domain on the hotspot machine and a destination physical machine with enough resources according to the migration policy. Finally, the migration manager sends the migration request to the hotspot machine to trigger domain migration. One exception is that, under memory overload, the VMM would try to allocate more memory resources to the overloaded domain first before performing the domain migration. Figure 3.2 shows the message flow of VM migration control in our system. In the figure, the upper part shows the flow corresponding to a memory overload that can be solved by allocating more memory for the overloaded domain, while the bottom part shows the flow that solving the overload problem by domain migration. After the finish of the VM migration, the resource usage information on the source machine of the migration is reclaimed.

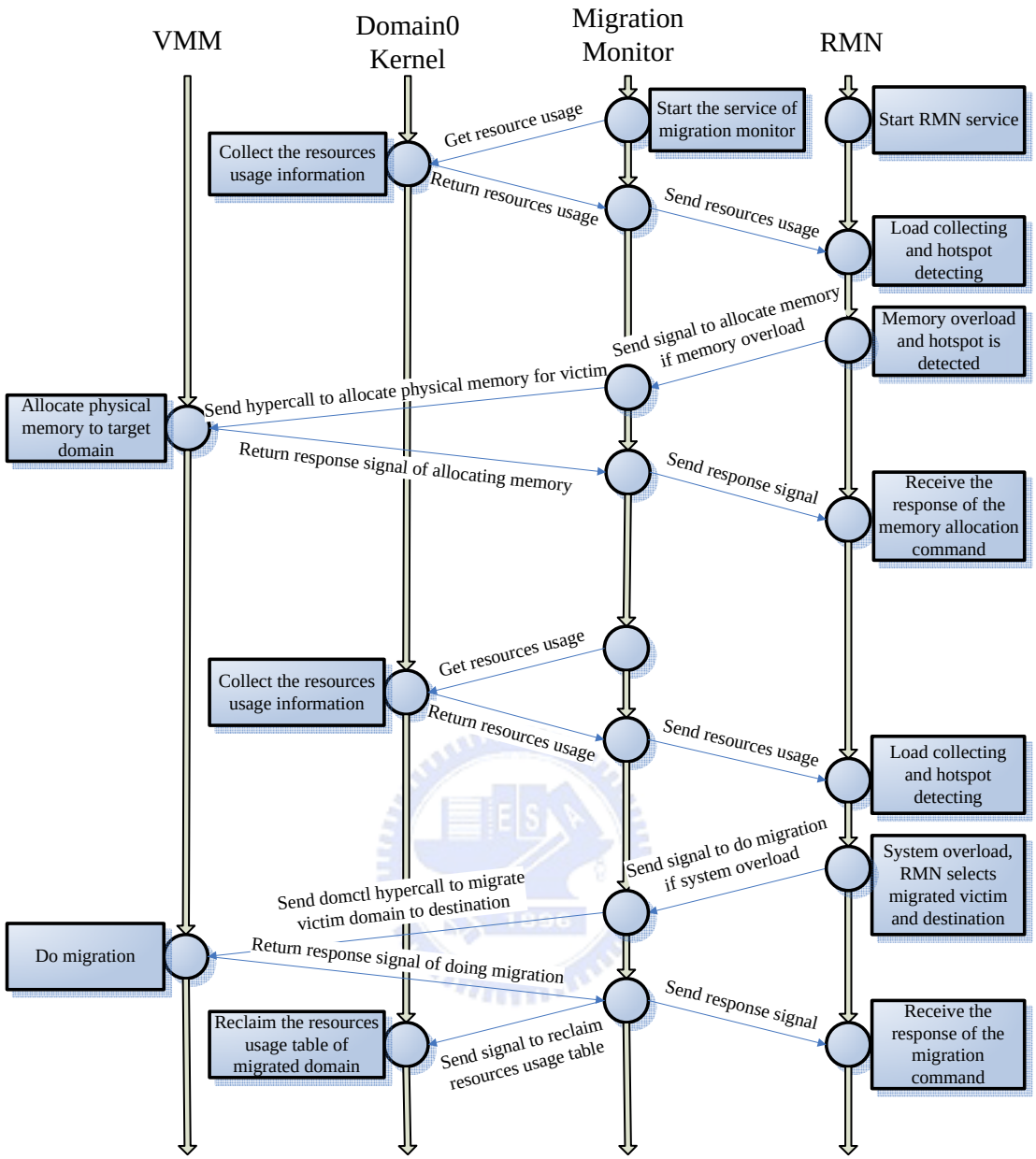

Figure 3.2 : The Message Flow of VM Migration Control

3.3 IDC Aware Migration Policy

IDC Aware Migration Policy

Thee basic idea of the policy is to group together domains that have moderate or heavy communication traffic among them and run the group on the same physical machine so as to utilizing the high performance of IDC. To simply the description, we assume that each group has an unique group ID (GID) are located at the same physical machine.

The basic abstraction of purposed IDC aware migration policy is migrate the highest load domain from source to light load destination. In addition, the chosen victim (i.e., the domain to be migrated) is with high throughput to corresponding destination. To choose the victim from hotspot machine easily and reasonably for all case of system overload, we defined the volume function V_{ijk} for domain with index j in the physical machine i and the destination machine is with index k as following:

$$V_{ijk} = CPU_{ij} * C_i + Mem_{ij} * M_i + NB_{ijk} * N_i \quad (3)$$

Where CPU_{ij} represented as CPU utilization, Mem_{ij} represented as memory utilization, NB_{ijk} is *Network benefit* for domain j in physical machine i and the destination is machine k. C_i, M_i and N_i are the overload flags for CPU, Memory and Network separately. *Network benefit* is meant that real reclaimed network traffic from physical NIC on the hotspot machine. Different from Sandpiper, they calculate network traffic including IDC and we exclude IDC. Sandpiper does not make sense because IDC does not pass through physical NIC.

If the physical machine got overload of CPU, the C_i would be set to 1, otherwise it would be 0. M_i and N_i are the same as C_i, but to consider the feature of IDC totally, we always set N_i to be 1. We can easily understand that the proper victim is with highest V_{ijk}, because it can release more resources to address the correct type of overloads. It is also a difference from Sandpiper. With this volume function, we can find the proper victim and destination easily as soon as possible.

Network Benefit

The real reclaimed network traffic, *network benefit*, is quite significant to avoid the erroneous decision of hotspot detection and choosing victim domain. Figure 3.3 and 3.4 illustrate the situations before and after migration, respectively. The definitions of the symbols are shown in Table 3.1. Suppose the victim VM_{i1} has to be migrated from source machine PMi to PMj. VM_{i1} has three types of communication:

Is, denoting the volume of IDC, Nsd, denoting the volume of traffic from VMi1 to the destination machine, and Nso, denoting the volume of traffic from VMi1 to the other physical machines. After the migration, the Is would become Nds, the connection from destination to source via NIC, the Nsd would become Id, the IDC on the destination, and the Nso would become Ndo, the connection from VMi1 (on the destination) to the other physical machines, still remains. Without losing the generality, we assume that Ndo is equal to Nso.

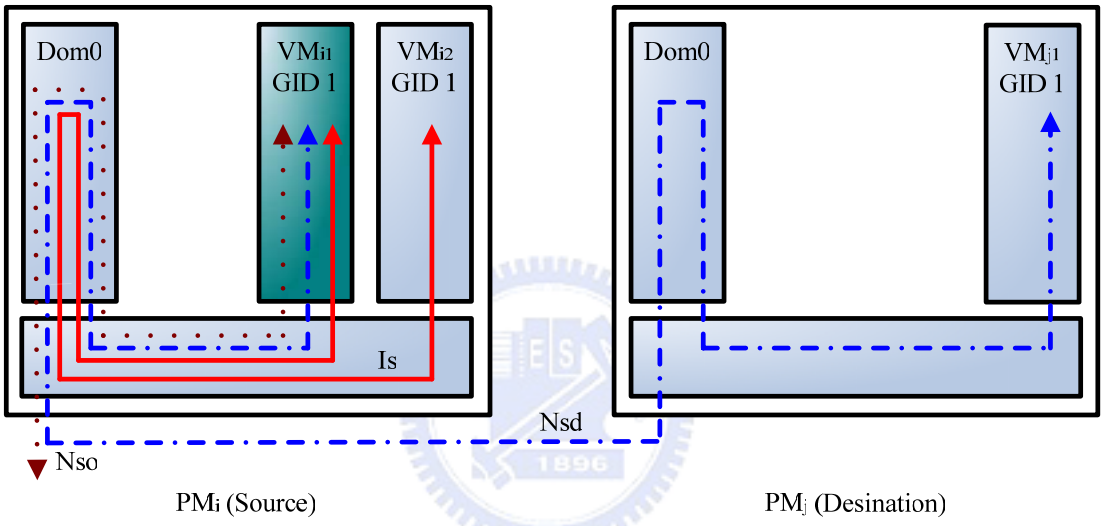

Figure 3.3 : The Network Communication before Migration

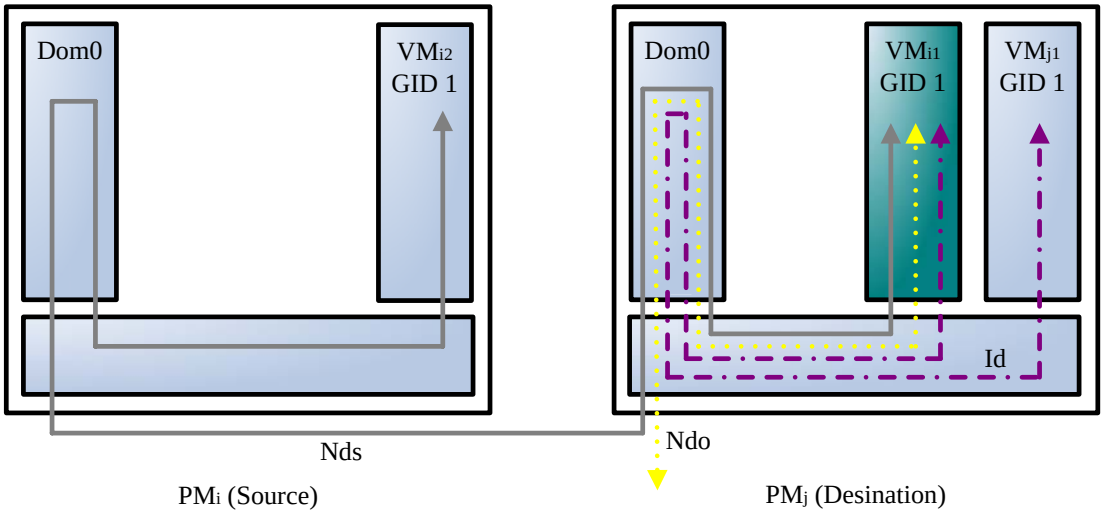

Figure 3.4 : The Network Communication after Migration

Table 3.1 : Symbol Definitions

Symbol	Definition
Is	Inter-Domain Communication on Source.
Id	Inter-Domain Communication on Destination.
Nso	Volume of traffic from Source to Other machine.
Nsd	Volume of traffic from Source to Destination.
Ndo	Volume of traffic from Destination to Other machine.
Nds	Volume of traffic from Destination to Source.
AB	Available Bandwidth after migration.
NB	Network Bound for one domain in virtual machine.

According to above description, we define *network benefit* that given as following:

$$Network\ benefit = Nso + Nsd - Nds \quad (1)$$

The Network benefit is like the utilization of network, but it is more accurate in releasing network resource in the virtual machine. The main ideas of the design of network benefit is the source can reclaim accurate network bandwidth and get better performance with consideration of IDC and destination. Generally, we consider the available resources that after migration are equal to the released resources form victim domain, but available resources are less than released resources. The reason is that there are IDC between another domains and victim domain at source machine. After migration, the IDC becomes the original network traffic between source and destination machine, so that the accurate available resources in network is *Nso + Nsd – Nds*, named *Network benefit* to distinguish form the original network utilization.

For example, with the concept of group that we mention as above, there were

several groups in the source machine and we supposed that we have two candidate domains with equal resources, named A and B. A was with higher IDC in one of groups. B was a single domain in source machine and connected to domains which are located in other machines. B had higher network traffic than its IDC, and A was opposite. That is to say, B had higher Nso + Nsd and lower Is, and A had lower Nso + Nsd and higher Is. According to our model, Nds were higher related to Is because Is will change to Nds after migration. If one domain's IDC is higher than another, its Nds traffic would has the same situation. It we choose A to be the proper victim, we could not release more network resource and would get lower performance because we decrease the IDC of A. With the *Network benefit* function, B gets the bigger value, so that we choose it to be the proper victim. As our expectation, *Network benefit* can help us to choose the proper victim with lower IDC to be migrated and release more network resource. We can summarize above, it is a good choice if the domain has higher network benefit, because the victim has higher network traffic and lower IDC. We should make the domain stayed on the source machine if the domain has lower network benefit.

But we have a significant challenge in calculating *Network benefit*; it is how to get the correct value of Nds. We can get Nso, Nsd and Is by tracking the network traffic on source machine, but we have no idea about the real values of Nds and Id after migration. To let *Network benefit* be more reasonable and accurate, we need to predict the value of Nds. It is a hard work because our system is based on the virtual machine environment, it caused that we did not have a well prediction of Nds by using previous researches[29]. The previous researches about predicting the future network traffic all focus on the more stable environment than virtual machine. That is to say, they did their researches on single machine and there are no other effective factors except the running applications. On the virtual machine environment, we

should consider the running applications on the domain as well as the effects of other domains on the source machine. So, we present a simple algorithm to predict Nds without consider the all effective factors; it can reduce the dimensions of prediction algorithm and has reliability of a certain degree in normal case. Our simple prediction algorithm is as following:

$$Nds = min\ (Is,\ AB,\ NB) \quad (2)$$

In our model, we introduce AB and NB, AB is available bandwidth after migration and NB is the network bound in virtual machine. Virtual machine has lower network performance than native physical machine because the domains can not direct access the physical network devices. It restricts the real network bandwidth of virtual machine, and to make our prediction more accurate, we must define the value of network bound of virtual machine, NB. In the case of network overload, Nds is related with Is, AB and NB, and we consider these conditions as following:

1. Nds was higher related to Is because it was exchanged from Is, Generally, the value of Nds was lower than Is in the high network performance environment.
2. Nds is defined as the original network traffic after migration, so it can not exceed the available bandwidth of the source machine.
3. As the mention as above, the network throughput can not reach the limit of gigabit network interface card in the virtual machine environment. The Nds was bounded in network bound of virtual machine, NB.

According to above, we predict the value of Nds to be the minimum of Is, AB and NB

In the normal case of choosing destination machine, we also want to consider the effect of IDC, but predict the Id that was changed from Nsd is harder than predict Nds. So we only consider if there is enough resources in the destination, but not the effect of the IDC in the destination.

In following discussion, we suppose the destination machine have enough resources to solve system overload by using migration. In the case of having CPU or network overload, as the mention as above, we find the victim with maximal volume value and destination that have enough resources. If the destination does not have enough resources to contain the victim domain with maximal volume value, we would choose the next bigger one from volume list until the destination machine can contain it. Moreover, if there are two domains which have the same volume value and destination machine also has enough resources, we would choose the domain with higher network traffic to be the victim.

In the case of having memory overload, it is the same as having CPU or network overload partially. The difference of solving overload is that we provide the dynamic provisioning technique to allocate physical memory to the target guest domain before doing migration. When the memory overload happened, we will find the victim with bigger volume value and allocate extra physical memory to victim domain to eliminate the frequent swap. If it still has memory overload, we will repeat this process until the physical memory of this machine is empty. The rules of dynamic provisioning were described as following:

1. The extra additional physical memory is a fixed amount of quantity for victim guest domain at every allocation.
2. Any guest domain can not add the extra physical memory unlimitedly. If the allocated physical memory of target domain has already achieved the maximal amount, the memory reallocation of target domain would not be allowed.
3. The memory reallocation should be stopped if there is no any free physical memory on the physical machine.

Moreover, if there are CPU and memory overloads in the same time, we would solve

memory overload first because eliminating the memory overload sometimes can solve CPU overload.

3.4 Implementation

We present the implementation details in this section.

3.4.1 Information Collection

Information Collection Setup

The information collection can be broken down into three parts: CPU, memory and network. To make our implementation more simple and flexible, the code modifications are only done in the privilege domain (i.e., domain 0). We build two data structures *domain_info* and *net_info* for information collection, as shown in Figure 3.5. The domain_info records the necessary information of each domain, such as domain ID, CPU usage, the amount of allocated memory, amount of swap operations and network information. The net_info structure records the volume of a specific kind of traffic (e.g., IDC from domain 1 to domain 10). Each domain_info consists of four lists of net_info structures for recording Tx and Rx volumes of the NIC and IDC traffic. Each net_info consists of the IP address, MAC address and total traffic bytes of the remote domain. For IDC, remote domain id is also included in the net_info structure.

```
typedef struct net_info *netinfo_ptr;
typedef struct net_info{
   domid_t                domid;        // domain id (only for IVM)
   __be32                 ip;           // ip address
   u8                     mac[6];       // mac address
```

```
    unsigned long           total_bytes;    // total bytes of traffic
    netinfo_ptr             prev;           // point to previous net_info
    netinfo_ptr             next;           // point to next net_info
}net_info_t;

struct net_info_header{
    int                     count;          // total count of list
    netinfo_ptr             front;          // the first net_info item
    netinfo_ptr             tail;           // the last net_info item
    spinlock_t              lock;           // for list protected.
};

typedef struct domain_info *dominfo_ptr;
typedef struct domain_info {
    domid_t                 domid;          // domain id
    __be32                  ip;             // ip address of domain
    u8                      mac[6];         // mac address of domain
    unsigned int            swap_blkif;     // id for swap partation
    unsigned int            root_blkif;     // id for root disk partation
    struct timespec         time_stamp;     // time stamp
    struct timespec         time_period;    // time period
    unsigned long           cpu;            // cpu execution time
    unsigned long           mem;            // current allocated memory
    unsigned long           max_mem;        // the maximun of memory
    unsigned long           swap_r ;        // amount of swapping read page
    unsigned long           swap_w ;        // amount of swapping write page
    struct net_info_header  itx;            // TX net_info_header for IDC
    struct net_info_header  otx;            // TX net_info_header for network
    struct net_info_header  irx;            // RX net_info_header for IDC
    struct net_info_header  orx;            // RX net_info_header for network
    dominfo_ptr             prev;       // point to previous domain_info
```

```
    dominfo_ptr             next;           // point to next domain_info
}domain_info_t;
```

Figure 3.5 : The Main Data Structures for Information Collection

To create a domain, the function do_domctl() is invoked, which in turn invokes the do_xen_hypercall(). In the hypercall, the privilege domain execute privcmd_ioctl() to ask the VMM to create a domain. VMM then chooses a proper domain id for the new domain and create a new guest domain. Next, VMM sets up the event channel, creates the grant table, performs other domain initialization jobs, and adds the domain to the run queue for domain scheduling. Finally, VMM sends an event to the privilege domain to initialize the back-end drivers of the newly-created guest domain such as xenbus, balloon driver, virtual block device interface, virtual network interface and etc..

The domain_info should be registered at the boot time of each guest domain without the modifications of guest domain. Modifying the guest operating system for such registration would raise the portability problem since it is hard to support different operating systems on the guest domains. Moreover, we also want to reduce the frequencies of expensive system calls or hypercalls so as to decrease the overhead of collecting resource usage information. According to the design goals above, we choose to register the data structure of domain_info upon the initialization of the Xenbus driver. Such an approach results in a lower implementation cost since no modifications to the VMM and guest domains are needed. Then, we create a mapping between swap partition and virtual block device to records swap information into the domain_info at the starting time of virtual block device[21]. The mapping helps us to separate swap read/write operations from other disk IO operations. Finally, we record the MAC and IP addresses into domain_info of guest domain at receiving/transmitting

the first packet for that domain. After the description about the setup of the domain_info and net_info, we next describe the information collection approach. All the collected information is sent by the Profiling Engine to the RMN periodically, which is done via TCP/IP.

CPU and Memory Information

The CPU execution time can be obtained easily by using Xenstat, the tool provided by Xen. Xenstat utlizes hypercalls to obtain the detailed resource usage information from the VMM, such CPU execution time, total network traffic, current memory size and maximal size of memory. To obtain the frequency of swap writes, we have to track the swapping disk I/O in the backend block device interface. Upon a block request, if the target device id is equal to a swap partition id of a guest domain, we regard the request is a swap operation.

Network Information

The collection of network information has the largest overhead in our system because we need to parse packets to classifiy them into IDC and NIC traffic.

Upon receiving a packet, either from a guest domain or from the NIC card, we check the packet's IP addresses to see if the packet belongs to intra-cluster communication (i.e., both source and destination are in the cluster) and MAC address of each packet if the same as MAC address of remote guest domain. Only intra-cluster communication is recorded for saving the recording cost. If the condition holds, we count this packet into the corresponding field of the net_info structure of the guest domain. To reduce the time of searching a net_info structure, the most recently accessed net_info is cached.

3.4.2 Migration Monitor and RMN

In this section, we describe the detailed implementation of the migration monitor and the components in the RMN.

Migration Monitor

The migration monitor handles two kinds of requests sent from the RMN if the latter detects a hotspot. One is memory allocation request and the other is domain migration request. Upon receiving a memory allocation request that is accompanied with the ID of the target domain and the request memory size, the migration monitor issues a memory allocation hypercall for allocating that size of memory for the given domain. Upon receiving a domain migration request, which is accompanied with the victim domain and the destination machine, the migration monitor issues a migration command to migrate the victim domain to the destination.

Load Collector

As the per-domain resource usage information is sent to the load collector, it arranges the information into a history table It also establishes another table for recording the traffic among all domains and sort the information according to the traffic volumes for efficient search.

As the migration command is sent to hotspot machine, the *dom_info* of victim guest domain was registered on the destination machine, which leads to the existence of two *dom_info*s for the same guest domain on the source and destination. To synchronize the statistics from different machine after migration, we should check the lifetime of the victim from source and destination machine separately. Otherwise, there will exist that history table of a domain belong to different machines Figure 3.6 shows the process of load collector and lifetime of victim during the time of migration.

The Profile engine sends the resource usage information for victim to load collector, which will collect the resource usage information if the state of victim is not paused (paused means the domain is unavailable) to avoid non-synchronized statistic. More accurately, load collector record the information of migrated domain only when its state is running. After the time of migration finished, the profile engine on the source will be weak up to send statistic information.

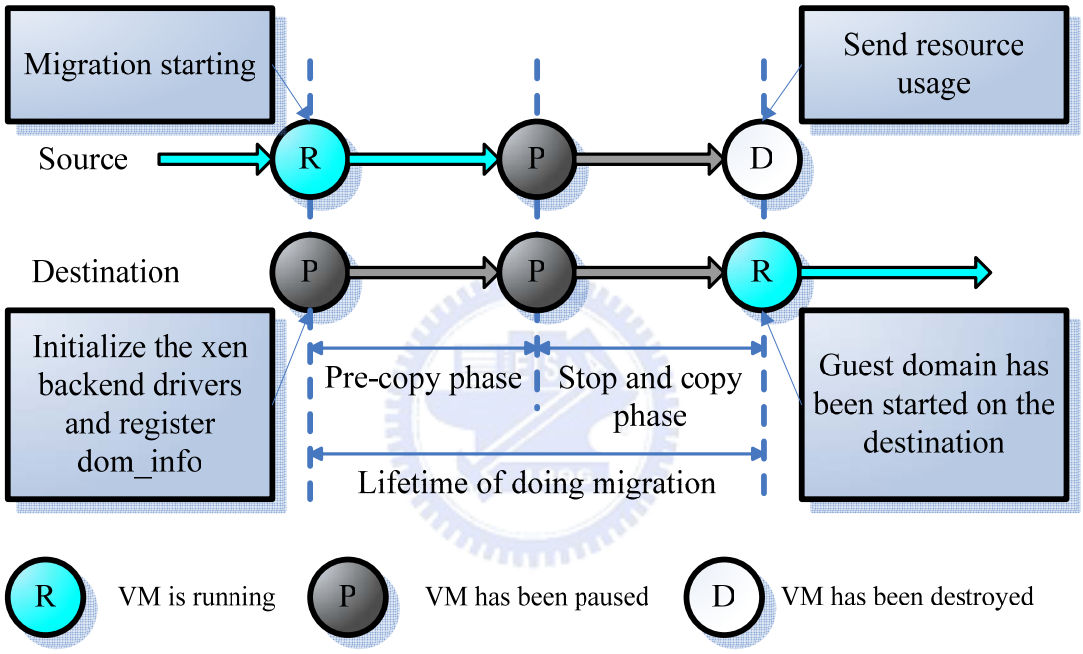

Figure 3.6 : The Lifetime of Migrating Victim Domain.

Hotspot Detector

An instantaneous system overload would not be regarded as a hotspot. If the current resource usage information shows a system overload, the hotspot detector predicts the load of such potential hotspot. VM migration is triggered only when the result of the prediction also shows a system overload. This avoids VM migration due to sudden utilization peaks. According to the history table, the hotspot detector predicts the utilization of the next round by using the autoregression of order 1. Given the sequence of history volumes: $u_1, u_2, \ldots, u_k$, where u_k is the utilization of the k-th round, we predict u_{k+1} by using the following AR(1) predictor,

$$u_{k+1} = \mu + \varphi(u_k - \mu) \quad (4)$$

where μ is the mean value of sequence and φ is the parameter of time series.

If both the current and the predicted utilizations exceed a threshold, the source machine is regarded as a hotspot. Note that, the hotspot detector would not detect the loads of the machines that are involving VM migration, since the loads are not stable during that period.

Migration Manager

When the hotspot is detected, the migration manager will try to send memory allocation requests to the overloaded machine if memory overload. If the overload cannot be solved by the memory allocation requests, the migration manager would choose the victim and the destination machine according to the IDC aware migration policy. Next, it sends the migration request and the required information, such as the domain id of the victim and the IP address of the privilege domain in the destination machine, to the migration monitor corresponding to the victim domain.

Chapter 4 Performance Evaluation

In the chapter, we present the performance evaluation of the proposed IDC-aware migration policy. Section 4.1 describes the experimental environment. In Section 4.2, we demonstrate that the policy can solve resource overload problems and compare the performance the proposed policy with Sandpiper[24]. In Section 4.3, we show the overhead of our system.

4.1 Experimental Environment

The experimental environment consists of a server cluster of four nodes, a client node, and a RMN node. Figure 4.1 illustrates the experimental environment and Table 4.1 shows the specifications of each node. Each cluster node is equipped with an Intel Pentium 4 2.8 GHz processor, 2GB DDR RAM, an Intel Pro 100/1000 Ethernet adapter and an 80 GB hard drive. We run Xen 3.1.0 with Linux kernel 2.6.18 on each node. The client machine is equipped with an Inter Pentium 4 3.2 GHz processor, 3GB DDR2 RAM, an Intel Pro 100/1000 Ethernet adapter and an 80 GB hard drive. Xen 3.1.0 with Linux kernel 2.6.18 is run on the client machine.

The RMN node is equipped with an Intel Pentium 3.2 GHz processor, 3GB DDR2 RAM, an Intel Pro 100/1000 Ethernet adapter and a 500GB hard drive. Note that, the RMN node also acts as the NFS storage server for the guest domains in the cluster. All the nodes are connected via the D-Link DGS-1024D gigabit switch, and we use SEPCweb2005 and netperf-2.4.2 as the benchmark software for performance evaluation.

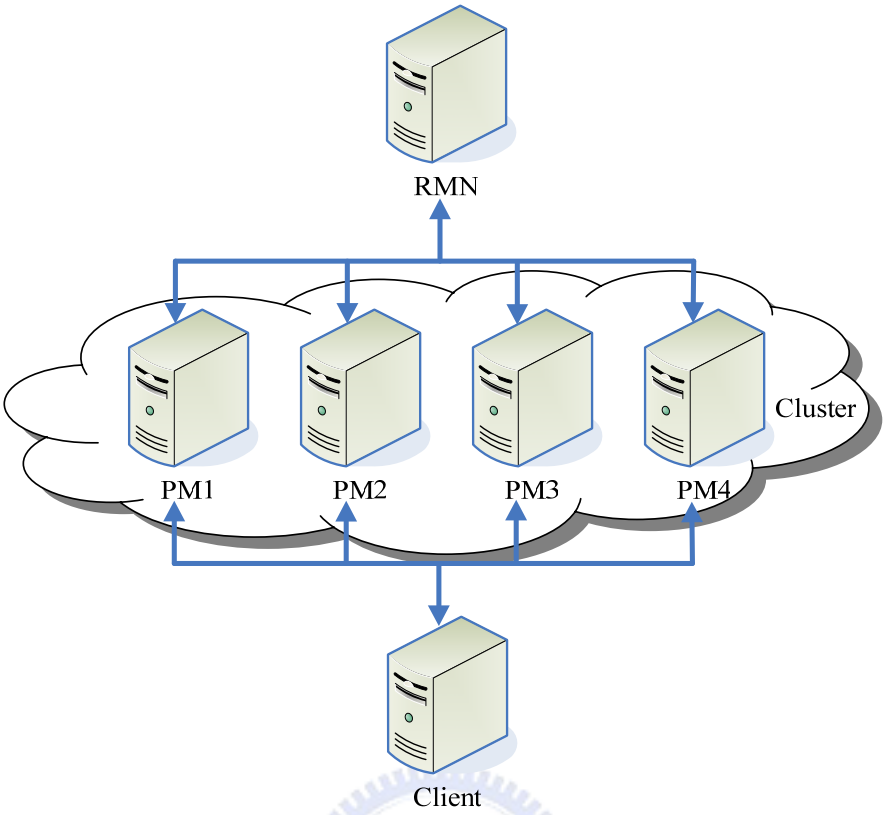

Figure 4.1 : The Experimental Environment

Table 4.1: The Specification of Each Machine

	Cluster Node	Client	RMN
Number of nodes	4	1	1
CPU	Intel P4 2.8G Hz	Intel P4 3.2G Hz	Intel P4 3.2G Hz
Memory	2GB DDR RAM	3GB DDR2 RAM	3GB DDR2 RAM
NIC	Intel Pro 100/1000	Intel Pro 100/1000	Intel Pro 100/1000
HD	Seagate IDE 80GB	Maxtor SATA 80GB	Seagate IDE 500GB
VMM/OS	Xen 3.1.0/Gentoo Linux kernel 2.6.18	Xen 3.1.0/Gentoo Linux kernel 2.6.18	None/Gentoo Linux kernel 2.6.22.9
Benchmark Software	SPECweb2005 Server Netperf-2.4.2	SPECweb2005 Client	None

In the following experiments, the thresholds of CPU, memory and network overloads are defined as 85% CPU utilization, 150 swap page writes per second, and

80% network utilization, respectively, which differs from Sandpiper in two aspects. First, Sandpiper counts both read and write operations of swap pages while the proposed system counts the swap writes only. This is because that a high frequency of swap reads does not indicate the lack of the physical memory. Second, Sandpiper includes IDC in the network traffic while the proposed system excludes IDC from the network utilization. In our system, only the traffic that passes through the physical NIC is regarded as network traffic. The resource usage information is sent to the RMN node every 10 seconds.

4.2 Performance Evaluation

In this section, we compare the performance results of the proposed IDC aware migration policy with the no-migration policy and the policy used in Sandpiper under different overload situations. Before the performance comparison, we introduce the migration policy of Sandpiper, which uses the following equations for choosing the victim.

$$Volume = \frac{1}{1-cpu} \times \frac{1}{1-mem} \times \frac{1}{1-net}, \quad (5)$$

$$VSR = Volume/Size, \quad (6)$$

where *cpu*, *mem, net* denote the CPU, memory and network utilizations of the given domain, respectively, and *Size* denotes the memory size of the domain. The victim domain is the one with the largest *VSR* value, and has to be migrated to the machine with the lightest load.

4.2.1 CPU overload

Table 4.2: The Configurations of Each Domain in Case of Migrating Isolated Domain

	Workload type	Run-seconds (seconds)	Sessions	Group ID	Physical memory	Initial location

VM1	Support(server)	1800	55	1	256MB	PM1
VM2	Support(database)	1800	55	1	256MB	PM1
VM3	Support(server+database)	1800	50	2	256MB	PM1
VM4	Support(server+database)	1800	60	3	256MB	PM1
VM5	Support(database)	1800	55	5	256MB	PM2
VM6	Support(server+database)	1800	100	4	256MB	PM2
VM7	Support(server)	1800	55	5	256MB	PM3

This experiment demonstrates the use of the domain migration to solve CPU overload. Table 4.2 shows the configurations of the guest domains. Four guest domains (VM1-VM4) run on the physical machine 1 (PM1). VM1 and VM2 are in the same group, meaning that they are the two tiers of the same web service. Therefore, IDC exists between the two VMs. VM5 and VM6 runs on PM2, and VM5 communicates with it group member VM7 residing on PM3. All the VMs run the *support* test of the SPECWeb 2005[31].

Such configuration makes CPU overload on PM1. In this situation, Sandpiper migrates VM1 to PM3, the physical machine with the lightest load, since VM1 has higher CPU utilization and network traffic to VM2. Although Sandpiper can solve the CPU overload, it divides the group 1 on the different machines and thus degrades the performance of that group. With the consideration of IDC, IDC policy migrates VM4 instead. Thus, we can solve CPU overload without degrading the performance. Figure 4.2 shows the response times of different policies. It reveals that migration helps to eliminate the resource overload and achieve a better overall system performance. Compare with Sandpiper, IDC policy reduces the response time by 14.7% in group 1 and 23.3% in group 3, respectively. In group 5, Sandpiper outperforms IDC policy by 6%. This is because the increased performance of VM4 causes heavier network interference with group 5 (i.e., VM 7) on PM3. Moreover, VM4 processes 60 concurrent sessions while VM1 only processes 50 sessions. This also causes heavier network interference with group 5.

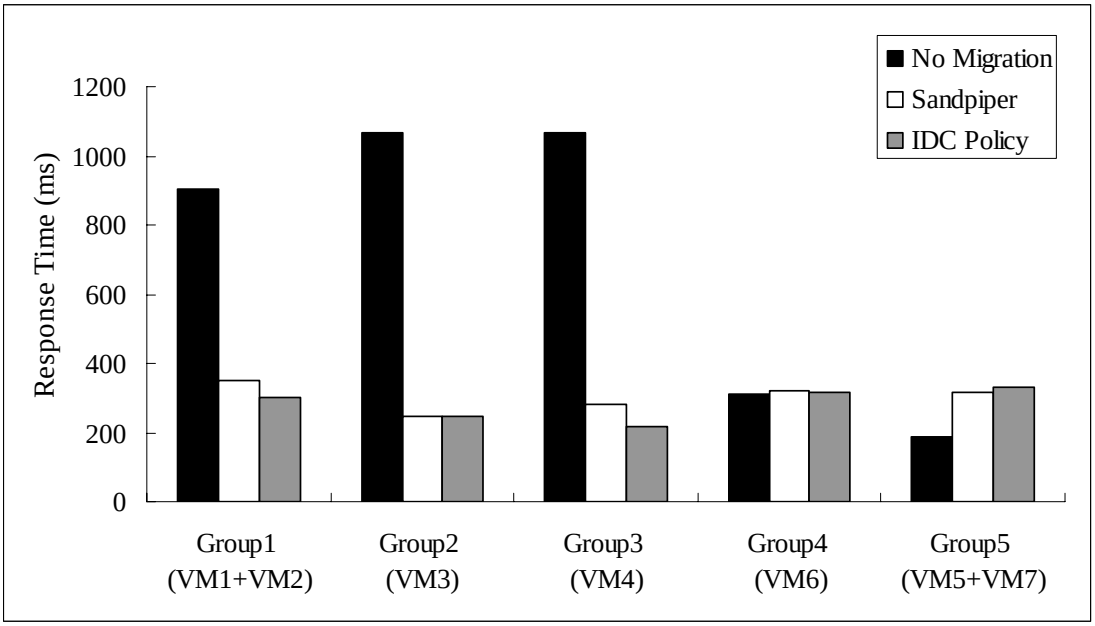

Figure 4.2 : The Response Time in Experiment of Migrating Isolated Domain.

Figure 4.3 and 4.4 show the CPU and network utilizations under different policies. As shown in the figures, less resources are used by IDC policy, as compared to the Sandpiper.

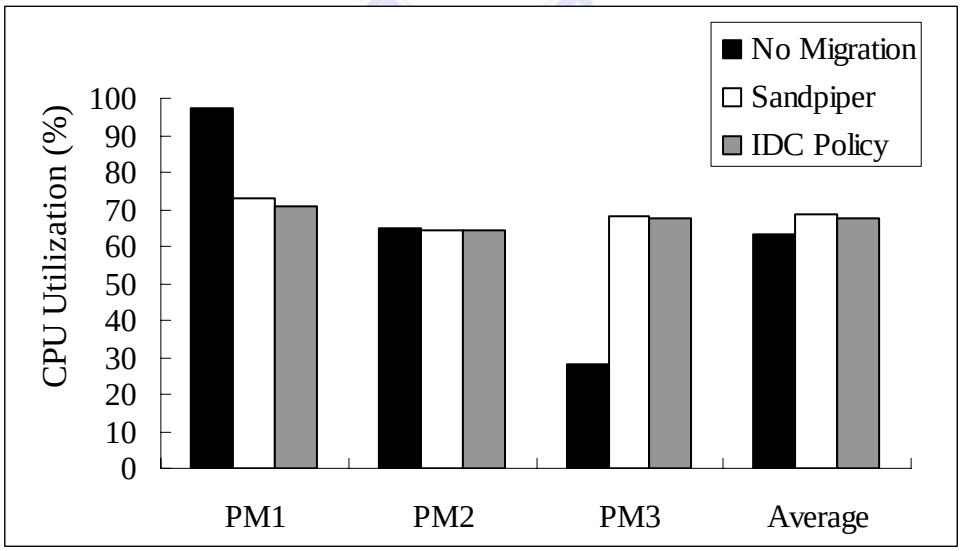

Figure 4.3 : CPU Utilization of Each Physical Machine

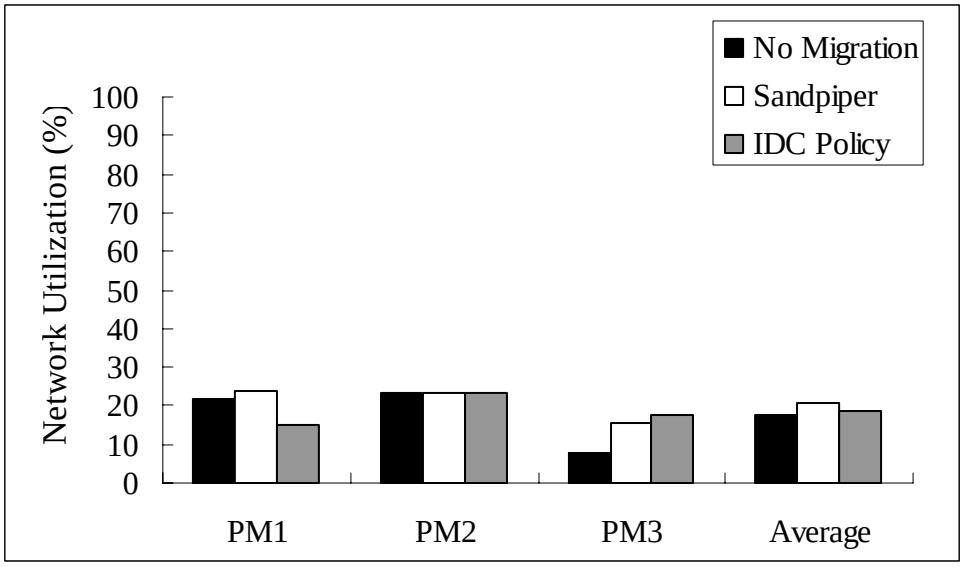

Figure 4.4 : Network Utilization of Each Physical Machine

Table 4.3 : The Configurations of Each Domain in Case of Group Reunion

	Workload type	Run-seconds (seconds)	Sessions	Group ID	Physical memory	Initial location
VM1	Support(server)	600	85	1	256MB	PM1
VM2	Support(database)	600	85	1	256MB	PM1
VM3	Support(server)	600	80	2	256MB	PM1
VM4	Support(database)	600	80	2	256MB	PM2

Next, we demonstrate the capability of group reunion of IDC policy. In this experiment, four guest domains are run on two physical machines. Table 4.3 shows the configurations of each domain. VM1, VM2 and VM3 reside on PM1, and the former two form a group. VM3 communicates with it group member VM4 on PM2.

Such configuration also cause a CPU overload on PM1. Figure 4.5 shows the response time of each group. As shown in the figure, both IDC policy and Sandpiper can solve the CPU overload and effectively reduce the response time by more than 80%. Moreover, compared with Sandpiper, IDC policy reduces the response time of the both groups by 24%. The reason is as follows. Sandpiper selects VM1 as the victim, which divides the group 1 onto different PMs. Instead, IDC policy migrates VM3 to PM2 and thus leads to a performance improvement caused by group reunion.

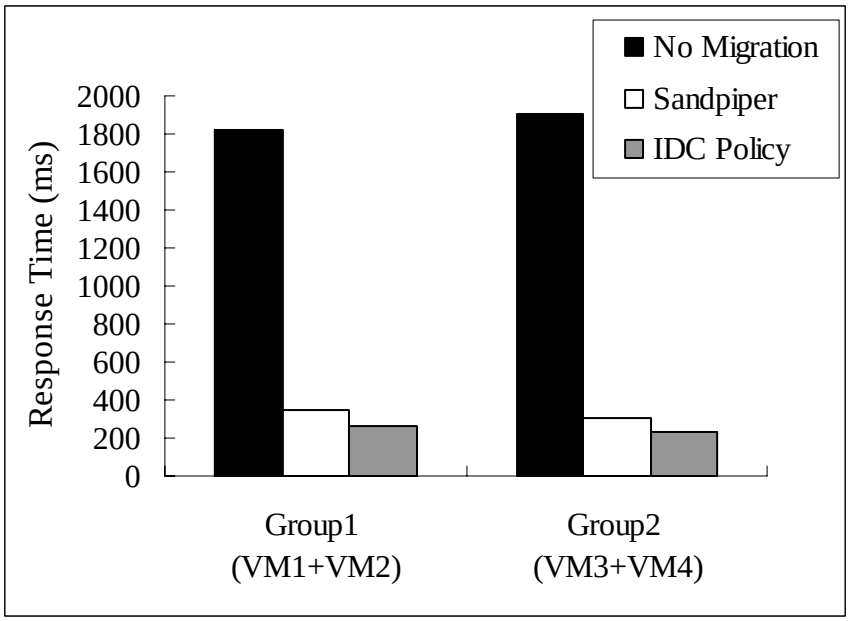

Figure 4.5 : The Response Time in The Experiment of Group Reunion.

From Figure 4.6 and 4.7, we can see that IDC policy can achieve a lower resource consumption since the resources can be used more efficiently.

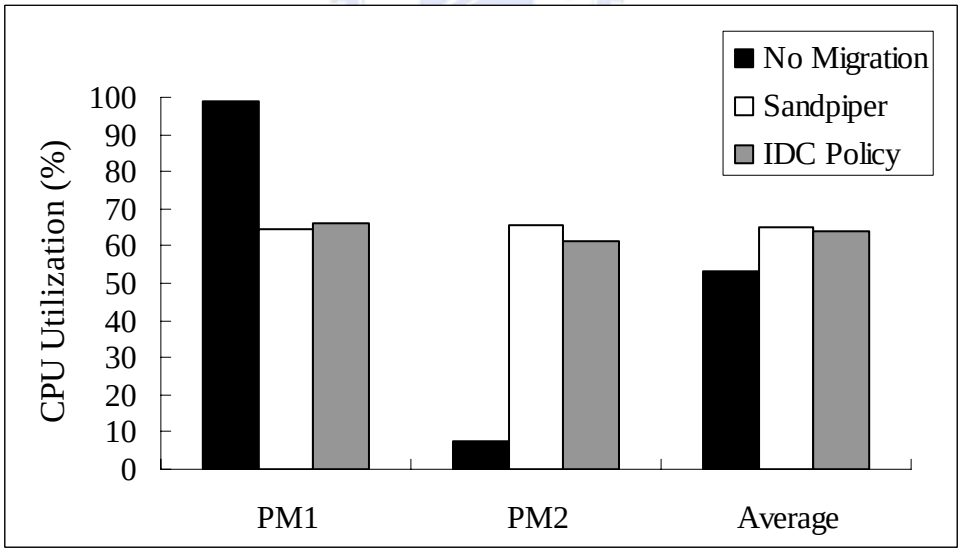

Figure 4.6 : CPU Utilization of Each Physical Machine in The Experiment of Group Reunion.

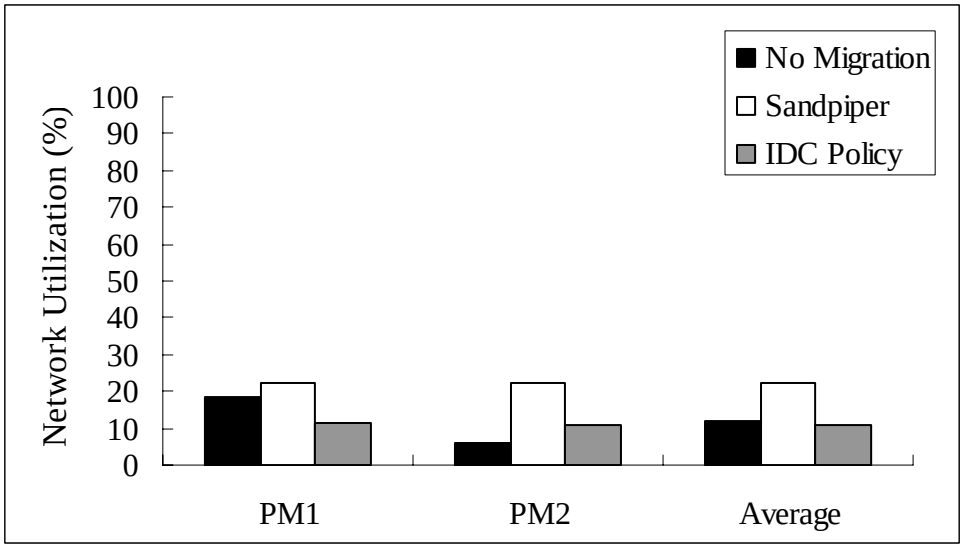

Figure 4.7 : Network Utilization of Each Physical Machine in The Experiment of Group Reunion.

4.2.2 Memory Overload

Table 4.4 : The Configurations of Each Domain (Memory Overload)

	Workload type	Run-seconds (seconds)	Sessions	Group ID	Physical memory	Initial location
VM1	Support(server)	600	70	1	64MB	PM1
VM2	Support(database)	600	70	1	192 MB	PM1
VM3	Support(server)	600	70	2	160MB	PM1
VM4	Support(database)	600	70	2	192 MB	PM2

In this experiment, we show that IDC policy can deal with memory overload by memory allocation and domain migration. Table 4.4 shows the configurations of each domain. The settings are the same as those in the previous experiment except for the session numbers and the physical memory of each domain. In this experiment, both PM1 and PM2 have 1GB physical memory, in which 34MB are reserved for the on-board graphic card and 512MB are reserved for domain 0. We allocate 64MB, 192MB, 160MB and 192MB for VM1 to VM4, respectively. Such configurations lead to memory overload on VM1 and PM1. Figure 4.8 shows the response time under different policies. The values of the no-migration policy are not shown due to that

they are significantly large (i.e., 7.075 seconds for group 1 and 0.905 seconds for group 2). IDC policy outperforms Sandpiper by 22.8% for group 1 and 17.3% for group 2. The reasons are as follows. First, only allocating extra memory for VM1 is of little use since there is little memory left on PM1. Second, Sandpiper do the mechanism of memory allocation until achieve limit of max-memory for each domain or there is no more free physical memory on the source machine. At first memory overload, Sandpiper allocates 32MB physical memory to VM1, however it still can not solve the memory overload. At the next memory overload, Sandpiper moves VM1 to destination, which is PM2 ,and then VM1 can get extra physical memory by memory allocation on PM2; In IDC policy, we have the same behavior at first memory overload. The proposed policy migrates VM3 to PM2 according to IDC at second memory overload and thus VM1 can get the physical memory released by VM3.

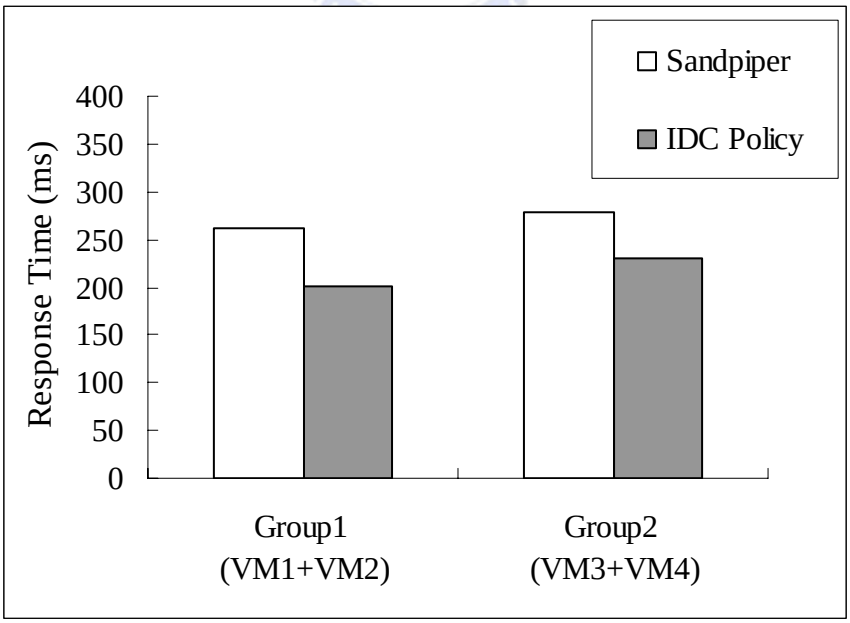

Figure 4.8 : The Response Time (Memory Overload)

Figure 4.9 and 4.10 show CPU and network utilizations under the memory overload. The same as above, IDC policy achieves lower resource utilizations than Sandpiper,

especially in network utilization. The reason is IDC policy transforms network traffic to IDC by migration.

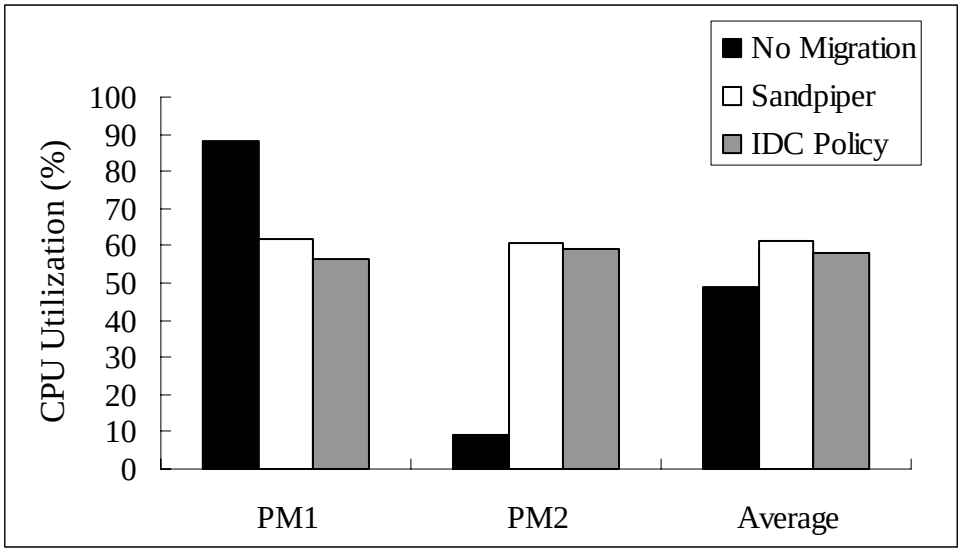

Figure 4.9 : CPU Utilization of Each Physical Machine (Memory Overload)

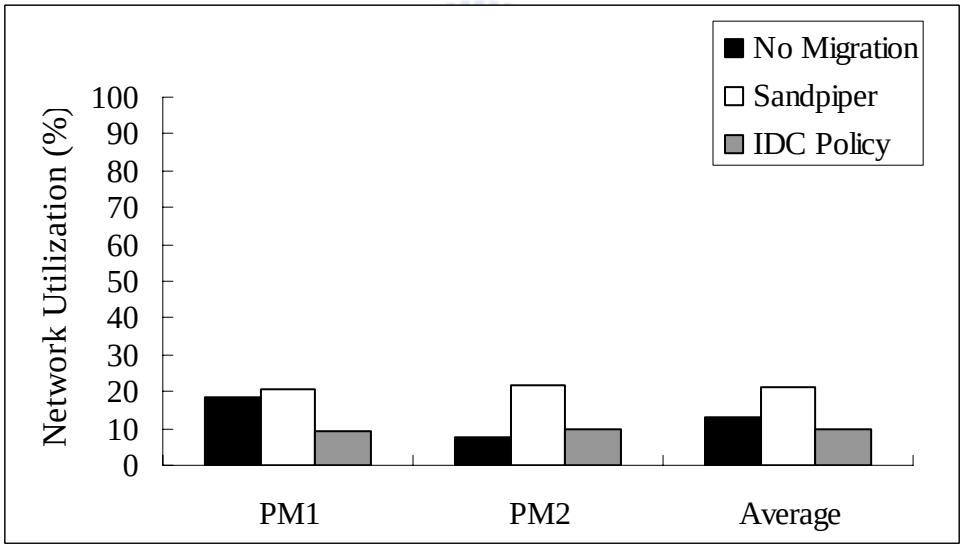

Figure 4.10 : Network Utilization of Each Physical Machine (Memory Overload)

4.2.3 Network overload

Table 4.5 : The Configurations of Each Domain (Network Overload)

	Workload type	Run-seconds (seconds)	Group ID	Physical memory	Initial location
VM1	Netperf client	600	1	256MB	PM1
VM2	Netperf client	600	2	256MB	PM1
VM3	Netperf client	600	3	256MB	PM1

VM4	Netperf server	600	1	256MB	PM2
VM5	Netperf server	600	2	256 MB	PM2
VM6	Netperf server	600	3	256MB	PM3

The network overload situation is caused by the configurations shown in Table 4.5. In this experiment, three two-domain groups are run on four physical machines and all the domains communicate with their group members on different physical machines. The domains residing on PM1 (i.e., VM1, VM2 and VM3) run the netperf client, while the other domains, residing on PM2 and PM3, run the netperf server[31]. In addition to PM1-PM3, an idle machine PM4 is used in this experiment.

Such configuration causes network overload on PM1, and both Sandpiper and our IDC policy choose VM3 as the victim. This is because that VM3 has higher network traffic with its group member VM6, which does not compete with other domains on PM3 for resources. Sandpiper migrates VM3 to the PM4, the machine with the lightest load. In IDC policy, however, we migrate VM3 to PM3 due to the consideration of IDC. Different to Section 4.2.1, we demonstrate that group reunion can also be achieved by choosing the right destination. Figure 4.11 shows the network throughput of each group. We can easily see that migration helps to improve the performance. Comparing with Sandpiper, IDC policy achieves a similar performance for group 1 and 2. For group 3, however, IDC policy outperforms Sandpiper by 102% due to the consideration of IDC. Figure 4.12 shows CPU utilization of each physical machine. Different from previous experiments, IDC policy results in a higher CPU utilization than Sandpiper on the PM3 due to the fact that we migrate VM3 to PM3 instead of PM4. However, IDC policy still has a lower average CPU utilization than Sandpiper. Figure 4.13 show that IDC policy can reduce network traffic and achieve better performance by transforming network traffic into IDC.

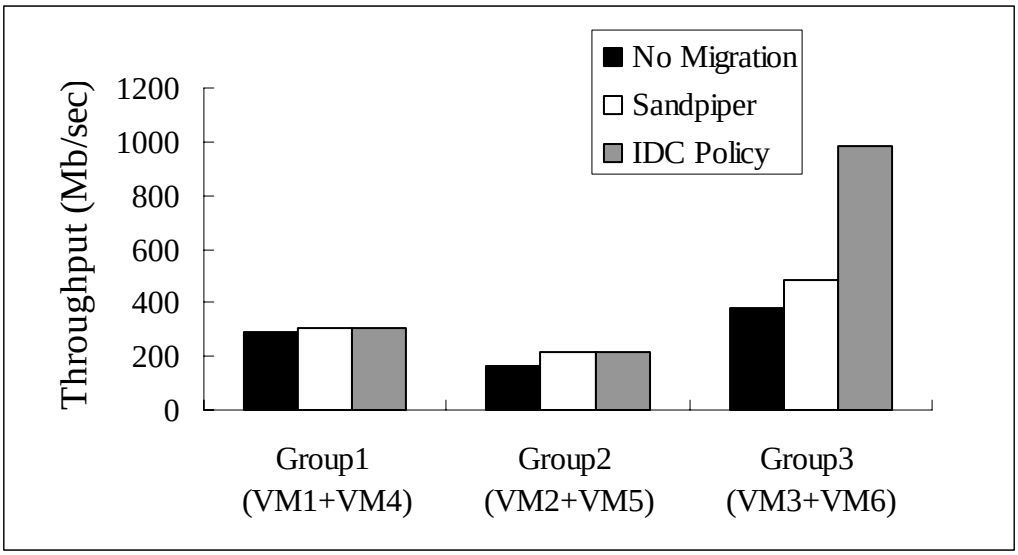

Figure 4.11 : The Performance Results (Network Overload)

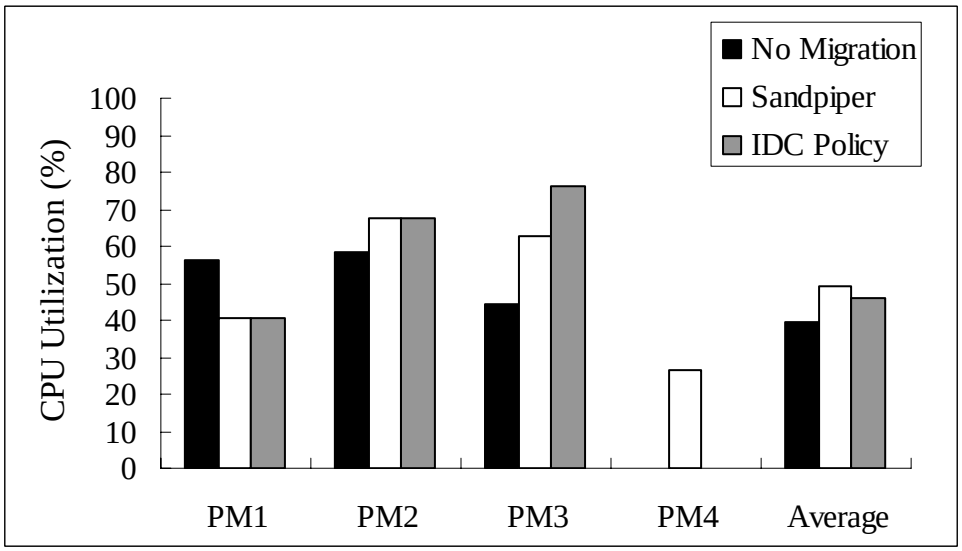

Figure 4.12 : CPU Utilization of Each Physical Machine (Network Overload)

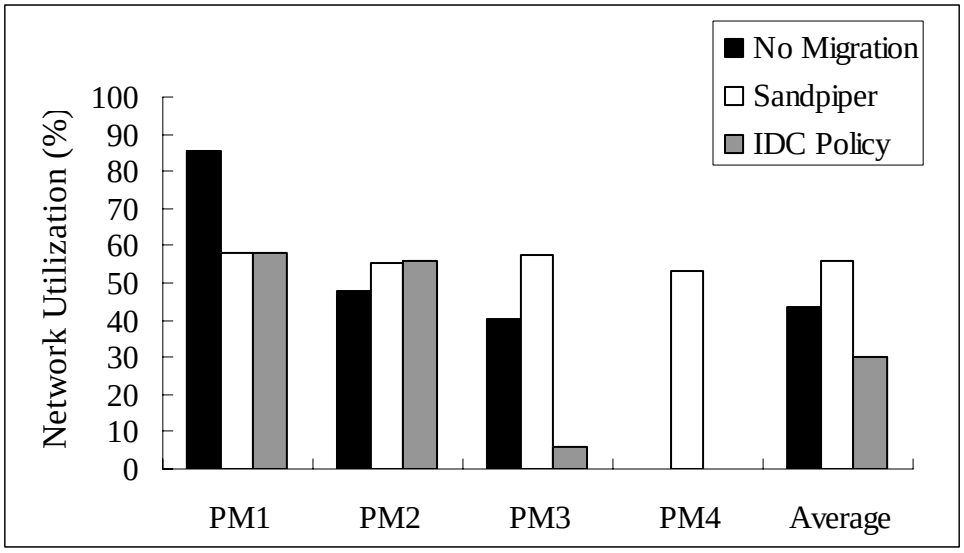

Figure 4.13 : Network Utilization of Each Physical Machine (Network Overload)

4.2.4 Solving multiple types of overloads

Table 4.6 : The Configurations of Each Domain (Multiple Overloads)

	Workload type	Run-seconds (seconds)	Sessions	Group ID	Physical memory	Initial location
VM1	Support(database)	600	70	1	192MB	PM1
VM2	Support(server)	600	70	2	64MB	PM2
VM3	Support(database)	600	70	2	96MB	PM2
VM4	Support(server)	600	70	1	128MB	PM2
VM5	Support(server)	600	65	3	128MB	PM2
VM6	Support(database)	600	65	3	192MB	PM3

We show that our system can solve multiple overloads in this experiment. Table 4.6 shows the configurations of this experiment. We use three physical machines, each of which has 1G DDR2 RAM. Moreover, we do not allocate enough memory for VM2, VM4 and VM5 in order to cause memory overload. We create several memory and CPU overloads on the PM2 and only two CPU overloads of them need to be solved by migration. Figure 4.14 shows the CPU utilizations, and Figure 4.14 shows the frequencies of the swap writes. At 80 seconds, there is a large amount of swap writes on PM2 (marked as A in Figure 4.15). Then, RMN allocates extra memory to

VM2 to eliminate the memory overload. At 130 seconds, the CPU loads of each VM are increasing. Then, there is a CPU overload on PM2 and solved by migrating VM4 to PM1. This migration successfully solves CPU overload and reclaim enough physical memory in PM2. Next, there are three times of memory allocations for VM2 (marked as C, D and E in Figure 4.15) to solve the continue CPU and memory overloads in the same time. Notice that, when CPU and memory overloads occur in the same time, we will solve memory overload first because reducing the frequent swapping can sometimes decreases CPU utilization. After 170 seconds (marked as E in Figure 4.15), we eliminate the memory overload that caused by VM2, but there is still CPU overload. At 210 seconds (see Figure 4.14), RMN tries to do migration and migrates VM5 to PM3 to achieve the group reunion. Then, the memory usages of group 1 and 3 are increasing, PM1 and PM3 allocate extra physical memory to VM4 and VM5 to eliminate memory overloads (marked as G and H in Figure 4.15).

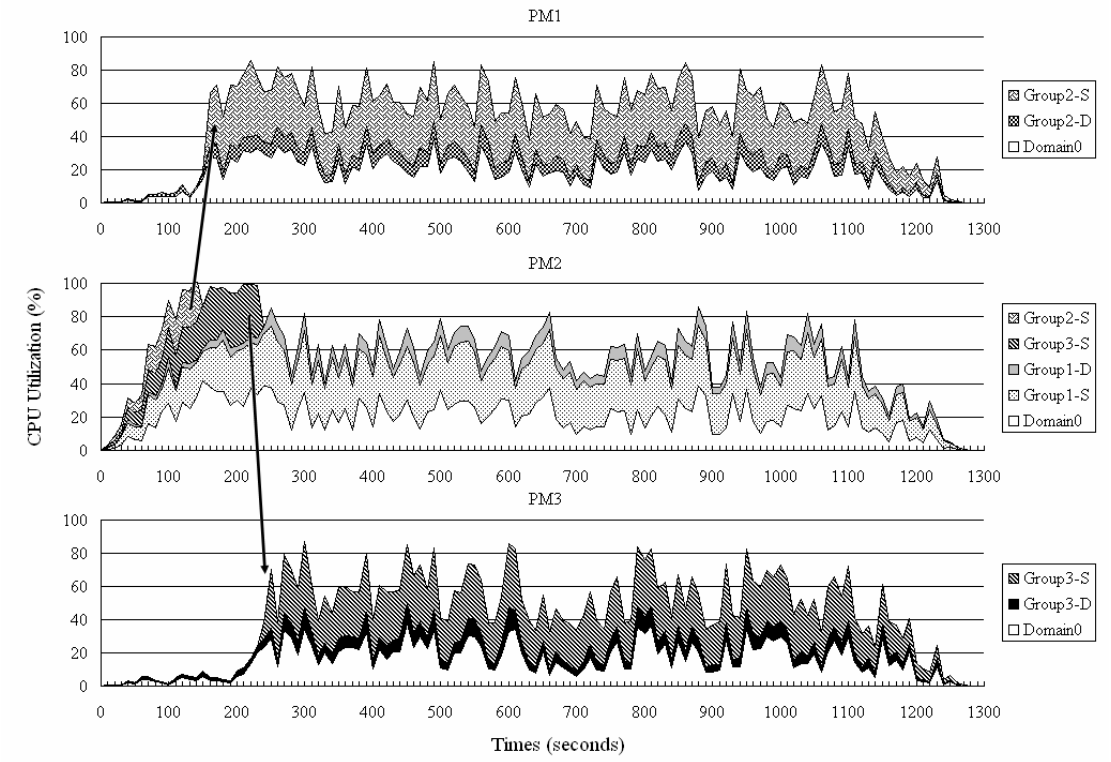

Figure 4.14 : A Series of Migrations for Solving Multiple Overloads.

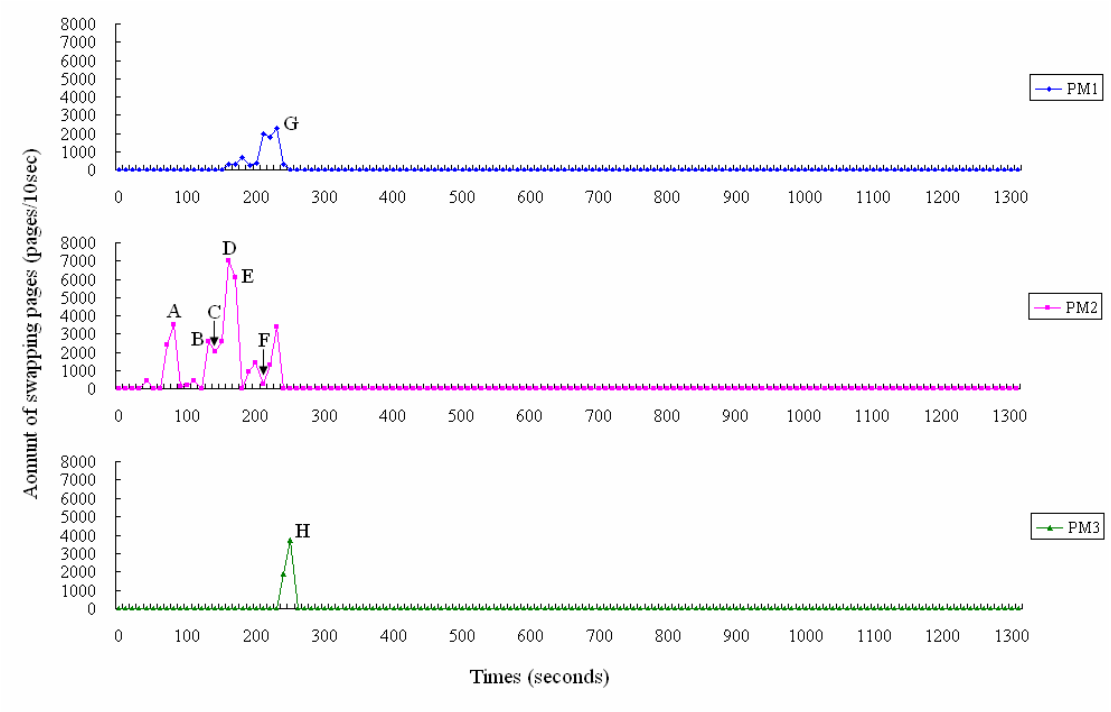

Figure 4.15 : Amount of Swap-Write Pages Per 10 Seconds in Each PM.

4.3 Overhead Evaluation

We evaluate the CPU and memory overhead of the proposed system in this section. We use iperf-2.0.2 [32] to evaluate the CPU overhead with various throughputs and numbers of VMs. Figure 4.16 shows the CPU utilizations under different throughputs. In this experiment, two iperf clients running on two VMs on the target machine connect to two iperf servers running on two VMs on a different physical machine. Figure 4.17 shows the CPU utilizations under different numbers of VMs. In this experiment, we create different numbers of VMs form 2 to 10 on the target machine, and each of them runs the iperf client. The iperf servers run on top of two physical machines, and the iperf throughput is fixed as 600Mbps. Both figures show that the monitoring mechanism for supporting the IDC aware migration policy causes little CPU overhead.

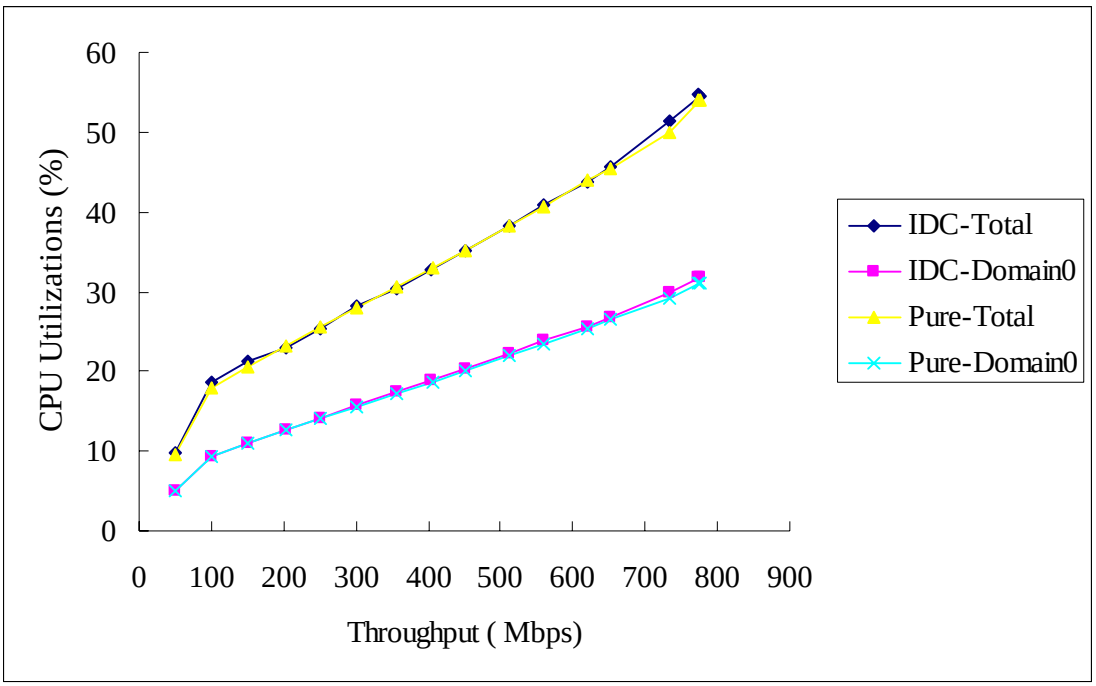

Figure 4.16 : CPU Utilizations with Different Throughputs

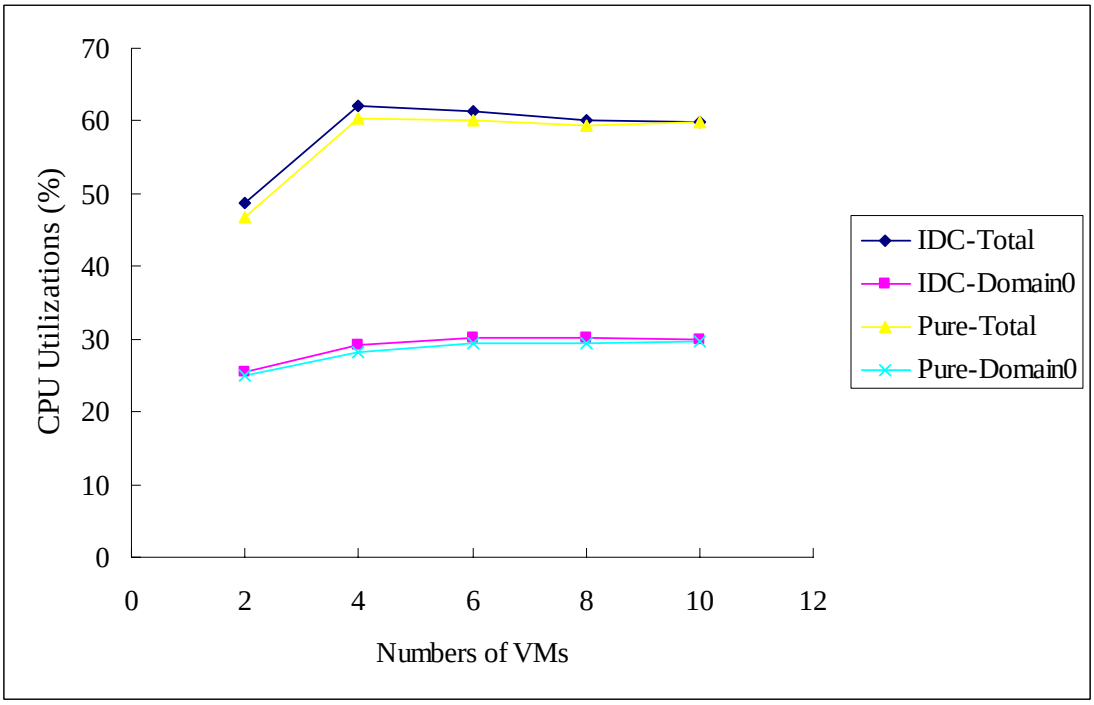

Figure 4.17 : CPU Utilizations with Different Numbers of VMs

Next, we analyze the memory cost of our system. We define CD_i as the number of domains communicating with domain i. Thus, the extra memory cost of the resource usage information for domain i can be represented as follows:

$$\text{mem\_cost}(i) = 100 + CD_i * 48 \text{ Bytes}, \quad (7)$$

where 100 and 48 are the sizes of the *domain_info* and *net_info*, respectively. Therefore, a cluster with N virtual machines, the whole memory cost of the resource usage information would be

$$100 * + 48 * \sum_{i=1}^{N} CDi \, . \quad (8)$$

Finally, we show the implementation cost. The system is implemented with less than 3790 lines of C code. We modify/add 1130 lines in the kernel of domain0, 1970 lines in RMN and 690 lines in migration monitor and profile engine.

Chapter 5 Conclusions and Future Works

5.1 Conclusions

We present an automatic load management system with IDC aware migration policy. It can solve system overloads immediately and effectively and the implementation overhead of our system to support IDC aware migration policy is insignificant. Compare to other migration policy, IDC aware migration policy can get better performance by achieving group reunion with lower CPU cost.

5.2 Future Works

In this thesis, we present IDC aware migration policy, but we use a simple predicted algorithm of Nds. However, it is not a precise way to evaluate the value of Nds, and so is Id. Maybe we can predict it according to disk I/O, CPU utilization, memory size and dependence of network traffic of other domains. In the policy of choosing destination, we just check if the remainder resource on the destination is enough to contain the victim domain now. But we do not consider that the resource usage of victim will be changed after migration. Without the well prediction of resource usage, it maybe causes the destination does not contain the victim and want to do migration because another system overload is happened.

Moreover, K. Kim *et al.*[26] and X. Zhang *et al.*[28] present different ways to increase the performance of Inter-Domain Communication. With the implementation of their works, our IDC aware migration policy should be more significant.

References

[1] M. M. Theimer, K. A. L., and D. R. Cheriton, “Preemptable Remote Execution Facilities for the V-System”, In Proceedings of the 10th ACM Symposium on Operating Systems Principles, pp. 2-12, Dec. 1985.

[2] M. Nelson, B. Lim, and G. Hutchins. “Fast Transparent Migration for Virtual Machines”, In Proceedings of the USENIX 2005 Annual Technical Conference, pp. 391-394, Apr. 2005.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A.Warfield, “Live Migration of Virtual Machines”, In Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, pp. 273-286, May, 2005.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization”, In Proceedings of the 19th ACM symposium on Operating Systems Principles, pp. 164-177, Oct. 2003.

[5] M. L. Powell and B.P. Miller, “Process Migration in DEMOS/MP”, In Proceedings of the 9th Symposium on Operating Systems Principles, pp. 110–119, Oct. 1983.

[6] R. Finkel, "The Arachne Kernel", Technical Report TR-380, University of Wisconsin, Apr. 1980.

[7] R. F. Rashid and G.G. Robertson, "Accent: A Communication Oriented Network Operating System Kernel", In Proceedings of the 8th Symposium on Operating System Principles, pages 64–75, Dec. 1981,

[8] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In 1st Workshop on Operating System and Architectural Support for the On-Demand IT Infrastructure, Oct. 2004

[9] M. Litzkow, M. Livny and M. Mutka, “Condor - A Hunter of Idle Workstations”, In Proceedings of the 8th International Conference on Distributed Computing

Systems, pp. 104-111, 1988.

[10] A. Barak, S. Guday and R. G. Wheeler, “The MOSIX Distributed Operating System: Load Balancing for UNIX”, LNCS 672, Springer, Berlin, 1993.

[11] J. Casas, D. Clark, R. Konoru, S. Otto, R. Prouty and J. Walpole, “MPVM: A Migration Transparent Version of PVM”, Oregon Graduate Institute School of Science & Engineering, Technical Report: CES 95-002, Feb. 1995.

[12] F. Douglas “Transparent Process Migration in the Sprite Operating System”, (PhD Thesis, University of California, Berkeley), Sep. 1990.

[13] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, “Process migration”, ACM Computing Surveys, pp. 241-299, Sep. 2000.

[14] D. Milojicic, P. Giese, and W. Zint , “Experiences with Load Distribution on Top of the Mach Microkernel”, In Proceedings of the USENIX Symposium on Experiences with Distributed and Multiprocessor Systems, Sep. 1993.

[15] P. Krueger and M. Livny, “A Comparison of Preemptive and Non-Preemptive Load Balancing”, In Proceedings of the 8th International Conference on Distributed Computing Systems, pp. 123-130, Jun. 1988.

[16] V. S. Sunderam, G. A. Geist, J. Dongarra and R. Manchek, “The PVM concurrent computing system: Evolution, experiences and trends”, Parallel Computing, pp. 531-545, Apr. 1994.

[17] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam and M. Rosenblum, “Optimizing the migration of virtual computers”, In Proc. of the 5th Symposium on Operating Systems Design and Implementation, pp. 377-390, Dec. 2002.

[18] D. Gupta, R. Gardner and L. Cherkasova, “Xenmon: Qos monitoring and performance profiling tool”, Technical Report HPL-2005-187, HP Labs, Oct. 2005.

[19] J. G. Hansen and E. Jul, “Self-migration of operating systems”, In Proceedings of the 11th ACM SIGOPS European Workshop, pp. 126.-130, Sep. 2004.

[20] R. Bradford, E. Kotsovinos, A. Feldmann and H. Schioberg, “Live wide-area migration of virtual machines including local persistent state”, In Proceedings of the 3rd international conference on Virtual Execution Environments, pp. 169–179, Jun, 2007.

[21] S. Jones, A. Arpaci-Dusseau and R. Arpaci-Dusseau, “Geiger: Monitoring the buffer cache in a virtual machine environment”, In Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 13–23, Oct. 2006.

[22] D. Menasce and M. Bennani, “Autonomic Virtualized Environments”, In Proceedings of International Conference on Autonomic and Autonomous System, Jul. 2006.

[23] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen, “Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure”, in Proceedings of the 2006 IEEE International Conference on Autonomic Computing, pp. 5-14, Jun. 2006.

[24] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, “Black-box and Gray-box Strategies for Virtual Machine Migration”, In Proceedings the 4th Symposium on Networked Systems Design and Implementation, pp. 229-242, Apr. 2007.

[25] C. Hyser, B. McKee, R. Gradner and B. J. Watson, “Autonomic Virtual Machine Placement in the Data Center”, HP Laboratories, HPL-2007-189, Feb. 2008.

[26] K. Kim, C. Kim, S. I. Jung, H. S. Shin and J.S. Kim, “Inter-domain Socket Communication Supporting High Performance and Full Binary Compatiblity on Xen”, In Proceedings of the 4th ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments, pp. 11-20, Mar. 2008.

[27] J, Wang, K. L. Wright and K. Gopalan,“XenLoop: A transparent High Performance Inter-VM Network Loopback”, In Proceedings of the 17th international symposium on High performance distributed computing, pp. 109-118, Jun. 2008.

[28] X. Zhang, S. McIntosh, P. Rohatgi and J. L. Griffin, “XenSocket: A

High-Throughput Interdomain Transport for Virtual Machine“, In Proceedings of Middleware, Aug. 2007.

[29] M. Mirza, J. Sommers, P. Barford, and X. Zhu, “A Machine Learning Approach to TCP Throughput Prediction,” In Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, June. 2007.

[30] Netperf-2.4.2 Available at http://www.netperf.org/netperf/

[31] SpecWeb2005 Available at http://www.spec.org/web2005/

[32] Iperf-2.0.2 Available at http://dast.nlanr.net/Projects/iperf