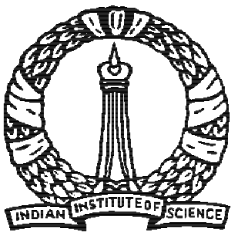
Adaptive Virtual Machine Placement supporting performance SLAs

A Project Report
submitted in partial fulfilment of the
requirements for the Degree of

Master of Technology
in
Computational Science

by

Ankit Anand



Super Computer Education and Research Centre
Indian Institute of Science
Bangalore – 560 012 (INDIA)

JUNE 2013

Dedicated

To

My Late Grandmother

Acknowledgements

It is not often that you get chance, space and time to acknowledge people who have mattered or played a part in your life. It is not because you are too busy in your schedule but because of fact that by the time you realize how your life would have been without the interference of these people, you have moved far too ahead in your life. The impact or imprint they have left comes too late to be visible to you and the world.

Hence, I hereby feel a privileged lot and thank the Almighty for giving me a chance to recapitulate and show gratitude to the people for shaping my life in two of the most struggling as well as learning years.

First of all, I would like to thank my parents for believing in me and encouraging me to pursue a masters degree. They have been with me throughout my career and without them, I would not have been where I am.

More importantly, this project started under the able guidance of Prof S.K.Nandy and Dr. J.Lakshmi. They have been a guiding light throughout the last year in giving this work its present shape and encouraging me to learn new things. A special thanks to Lakshmi Mam for all the support, guidance and belief which cannot be expressed in words.

A special thanks to our Chairman Prof. R. Govindarajan for making SERC a scholarly place and for all the help throughout the course duration. The first year courses taught by Prof. Sathish Vadhiyar, Prof. Mathew Jacob , Prof. Sathish Govindarajan , Prof. Atanu Mohanty deserve a special mention for laying the foundation of my career in technical field.

A special thanks to Mohit Dhingra and Anurag Murty for all the help, technical discussions, fights, cycle trips and the MAD(Murty-Anand-Dhingra)ness which we created here without which Bangalore would have had a bland taste.

The environment created in lab and around by Mohit, Siva, Nitisha and Aakriti is worth mentioning without which our lab would not have been so friendly as it is now. Also, the family of my SERC batch-mates deserve a special place in my heart forever.

IISc friends Akash Gupta and Priyanka Singla have been wonderful throughout these two years and mess chats and roaming in and around this beautiful campus was great with both of you.

A special thanks to Vaibhav, Aravind, Rooparam and all my seniors of SERC for guiding us how to survive in this department and being in constant touch with us whenever we required any help. The farewell given by our junior batch to us and the interactions with them throughout the last year would be unforgettable to us, special thanks to all of them.

Also, I would like to thank Make A Difference (MAD), Bangalore chapter for giving me the privilege to be a part of it and providing me beautiful breaks in between my work.

I would also like to thank Malika mam, Shekhar sir, SERC and Cadlab staff for providing me all the resources whenever I needed and making this place an ideal place to work at.

Finally, I express my gratitude to Jamshedji Nusserwanji Tata for having a vision to create this beautiful temple of learning and thank the Almighty for giving me a chance to work at this serene beautiful institute.

Abstract

One of the important problems in data centers, supporting cloud computing model, is the placement of virtual machines (VMs) requiring varying resources over physical hosts of fixed capacity. This problem has been dealt well in the literature as a multi-dimensional bin-packing problem. Almost all of these solutions overlook the concern of provisioning that supports performance Service Level Agreements (SLAs). It is a well-reported issue that virtualization overheads manifest as extra resource usage by the hypervisor, particularly in the case of I/O workloads. Provisioning algorithms must consider these overheads, which are virtualization architecture specific, while resolving VM placements that need to honour SLA guarantees. Also, VM placement decisions are dynamic in nature that need to be taken whenever a new workload arrives or an existing workload changes due to elastic behaviour. Prevalent technologies handle the elastic needs of a workload using VM migration. However, VM migration is a non-trivial and expensive operation, particularly for I/O workloads. In this paper we propose VM placement approaches that consider performance SLAs and VM migration costs while optimizing VM placements over a minimal set of physical hosts. We capture these constraints in the classical Integer Linear Program (ILP) model and solve for the minimal set of physical hosts. This approach is resource intensive and as the number of VMs versus physical hosts increases, the solution time also increases and is not very useful in situations where real-time scheduling decisions are needed. To reduce the solution time we recast the problem into First Fit Decreasing (FFD) algorithm. Experimental results demonstrate that FFD yields near optimal solution in time scales that can aid real-time scheduling decisions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The end of 20th century and the beginning of 21st century can be marked as a birth of computing era. During this period, computing and digital solutions expanded their place from universities and research labs to the hands of common man. Moreover, the phenomenon grew exponentially in developing world where enterprises digitized most of their business requirements from internal management tasks to core businesses. This expansion was particularly accelerated by the twin developments of Desktop computing and World Wide Web (WWW). Most of these tasks are executed by applications hosted on dedicated servers.

Availability of cheap hardware and computing facilities led to sprawling of these servers across datacenters. The same not only increased the overhead costs of air-conditioning(to vent out waste heat) and maintenance facilities but also required large space in datacenters. Interestingly, on the contrary most of the applications utilize less than 20 percent of the server resources allocated to them [1]. One of the major reason for this low overall utilization is majority of the enterprise applications are dynamic as well as resource centric. By dynamism, we mean workloads and hence resource requirements vary according to the time of day and season. For instance, a bank web-server would be experiencing high request rates during day hours compared to night hours. The resource centric nature implies that applications utilize one resource completely while other resources remain idle. For example, in case of I/O bound

applications, CPU resources remain idle while in case of CPU bound applications, I/O resources are not utilized fully. The other major reason for under utilization is over provisioning of resources that is applications are provisioned resources for peak workloads although this peak workload may occur for a very short duration of time. The same leads to idle resources for the remaining majority of time durations.

The simple solution thought to solve this problem was sharing resources between different applications. The same concept is used in case of processes in multi-programming system where CPU is context switched to another process while a process is waiting for a slow I/O operation. But in this case, instead of hosting an application as a process, we may require a separate server for each application so as to ensure complete isolation for the application. One way of providing isolation to these applications is by separating the hardware layer from the O/S layer, the technique popularly called as virtualization. In this case, O/S runs not directly on physical hardware but on a abstraction of it called virtualized hardware.

1.2 Virtualization

Virtualization is not a new concept. It dates back to 1960s when large mainframe systems were in use. These big mainframe systems were largely underutilized. Hence, virtualization was invented to run multiple applications on these machines by logically partitioning the hardware resources of the system and thus improving the overall utilization of systems. Also, hardware was continuously evolving at that time, so virtualization was used as a tool to support legacy applications which were not compatible with new hardware.

But with passage of time, due to decrease in prices of hardware and the advent of desktop computing, this field became dormant for two-three decades. But currently, there has been a resurgence of interest in virtualization due to the emergence of new concepts of utility and cloud computing. Before defining these new technologies, we give a brief over view of virtualization, how it emerged and what are challenges in virtualization of existing processor architectures.

1.2.1 Definition and Basic Concepts

A virtual machine is an abstraction of the underlying physical hardware. The guest operating system runs in this virtual machine and the applications in turn runs in guest O/S. The layer which provides this abstraction is called Virtual Machine Monitor (VMM) or hypervisor and thus it manages the execution of all VMs.

Formally, the definition of VMM was provided by Popek and Goldberg in a classical paper on virtualization [2]. According to them, a VMM is a software which has the following characteristics:

1. VMM provides an identical environment to the application as provided by a real machine.
2. The applications runs in a VM with minimal loss of performance.
3. VMM has complete control of all physical resources i.e no VM can use the resources of other VM without intervention of VMM.

To ensure these characteristics of performance and control, we need the model of direct execution [3]. The model of direct execution means most of the instructions execute directly on the CPU. Only those instructions which change or effect the resource control, should be trapped to hypervisor which then takes necessary action to perform the required operation. This can be implemented by executing Guest (O/S running in VM)'s privileged (kernel code) and unprivileged code in unprivileged mode of CPU and to run hypervisor in privileged state. Whenever an instruction effects control of resources, the processor goes to privileged mode and hypervisor then executes the required instruction.

1.2.2 Problem with Current Architectures

Unfortunately, the modern processor architectures including the x-86 architectures are not virtualizable. In x-86, the POPF instruction is used to set/clear the interrupt flag, but when this instruction (of guest O/S) executes in unprivileged mode, it does not trap and just ignores the interrupt register. So, running this instruction under direct execution model would not effect the interrupt flag and the code would not have desired effect. Also, certain instructions are there which directly read the mode of processor. Analogous to previous case, if these instructions (in guest O/S) are not trapped by hypervisor, wrong VM CPU state is recorded. Hence, to

eliminate these limitations of traditional x-86 architecture and make it virtualizable, different types of hypervisors and virtualization technologies emerged. The same are classified in the next sub-section.

1.2.3 Types of Virtualization Technologies

The prevalent Virtualization technologies can be classified into three different types [3] :

1. **Paravirtualization**: In this case, the guest OS code is modified so that the instructions which are not virtualizable are replaced with solutions which do not directly effect the CPU and other resources. The advantage gained in this case is less trapping to hypervisor for execution of privileged instructions and hence gain of performance. The downside of this is, since we are modifying guest O/S, the legacy O/S may not run and difficulty of porting guest O/S to a new architecture. Xen hypervisor is a popular example of this class.

2. **Emulation**: In this solution, guest O/S is unmodified and all virtualizable instructions are executed directly on hardware. The non-virtualizable instructions are first translated using a binary translator and then run on CPU. In this case, we translate instructions while they are executing, so, this technique incurs significant overhead and effects performance. VMware Workstation is an example of hypervisor of this kind.

3. **Hardware Assisted Virtualization**: To reduce the impact on performance and eliminating the need of modifying guest O/S, Intel and AMD have introduced hardware changes in x-86 architecture to support virtualization. Many hypervisors have emerged recently which uses these virtualization extensions to hardware. A new mode called guest mode was added to CPU which in itself has unprivileged and privileged modes in addition to existing privileged and unprivileged modes. These hypervisors not only give performance benefits compared to Binary translation but also guest O/S can be run unmodified. KVM is a popular hypervisor belonging to this class.

1.3 Cloud Computing and Utility Computing

Using hypervisors of any of the above given types, Virtualization is a key enabler in cloud and utility computing today. Utility computing is a business model where all the computing resources namely, physical servers, network and software can be used as a service/utility and can be procured on demand. This eliminated the need for purchasing computing resources and all the computing needs (both hardware and software) can be rented whenever required.

The underlying model on which utility computing is designed is Cloud Computing. The term 'Cloud Computing' was first used academically by a management professor Ramnath Chelappa in his talk on "Intermediaries in Cloud Computing - A new Computing Paradigm" in 1997 [1]. He explains this as "computing paradigm where the boundaries of computing will be determined by economic rationale rather than technical limits alone". As per National Institute of Standards and Technology (NIST), USA , Cloud Computing is defined as follows

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

It is a broader concept than that of utility computing where it includes sharing of resources with in an organization as well, a term called as "Private Cloud". It basically includes three paradigms:

1. Infrastructure as a Service(IaaS) where infrastructure resources like computer servers, storage and network resources are provided as a service and can be provisioned on demand .e.g Amazon's EC2 as a service etc.

2. Platform as a Service(PaaS) where platforms are provided as a service. These include operating system, programming language platform, database etc.Examples of this include Google App Engine, Microsoft Azure etc.

3. Software as a Service(SaaS) where software is provided as a service to end user. The examples of same include Google Docs/Drive, Microsoft Office etc.

[1] http://www.bus.emory.edu/ram/

In this work, we concentrate on IaaS where hardware resources viz. physical machines(hosts) are used as a service. A typical Cloud computing model is shown in Figure 1.1. As shown, the users access the physical resources with an abstraction of Cloud in between. This layer of cloud provides ubiquitous, convenient and on demand network access of physical resources. It also shows the layer of hypervisor between VMs and hardware. From now onwards, we will use the term physical machine and host interchangeably in this thesis.

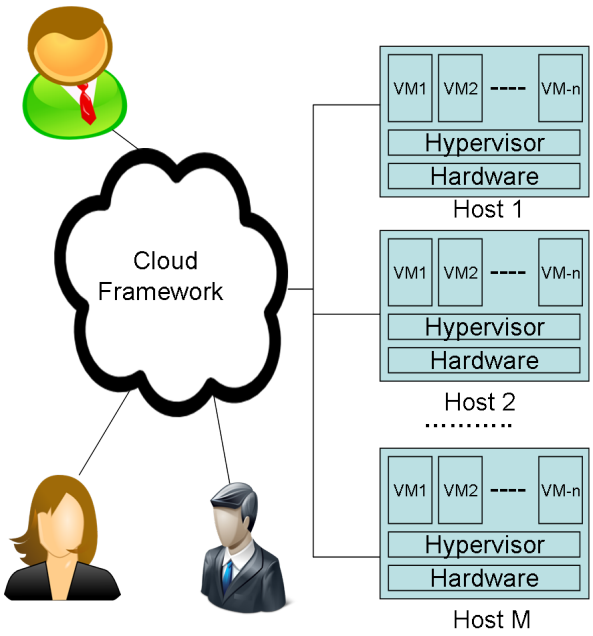


Figure 1.1: Typical Cloud Framework

An important problem in the Cloud Environments while managing clouds is placement of Virtual Machines (VMs) over hosts. This work highlights some important considerations missing in existing Virtual Machine Placement Problem (VMPP) and provide novel solutions to incorporate the same. The next section defines the problem statement in detail.

1.4 Problem Statement

The placement of Virtual Machines over physical hosts is an important activity in all datacenters and a vital component of any cloud management framework. While provisioning resources to VMs, a cloud provider wants to maximize resource utilization by placing VMs over a minimal

set of physical hosts. The problem is a variant of multidimensional bin-packing in this setting where items(VMs) of various sizes (resource vector in VM case) have to be packed in bins (hosts) of fixed capacity (capacity vector of host).

On the other hand, from a user's perspective, there should be minimal degradation of performance when applications are relocated to clouds. But as discussed in [4] and [5], when applications are executing in virtualized scenarios, virtualization overheads manifest as extra resource usage by hypervisor (or virtual machine monitor [2]) and this extra usage depends on virtualization technology used (paravirtualization or full virtualization). Particularly in case of I/O applications which are predominant in clouds, this overhead is significant and degrades performance of applications. Therefore, to honour performance Service Level Agreements (SLAs), this overhead must be captured in the problem definition of VM placement.

Also, the Virtual Machine Placement Problem (VMPP) is dynamic in nature i.e after the initial placement, not only new VMs with varying resource requirements arrive and existing VMs leave but also the resource requirements for VMs change continuously. So, after doing initial placement, scheduling decisions need to be taken periodically at each scheduling cycle. And VMPP algorithms need to be invoked at each scheduling cycle catering to these elastic requirements of workloads. The existing technologies handle these elastic needs by migration of VMs from one host to other. Also, the VM migration is proposed as a solution to meet SLAs in case of changing resource requirements [6]. However, migration in itself is a costly operation. It not only degrades the performance of applications for a certain time (during migration) but also introduces additional network and other overheads while migrating [7]. This migration overhead depends on type of application and resources used by it, for example, applications having more memory and I/O footprint would have higher migration overheads compared to pure CPU intensive applications. To reduce these overheads and to honour SLAs, the provisioning algorithms must consider the previous placement of VMs for minimizing migration overhead and number of hosts used.

In this work, we introduce the consideration of virtualization overhead in VMPP , propose a metric for quantification of migration overhead and then modify the VMPP to minimize this migration overhead. Also, we intend to solve VMPP in practical scenarios where scheduling

decisions have to be taken in real time.

1.5 Thesis Organization

The aim of this thesis is to study Virtual Machine Placement problem with respect to its effect on performance SLAs. It proposes the consideration of virtualization overhead in algorithms to preserve performance and making VMPP algorithms migration aware so that there is minimum migration overhead of existing VMs. Also, all these modifications are also proposed considering the fact that real time scheduling decisions can be taken. The rest of the thesis is organized as follows:

Chapter 2 deals with the related work and literature in the field of VM placement problem. We provide account of different techniques varying from exact solution methods to metaheuristics to simple first fit decreasing methods. Algorithms used in contemporary cloud management tools is also provided in this chapter.

Chapter 3 gives a simple case study illustrating why virtualization overhead should be considered . It illustrates the effect on performance using a web-server application hosted in Virtual Machine. This chapter also deals with issue of dynamic resource requirement and quantize the migration overhead by providing a metric for the same.

Chapter 4 provides the methodology for handling VMPP. It gives various algorithms incorporating the effect on performance SLAs by considering migration and virtualization overhead. We initially give the Integer Linear Programming formulation for the same. But to obtain solution in real time scenarios, we introduce FFD heuristics. We also compare different heuristics theoretically in the same.

Chapter 5 mainly deals with the monitoring tools developed to measure the Virtualization overhead and other resources in case of KVM hypervisor. It first explains the basic architecture of KVM with the help of a network application. The basic design of Linux Perf is then explained

and how the same can be used to extract CPU usage of hypervisor on behalf of each virtual machine. The chapter also describes different monitoring tools used in our study to measure other resources used in each VM like memory, network B/W ,Disk I/O etc. for KVM hypervisor. Also it analyses the performance of web-server and mail server applications in virtualized settings.

Chapter 6 discusses about the experimental setup and the results obtained by running different algorithms described in Chapter 4. It explains the different workloads we used in our experiments and what kind of distribution on the same was employed. It also gives the comparison of different algorithms in terms of minimizing the number of hosts used and number of migrations.

Chapter 7 gives the conclusions and the future work. It provides major contributions of this work and major gains obtained by strategies discussed. Then, it gives some future directions where this work could be extended and some untouched aspects in this field.

1.6 Summary

The chapter describes the resurgence of virtualization and how it has become a key enabler in cloud computing. It further explains the basics of cloud computing models namely, IaaS, PaaS and SaaS. The Virtual Machine Placement Problem is then elaborated and the concerns of virtualization and VM migration overheads are brought out in this work. Further, we highlight the need to use these algorithms for making scheduling decisions in real time.

Chapter 2

Related Work

The Problem of server consolidation is well explored in literature. It is a variant of one of the classical NP-hard problem of bin packing where different sized items are to packed in bins of fixed capacity and one aims to minimize the number of bins used. Here, the problem translates into a vector packing problem where VMs are items to be inserted, physical machines are bins and resources like CPU, memory, bandwidth etc. form different dimensions of vector. This chapter explains the different approaches tried to solve Virtual Machine Placement Problem (VMPP).

2.1 Classical ILP and Meta-heuristic based approaches

The problem of VMPP is posed classically as an Integer Linear Program problem where objective function captures minimization of physical hosts used and the capacity restrictions of hosts are expressed as constraints of the integer linear program. In [8], Gupta et. al. have given a two stage heuristic algorithm for solving this server consolidation problem with item-item and bin-item incompatibility constraints. The case of item-item constraint is a potential scenario where by two I/O bound VMs have not to be placed together on a single host. Similarly, bin-item arises in cases whereby a virtual machine could have a possible restriction of not to be placed on particular physical hardware configuration .

Then, Agarwal et al. in [9] deals with the same problem of bin-item and item-bin constraints

by a totally new approach of Group Genetic Algorithm and shows how a Genetic Algorithm can prove to be better than other approaches. The above genetic algorithm owes its origin to Falkenauer [10] 's group genetic algorithm where a chromosome is composed of a fixed part and a variable length part. It also gives the counter algorithm where a fixed length chromosome of all items to be packed is a bad approach to deal with in the cases of server consolidation.

Recently, Wu et al. in [11] have given a simulated annealing based algorithm to tackle this problem where initial solution is obtained by first applying FFD based heuristics and then with some compromise over time , significant improvements can be obtained.

For all the above approaches, it has been discussed that beyond a few hundred VMs, genetic algorithms, ILP based approaches and other meta-heuristics have very high time complexity. Hence, these are not seen as potential solutions in real world scenarios affecting scheduling decisions.

2.2 Real time heuristics for Vector packing

To reduce time complexity, algorithms based on greedy heuristics like First Fit, Best Fit etc. have been implemented. These give close to optimal solutions in one dimensional case [12]. Here, items are arranged according to sizes and then, packed into bins based on algorithm used. In case of multi-dimensional problems, it is not clear what combination of vector should be used to generate a scalar(size) so that one-dimensional algorithms can be used.

Panigrahy et. al. [13] in their report on heuristics for vector bin packing explore different combinations of vectors to generate the scalar(size) and propose the new concept of geometric heuristics. But it does not justify why one heuristic outperforms others and given a workload, which heuristic should be applied.

A recent work by Mishra et al. [14] explains and points out shortcomings in existing technologies and present a new approach based on vector algebra for VM placement. We evaluated their approach of resource imbalance vectors and show that the same would be outperformed by Euclidean distance/Norm-2 approach in many cases ,giving specific counter examples for the same.

There are also automated monitoring and provisioning systems like Sandpiper [15] which uses volume metric for detecting hotspots and enable VM migration.Volume in this case is defined as follows:

$$volume = 1/(1-cpu) \times 1/(1-net) \times 1/(1-mem) \tag{2.1}$$

where cpu, net and mem are the corresponding utilizations of that resource for the virtual or physical server. Virtual Machines are arranged in order of decreasing volume to size ratio while physical machine are ordered according to decreasing volume. Then, VMs are dynamically moved from high volume to low volume physical machine when a hotspot is detected. The shortcomings of the same are shown as well in [14].

2.3 Algorithms in contemporary cloud management tools

A recent paper by Lee et. al. of Topology Aware Resource Allocation (TARA) [16], brings out a new dimension where by network topology related information and application's requirements are used to allocate data intensive workloads. They do so by a naive genetic approach that is different than discussed in [9].

The problem of migration control is described in [17]. They put a constraint where a static workload would be never migrated. In this case, one may end up getting less packing efficiency. We point out that such a migration control should be dependent on type of workload and migration cost involved for migrating that VM.

There are many prevalent datacenter management tools like OpenNebula, Eucalyptus etc.. They have diverse strategies for VM placement. For instance, the OpenNebula [1] implements a rank scheduling policy for VMs where by hosts are ranked using a formula on available monitoring information. The pre-defined policies exist for both Packing(Minimum number of hosts) and load balancing. The metrics used by these are number of running VMs or Free CPU available on that host. These are coarse grained and high level metrics because what type of VMs are running and their resource requirements are more important than number of running

[1] http://opennebula.org/documentation:archives:rel3.2:schg

VMs. Also, a decision should not be taken on a single resource like Free CPU. OpenNebula also provides an option to user to define its own ranking policy. But it is not possible to obtain SLA performance guarantees for the applications without using fine grained monitoring information (Hypervisor's CPU usage). On the other hand, Eucalyptus follows first fit or next fit based algorithms for VM provisioning on hosts.

2.4 Key Contributions

To the best of authors' knowledge, none of these solutions consider the virtualization and migration overhead while making the provisioning decisions. The contributions of this work in this context are as follows:

1. Including architecture specific virtualization overhead in problem definition to honour performance SLAs.

2. Considering previous placements of VMs and type of VMs to minimize migration overheads in case of dynamic placements.

3. Comparing the existing algorithms and considering the above constraints so as to aid real time scheduling decisions.

2.5 Summary

This chapter explores the related literature in the field of Virtual Machine Placement. Various works dealing with simulated annealing, Genetic algorithms and ILP based approaches are discussed. The chapter also describes various approaches which use FFD based heuristics and techniques used in contemporary tools like OpenNebula and Eucalyptus. The consideration of virtualization and VM migration overheads are emphasized as key differences of present work with related literature.And, the need to integrate the placement algorithms for taking real-time scheduling decisions is also highlighted.

Chapter 3

Impact of virtualization on application performance

In this chapter, firstly, we present a case study showing the impact of virtualization overhead on performance SLAs of applications. As discussed in [18] and [4], in case of I/O applications virtualization overhead is manifested as extra CPU usage of hypervisor apart from usual guest usage. We show that not considering this overhead in problem domain leads to severe degradation of applications' performance and therefore SLA violations.

3.1 Case Study

We present two different scenarios, in Case-1 virtualization overhead is considered in resource provisioning while in Case-2, no overhead is considered. We use KVM [1] as a hypervisor for our experiments. Since in KVM each VM is treated as a separate process, we take sum of CPU used by guest and CPU usage of hypervisor as total CPU requirement in Case 1. Only the guest CPU is considered in Case-2 where no virtualization overhead is considered. The other resources considered are memory, disk I/O bandwidth and network bandwidth.

In order to validate above scenarios, we first show the difference in resource requirements in

[1] http://www.linux-kvm.org/

two cases by analysing resources used in case of web-server application. We hosted apache web-server application in virtualized and non-virtualized environments. The machine specifications for the two cases are shown in Table 3.1.. To simulate workloads, we used httperf which is commonly used benchmarking tool for generating representative workloads. Figure 3.1 plots the total CPU used for different request rates for the two cases discussed above. We analyse that the CPU requirement in virtualized case is significantly high compared to non-virtualized scenario. Further breaking down the CPU requirement in different components in figure 3.2, we observe that extra CPU requirement is due to CPU used by different modules in hypervisor in case of VM.

Table 3.1: Machine Specifications

Hw-Sw	Physical Machine	Virtual Machine
Processor	AMD 2.4 Ghz 12 cores	AMD 2.4GHz (Requirement based)
Memory	16GB	Requirement based
OS	OpenSuse 12.1	Opensuse 11.4
Network Bandwidth	1Gbps	Best Effort
Disk I/O Bandwidth	7000 KB/s	Best Effort

Next, we analyse the impact of not considering this CPU overhead in scheduling decisions. For our experiments, we hosted VMs with four different applications as shown in Table 3.2.. The table also shows the metric for performance evaluation, how the workload is generated and the CPU requirement for each application. The other resources like memory, Network I/O and Disk I/O do not effect the scheduling decision in this case and their actual requirements are shown in Table A.1 and Table A.2 .

We observe from figure 3.1 that each web-server VM requires only 1 CPU if no overhead is considered while there is requirement of 3 CPUs in virtualized environment. The scheduler aims on consolidating workloads to minimum number of physical machines packs all VMs on a single host in no overhead case while it needs two hosts in other case. We analyse the effect on performance of web-server application in two Cases. Figure 3.3, plots request rate with response time which is most common metric for defining SLAs in case of web-servers. Since only 1 CPU is given to web-server VM while not considering virtualization overhead, we observe

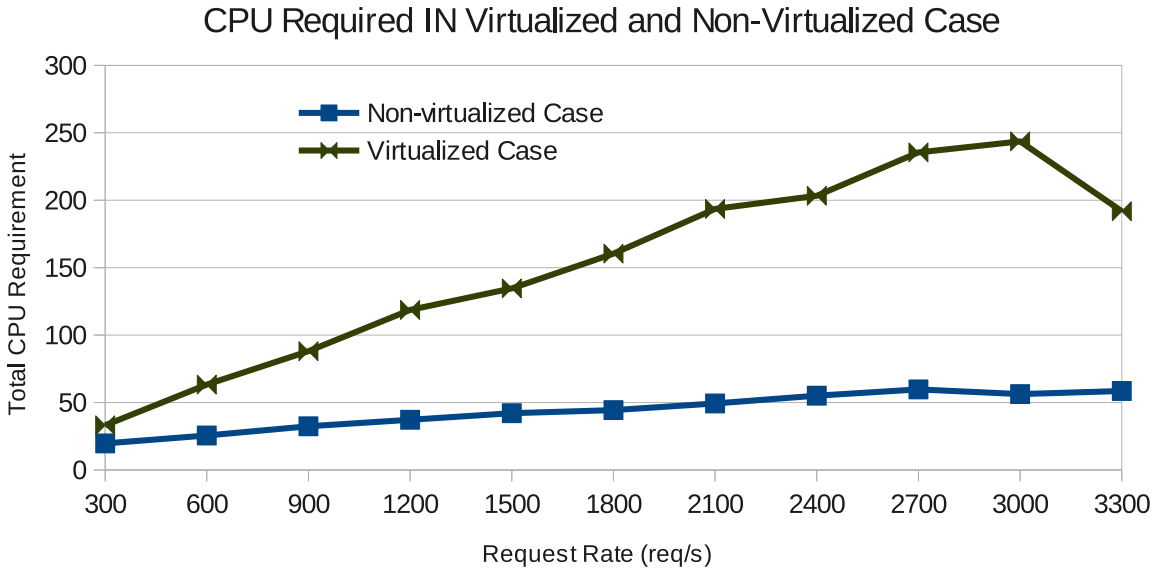


Figure 3.1: CPU Usage in Virtualized and Non-Virtualized Settings

that response time begins to rise sharply at around 1000 requests/second. This is in contrast to other scenario where we can achieve reasonable response time even at 3000 requests per second. Hence, if SLA is such that there should be less than 100 ms response time for request rates till 3000 req/s, we would be achieving the SLAs for only 30 percent of requested rate.

Hence, this study clearly points out that while moving applications to cloud, we must consider the resource requirements in virtualized settings which are significantly high compared to host usage and effect performance SLAs of application drastically.

Table 3.2: Types of applications

Type of appl. (Workload intensity)	CPU requirement	Performance Metric	Load generation method
web-server (3000 rq/s)	1CPU / 3CPUs	response time (ms)	httperf [19]
smtp mail-server(6000 req/s)	1 CPU	delivery time of mail(s)	smtp-source [2]
Prime numbers (Till 100000)	8 CPUs	Task completion time(s)	sysbench
Matrix Multiplication (1024*1024)	1 CPU	Task completion time(s)	size of matrix
Matrix Multiplication (1024*1024)	1 CPU	Task completion time(s)	size of matrix

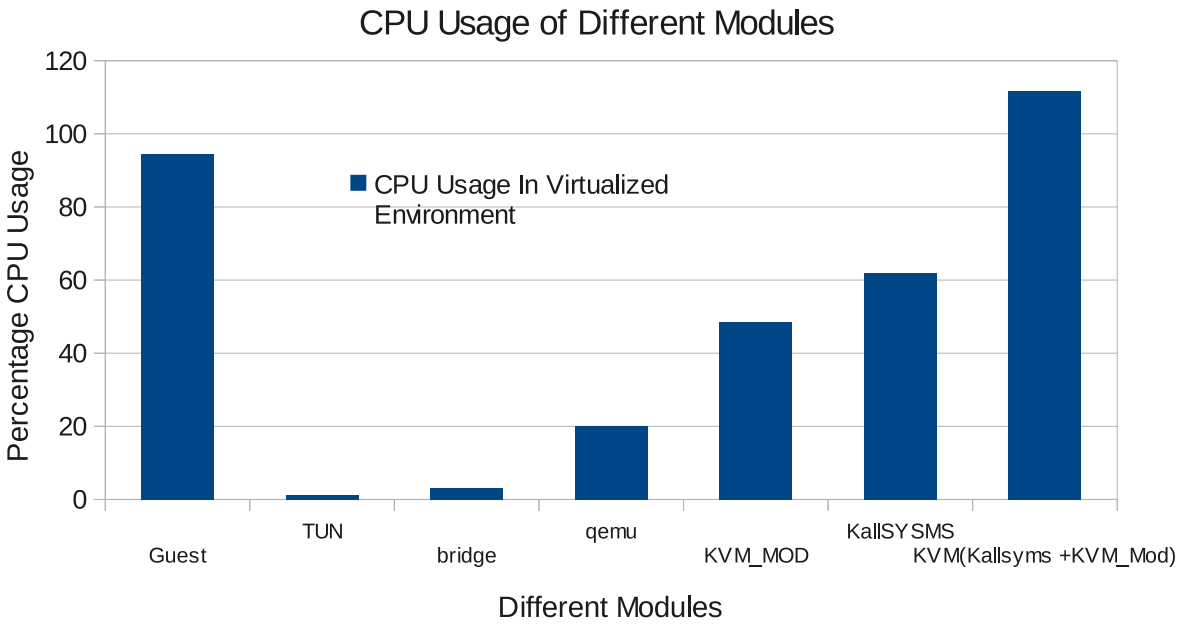


Figure 3.2: CPU Usage Breakdown for Saturation Points

On the other hand, owing to dynamic nature of problem and to cater to elastic workloads, VMPP algorithm needs to be invoked at every scheduling cycle. The new solution in order to minimize the number of hosts may lead to migration of VMs from one host to another which itself is expensive process and incurs additional overheads.

We quantify this migration overhead in terms of type and resource requirements of applications. We argue that this overhead is more for applications having high I/O (file descriptors) and memory footprint compared to pure CPU intensive applications. So, CPU intensive applications should be given more priority for migration. We quantify this total migration overhead(α) by weighted addition of different resource requirements (r_i) for each VM as shown in equation 3.1. The relative weights(w_i) are 0.8 for network bandwidth resources, 0.6 for disk I/O resources, 0.4 for memory and 0.1 for CPU resource.

$$\alpha = \sum_{i=1}^{i=d} r_i * w_i \tag{3.1}$$

where, d is number of resources to be considered.

These values can be calculated experimentally by undertaking actual migrations but the

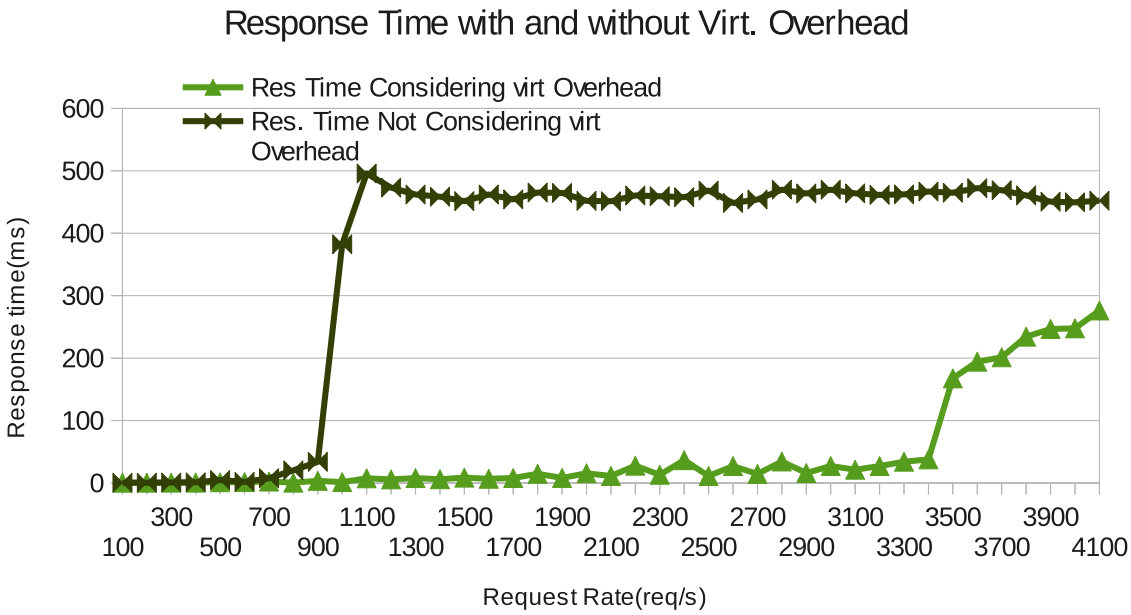


Figure 3.3: Virtualization overhead impact on application performance

relative weights will remain the same for different resources [7]. We modify the VMPP by including this migration overhead and then, try to minimize both the number of hosts and the migration overhead. The actual problem definition and algorithms to solve the same are described in next chapter.

3.2 Summary

The chapter describes a case study to compare VMPP algorithms with and without virtualization overhead. It shows significant performance degradation in case of web-servers affecting performance SLAs. It also the quantifies the migration in terms of type of VM and its resource requirement.

Chapter 4

VM Placement Algorithms

In this chapter we propose various algorithms to solve Virtual Machine Placement problem along with modifications introduced in this work. The first section proposes an ILP formulation for the problem. But due to enormous high time complexity of ILP solutions for large problem sizes, we shift to First Fit Decreasing (FFD) heuristics in the next section.

4.1 ILP Formulation for VMPP supporting performance SLAs

The problem of VM placement is formulated traditionally in the form of an ILP. Here, in this section, we give an ILP formulation of VMPP along with modifications to address issues of migration and virtualization overhead discussed in previous sections. The problem definition is shown below along with description of notations in Table 4.1.

Minimize:

$$F_{obj} = \sum_{i=1}^{i=m} P_i - \sum_{i=1}^{i=m} \sum_{j=1}^{j=n} \alpha_j * V_{ij}^{prev} * V_{i,j} \tag{4.1}$$

Constraints:

$$\sum_{i=1}^{i=m} V_{ij} = 1 \; \forall j \tag{4.2}$$

$$P_i * CPU_i \geq \sum_{j=1}^{j=n+t} V_{ij} * \left(cpu_j^{Hypervisor} + cpu_j^{vm} \right) \; \forall i, j \tag{4.3}$$

Table 4.1: Notations

$V_{i,j}$	binary variable,1 if VM j is placed on host i
m	No of Hosts
P_i	binary Variable,1 if host i is used for any VM
n	No of Previously hosted VMs
α_j	Migration Coefficient of VM j
t	No of new VMs to be allocated
R_i	Capacity of R resource in Host i
r_j	Requirement of r resource for VM j
$V_{i,j}^{prev}$	values of $V_{i,j}$ according to previous allocations
$cpu_j^{Hypervisor}$	CPU used by hypervisor for VM j
cpu_j^{VM}	CPU used by VM j

$$P_i * R_i \geq \sum_{j=1}^{j=n+t} V_{ij} * r_j \; \forall i, j \tag{4.4}$$

Here, the objective function (eqn. 4.1) of problem consists of two parts, the first part focuses on minimizing the number of hosts while the second part relates to minimum migration overhead. Here, α_j is calculated by weighted addition of resource requirements as described in previous section. So, eqn. 4.1 would minimize the objective function if both $V_{i,j}^{prev}$ and $V_{i,j}$ are one. Hence, if some allocation with same number of physical hosts exists with and without migration , the objective function would choose an allocation with minimum migration overheads.

In constraint set, eqn. (4.2) describes the fact that a VM is allocated to only one host. Eqn. (4.3) shows the second modification to the problem, where virtualization overhead is considered. We have added CPU used by hypervisor and CPU used by VM in this case Finally, eqns. (4.4) puts the capacity constraints on different resources. Also, it is assumed that maximum disk I/O bandwidth and other resource capacities are not effected much in virtualized settings and if they differ significantly, the values should be considered in virtualized settings as well. We used lpsolve [1] package for solving the exact ILP in serial form which uses branch and bound method for solving the same. To further reduce time in big instances, we used scip multi-threaded package [2] and ran the algorithm upto 12 pthreads. According to a classical result by Lensra [20], any ILP with n variables and m constraints can be decided in $O(c^{n^3} - m^d)$ time where c

[1] http://lpsolve.sourceforge.net/5.5/
[2] http://scip.zib.de/

and d are constants. In our case, for p virtual machines and q physical hosts, we have $(p*q+q)$ variables and $(p+4q)$ constraints which gives a highly exponential time. The results for the ILP formulation for small problem instances is shown in chapter 6 . The next section captures the same problem definition in different FFD heuristics to reduce the exponential time.

4.2 FFD Vector Heuristics for VMPP

The above Integer Linear Program provides an easy formulation of the problem and all the constraints and requirements are expressed directly capturing the problem domain naturally. But solving ILP is an NP hard problem and so can't be used in practice for real time solutions. However, we can do some modifications to the same by having a linear programming relaxation of the original ILP. These may give some early results for small cases but as problem size grows, these algorithms are not suitable for obtaining near optimal solutions within the scheduling cycle [13]. Therefore, FFD heuristics might be a better trade off.

The theoretical bounds of 11/9 of optimal number of bins are shown for one dimensional case in [12] for these heuristics. In average cases, FFD performs reasonably well and that too in small duration of time such that solving a new problem with large number of VMs in every scheduling cycle is feasible.

The first problem in this case is how to incorporate the notion of a vector in FFD. Different heuristics for the same were suggested in [13] and [14]. We choose 3 different FFD based methods viz. Dot Product, Euclidean Distance and Resource Imbalance Vector method as suggested in the literature and then modify those to our problem settings. Firstly, we analyse these methods and explain what these methods are capturing in different scenarios. The three strategies differ in the ordering of VM placement in each case. Before proceeding to strategies, we define some acronyms, RRV is Resource Requirement Vector defined for all VMs, RUV is Resource Utilization Vector defining currently used resources of a host and RCV is Residual Capacity Vector defining remaining capacity for all used hosts.

In Dot Product approach, we take the unit vector corresponding to RCV of currently used host and take its dot product with unit vector of all unallocated valid VMs (VMs which satisfy

resource capacity constraints) and the VM having maximum dot product is chosen for allocation. The intuition behind the same is that a large Dot product value means small angle between the RCV of host and RRV of VM. This in turn means better alignment of VM vector with the host vector, thus, making VM a good choice to be placed on that host. The issue with this approach is that although it captures the direction sense pretty well, it does not capture the remaining capacity in absolute sense.

The second approach of Resource Imbalance Vector (RIV) was suggested by Mishra et. al. in [14]. They take the projection of RUV on normalized resource diagonal i.e (1,1,1..1) vector and subtract it from RUV to get RIV of that physical machine(fig 4.1). Similarly, we get RIV for every VM. Then, by adding these two RIVs, we choose the one with minimum magnitude. The intuition behind this is that complementary workloads across diagonal will be placed on same machine. This leads to better host utilization. We argue that in a number of cases, there would be many such vectors which would balance the RIVs exactly but not all are good candidates for placement. As shown in figure 4.1, all the vectors a,b,c,d of the VMs give the same total RIV but only c is the candidate which actually captures the remaining capacity exactly and should be chosen for placement.

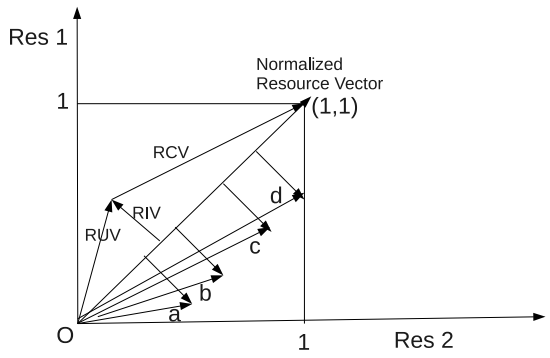


Figure 4.1: Resource vectors in 2-D plane

Thirdly, we look at the euclidean distance strategy where we calculate euclidean distance between RCV of PM and RRV of VMs and choose the VM having minimum Euclidean distance with the host. Euclidean distance method inspite of its simplicity captures both the direction and magnitude aspects correctly. It is able to capture the direction sense of a vector because with increase in angle between two vectors, the euclidean distance also increases. Also, most

of the workloads in real world are specific resource centric i.e they use one resource more than the others. So, the solution should not only capture its alignment with host capacity as done in previous two approaches but a good fit overall (across most of dimensions).

```
for All dimensions do
    Normalize all resource capacities and requirements to 1;
end
Initialize a host;
while All VMs Not Placed do
    Initialize pointer to start of VM list;
    while Pointer not At the end of VM list do
        if VM can be packed into current Host then
            calculate score by RIV or Dot product or Euclidean distance ;
            if VM placed on same host in Previous Mapping then
                Update score of each VM by adding migration overhead according to
                heuristic,;
            end
        end
        Goto next VM in the list ;
    end
    if No VM can be placed but remaining unplaced VMs then
        Initialize and add a new host;
        Goto statement 4 ;
    end
    Find VM with minimum score in RIV and Euclidean distance and maximum score in
    Dot product;
    Place that VM over host and remove it from VM list;
    Update host's current capacity;
end
```

Algorithm 1: FFD previous mapping aware heuristic

Our basic strategy is underlined in Algorithm 1. First, we normalize all capacities and resource requirements in all dimensions to 1. We initialize a host and to place the next VM, we consider all the valid VMs (which meet resource requirements) and find the most appropriate VM according to different heuristics based on what we call 'the score'. The key contribution of our algorithm is that we change (increase in Dot product or decrease in RIV/Euclidean distance) the score of a VM if the VM is getting placed on the previously allocated host. This increases the chances of a VM being placed on the same host again compared to other VMs. This change of score is quantified in terms of migration overhead α_j as described in previous section. The

issue of meeting performance SLAs is dealt in this case, by adding CPU usage of hypervisor as described in previous chapter. Analysing our algorithm's complexity, in the worst case, we select 1 VM in every pass and in total, we make number of passes equal to number of VMs, so to place n VMs ,we will take $O(n^2)$ time which is significantly less compared to exponential time of ILP. The calculation of score is a constant in all algorithms whose value depends on heuristic used.

Also, since all of these are heuristic solutions for the problem, the counter examples for all of these can be found. But the choice of heuristic should depend on the nature of workload and problem context. In the next chapter, we state our experimental setup and results.

turnaround time in case of CPU and memory intensive case.

4.3 Summary

This chapter describes various methods to solve VMPP considering virtualization and VM migration overheads. Both ILP formulation and FFD heuristic based methods are illustrated for the same. Further, three different strategies of Dot product, Euclidean distance and RIV based approaches are evaluated to use FFD heuristics for vector packing. Logical arguments support Euclidean distance strategy for it captures both magnitude and distance of vectors naturally.

Chapter 5

Workload generation and Monitoring

One of the important considerations when applications are moved to VMs is virtualization overhead. To account for such overhead, we need to monitor resources used by different applications while running multiple virtual machines in a physical server. Resource monitoring in virtualized scenario is a bit difficult as hardware performance counters are not virtualized for VMs[21]. There are many tools available which give host specific resource usage for different hypervisors like Xen, KVM etc. Moreover, resource usage and virtualization overhead depends on hypervisor and virtualization technology used. In recent years, to make development of virtual machine monitors easy, hardware vendors like AMD and INTEL have added virtualization extensions to x86 processors which were initially difficult to virtualize.

The KVM (Kernel based Virtual Machine) is a relatively new VMM which utilizes these hardware extensions and have found its way in Linux kernel. It is a full virtualization solution, which requires no changes in guest operating system. Hence, in this work, we choose KVM as a hypervisor for managing VMs. Also, it is open-source, so becoming hypervisor of choice in most recent studies. Since it is relatively new hypervisor, not many tools are available to extract resources used by application in virtualization settings. In this chapter, we give details of tools used to generate diverse workloads and monitor different resources in KVM based machines. To extract the fine grained monitoring information in case of KVM, we first study the architecture of KVM in the next section.

5.1 KVM architecture

Kernel based Virtual Machine (KVM) is a relatively new hypervisor which uses virtualization extensions and has found its way in Linux kernel.It consists of two modules, namely, kvm.ko and an architecture dependent kvm- amd.ko or kvm-intel.ko module. Under KVM, each VM is spawned as a regular linux process named KVM and scheduled by the default linux scheduler. In this case, processor has three modes of operation namely, guest, user and kernel modes. The guest mode is added to support virtualization and the virtual machine runs in guest mode. The guest mode in turn has user and kernel modes in itself.

Since all the virtual machines are running on same host, these share the I/O hardware of host machine. For using shared I/O hardware, these virtual machines interact with Qemu emulator in host user space which provides emulated I/O devices for VMs. For instance, in the case of network related applications, Qemu provides emulated Network Interface Card (NIC) to VMs. A software bridge is also configured in the host kernel for distributing packets to different VMs.

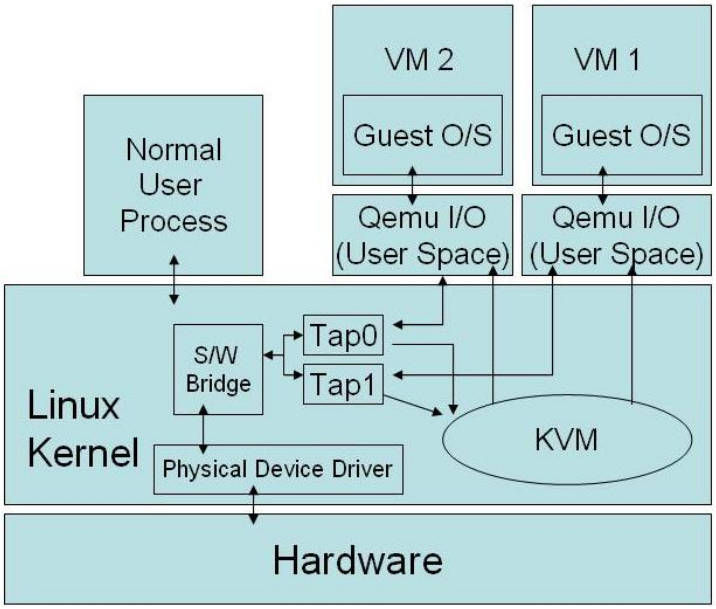


Figure 5.1: KVM Architecture Source [22]

Figure 5.1 shows the typical KVM architecture with reference to a network related application. A typical network packet flows through the KVM virtualized host in the following way. As

depicted in picture, when a packet arrives at physical NIC, interrupts generated by NIC are handled by the physical device driver. The device driver forwards the packet to software bridge. The bridge, then pushes the packet to the tap device of the corresponding VM. The tap device is a virtual network device that operates at layer-2 (link layer). This tap device is attached to user space Qemu program . On receiving the packet, tap device sends a signal to KVM module. KVM module in turn, generates a virtual interrupt to the user space Qemu of the target VM. Qemu then copies the packet from tap device. Also, Qemu generates the interrupt for the guest OS emulating the virtual NIC. Then, the physical device driver in the guest OS handles the packet transfer to the VM's address space. The packet is then delivered to a process as done in a normal non-virtualized setting.

Consequently, for a VM process in KVM virtualized server, guest mode execution (both kernel or user mode) corresponds to execution within a VM while other modules in user mode (Qemu) and kernel mode (KVM module, tun-tap module, bridge module, etc.) correspond to hypervisor execution. Among these, Qemu I/O module runs separately for each VM but is co-ordinated by a single KVM module which manages all VMs by signal and virtual interrupts. Hence, in a virtualized setting there is a CPU used by guest mode which is separate for all processes and can be run in parallel on a multi-processor machine. In addition, there is a CPU used by hypervisor which is working on behalf of all VMs and this serial code of hypervisor can become a potential bottleneck and can effect the performance of applications.

There are many tools available to monitor CPU resources in non-virtualized environment like top, o-profile etc.. But to extract fine grained VM information, we need a tool like o-profile which uses hardware performance counters. But to the best of author's knowledge, o-profile has no functionality available to differentiate CPU usage in guest and host mode. One of the tool which is in close match with KVM in this context are Linux Perf Counters [1]. Perf is a profiling tool for linux based systems and so, makes a good integration with KVM which is also embedded in Linux kernel itself.

[1] https://perf.wiki.kernel.org/index.php/Tutorial

5.2 Performance Monitoring in Linux -Perf

Perf internally uses hardware performance counters for profiling. Initially, certain events like instructions executed, cache misses etc. are selected based on which profiling needs to be done. A counter is incremented whenever such an event occurs and when the counter reaches a predefined value, an interrupt is generated and the program counter value at that time is recorded, with counter being reset again. The interpretation of recorded events gives the percentage CPU used by different programs at a finer granularity of function level.

Perf apart from profiling like o-profile has a special command “perf kvm” which can be used to profile the guest kernel much like the host kernel. Also, it gives a clear percentage of CPU used in guest mode, host user and host kernel mode. In addition, we can get the CPU profile of guest kernel as well .

Using the architectural details of KVM and the above information provided by perf kvm, we extract the CPU usage of hypervisor in each module on behalf of a particular VM .

5.3 System Design for performance counters

The basic system model is shown in Fig 5.2. As described in previous section, each VM under KVM hypervisor is a Linux process. Initially, we extract the PIDs of VM linux processes named KVM. Then, for a given monitoring period, resource profiles are generated across all CPUs by a profile recorder like **perf kvm record** and fed into the system along with PIDs. Using these PIDs, we segregate the process wise CPU usage using the corresponding interpreter of the profile recorder used in initial phase. In our case, we have used **perf kvm report** for the same. The important point to note is that profiler must be enabled in virtualization mode for e.g we have to use suffix “kvm” to record profiles for guest mode separately in case of perf.

Then, profiles obtained for VMs are used to obtain CPU used by the guest and host user and kernel modes separately as shown in figure 5.2. An example CPU profile for web-server application is shown in figure 3.2. The CPU information is extracted from the above tool only for web-servers. In the next section, we consider mail server application , how workload is generated and how the resources are monitored for the same.

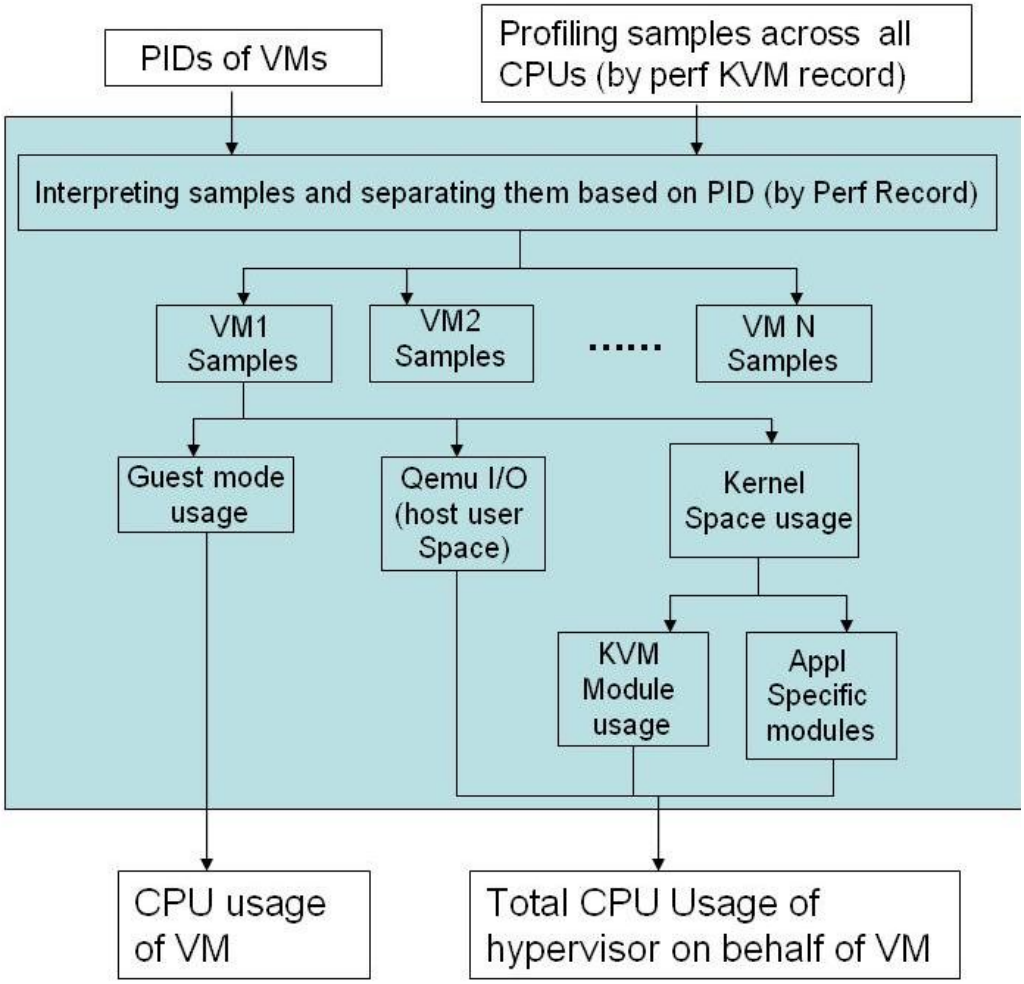


Figure 5.2: System Model

5.4 Mail Server

The second application we considered is also one of most commonly hosted applications in cloud and which uses a different resource as compared to web-server - the Mail server. To understand how to generate workload for the same and monitor it, we first give a brief introduction of mail send and receive process.

A mail send and receive process in itself comprises of a series of steps. A user accesses a mail delivered to its inbox by a mail server through an Email Client or Mail User Agent (MUA) program. But how a mail is delivered to an inbox (a file or a database) is one of the most important components in Electronic Mails. Mail transfer from one mail server to another

occurs by following a protocol. One of the most common mail transfer protocol in this domain in Simple Mail Transfer Protocol (SMTP). The mail is transferred using a simple client -server application where the sending system is called as client and delivers the mail to receiving system called server. Since the communication can be in both directions in this case, both the client and server programs are implemented in a single application called Mail Transfer Agent (M.T.A). Typical MTA 's are sendmail and postfix mail transfer agents with postfix replacing the sendmail in most of new systems.

We hosted a mail server in a Virtual Machine and parallel mails to that server from an outside client were sent. The client program we used for the same was **smtp-source**. smtp-source is a simple mail benchmarking program which sends mails in parallel to a mail server. We compared the performance of the mail-server in single VM case and case where two VMs were consolidated on a single server. The results obtained are described below. To measure the performance of mail server, we extracted the average delivery time of mails from the logs of mail delivery. We monitored all the resources for the system and observed that Disk I/O becomes the bottleneck in this case.The file system was used in sync mode so that mails are written directly to Disk without storing the same in buffer cache initially. Also, the file system used was EXT2 so that journalling does not effect the Disk I/O monitoring information. We used iotop tool for measuring the disk i/o activity and segregated the same for each Virtual Machine. Since in KVM each VM is treated as a separate process, we filter I/O activity based on PID as was done in CPU monitoring case earlier.

In figure 5.3, we plot the mail rate (mail/sec) generated with smtp-source with the average delivery time of mails. We observe that in case of 1 VM, we have less than 5 second delivery time upto 5600 request rates. This delivery time begins to rise sharply at 2400 request rates in 2 VM case. This mainly happens because disk is shared between VMs and maximum write speed is limited by the disk write speed.

To analyse the bottleneck, we plot the Disk I/O per VM in two cases with mail rate in figure 5.4. We observe that total disk rate goes close to 5000 KBps in first case of 1 VM, but since Disk is shared between the two VMs, the performance drops to less than half in the second case. This clearly shows that the placement decision must consider the resource usage carefully where VMs

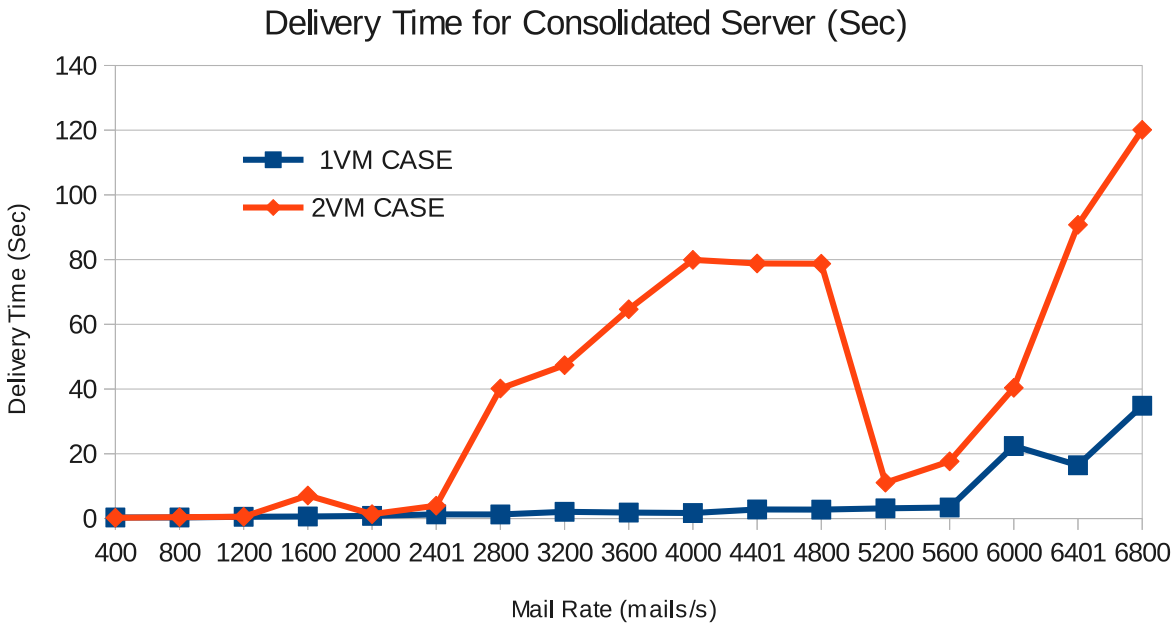


Figure 5.3: Delivery time for Virtualized Case

sharing the same resource should not be placed on the same host. Also, the cumulative Disk I/O is less than single VM case in second case. This may occur due to interference effects of Disk I/O request rates of two VMs. We will use mail server as one of workloads while considering the diverse workloads for consolidation.

5.5 Other workloads and Monitoring

The other workloads we used in our study are matrix multiplication and computation of prime numbers. While performance of matrix multiplication depends on memory available , the computation of prime numbers is pure guest CPU intensive workload. For monitoring of different resources like Network B/W, CPU, Disk I/O, Memory etc., we used the following tools as described in Table 5.1.

We generated these diverse workloads and measured resources using tools and techniques described above. All this was done to illustrate that resource monitoring is critical in clouds and to test the algorithms described with a good mix of workloads for VM placement. The results shown in next chapter are generic in nature and the algorithms can be employed on any generic

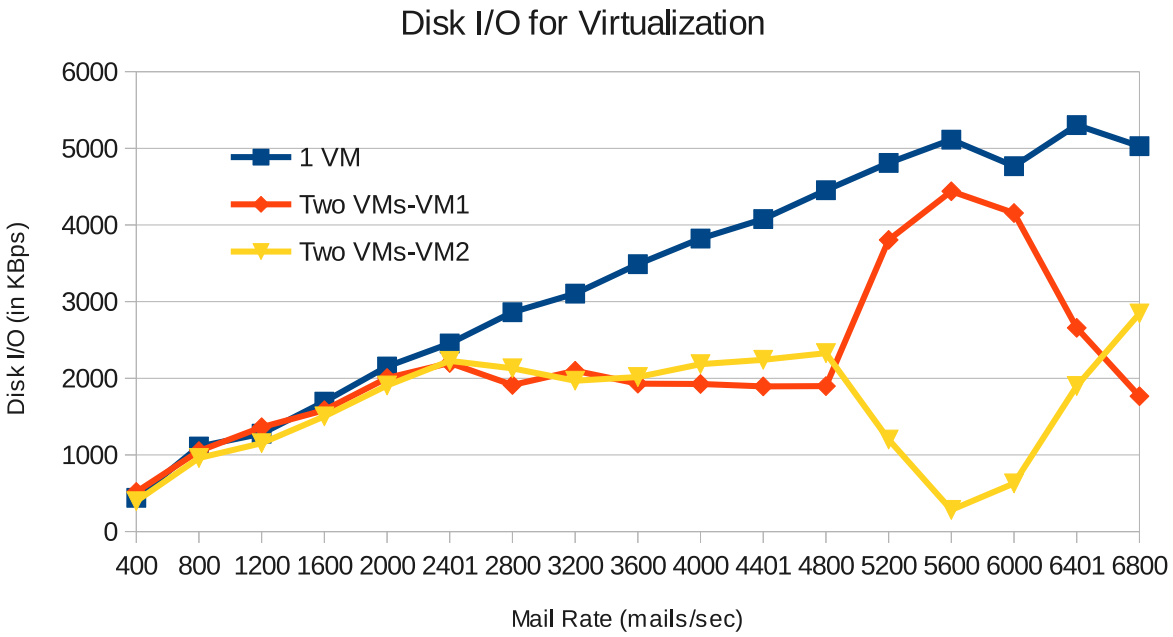


Figure 5.4: Disk I/O in Virtualized Case

Table 5.1: Monitoring Tools for Physical Resources

Resource	Monitoring Tool Used
CPU	Linux perf (perf kvm command)
Memory	free command
Network Bandwidth	bwm-ng
Disk I/O Bandwidth	iotop

data monitored in real datacentres.

5.6 Summary

The chapter describes various resource monitoring tools useful on virtualized servers based on KVM hypervisor. It explains the architecture of KVM and depicts the flow for generating fine grained CPU monitoring information to measure virtualization overhead. Also, it describes what kind of workloads are used and how synthetic data is generated to test various algorithms described in previous chapter.

Chapter 6

Experimental Setup and Results

6.1 Experimental Setup

In this chapter, we evaluate different heuristics and ILP in a setup consisting of four different types of workloads as described in Table 3.2. In our work, we consider four dimensions of resources, Total CPU usage of VM and hypervisor, memory requirements, disk I/O bandwidth and network bandwidth. We assume sufficient storage (30 GB) for all applications although same can be added as an extra dimension without any modification. To select a good mix of workloads, we choose a uniform random distribution across these four types of workloads. Then for each type of application, we generate different workloads and calculate resource requirements (shown in Table A.1 and Table A.2) for the same by actual experiments. For our experiments, we choose ten different request rates for web-servers and fifteen different request rates for mail servers and use a normal distribution over the same to emulate real world scenario. Similarly, five different CPU workloads and four different memory intensive workloads are chosen.

The reason for choosing these kind of workloads is these are most commonly used in clouds and each of them uses a different resource, thus providing a good mix of workloads. Also, these are similar to SPECvirt [1] workloads for benchmarking virtual servers.

[1] http://www.spec.org/

6.2 Results

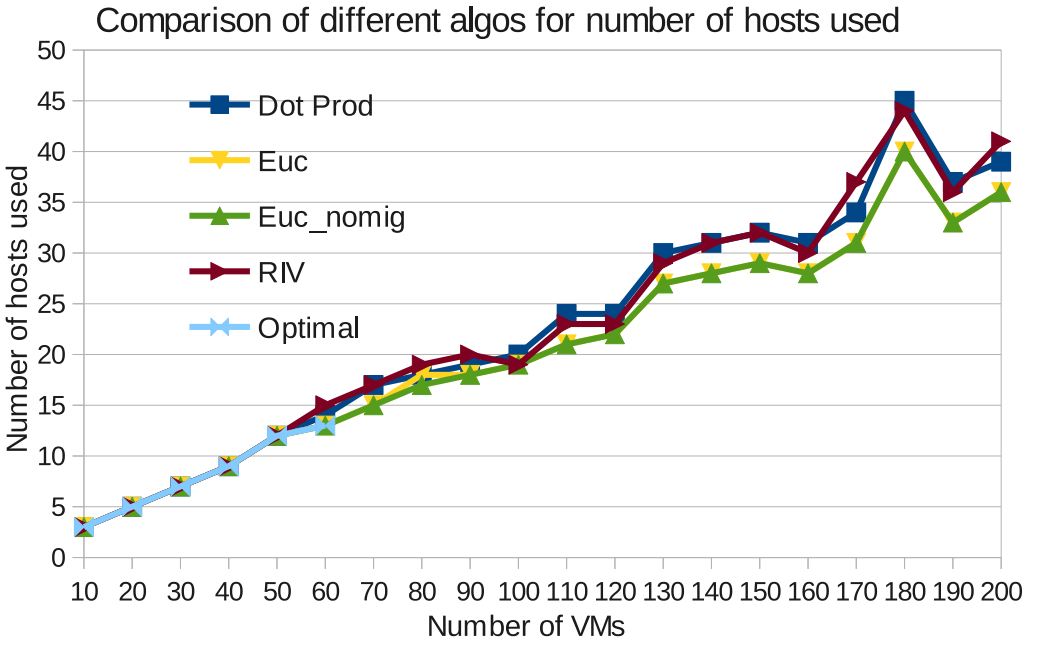


Figure 6.1: Comparison of FFD heuristics

We tested the above three algorithms, varying the total number of VMs from 10 to 200. In figure 6.1, the number of hosts needed to place all VMs is plotted against number of VMs. Due to exponential running time, ILP based optimal solutions are available till only 60 VMs beyond which it was taking many hours even with twelve parallel threads. As we see, all the heuristics are near optimal for small instances, but for bigger problem instances Euclidean distance strategy starts to outperform others by around 10 percent. Also, we show number of hosts used both with and without migration overheads (for euclidean distance) in figure 6.1. We observe less than 2 percent increase in number of hosts when migration overheads are considered. This points out that multiple solutions are available in most of the cases and by making our algorithm migration aware, we are able to choose those solutions which take minimum migration overhead. In figure 6.2, we plot number of migrated VMs against number of VMs with and without consideration of migration overheads for euclidean distance strategy. We observe that our strategy reduces the migrations by more than eighty percent in almost all cases.

From above results, we clearly observe that we are able to meet SLA performance guarantees

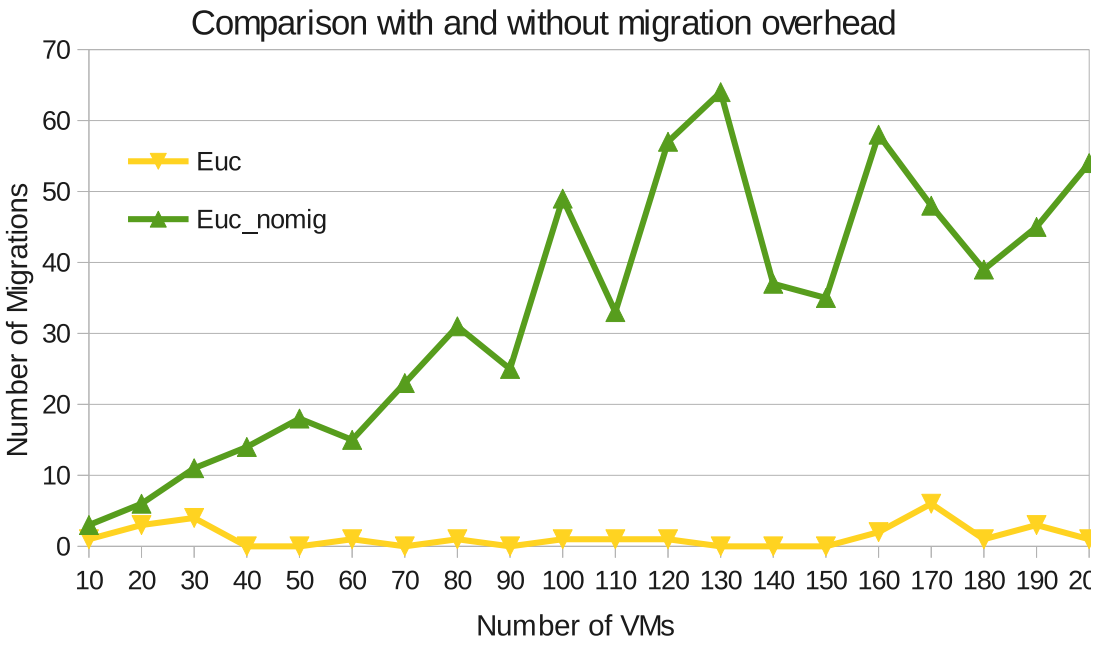


Figure 6.2: Comparison of FFD heuristics

by including overhead aspect in problem definition. Also, by modifying problem statement, we are making problem aware of previous assignments. Hence, we get a mapping with minimum migration overhead without increasing the number of hosts used.

6.3 Summary

The chapter describes experimental setup for various algorithms. Also, it shows that Euclidean distance method outperforms other heuristics by around 10 percent. It also shows a significant reduction of more than 80 percent number of migrations by making the algorithms migration aware as described by considering the migration overhead.

Chapter 7

Conclusion and Future Work

This work deals with the problem of Virtual Machine Placement Problem in datacenters. The problem has assumed great importance in today's world where power consumption and green computing is the need of hour. In our work we aimed to minimize the number of hosts used while placing virtual machines over physical hosts of the data center.

In this work, we point out modifications to VMPP by including performance SLA guarantees and considering the VM migration overheads, which has not been considered before. We also illustrate that considering virtualization overheads is paramount for providing performance SLA guarantees. Also, we observe that making VMPP algorithms migration aware reduces the number of migrations by 80 percent along with obtaining a solution near to optimal number of hosts. We further compare various heuristics to obtain solutions in real time and observe that Euclidean distance based FFD approach provides better gains than other FFD based heuristics. This approach along with modifications proposed can aid real time scheduling decisions.

In future, we would like to investigate the effect of consolidation of multiple VMs on KVM. Also, we would like to test these algorithms for real time elastic workloads. We would also like to evaluate effect of other architectural considerations like cache interference while taking placement decisions.Furthermore, we would like to explore and define a formal packing metric in this multi-dimensional resource case to evaluate fragmentation effects in different algorithms.

Appendix A

Resource Requirement Values for mail-server and Web-server

In this appendix, we provide resource requirements for different request rates for mail-server and web-server. We monitored resources for 9 different request rates in case of web-server and 15 different mail rates in case of server. The resources monitored includes Disk I/O bandwidth, Memory, Network bandwidth and CPU cycles used by guest as well as hypervisor. The data provided here was used in different algorithms described in this work.

Table A.1: Resource requirements for web-server having response time less than 100 milliseconds

Request rate	Disk I/O	Memory(GB)	Network B/W (Mbps)	Hypervisor's CPU(percent)	VM CPU(percent)
100	0.3552	0.45038	10.3959	9.69	4.96
200	85.8906	0.45023	20.7628	17.19	11.1
300	7.20042	0.44788	31.1061	23.56	11.71
400	11.3644	0.44781	41.5110	28.43	14.33
500	33.0323	0.44857	51.7151	30.34	16.8
600	30.4944	0.45174	61.9890	26.11	17.6
700	46.1489	0.47109	76.8173	27.54	21.02
800	74.372	0.53614	83.9530	30.02	21.94
900	71.3733	0.54164	88.2553	30.17	22.62

Table A.2: Resource requirements for mail-server having delivery time less than 5 seconds

Mail rate	Disk I/O	Memory(GB)	Network B/W (Mbps)	Hypervisor's CPU(percent)	VM CPU(percent)
600	648.052	0.5079	0.3710	2.08	7.73
1200	1373.38	0.5558	0.7496	3.84	5.96
1800	1944.1	0.60919	1.1145	5.23	9.16
2400	2472.13	0.63842	1.4393	7.55	13.04
3000	3685.53	0.63096	1.8411	12.54	16.98
3600	3685.68	0.6771	2.2419	10.68	16.17
4200	4223.03	0.69432	2.5875	12.19	19.96
4800	4542.14	0.69722	2.7729	14.29	22.64
5400	5031.67	0.72077	2.9412	15.09	28.53
6000	5674.32	0.71144	3.6031	16.74	25.73
6600	6107.63	0.71372	3.8431	14.83	26.6
7200	6359.35	0.70488	4.2423	17.12	29.01
7800	6797.89	0.69901	4.4503	19.14	31.52
8400	6719.09	0.69964	4.5696	23.05	34.51

Bibliography

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[2] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[3] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39–47, 2005.

[4] M. Dhingra, J. Lakshmi, and S.K. Nandy. Resource usage monitoring in clouds. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, pages 184 –191, sept. 2012.

[5] J. Lakshmi. *System Virtualization in the Multi-core Era, a QoS Perspective*. Dissertation, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, 2010.

[6] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 119–128, 2007.

[7] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo. Dynamic resource management using virtual machine migrations. *Communications Magazine, IEEE*, 50(9):34–40, 2012.

[8] R. Gupta, S.K. Bose, S. Sundarrajan, M. Chebiyam, and A. Chakrabarti. A two stage heuristic algorithm for solving the server consolidation problem with item-item and bin-item

incompatibility constraints. In *Services Computing, 2008. SCC '08. IEEE International Conference on*, volume 2, pages 39 –46, july 2008.

[9] Shubham Agrawal, Sumit Kumar Bose, and Srikanth Sundarrajan. Grouping genetic algorithm for solving the serverconsolidation problem with conflicts. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, GEC '09, pages 1–8, New York, NY, USA, 2009. ACM.

[10] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing, 1996.

[11] Yongqiang Wu, Maolin Tang, and W. Fraser. A simulated annealing algorithm for energy efficient virtual machine placement. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 1245–1250, 2012.

[12] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for np-hard problems. chapter Approximation algorithms for bin packing: a survey, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.

[13] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. Technical report, Technical report,Microsoft Research, 2011.

[14] M. Mishra and A. Sahoo. On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 275–282, 2011.

[15] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, NSDI'07, pages 17–17, Berkeley, CA, USA, 2007. USENIX Association.

[16] Gunho Lee, Niraj Tolia, Parthasarathy Ranganathan, and Randy H. Katz. Topology-aware resource allocation for data-intensive workloads. *SIGCOMM Comput. Commun. Rev.*, 41(1):120–124, January 2011.

[17] Tiago C Ferreto, Marco AS Netto, Rodrigo N Calheiros, and César AF De Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27(8):1027–1034, 2011.

[18] Ankit Anand, Mohit Dhingra, J. Lakshmi, and S. K. Nandy. Resource usage monitoring for kvm based virtual machines. In *Proceedings of the 18th annual International Conference on Advanced Computing and Communications (ADCOM 2012), To Be Published*, dec. 2012.

[19] David Mosberger and Tai Jin. httperf-a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, December 1998.

[20] Stefan Kratsch. On polynomial kernels for sparse integer linear programs. *CoRR*, abs/1302.3494, 2013.

[21] Jiaqing Du, Nipun Sehrawat, and Willy Zwaenepoel. Performance profiling of virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, pages 3–14, New York, NY, USA, 2011. ACM.

[22] Shan Zeng and Qinfen Hao. Network i/o path analysis in the kernel-based virtual machine environment through tracing. In *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pages 2658 –2661, dec. 2009.