A Comparative Analysis of Server Consolidation Algorithms on a novel Software Framework

M. Tech. Project Dissertation

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology

by

Anwesha Das
Roll No: 09305913

under the guidance of

Prof. Purushottam Kulkarni and Prof. Varsha Apte

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Dissertation Approval Certificate

Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

The dissertation entitled **"A Comparative Study of Server Consolidation Algorithms on a Software Framework in a Virtualized Environment"**, submitted by **Anwesha Das** (Roll No: 09305913) is approved for the degree of **Master of Technology** in **Computer Science and Engineering** from **Indian Institute of Technology, Bombay**.

Prof. Purushottam Kulkarni **CSE, IIT Bombay** **Supervisor**	**Prof. Varsha Apte** **CSE, IIT Bombay** **Co Supervisor**
Prof. Umesh Bellur **CSE, IIT Bombay** **Internal Examiner**	**Dr. Subhasri Duttagupta** **TCS** **External Examiner**

Prof. Kannan Moudgalya
CHEM, IIT Bombay
Chairperson

Place: IIT Bombay, Mumbai
Date: 9th July, 2012

Declaration

I, Anwesha Das, declare that this written submission represents my ideas in my own words and wherever others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above can be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signature

Name of Student

Roll Number

Date

Acknowledgements

It gives me immense satisfaction and great pleasure to thank all the people who have helped me in making this project a real learning experience.

First and foremost, I would like to thank and express my heartfelt gratitude towards **Prof. Purushottam Kulkarni** for his valuable guidance, useful suggestions and most importantly for his patience & tolerance towards me during difficult times. I am deeply indebted to him for his continuous support throughout the project. Without his excellent guidance, this project would not have been possible. It was a real privilege to work with him.

I would like to thank **Prof. Varsha Apte** for helping me in better understanding and answering all my doubts regarding the project whenever needed. Her suggestions were useful throughout the project.

I would also sincerely like to thank **Ram Pangeni** for assisting me and helping me whenever needed in the entire duration of the project.

Anwesha Das
I. I. T. Bombay
July 9^{th}, 2012

Abstract

Researchers have shown virtualization to be a lucrative approach for improving overall system's resource utilization. This enables data centres to multiplex its resources across several hosted applications. In this context Server Consolidation and Load Balancing are two key areas being studied in the academia and industry. Evaluation of solutions for these tasks is needed to understand the efficacy of one over the other, which needs a system set-up for instrumentation and analysis. In previous work, we see how every group of researchers formulate their own algorithms and highlight the performance benefits over the already existing ones. Every system set-up is designed keeping in mind the formulated algorithm. The set-up may be tightly coupled to one particular algorithm and may not suffice for the others. Also no methodical quantitative comparison of such algorithms is done yet. This is important since different algorithms with different formulated metrics have different aims. It is not clear which one works better and in what circumstances. To do that, we need to perform a structured study of few carefully chosen algorithms, on a generic set-up which has the essential features of monitoring, profiling and storage of statistics to facilitate quick prototyping and comparison.

Our work involves the comparative study of algorithms by developing a flexible tool which aids in rapid prototyping of VM migration algorithms via the use of feature-rich APIs through experimentation. This thesis elucidates the comparative study of three algorithms on such a tool developed by us. We discuss the design and implementation of such a framework where rapid prototyping of algorithms can be easily achieved and then discuss the empirical evaluation of prototyped algorithms on it. Designing of API based software framework, prototyping of three VM migration algorithms with a comparative study for understanding their behaviour, is our project contribution.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the advent of Virtualization Technology in data centers, new techniques have been formulated by researchers to govern the resource management and administration. In a virtualized cluster environment, Server Consolidation and Load balancing are some of those techniques which have gained paramount importance for on-the-fly resource management. In a virtualized cluster, many applications run on a virtual machine(VM) and one or more VMs are mapped onto each physical machine of the cluster. Migration of a fully functional and running virtual machine across distinct physical hosts is referred to as live migration. Server consolidation is an approach to reduce the total number of servers used in data centres while eliminating hotspots and coldspots that arise in physical machines at data centersas as discussed in *Section 1.2*. The use of dynamic live migration of VMs has enabled more effective sharing of resources over many hosts by performing server consolidation. Live VM migration can also help in balancing the overall system load in virtualized clusters. Inorder to ensure good system performance it is essential to place VMs on underutilized machines instead of overloading any system. Load balancing heuristics try to do the same. The presence of different kinds of VM migration heuristics, different ways of quantifying metrics considering different parameters, and input workloads considered necessitates the need for a systematic comparison. Analysis of such techniques of server consolidation and load balancing requires a common platform which is absent. This fuels the need to perform a systematic study of consolidation algorithms on a designed framework, where such activities can be seamlessly performed without bothering much about the basic pre-requisites. In the following sections, we have provided a brief background and overview of the above mentioned techniques.

1.1 Challenges of Data Center Administrators

Data Center owners host applications from different enterprise clients on a shared pool of resources. These resources include virtualized physical machines which host applications, shared storage devices like NFS (Network File Storage), backup servers etc. As the workload on these applications vary from time to time resulting in varying resource requirements, dynamic efficient use of these shared resources is a major technical challenge for them.

A contract in the form of Service Level Agreements (SLAs) is negotiated by the clients with the service providers in which the service provider(data center owners) gives some application performance guarantees about the hosted applications. Such performance guarantees may include expected average response time, average cpu usage, maximum downtime and average through-

put of the application. Violation to such SLA guarantees invites penalty. In any case, performance degradation leading to dissatisfied clients may ultimately incur financial loss for the service providers. Over-provisioning of resources can ensure availability of resources and hence prevention of SLA violations but it leads to inefficient resource usage and expensive resource management. Data center administrators usually target to make their customers happy by adhering to SLA conditions, which implies preventing SLA violations as much as possible.

The main culprit for such SLA violations is the time-varying workload dynamics. The applications serving requests have varying levels of workloads. Sometimes bursty data may barge in consuming more resources than required on average, again sometimes in low load situation, the resources may be under utilized, there may be cyclic workload conditions, where the workload changes periodically between high and low, finally high load conditions may exist in certain times when the resource consumption remains fairly high for considerable duration. In all such circumstances utilizing the data center resources optimally to improve usage as well as adhere to SLA conditions becomes a challenge for the data center administrators.

Figure 1.1 shows variation in number of requests arrived at 18 commercial web sites over a period of one week. We can infer from the figure that there is a lot of variation in the workloads served by the websites over time. Many of the webservers hosting these websites are underutilized during night-time and during weekends/holidays as can be seen from the figure. Thus, allocation based on those instances will keep the servers under-utilized for most of the time.

Hence, researchers have formulated many heuristics which try to reduce operational costs of data centers by consolidating servers and maximizing resource usage, preventing SLA violations by dynamically reallocating resources using live virtual machine migration. All these heuristics try to alleviate the problem of over-provisioning which wastes resources, improve resource usage and thereby maintain SLAs which keeps the customers happy. We shall use heuristics/algorithms as described in *Section 1.3*, in this write-up to mean the same.

1.2 Server Consolidation and Load Balancing

Sometimes multiple under-utilized servers consume more resources compared to the workload being serviced. This leads to ***server sprawl*** which refers to the usage of multiple under-utilized servers in data centres incurring high system management costs. To prevent ***server sprawl***, server consolidation aims at reducing the number of server machines used in the data centres by consolidating load, enhancing resource utilization of physical systems along with provision of isolation & security of the applications hosted. Such heuristics try to pack as many VMs as possible in a physical machine(PM) considering the availability of resources in the PM which houses the VM. Thus all heuristics formulated to aim at server consolidation are some variation of the traditional "bin packing" problem. In our work, we define server consolidation as the procedure to reduce the number of PMs by packing VMs as much as possible while mitigating hotspots and colspots that arise at the physical machines due to varying workload dynamics.

Load balancing heuristics try to balance the overall system load. This means it prevents physical machines to get overloaded by moving VMs to underloaded PMs. Load is a crucial metric which can be defined in various ways. How we quantify load determines the efficacy of the algorithm and its accuracy in predicting future load.

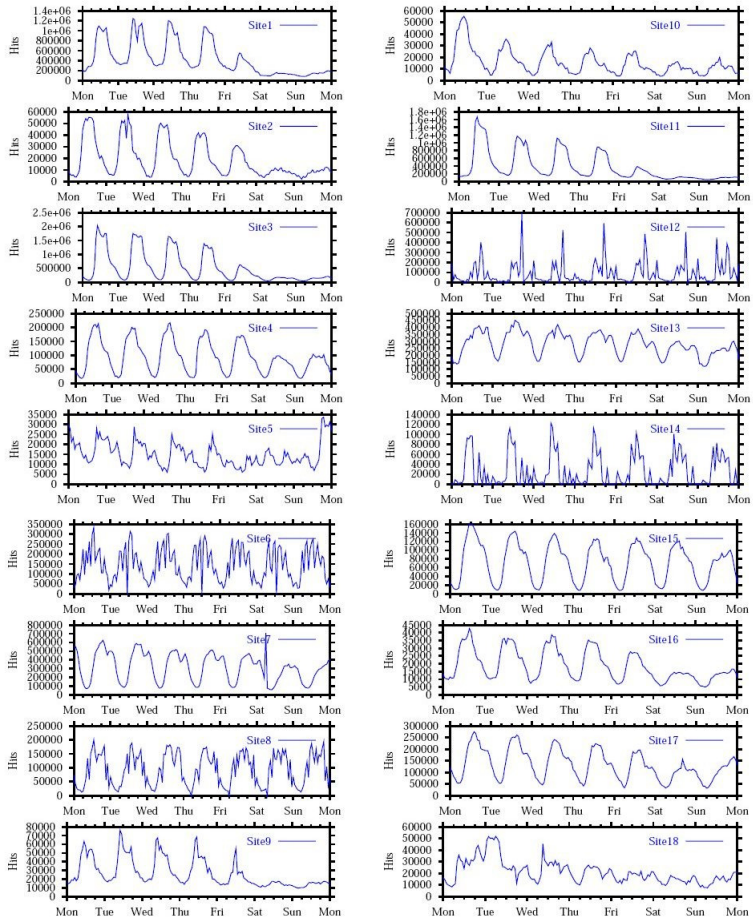


Figure 1.1: Workload Variation over a period of one week for 18 commercial websites: Source [25]

Both the above mentioned techniques to improve system performance requires *dynamic VM migration* which we discuss in *Section 2.1.* Most researchers go through the tedious process of designing their own set-up to test their formulated algorithm. Our work attempts to lessen their burden by generalizing the set-up without making it tightly bound to any specific algorithm and catering to the requirements of other operations in the resource pool like starting a VM, allocating resources to it etc. We have also selected three algorithms which target server consolidation and thus hotspot mitigation for comparison. We have prototyped them using our software framework and tried to analyse the algorithms in face of varying workloads.

1.3 Heuristics/Algorithms

Several heuristics have been formulated to target server consolidation and load balancing. We have done a thorough study of those which we discuss in ***Chapter 2*** and have decided to do

a comparative study based on three chosen algorithms. In computer science and optimization, a heuristic is a "thumb rule" developed from past experience. In the context of server consolidation, we have come across the popular "bin packing problem" of computer science which is classically known to be NP-hard problem. Researchers have used "heuristics" to improve the efficiency of optimization algorithms, either by finding an approximate answer when the optimal answer would be difficult or make the algorithm run faster. Nevertheless, NP-hard problems in theoretical computer science make heuristics the only viable alternative for many complex optimization problems which are significant in the real world systems research.

In our context of work, "algorithms" entail the entire process of using the formulated heuristics to achieve goals of server consolidation, load balancing or only hotspot mitigation. Our work chooses three such algorithms which aim at server consolidation and hotspot mitigation, and tries to study their behaviour. As mentioned earlier, we shall use heuristics/algorithms in this write-up to mean the same.

1.4 Thesis Goals

Our primary aim is to do a comparative study of three algorithms trying to perform server consolidation. The need to perform server consolidation arises in data centers to prevent server sprawl and reduce operational costs by better multiplexing and utilization of resources.

To focus on comparison of server consolidation algorithms we need a software framework which does measurement,monitoring & cluster management activities. The provision of good API support from the framework can help in easy prototyping of algorithms which we want to compare. Thus, our work has three main objectives:

- Development of a measurement & monitoring framework with API support
- Prototyping and validation of algorithms whose comparative study we intend to do and
- Benchmarking and experimentation with workloads using those algorithms for quantitative analysis.

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 provides an overview of the volume of existing literature on dynamic resource management using live virtual machine(VM) migration. This chapter further describes the major classification or categories of live VM migration applications, all of which intend to reallocate resources dynamically inface of changing workloads. Chapter 3 describes the motivation, problem statement and the thesis contribution, which includes the work done in stage-1 as well. Chapter 4 describes the implementation specific details of the development of the software framework, the prototyping of algorithms, their verification etc. Chapter 5 describes the Application specific details, what applications were used for workload generation, how varied loads were generated for experimental runs etc. Chapter 6 discusses the experimentation test-bed, hardwares and softwares used, experimentation process, experiments performed, their evaluation and the results obtained. Finally, chapter 7 provides conclusion with some possible future directions of this project work.

Chapter 2

Survey of VM Migration Algorithms

In this chapter, we describe what is live migration and provide a brief overview of all the areas where live migration is triggered to enable better resource utilization. This chapter intends to provide a holistic view of the wide applications of live migration in multiplexing data center resources and describes ways of dynamic re-allocation of resources inface of fluctuating workloads. This technical survey lays emphasis on how much the live migration process of a virtual machine plays a pivotal role in managing data center resources. We have touched all the facets of dynamic resource management involving live VM migration and have presented a methodical structure of the same.

2.1 Live Virtual Machine Migration

As we know, Virtualization provides a "virtualized" view of resources used to create virtual machines. A virtual machine monitor (VMM) or hypervisor manages and arbitrates access to the physical resources, maintaining isolation between virtual machines. As the physical resources are virtualized, several virtual machines, each of which are self-contained machines with their own operating systems, can execute on a physical machine. A virtual machine, like a physical machine, has associated cpu, memory and disk capacity. Migration of a fully functional and running virtual machine across distinct physical hosts is referred to as live migration as shown in 2.1. Introduced by Clark et.al.[14], live virtual machine migration transfers the memory "state" of a virtual machine from one physical machine (PM) to another facilitating uninterrupted services to the running applications. Also, the disks of virtual machines are stored on some shared storage, popularly NFS(network file storage) accessible from the entire network, for which disks are not transferred.

There are different ways by which this memory transfer of the virtual machine is done, like post-copy, pre-copy, suspend and copy approaches which we donot discuss in details as we are not concerned about that in the context of our project work. Whenever there is a resource crunch on a physical machine which hosts virtual machines, a virtual machine is chosen for live migration. Any live VM migration incurs migration downtime during which the services running on that VM gets affected, hence the lower the better. Due to changing workloads on each of the VMs, whenever resource usage increases, hotspots are said to have developed, and migration of VMs running applications, to some other host help mitigate the hotspots. Similarly, when the

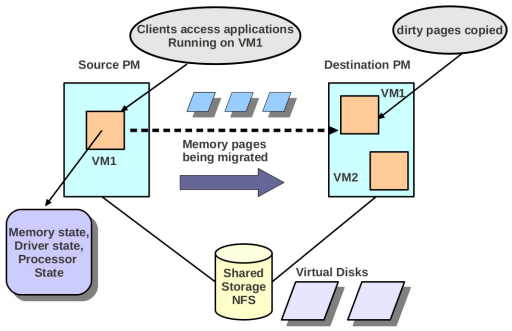


Figure 2.1: Live virtual machine migration mechanism: Source [27]

resource usage levels drop, coldspots develop which can be mitigated, again by migrating VMs on hosts to enable better packing of VMs improving resource usage and reducing the number of physical servers, if possible.

Now, all the algorithms which try to efficiently allocate resources on-demand through live migration answer three questions - a) *when to migrate* b) *which VM to migrate* and c) *where to migrate*. Following table describes them succintly:

Table 2.1: VM migration Heuristics

Goals	Coldspot Mitigation	Load balancing	Hotspot mitigation
When to migrate?	Coldspots on PMs	Load Imbalance on PMs	Hotspots on PMs
Which VM to migrate?	VMs from Lightly loaded PMs	VMs from overloaded PMs	Bunch of VMs from hotspot-PM
Where to migrate?	Higher loaded PMs	Lightly loaded PMs	PM which has enough resources to house

The answer to "*when to start the algorithm?*" is either ***time triggered*** or ***event triggered***. In the former case, periodically PMs are scanned for hotspots or SLA violations, in the latter case they are continually scanned, and whenever found, necessary migrations are triggered based on the algorithm used. We will study in details the goals of migration in *Section 2.3*.

Across all the heuristics, the source PM from where a VM is chosen for migration is the one, where SLA has been violated in terms resource usage/response time etc. or hotspots/coldspots have been formed (discussed under *Section 2.3*). On every host, the VMs are ***sorted*** usually in some order, which may be ascending or descending in terms of resource usage or residual capacity. As per the order, a VM is chosen for migration on a destination PM. The destination PM is selected again based on some formulated metric which is always a function of resource usage. Resource usage determines load on a PM and based on some such metric, the destination PMs are sorted. Accordingly, a PM is chosen which has some spare capacity to hold the VM and VM is migrated on that. Now, this destination PM selection may be based on popularly known First Fit (FF), Best Fit (BF) or Worst Fit (WF) schemes. Needless to say, any PM which doesnot have sufficient capacity to host the VM cannot be a potential destination candidate PM for it.

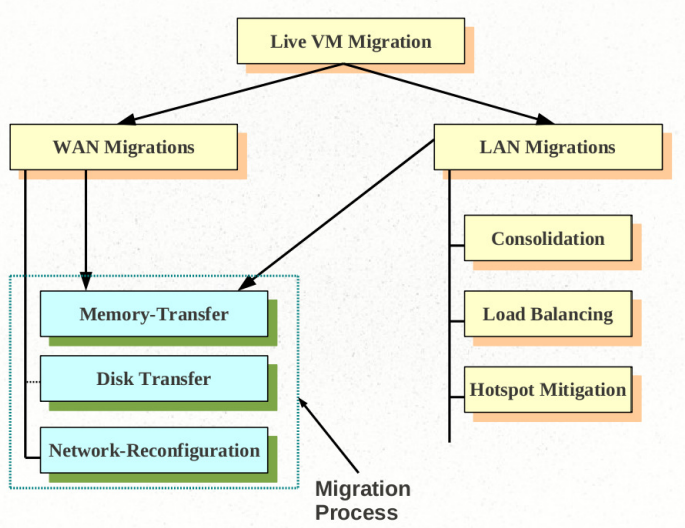


Figure 2.2: Application of live VM Migration

2.2 LAN Migrations

From the above *Figure 2.2*, we can see that live VM migrations can be over LAN which is most popular or over WAN across different geographical locations. In LAN based migrations, the disk images of the VMs are stored on some shared storage like NFS for which the disks never need relocation in the course of VM movement. LAN migration involves the transfer of the memory state of the VM alone. Also since, the VM is moved on a new destination PM on the same LAN, network re-configurations are not needed. But both disk transfer and network reconfigurations are needed for WAN based migrations as migrations are across widespread regions, making it even more challenging. We have discussed WAN based migrations in *Section 2.4*. Unless explicitly mentioned, all reference to live migrations are over LAN only.

2.3 Migration Goals

There are three goals behind any live VM migration over LAN, all for dynamic resource management. Following are those:

- **Server Consolidation** - We have mentioned this earlier in *Section 1.2*. Inorder to reduce *server sprawl* in data centers, server consolidation algorithms are required. These algorithms are VM packing heuristics which try to pack as many VMs as possible on a PM so that resource usage is improved and unused or under utilized machines can be turned off. This saves power consumption thus reducing overall operational costs for data center administrators. But in the process of doing consolidation we have to ensure that, hotspots present if any, are mitigated. If resource usage violation exceeds a pre-defined upper threshold we cannot consolidate without mitigating such hotspots. And as the description implies coldspots or underutilization of resources in PMs can be eliminated by redistribution of VMs from underutilized PMs to other PMs which can hold the VMs. Thus, in our work we define and refer to consolidation as a procedure to reduce the number of physical ma-

chines making sure that all hotspots and coldspots that arise in physical machines are eliminated. Live migration of VMs running applications help in achieving this. Based on varying load conditions under loaded machines having resource usage below a threshold, and overloaded machines having resource usage above a certain threshold are identified, and migrations are triggered to tightly pack VMs to increase overall resource usage on all PMs and free up resources/PMs if possible. In other words, a remapping of VMs on PMs is done thus reallocating resources to the heavily used VMs undergoing resource crunch. [12, 34, 24, 21, 8] are some algorithms amongst the huge body of work done in the context of server consolidation.

- **Load Balancing** - As earlier mentioned in *Section 1.2*, load balancing reduces the disparity of resource usage levels across all the PMs in the cluster. This prevents some machines from getting overloaded in the presence of lightly loaded machines which have sufficient spare capacity. Live migration can be employed here as well. The overall system load can be balanced by migrating VMs from overloaded PMs to underloaded PMs. This reduces the **standard deviation of the imbalance metric** which is usually formulated in terms of functions of some resource usages by researchers. [10, 20, 16] are some references which discuss load balancing heuristics in details.

- **Hotspots & Coldspots Mitigation** - The detection of hotspots and coldspots are always based on ***thresholds*** which are set by the data center owner or based on the SLAs specified by the clients. Usually, a higher resource usage value close to maximum is set as the upper threshold and comparatively low resource usage value below 50% is set as the lower threshold. PMs having resource usage values beyond the upper threshold is said to have formed hotspots, and whose usage values are below the lower threshold are said to have formed coldspots. The former implies over-utilization and the latter implies under-utilization, applicable across any resource dimension. These conditions are inherently taken care of in the above mentioned consolidation and load balancing algorithms. But, there do exist algorithms which only tried to mitigate hotspots without achieving consolidation or load balancing. [36] is one such paper which does hotspot mitigation alone without aiming for consolidation.

There may be algorithms which aim to achieve one or more or any combination of the above goals. VM migration algorithms try to adapt the cluster to changing workload conditions by turning the knobs of resource allocations through triggering migrations. Thus, live VM migration has become an *indispensable* tool for resource provisioning and virtual machine placement in a virtualized cluster. [18, 23, 19] are more such works which one can refer, to know more in details about them.

2.4 WAN Migrations

Several enterprises have world-wide offices and data centers are spread around the world. Migration of VMs running applications across such data centers which are geographically distant poses even more challenges. Refering to *Figure 2.3*, the main differences between virtual machine migration over LAN and WAN are as follows:

- **Disk Storage Transfer** - PMs within a LAN are usually connected via a shared storage system, hence LAN based migrations donot need storage transfer, as wherever the VMs move within the LAN their disks are accessible from within the network. Hence only

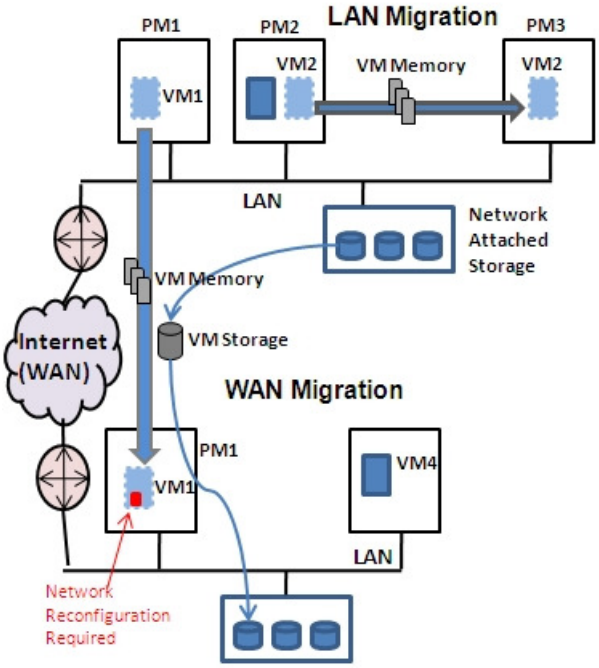


Figure 2.3: WAN and LAN Migration Process :Source [27]

memory state migration is needed in LAN based migrations. In WAN, there is rarely any shared storage device from where disks can be accessible from anywhere across multiple cities. Hence, WAN migration entails not only transferring memory state, but migration of local disk state as well. This if done in a low bandwidth high latency link, incurs considerable migration downtime. Hence, low latency, high bandwidth links are practically essential for such disk state transfers.

- **Network Reconfiguration** - Migration across WANs over different locations necessitates network reconfiguration. Since the VM moves into a new subnet, a new IP address has to be assigned to the VM, as the old address become obsolete. This results in disruption of network connections, and obviates the need for applications to be made aware of the network changes. Thus, WAN migration needs IP reconfiguration seamlessly while moving across different networks to avoid unacceptable performance degradation. Unlike WAN, LAN based migrations donot require any IP reconfiguration as they move to a different PM on the same subnet across the same LAN, without breaking any existing connections.

The challenge of disk storage transfer is accomplished by schemes based on pro-active storage replication and use of content-based hashing (for redundancy elimination) as discussed in [37]. Bearing the downtime costs for disk migration whether reactively or pro-actively is inevitable. Redundancy elimination tries to speed up the process of disk transfer by transferring the common blocks just once, but it may be a futile attempt in high latency, low bandwidth end to end links where the disk transfer time is too high to realize minor optimization benefits.

Researchers have proposed some IP re-configuration mechanisms. Bradford et. al, [13] describe DNS-resolutions to do the job. The seamless change in original IP address and resolution of new IP address while the VM migrates across different networks is done through IP tunneling. When virtual machines migrate, they maintain their canonical names and the new IP address is registered with the name server. DNS lookups for the virtual machine based on the canonical name, after migration, will resolve to the new IP address. Tunneling is a mechanism of providing a path across networks/LANs of different IP configurations by taking help of the gateways encountered on the way to the destination network(where the desination host resides). Gateways are provided with tunnel end-points to prevent any intermediate loss of connectivity. But this solution places the burden on managing end-points on the applications, i.e., they need to be aware of the IP address change. An alternative solution to address the re-configuration issue is to address the problem in the network protocol stack as proposed by Cloudnet [37]. They employ combination of Layer 3 Virtual Private Networks (VPNs) and Layer 2 Virtual Private LAN Service (VPLS) to provide end-to-end routing across multiple networks and to bridge LANs at different locations. The unified virtual network provides the view of a virtual LAN for migrating virtual machines, where virtual machines maintain a single IP addresses all through.

Although WAN migration incurs lots of overhead and operational costs, they are beneficial in the following situations:

- **Collaborative Projects** - Researchers across the world can work on the same project by using the same VM containing the project without maintaining replicas thus reducing storage overhead and eliminating consistency issues.
- **Time biased Relocation** - VMs can migrate to certain locations during certain times of the day to be "closer" to specfic users in a geographic region based on requiements and intensity of traffic in a region.
- **Maintenance and Upgradation** - While maintenance operations process on one site, VMs can be moved from that site to another, to keep services running on them.
- **Cloud Bursting** - When a data center has resource crunch or is saturated, WAN migrations can facilitate VM hosting at some other data center or remote infrastructures.

2.5 Techniques of migration

To achieve the goals of migration as dicussed in *Section 2.3* inorder to dynamically manage resources, there are certain techniques based on which the VM migration algorithms can be classified. Based on our study we have come up with the following major types:

- **Affinity aware** - This category of heuristics try to optimize some resource usage like network or memory. If the hotspot is the result of network resource crunch in PMs, then placing two VMs communicating with each other on the same PM, will reduce the network overload. One way to do this is to monitor and capture the intra-PM and inter-PM network traffic for a physical machine. If the latter is more, the VM residing on that physical machine is migrated to the new physical machine with which it frequently communicates. Similar idea can be applied for memory by co-locating those VMs which share memory pages. The co-location of the memory sharing VMs also improve the migration efficiency as the common memory (used by both the VMs) is required to be transfered only once. [18, 34] are some such work.

- **Workload aware** - This strategy tends to capture the type of workload used while migration decision making. Workload aware migration mechanisms typically consider the different workload mix (cpu or I/O intensive etc.) while triggering migrations for dynamic resource provisioning. [22] discuss such a strategy. Figure 2.4 illustrates the performance of various workloads when mgrating to VMs containing different kinds of workloads. *Baseline* refers to migration in an idle VM running no applications.

	Baseline	**Mig. to Jbb VM**	**Mig. to File VM**	**Mig. to Database VM**	**Mig. to Web VM**
SPECjbb (bops)	14849	11996	13770	14069	12820
IOzone (Kbytes/sec)	414560	243604	109350	142072	213602
Sysbench (s)	88.10	110.96	126.86	97.04	117.32
Webbench (Pages/min)	1994840	1565262	1677740	1807098	1115668

Figure 2.4: Performance comparison of different workloads :Source [22]

From Figure 2.4 we decipher that all the workloads achieve bad performance when they migrate to their own "type" except Sysbench. Because of single resource contention it may not be befitting to migrate to PMs hosting workloads of the same type as of incoming VM. SPECjbb is CPU intensive, when migrating to another SPECjbb virtual machine, cpu will be the bottleneck although other resources may have spare capacity. Sysbench database application gets good performance as it doesnot consume only one kind of system resources. We also observe that co-colocation of SPECjbb and Sysbench workloads produces least performance loss, while IOzone and Sysbench workloads are least suitable for co-location. SPECjbb is CPU intensive while Sysbench is I/O intensive consuming a small amount of CPU, consolidating the two virtual machine workloads together can make decent use of various system resources. But, IOzone and Sysbench are both I/O intensive, consolidating them together leads to I/O performance bottleneck. Moreover, IOZone may be hosted on a separate PM since it may undergo heavy or slow I/O operations to prevent other co-located VMs to get affected. Such kind of analysis are usually considered in workload aware migration strategies.

- **Power/Energy aware** - Power aware techniques have also been formulated to save power by estimating or modeling power consumption during migration and trying to minimize power consumption while formulating migration algorithms. A separate power management module is usually incorporated in the migration framework which provides power consumption estimates based on the resource usage levels of specific applications used. The types of applications used and variability of workload with time determines the range of power usages and impacts the power aware placement decision. Usually a power minimized allocation is provided in such heuristics. Sometimes software scaling and hardware scaling techniques are employed to provide soft power states on Xen Hypervisor to capture application specific power-performance policies. [8, 30] explicitly estimates power consumption and describes them in details.

- **Multiple VM Migrations** - We have all throughout discussed heuristics which trigger a single VM migration. Pro-actively tracking identical contents of co-located VMs and transfering those contents only once while migrating all those VMs simultaneously to another PM is something known as "Gang Scheduling" of virtual machines. It optimizes both memory and network overhead of migration. Such mechanisms can be fruitful when an entire rack of servers have to be evacuated and all the co-located VMs running on them have to be shifted to a different location. [15, 22] investigates such gang migration strategies.

One or more different combination of such techniques may be employed in the migration algorithms to achieve the goals of consolidation, load-balancing and hotspot mitigation.

Thus, leveraging live virtual machine migration for dynamic resource management for changing and unpredictable workloads is indeed a lucrative mechanism. Based on our extensive literature survey, we have highlighted the various migration techniques used for resource provisioning and placement laying emphasis on, resource optimization, reduction of energy consumption, prevention of SLA violation and performance degradation, consideration of workload/application types, all of which are underlying requirements of the migration goals itself. Such techniques help to achieve the migration goals of consolidation, load balancing and hotspot mitigation for on-the-fly adaptive resource management. In today's data centers, live migration is indeed an incredible way to perform adaptive cluster management.

2.6 Choice of Algorithms for Our Comparison

For our work of consolidation we have chosen three algorithms. First is *Sandpiper* [36] which aims at hotspot mitigation alone and not server consolidation. Since this heuristic doesnot take into account consolidation, we are unsure of its performance when compared with other consolidation algorithms. Also, major consolidation algorithms do hotspot mitigation which is Sandpiper's sole objective. The reason behind choosing *Sandpiper*, is its simplicity in terms of implementation on our framework, and to investigate its relative behaviour amidst other consolidation algorithms. It is expected to efficiently mitigate hotspots but we are curious to figure out its behaviour in the comparative study through various experimentations.

The remaining two algorithms aim at server consolidation. Second is *Khanna's Algorithm* [24]. They have considered the residual capacity of the PMs while deciding about migration and have accounted for both hotspots and coldspot mitigation. They intend to keep the resource usage levels of the PMs within specific bounds while achieving efficient packing of VMs. This seemed a good choice for comparative study, simple to implement. It unlike sandpiper considers coldspot mitigation as well. Third is *Entropy* [21], which uses a Constraint Satisfaction Programming based technique to figure out the minimum possible PMs needed to house the VMs. Subsequently, it does VM placement by trying to minimize the migration costs in the process. We chose this as its approach is different from *Khanna's Algorithm* and a comparative study should incorporate algorithms, which use diverse techniques but have identical goals. We have discussed all three algorithms in details in *Section 3.4.2*.

Since these algorithms are widely different in their migration heuristics, we have carefully chosen them, conjecturing that they will be good candidates of comparison. Choosing very similar algorithms for comparison is not a good option as intuitively they are expected to behave in a similar manner. Moreover, deciphering the trade-offs in such cases, may lack clarity because of the similarity in the nature of heuristics. Rather, algorithms with same aim but relatively disparate approaches are good choices of comparative study. They may facilitate in investigating the conditions when or where one works better than the other. With these hopes we have decided on these three algorithms. Also, needless to mention, an obvious fact is that it makes no sense to compare algorithms whose goals are different like load balancing and consolidation. Considering the time constraints for project completion, the above selection seemed feasible to achieve.

Chapter 3

Problem Statement

In this chapter we have provided the motivation behind our work, related work, followed by the problem statement and contributions.

3.1 Motivation

Several algorithms [12, 34, 24, 21, 8, 36, 10, 20, 16] have been formulated by researchers trying to efficiently prevent server sprawl or ensure non-disruptive load balancing in data centres. Such algorithms have different criteria to answer to questions like *when to migrate, which VM to migrate and where to migrate*. To answer the above questions, different algorithms formulate different metrics which primarily is a function of resources considered. Efficiency of an algorithm depends upon the resource parameters and metrics considered. It is not straightforward to state which algorithm is better than the other and in which case one might be beneficial because an algorithm which tries to minimize system load will have different evaluation metrics over the one which tries to consolidate servers. Cost of migration is also captured in different ways. Migration Cost can be captured in terms of number of migrations triggered, the cpu or memory utilizations of the VMs which are migrated, the amount of load transferred per unit bytes, migration downtime etc. All the algorithms try to minimize the migration cost or migration overhead. Since this cost metric is different for different algorithms, we have to carefully analyse the migration overhead when we try to compare the algorithms. Evaluation metrics like number of migrations triggered, number of hotspots mitigated, number of PMs used, time taken to achieve consolidation are considered in our work and are very relevant in this context. Hence a structured comparative study is needed to analyze their applicability, goodness and incurred overhead. This instigates the need to methodically compare the various existing VM migration algorithms which have a common aim. We have considered this aim as *server consolidation*.

To perform such comparison we need to have a common set-up on which we can prototype them and run various experiments to compare them. We also need to ensure that all the other physical parameters including the resource pool set-up remain consistently same. Researchers always designed their own heuristic specific set-up. This is fine as each of the research groups have aimed at formulating their own heuristic to improve system performance. But can we have a common set-up which can prototype all the above algorithms? The challenge is to identify the common aspects needed for them and provide a common substrate. Periodic measurement, monitoring & provisioning of resources, history maintenance are some elementary operations required in such a set-up. This led to our problem formulation. The lack of a flexible experimental set-up

in the academic setting to perform various migration related tasks and the need for comparison of several heuristics on the same set-up motivated us to come up with a common platform and quantitative study. Presence of such a platform will aid multiplexing of many functionalities on the same set-up with no changes in the underlying measurement and control modules. At the application level itself, with the API support, specific operations can be prototyped with convenience, making the application developers job much easier than before.

Thus, with primary focus on comparison, we made attempts to come up with our own open-source software tool and used it for prototyping chosen algorithms (as mentioned in *Section 2.6*) for comparative analysis. HTTPERF and RUBiS applications are used to generate workload and parameters like no. of migrations triggered, no. of hotspots mitigated, no. of PMs used, and time taken to achieve consolidation are considered as evaluation metrics in this work.

3.2 Related Work

Existing research can be classified into (a) Open source cloud computing platforms providing IaaS (Infrastructure as a service) and (b) Algorithms aiming at server consolidation, load balancing and hotspot elimination.

3.2.1 Open Source Cloud Computing Platforms

The first class of literature pertains to the recent releases of open source cloud computing platforms. Xen cloud platform[6] and Eucalyptus[1] are two such platforms which offer IaaS. IaaS refers to the services provided to the users to lease the processing power, network, storage and other basic computing utilities including operating systems and applications. For all these services, the users need not manage or control the cloud infrastructure, including network, server, operating system, storage and even the functioning of various applications. These platforms develop an architecture to perform automation of tasks, resource and cluster management, run cloud applications, without worrying about the basic infrastructure needed to execute them. They attempt to provide scalability and efficiency enhancing private and hybrid clouds within an organisation's IT infrastructure. Since they provide *Hardware as a Service* the service provider is responsible for housing, running and maintaining the platform as well.

Based on our short past experience with these existing tools, we felt that they are little complicated for the purpose of comparative study of algorithms. Different platforms have different trade-offs and limitations which make them suited for environments compatible to their architecture. Morever, the architecture specific configurations to be done and heavy protypying overhead in the context of our work (comparative analysis of migration algorithms) motivated us to develop our own tool. Our work also tries to automate tasks and do cluster management but **without providing hardware as a service**. Our approach follows a simple way to make the prototyping easy for the application developer by providing a lightweight framework.

3.2.2 Algorithms for Server Consolidation/Load Balancing/Hotspot Mitigation

Previous research work [12, 34, 24, 21, 8, 36, 10, 16, 18, 19, 23] have designed frameworks to test performance benefits of their developed heuristics. For example, Sandpiper[36] had their own centralized way of monitoring, profiling and detecting hotspots on a prototype data center consisting of twenty 2.4 GHz Pentium-4 servers connected over a gigabit Ethernet with control plane

and nucleus, Khanna *et al.*[24] had their test-bed of IBM blade center environment with VMWare ESX server as the hypervisor deployed on three HS-20 Blades, Ameek *et al.*[7] used HARMONY set-up with ESX server and SAN (Storage Area Network) controller integrating both server and storage virtualization technologies to design an agile data center. Memory buddies[34] colocated *similar* VMs exploiting page sharing to mitigate hotspots. They too had a control plane with nucleus on the PMs like Sandpiper[36] performing initial VM placement, server consolidation and offline capacity planning using VMware ESX hypervisor. Their testbed is a cluster of P4 2.4 GHz servers connected over gigabit ethernet. Emmanuel *et al.*[10] developed a dynamic resource allocation framework based on their load balancing VM migration algorithm on a test-bed of three ESX servers, a SAN, VMware VC (Virtual Center) and VIBM Migration Handler. Starling[18] did affinity based VM placement and migration in a decentralized approach with a 8-node cluster of 2xdual-core AMD machines. Kellar *et al.*[23] proposed *Golondrina* multi-resource management system for operating system-level virtualized environments with client nodes, manager server and cluster gate. Autocontrol[33] proposed an automatic resource control system using two test-beds for evaluation, first consisting of HP C-class blades each equipped with two dual-core 2.2 GHz 64 bit processors and second Emulab-consisting of pc3000 nodes with single Xeon 3 GHz 64 bit processor as servers and pc850 nodes as clients.

Except Starling[18] and Autocontrol[33], most of the set-ups rely on centralized architecture. Our work closely resembles the work done by Sandpiper[36] and Memory Buddies[34]. We design a framework which does monitoring and control. Memory Buddies[34] exploits API support in the control plane to discover active hosts and initiate VM migration between hosts. Our work also leverages API support at the Management node to perform control operations. In addition, we stress on the agility of the set-up as we target making applications loosely coupled to the underlying infrastructure.

[36, 24, 10] consider cpu, memory and network, [7] additionally considers I/O, [34] considers only memory, [18] considers only network, [23] consider only cpu and [33] considers cpu and disk. [36] perform hotspot mitigation, [24, 21] and many more perform consolidation, [10, 20, 16] perform load balancing, [23] compares replication and migration mechanisms to do the same and [18] performs affinity aware migration to optimize communication overhead.

Such different kinds of algorithms considering different parameters and set-up performing different jobs obviates the need to perform a structured analysis to find out which algorithm should be used when, which performs better and under what cases etc, after we fix the aim of the algorithms under consideration. As said earlier, this complicated systematic study needs a common platform to efficiently find out the trade-offs.

3.3 Objectives

Given the necessary background and motivation following are the project's objectives:

- Design and implementation of a measurement and monitoring framework
- Development of API support for application users to enable rapid prototyping
- Selection, implementation and validation of algorithms for comparison
- Benchmarking and experimentation using those algorithms
- Comparative Analysis of results

The first two objective's were accomplished in the first phase of work. Software framework development was a joint work with Ram Pangeni. The rest of the objectives are done in the second phase.

3.4 Thesis Contributions

To achieve the first two objectives we have designed and developed a software tool whose architecture and description is as follows:

3.4.1 Design and Implementation of a Measurement & Monitoring Tool

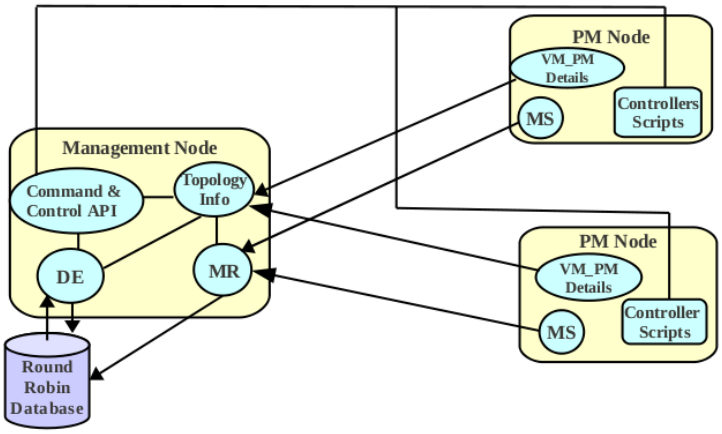


Figure 3.1: Architecture of the Framework

The system architecture of Sandpiper[36] forms the backbone of our preliminary design. Wood et al.[36] have followed a centralized approach for threshold based hotspot detection. It comprises of the control plane which profiles resources received from PMs and triggers migrations to eliminate hotspots. We too follow a centralized approach. We provide a modular and flexible framework for regular measurement and monitoring, history maintenance in a database for future prediction and topology maintenance. Moreover we provide APIs for quick prototyping of heuristics. The resource parameters we considered are **cpu, memory, disk** and **network**. In addition, we consider **response time** as application performance parameter.

We briefly mention the key components of our architecture:

- **Management Node**(MN): This is the controller node in the set-up which co-ordinates the PMs in the cluster. This node maintains a Round Robin database (described later) for the storage of resource utilization statistics and history maintenance. The history can be used for profiling and prediction of resources. It receives resource usage and topology update information from all the PMs, triggers various control operations like resource control, domain control and migration control. It also has the user interface with API support which means application developers and users of the framework can interact with the MN

for VM related operations without having to contact the PM or VM directly. MN retains the necessary details needed to communicate with a VM or PM at any instant of time. MN has three main sub-components:

  - **Measurement Reception (MR)** - This is needed for the reception of various resource usages sent by the PM.
  - **Topology Info** - This is required for the reception and storage of static details like PM/VM IP addresses, which VMs reside on which PMs, their allocated cpu amd memory, to generate the topology of the set-up.
  - **Decision Engine (DE)** - This contains the usage of APIs to trigger several operations like prototype migration heuristics or perform resource setting/VM management.
  - **Command and Control** - This is required for the user interface which again uses APIs to trigger several VM management operations.

- **PM node**: It is a physical host with XEN hypervisor installed which periodically collects various resource usage statistics and sends it to the MN for storage and maintenance. It has the **controller scripts** to trigger several operations related to resource control or domain control of VMs whenever instructed by the MN. PM node has two main subcomponents:

  - **VM_PM Details** - This is required to resolve IP addresses of VMs/PMs and collect details like allocated cpu/memory to send them to MN for topology maintenance.
  - **Measurement agent** - This is needed to have the measurement sender(MS) which collects statistics of resource usage and sends them to the MN.

- **RRD Tool**: [4] is an open source standard of logging and graphing of time-series data. Round robin is a technique that works with some fixed amount of data, along with a pointer to the current element in a **circular** buffer. When the current data is read or written, the pointer moves to the next element. Since circular queue has no start or an end, we can go on and on. When all the available places get used, the process automatically **reuses old locations**. Hence the name round-robin. This way, the dataset never grows in size and therefore requires no maintenance. RRD tool works with **Round Robin Databases** (RRDs). MN has the RRD installed to store measurement data it receives from the PMs. We used this tool in our work primarily for the following reasons:

  1. This tool requires time-series data which means it can store numerical values of data measured at several points in time. RRD can also store multi-time granularity statistics. It aptly suits our requirements to store history of resource usages like memory, cpu, disk, network & response times of machines thus capturing the dynamic time varying parameters in our framework.
  2. There is no extra overhead of database maintenance for developers since this task is handled by the RRD tool itself. The tool aids in database creation and storage once we feed the values through the tool, by providing commands to do so.
  3. We can conveniently use the tool to store and retrieve data averaged over some user specified time without any manual calculation whenever needed at the MN. This eliminates the computation overhead for users to calculate aggregate data since RRD itself does the computation.

  We have created a directory structure in the RRD to store the PM/VM statistics. A directory is created based on the unique IP address of PM/VM and the measured statistics

is stored in it. We decided on a **single level directory** structure instead of having subfolders of VM statistics inside PM folders, to avoid the restructuring of directories after migration to reflect the topology change. VM movement collapses the two level structure since the logs of a VM which moves to another PM should be also shifted to the destination PM's directory.

Each of the modules will be described in the next section. We designed the framework with the MN as the central controller feeding measured resource usage data into Round Robin database along with topology maintenance. Since the primary user interface is at the MN we incorporate the API support there.

Protocols and Modules of the framework

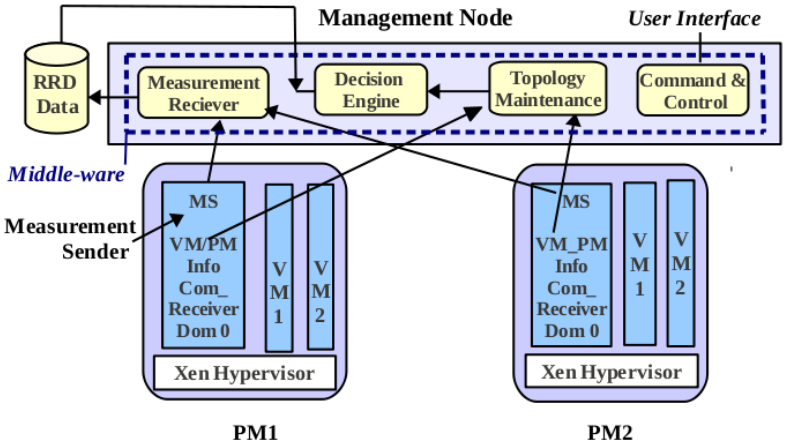


Figure 3.2: Interaction between components of the Framework

Figure 3.2 shows the interaction of various components in our framework. It is clearly seen as to how the **middleware** formed in the MN helps to co-ordinate different activities in the cluster. Measurement receiver in MN receives all resource usage dumps from the measurement senders of PMs, topology details received is also maintained in the MN using python dictionary structure for maintaining up-to-date topology of the resource pool. It contains the database storage, the decision engine to take decisions about migrations and the command and control module which has the user interface to trigger various control operations. API support aids the management and prototyping at MN. We have used XEN hypervisor[31] for our work, hence descriptions are given in the context of *Xen* for our work.

The major protocols developed for this framework are briefly described below:

- **Topology Update Protocol** By using the output of the *xm list* command in xen, We have formulated topology update protocol to reflect the change in topology at the MN whenever a VM/PM goes down or comes up or VM migration occurs. As we know *xm list* outputs the currently hosted VMs on that PM with vm-name, domain-id, no.of vcpus allocated and the amount of memory given to the VM. Initially, a sampling interval say t is provided by the user and the measurement statistics are sent from all the PMs periodically

- time period being equal to the sampling interval t. Now, we parse the output of *xm list* every t seconds, tracking the presence of VMs, to keep the MN updated at the granularity of the sampling interval. We donot perform any updation within the time window of t secs. We assume that, too many migrations/topology change in that t secs is unlikely to occur, which if not immediately reflected can affect the system adversely. Choosing appropriate t can make the set-up more efficient, for example 10 minutes may be high in case of dynamic time-varying workload and 5 secs may be less incurring unnecessary communication overhead. Currently this value is left as part of the user specification.

- **Push protocol - collection of measurement statistics** Who should initiate the data collection process, should the MN pull information from the PMs or should the PMs themselves push collected data to the MN? We decided on the latter because of the following reasons:

  - Our framework relies on **periodic** sampling and monitoring of measured data. In this case it makes sense to use *push protocol* since it avoids the MN's overhead of polling all the PMs periodically to collect measurement data. MN should be in listening state. This reduces the communication overhead at MN in querying for data at all the PMs.
  - Push avoids the need to ensure that only running PMs report data to MN. For pull protocol to work MN needs to find out the PMs which are up and running from the topology and query for data at those PMs only. In the mean while if a PM comes up, its statistics may get delayed in getting reported at MN because MN needs to explicitly keep track of the active PMs which are running and pull data from those PMs only. This adds an extra overhead of selectively communicating with the running machines considering the fact that the set of running PMs can change with time. Whenever a new PM is added, its provided with MN's IP address and port. The new PM also starts pushing data to the MN at specified time intervals.

  We found the push protocol more sensible in our framework and easier to implement compared to the latter. Ofcourse in any centralized set-up inherent central node failure problem exists. But as long as MN is working, *pushing* data from PM is more convenient than *pulling* data from MN.

- **Management Node Module** This module is run only on the manager node in the framework which co-ordinates and manages the cluster. It contains RRD Logs which tracks details of measurement statistics for each PM and VM based on its unique IP. It has a single level directory structure with VM and PM logs on the same level (instead of having subfolder logs of VMs in a PM) to prevent unwanted discrepancies during migration of VMs. This module recieves resource information from the PMs, maintains topology and triggers various control operations like resource control, domain control and migration control. Its main sub-components are as follows:-

  - **Measurement Reception**: This contains scripts pertaining to reception of measured statistics and their storage into RRD with time-stamps.
  - **Topology Maintenance**: This has scripts for reception of PM/VM information (which PMs house which VMs etc) for topology maintenance and updation.
  - **Decision Engine**: This module has APIs to fetch the required data from RRD, to trigger control commands at the specified PMs & scripts to prototype any operation using the APIs needed.

  - **Command and Control**: This is the main user interface module which enables user to perform various actions like resource control, VM control, migration control by selecting a certain action with specified inputs. It works by using the APIs defined in the framework.

- **PM Node Module** The scripts inside this module is run on dom-0 of all the PM nodes. It sends topology & resource usage related data to the MN. Its main sub-components are as follows:-

  - **Measurement Agent**: Periodically gathers resource utilization statistics by parsing the output of *xentop* command, then selectively reports usage data of those resources only, which are asked for by the MN, with time period being same as the *Sampling interval*.
  - **VM_PM Details**: Resolves IP addresses of PMs, VMs using *ifconfig*, *nmap*, parses the output of *xm list*, stores the required details and sends them to the MN for topology maintenance.

Tool with API Support

Figure 3.3 illustrates the flexibility and ease provided by the infrastructure developed with APIs for a simple VM migration process. As seen from *Figure 3.3* the IP addresses of PMs and VMs are obtained using *PM_Stats()* and *VM_Stats()* functions. Using the IP address as input, corresponding CPU usage values are fetched from the database using the function:
RRD_Fetch.cpu(<ip_address>,<time_granularity>)
Once the CPU usage values are obtained, one can check if any VM exceeds the specified threshold. If that happens, a PM with the required available CPU is searched for. If not found, migration cannot occur else that VM is migrated using function:
migration(<src_ip>,<vm name>,<dest_ip>)

For more information regarding implementation details and how to use the framework one can go to the following link and refer to the Documentation and Report:
`http://www.cse.iitb.ac.in/~anwesha/MTP/SoftFrame-Version-0.1.tar.gz`

3.4.2 Comparative Study of Algorithms

The second phase of work comprises of the following:

- Extensive Study of VM migration algorithms formulated for dynamic resource management which has been described already in *Chapter 2*.
- Selection of Algorithms and their implmentation - Based on the technical survey, we have chosen three heuristics (discussed in *Section 2.6*) for the purpose of comparison. The implementation details of the comparative study of algorithms are provided in *Chapter 4*.
- Validation of the correctness of the algorithms prototyped - Conducted tests on a small set-up in the lab to verify that migrations were triggered correctly. We have considered very simple test cases to do so.
- Benchmarking with **httperf** and **RUBiS** - Configuration of servers and clients for workload generation using RUBiS and httperf applications have been described in *Chapter 5*.

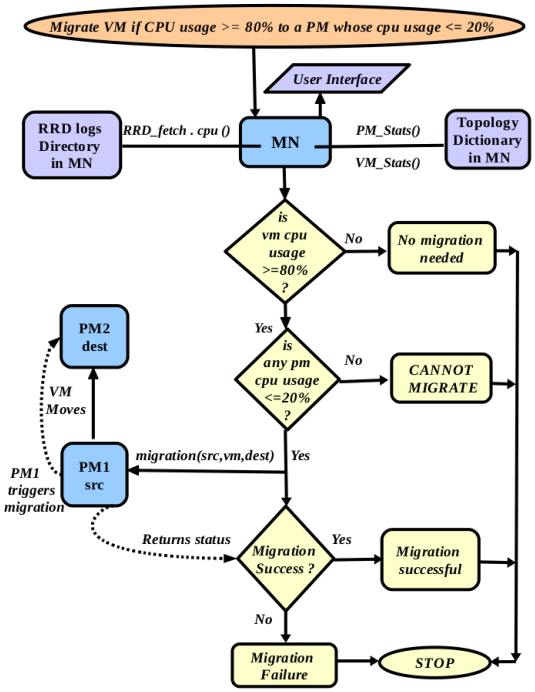


Figure 3.3: API usage for prototying

- Experimentation and Evaluation - The experimentation, evaluation and analysis details pertaining to comparison is provided in *Chapter 6*.

Table 3.1 describes the migration heuristics described below, succintly:

Table 3.1: Chosen Migration Heuristics

Algorithms	**Goal**	**When to migrate?**	**Which VM to migrate?**	**Where to migrate?**
Sandpiper[36]	Hotspot Mitigation	Resource usage exceed thresholds set	Most loaded VM	Least loaded PM
Khanna's Algo[24]	Server Consolidation	Resource usage violate the thresholds set	VM with lowest resourse usage	Best fit PM by residual capacity
Entropy[21]	Server Consolidation	if new config has fewer nodes or non-viable config exists	whichever minimizes reconfiguration cost	whichever minimizes reconfiguration cost

Now we describe each of the three chosen algorithms briefly:

- **Sandpiper** [36] - This algorithm aims at hotspot mitigation to prevent SLA violations. Hotspots are detected when cpu usage values are violated w.r.t the cpu thresholds set. Based on the thresholds set, PMs are classified as underloaded or overloaded. The PMs are sorted based on the descending order of their *volume* metric, and VMs are sorted based

on the descending order of their *vsr* metric, where *volume* & *vsr* are defined as:

$$vol = (\frac{1}{1-cpu}) * (\frac{1}{1-mem}) * (\frac{1}{1-net})$$

$$vsr = \frac{vol}{size}$$

where cpu,memory and network in the above Equation:(3.4.2 refer to the cpu, memory and network usages of the PMs and VMs respectively, and size in the Equation:3.4.2 refers to the memory footprint of the VM. To mitigate hotspot on an overloaded PM, the highest VSR VM is migrated to a least loaded PM amongst the underloaded ones. If the least loaded PM can't house the VM, next PM in the sorted order is checked. Similarly, if the VM cannot be housed in any of the underloaded PMs, next VM in the sorted order is checked. This way Sandpiper tries to eliminate hotspots by remapping VMs on PMs through migrations.

- **Khanna's Algorithm** [24] - This algorithm aims at server consolidation. Application response times are specifed as SLAs and based on an analysis of response time versus load, the authors come up with a formula to map the response times to a cpu usage value, which can serve as the upper cpu threshold. Upper and lower cpu thresholds are set based on which hotspots and coldspots are detected as discussed under 2.3. The PMs are sorted based on the increasing order of their residual capacities across any resource dimension like cpu. The VMs on each PM are sorted according to increasing order of their resource utilization like cpu usage. Migration costs of the VMs are determined based on their resource usage i.e least usage implies least costly migration. Whenever a hotspot is flagged on a PM due to violation of upper threshold, VM with least resource usage is chosen for migration to destination host which has the least residual capacity to house it (best fit policy). If a PM cannot house the VM, next PM in the sorted order is checked. Similarly, if the VM cannot be housed by any of the candidate destination PMs, next least usage VM from the sorted order is checked.

  Whenever coldspots are flagged, the least usage VM across all the underloaded PMs is chosen and migrated to a destination PM, only if addition of the new VM increases the *variance* of residual capacities across all the PMs, else we choose the next VM in order. If there is no residual space left for the chosen VM, then the heuristic for coldspot mitigation stops. *Variance* is defined as follows:

$$variance = \frac{(mean - res_{cpu})^2 + (mean - res_{mem})^2 + (mean - res_{net})^2 ...}{(m-1)}$$

$$mean = \frac{res_{cpu} + res_{mem} + res_{net} + ...}{m}$$

$$r_n = \sqrt{var_{p_1}^2 + var_{p_2}^2 + var_{p_3}^2 + ...var_{p_n}^2}$$

In the above Equation 3.4.2 *mean* is defined as the average of normalized residual capacities across 'm' different resources like cpu, memory, network etc. $res_{cpu}, res_{mem}, res_{net}$....

stands for the residual capacities across different resource dimensions in Equation 3.4.2. Equation 3.4.2 gives the standard formula of variance across all the resources considered. L2 norm r_n is the magnitude of the vector which comprises of the individual variances across 'n' physical machines. It is calculated by the standard formula as mentioned in 3.4.2 where var_{p_1}, $var_{p_2}....var_{p_n}$ refers to the individual variances (of 'm' resources) across 'n' physical machines $p_1, p_2....p_n$.

In this way, Khanna's Algorithm packs the VMs as tightly as possible trying to minimize the number of PMs by maximizing the variance across all the PMs. For more details [24] should be referred.

- **Entropy** [21] - Entropy proposes a consolidation algorithm based on constraint solving which works in two phases. In the first phase, based on the current topology and resource usage of PMs and VMs, entropy computes a tentative placement (mapping of VMs to PMs) and reconfiguration plan needed to achieve the placement, using minimum number of PMs required. In the second phase, based on another set of constraints incorporating optimization of migration cost, it tries to improve the reconfiguration plan by reducing the number of migrations required. They have used *Choco Solver* - a Java based CSP solver to solve the CSPs (Constraint Satisfaction Problem). Also, since obtaining the placement and reconfiguration may take considerable amount of time (as sometimes no optimum value can be found), the time given to the CSP solver is defined by the users, exceeding which whatever intermediate value the solver has computed is considered for dynamic placement of VMs. VMs are classified as *active* or *inactive* based on their cpu usages w.r.t the thresholds set. The authors define a *viable configuration* as one in which every active VM present in the cluster has access to sufficient cpu and memory resources on any PM. For example, if a PM has 4 cores atmost 4 active VMs can be housed on that PM, similarly a uniprocessor machine can have atmost 1 active VM. There can be any number of inactive VMs on the PM satisfying the constraints. The CSP solver takes this *viable condition* into account in addition to the resource constraints, while procuring the final placement plan. For further intricate details of this algorithm, [21] should be referred.

Table 3.2 describes some features of the migration heuristics:

Table 3.2: Features of the Chosen Algorithms

Algorithms	**Metrics Used**	**Resources considered**	**Approach Followed**
Sandpiper	Volume, Volume/Size	cpu, memory and network	Threshold Based
Khanna's Algo	Residual Capacity, Variance	cpu, memory	Threshold Based
Entropy	No. of Migrations, Memory Size of VMs	cpu, memory	Constraint Satisfaction Programming Based

Both Sandpiper and Khanna's Algo use a threshold based technique of triggering VM migrations. Entropy relies on the CSP solver4.4.1 to perform consolidation by providing a set of constraints, optimizing the number of PMs needed to house the VMs and the migration cost to determine the selection of reconfiguration plan.
In Sandpiper the migration cost is in terms of *vsr* metric (mentioned in Equation 3.4.2) whereas

Khanna's Algo considers the resource utilization as the migration cost metric. Entropy formulates the migration cost in terms of the memory allocated to the VMs. All of them intend to reduce migration overhead to prevent application downtime irrespectively. Entropy tries to obtain a *globally optimal solution* unlike the other two algorithms which distinguishes itself in its consolidation approach. Unlike *local optimizations* which sandpiper and Khanna's algorithm does, Entropy considers all the hosts in the topology, and based on their current resource usages, finds out an optimal solution which tries to decrease the migration overhead in terms of memory. The other algorithms try to achieve consolidation on a per host basis, making sure that resource threshold violations are prevented every time each host is scanned, and then the VMs are packed as closely as possible. We shall refer to these three algorithms as *Entropy*, *Sandpiper* and *Khanna's Algo* in the following sections.

Chapter 4

Implementation

This chapter describes the implementation details of the algorithms on the developed software framework 3.4.1. We have prototyped the chosen algorithms using the software tool we developed as has been discussed in details in Chapter 3.4.1. Our test-bed consists of four host PMs and seven VMs. All these four PMs have Xen installed. We have used one machine for the NFS set-up and a couple of machines for the client configurations which are used to generate load on the VMs. Also, a separate PM is configured as the Management Node which runs a specific algorithm in its Decision Engine.

4.1 Prototyping Algorithms

All the algorithms are implemented in python using required packages and libraries. We have used the APIs provided by the tool to fetch the necessary information regarding topology and resource usages in the required format whenever needed. This has helped in prototyping a lot. Here, we discuss in details the way each of the algorithms is prototyped by us.

4.2 Sandpiper

As mentioned before sandpiper is a hotspot mitigation algorithm which migrates VMs from overloaded PMs to underloaded PMs to prevent the PMs from overshooting the thresholds set. We have protoyped sandpiper considering *cpu, memory* and *network*. Following are the some major important points of sandpiper heuristic:

- Whenever the upper cpu threshold is violated the algorithm is triggered. It does nothing about coldspot mitigation. It doesnot do consolidation.
- The VMs are sorted according to the descending order of the VSR metric as discussed in 3.4.2. VSR is volume to size ratio, it attempts to migrate the maximum volume per unit byte. The volume metric captures the degree of overload of a PM or VM. The *migration cost* metric in this algorithm is thus the VSR metric.
- The PMs are sorted in ascending order of their volume, which has been discussed in 3.4.2.

- Whenever a hotspot is flagged, *highest VSR VM* is migrated from the PM with the *highest volume to the least volume PM* whichever can house the VM.
- If no underloaded PM can house the VM, next VM is chosen and scanned, the process is repeated for all the PMs where hotspots are formed.

4.3 Khanna's Algorithm

Khanna's Algorithm does consolidation through both hotspot and coldspot mitigation. It considers the residual variance for packing of VMs. We have considered cpu, memory and network resources. Following are some important points regarding this algorithm:

- The algorithm takes response times as inputs and provides a way to convert them to cpu usage values respectively which can serve as upper cpu threshold. We have taken the upper cpu threshold value directly as the input and not response times. This makes sense as for our comparative analysis purpose, we need to have the same upper cpu threshold for all the algorithms which have a threshold based approach.
- Whenever the cpu usage goes below the lower cpu threshold(coldspot) or higher the upper cpu threshold(hotspot), this algorithm is triggered.
- The VMs are ordered in the ascending order of their resource utilizations. There is a migration cost co-efficient associated with each of the resource dimensions. We have given *migration cost-co-efficient* as 2 for cpu and rest as 1. Thus we have a resource vector and multiplying with migration cost with the resource usage gives us the migration cost of that VM. Thus, lower the utilization, lower the cost.
- The PMs are sorted in ascending order according to their variance of the residual capacitiy vector. We find the residual capacities across all the resource dimensions considered and find out the variance this residual vector through standard formula, as discussed already in 3.4.2. The algorithm aims to keep the residual variance across all the PMs as high as possible.
- Whenever a hotspot is flagged, the lowest usage (hence cost) VM from the source PM is migrated to the PM having the least residual capacity but big enough to hold the VM in consideration for migration.
- If the VM cannot be housed anywhere, the next VM is scanned for migration.
- Whenever a coldspot is detected, the VM with least usage is chosen across all the PMs, it is migrated to a PM which has least residual capacity but big enough to hold the VM, such that the variance across all the PMs is increased. When the variance decreases, the coldspot mitigation heuristic terminates. The variance across all the PMs is formulated based on the individual PM's variance by l2 norm formula(magnitude of vector). as already discussed in section 3.4.2. The *L2 norm* r_n of the vector of all the PM's residual variances is calculated in the following manner using standard formula:

$$r_n = \sqrt{var_{p_1}^2 + var_{p_2}^2 + var_{p_3}^2 ...}$$

where var_{p_1} refers to the residual variance of the 1^{st} PM and so on..

- Whenever the cpu usage is within the thresholds nothing happens. We have set the lower cpu threshold as 20% in all the experiments described later.

4.4 Entropy

This algorithm attempts to compute a globally optimal solution from the information based on the number of PMs, number of VMs, the current topology and the resource usages and resource capacities. To determine whether a VM is active or inactive we have set a threshold as 50. If the resource usage is more than 50% then, the VM is an active VM, else if less than 50% the VM is inactive. We have considered cpu and memory for this algorithm. Following are some major points about this algorithm:

- Whenever there exists a non-viable configuration or based on the current topology the optimal number of PMs that can be used is lesser than the number of PMs being used, the heuristic is triggered. Unlike the other threshold based approaches, it checks whether we can consolidate further, if yes it reconfigures the topology.
- We have discussed about viable configuration in Section 3.4.2.
- Two CSPs (constraint satisfaction programming) have to be solved in case either of the conditions mentioned in the first point is met. The CSPs being solved in this algorithm is an ILP, Integer Linear Programming problem. The unknown variables take values 0 or 1.
- The first ILP obtains the optimal number of PMs needed to house all the VMs existing in the topology, based on the number of PMs, number of VMs, the resource usages and the resource capacites. It also gives a tentative PM-VM mapping which satisfies the optimal number of PMs obtained. *The objective function to be optimzed here is the sum of non-empty PMs.*
- The second ILP takes this optimal number of PMs found by the first ILP as input, takes the current topology or the existing PM-VM mapping as input and provides the reconfiguration plan with minimum migration cost. Thus, the second ILP optimizes the migration cost using the optimal number of PMs found by the first ILP. We get the new PM VM mapping with lesser number of PMs from the second ILP. *The objective function to be optimzed here is the sum of memory allocated to the VMs which needs migration for reconfiguration.* We have dicussed more about this in section 4.4.1.
- The migration cost is captured in terms of memory here. We have created all the VMs with 256 MB of RAM. Migration cost in entropy is the summation of the VM memory for each of the VMs migrated. Thus, lesser the number of migrations, lesser the cost.
- It has no upper cpu threshold below which the PM cpu usage has to lay. As long as a viable configuration is maintained, and the resource constraints are satisfied, one can stuff VMs on a PM.

4.4.1 Python CSP Solver - Google OR Tools

Originally the authors used choco solver in Java to implement Entropy. Their entire framework was in Java. Since we have used python in our framework, we thought of using a python CSP solver which can facilitate the ILP solutions needed in the context of our work. Also, integrating JAVA CSP solver with python is tedious over using a CSP solver in python right away, when there

are already some existing solvers available. The solver needs to have the required functionalities like minimization of a function, other than addition of constraints. After fiddling with some solvers available, we found a relatively new package available from *google*, *Google OR (operation research) Tools* [17]. We solved the ILPs needed for entropy using this solver. We formulated the ILP and coded the three major portions of the ILPs, the variables, the constraints and the objective function. Please refer to the google OR tools site and the entropy paper to understand or know more about these. Following are the set of constraints provided to the solver for solving the ILPs of entropy algorithm.

Let there be 'n' number of PMs and 'm' number of VMs in the system. Let the cpu capacity of i^{th} PM be denoted by ${C_i}^p$ and memory capacity be denoted by ${C_i}^m$. Let the cpu usage and memory usage of j^{th} VM be denoted by ${R_j}^p$ and ${R_j}^m$ respectively. 'i' is the index variable for number of PMs ranging from 0 to n and 'j' indexes number of VMs ranging from 0 to m. The first ILP in entropy needs the following, variable declaration and the set of constraints:

- **Variables** - Two binary variables h_{ij} and u_i are declared which are described below. The former determines the tentative reconfiguration plan i.e mapping of PMs and VMs using the optimal number of PMs. The later determines whether or not a PM is empty.
  - h_{ij} binary variables. If $h_{ij} = 1$ then i^{th} PM hosts j^{th} VM where 'i' ranges from 0 to n and 'j' ranges from 0 to m. If 0, then j^{th} VM is not hosted by i^{th} PM.
  - u_i binary variables. If $u_i = 1$ then i^{th} PM hosts atleast 1 VM, if 0 it is an empty host.
- **Constraints** - We make sure that the resource constraints are met, the summation of VM usages should not overshoot the PM's resource capacity. All the VMs have to be placed in exactly 1 PM. The condition of viable configuration should be satisfied. Finally, atleast 1 VM needs to reside in say PM 'i' for u_i to be 1. To summarize we have the following:
  - $\sum {R_j}^p * h_{ij} <= {C_i}^p$ for all i and j in 0 to n and 0 to m respectively.
  - $\sum {R_j}^m * h_{ij} <= {C_i}^m$ for all i and j in 0 to n and 0 to m respectively.
  - $u_i - h_{ij} >= 0$ for all i and j.
  - For every i^{th} PM $\sum h_{ji} == 1$ for all i and j. This implies that all the VMs have to be placed on exactly 1 PM.
  - Sum of active VMs in a PM should not exceed the number of cores of any PM.
- **Objective function** - We have to minimize the number of PMs, which means: if X=$\sum u_i$ for $0 <= i <= n$, then X should be minimum.

The outcome of this ILP are, X which provides the optimal number of PMs that can house all the VMs, and h_{ij} values which provides the tentative mapping of the PMs and VMs using X. The second ILP formulation takes this X as input and optimizes the reconfiguration plan my minimizing the migration cost. Following is the description of the second ILP.

- **Variables** - Two binary variables h_{ij} and c_{ij} are declared which are described below. The former determines the reconfiguration plan with mininum migration cost using X PMs. The later determines whether or not a VM needs migration.
  - h_{ij} binary variables. If $h_{ij} = 1$ then i^{th} PM hosts j^{th} VM where 'i' ranges from 0 to n and 'j' ranges from 0 to m. If 0, then j^{th} VM is not hosted by i^{th} PM.
  - mig_{ij} binary variables. If $mig_{ij} = 1$ then i^{th} PM hosting j^{th} VM needs migration, if 0, no migration is needed.

- **Constraints** - We make sure that the resource constraints are met, the summation of VM usages should not overshoot the PM's resource capacity. All the VMs have to be placed in exactly 1 PM. The condition of viable configuration should be satisfied. Finally, the migration cost in terms of memory allocated to the VMs have to be minimized. To summarize we have the following:
  - $\sum {R_j}^p * h_{ij} <= {C_i}^p$ for all i and j in 0 to n and 0 to m respectively.
  - $\sum {R_j}^m * h_{ij} <= {C_i}^m$ for all i and j in 0 to n and 0 to m respectively.
  - Let us assume that cur_{ij} is a matrix which provides the current topology (i^{th} PM hosts j^{th} VM). Then, if $cur_{ij} = 0, mig_{ij} = h_{ij}$ else $mig_{ij} = (1 - h_{ij})$ for all i and j. This is basically a XOR operation between the current topology and the topology provided by the solver to find out the number of migrations necessary.
  - For every i^{th} PM $\sum h_{ji} = 1$ for all i and j. This implies that all the VMs have to be placed on exactly 1 PM.
  - Sum of active VMs in a PM should not exceed the number of cores of any PM.
- **Objective function** - We have to minimize the migration cost, which means: if M=$\sum R_j \sum mig_{ij}$, then M should be minimum.

Thus the second ILP provides us with the optimal configuration plan with minimum migration cost and thus the migration overhead. M value gives the cost in terms of memory transferred for each migration and h_{ij} provides the new configuration plan.

We have used Google operation research solver to implement these ILPs and then we feed in the output of this solver to the entropy algorithm running in the decision engine at the Management node.

4.5 Verification of Correctness of the prototyped algorithms

Now, before we perform benchmarking and experimentation with synthetic workloads we verified that the algorithms are correctly triggering the migrations in accordance to the defined heuristics. We have manually formulated test cases to determine the correctness of the implemented algorithms. To start with, we discuss the verification of sandpiper.

4.5.1 Verification of Sandpiper

To verify the correctness of sandpiper, we manually created a simple test case, figured out the obvious outcome which is expected to happen, and tested our algorithm for the same. We gave as an input, the topology and the resource usage values of the case considered and checked whether sandpiper triggered the expected output. We considered 3 PMs and 2 VMs. Lets say PM1 hosts VM1 & VM2 and PM2 is empty. The normalized resource usage values across cpu, memory and network are as follows:

PM1 - 60% 34% 50% has VM1 - 10% 10% 20% and VM2 - 20% 12% 25%
PM2 - 40% 10% 20% Empty- no VMs
PM3 - 30% 15% 25% Empty- no VMs

The volume of PM1: $PM1_{vol} = (\frac{1}{1-0.60}) * (\frac{1}{1-0.34}) * (\frac{1}{1-0.50}) = 7.575$

The volume of PM2: $PM2_{vol} = (\frac{1}{1-0.40}) * (\frac{1}{1-0.10}) * (\frac{1}{1-0.20}) = 2.303$
The volume of PM3: $PM3_{vol} = (\frac{1}{1-0.30}) * (\frac{1}{1-0.15}) * (\frac{1}{1-0.25}) = 2.233$

Thus as per descending order of the volumes we have: $PM1_{vol} > PM2_{vol} > PM3_{vol}$
The memory footprints of all the VMs considered as 256 MB for this test case.

The volume of VM1 : $VM1_{vol} = (\frac{1}{1-0.10}) * (\frac{1}{1-0.10}) * (\frac{1}{1-0.20}) = 1.540$
The volume of VM2 : $VM2_{vol} = (\frac{1}{1-0.20}) * (\frac{1}{1-0.12}) * (\frac{1}{1-0.25}) = 1.886$
The VSR of VM1 : $VM1_{VSR} = \frac{VM1_{vol}}{256} = \frac{1.540}{256} = 0.006015$
The VSR of VM2 : $VM2_{VSR} = \frac{VM2_{vol}}{256} = \frac{1.886}{256} = 0.007367$

Thus as per decreasing order of the VSR ratios we have: $VM2_{VSR} > VM1_{VSR}$

To check the correctness of sandpiper we set the upper cpu threshold as 55% to trigger a hotspot at PM1. As per the algorithm, VM2 should be chosen for migration as it is the highest VSR VM. The least volume server is PM3. We can see that PM3 has enough resources available to house VM2. The new capacity of PM3 and PM1 after migration of VM2 from PM1 to PM3 would be:

PM3 - 50% 27% 50% and PM1 - 40% 22% 25%
Thus above is the expected outcome in the considered test case.

As we can see that the hotspot on PM1 is mitigated and no other PMs are having hotspots. VM1 remains in PM1. We fed the above values of resource usages and topology information to the algorithm. We ran the prototyped algorithm and found out that the algorithm indeed migrated VM2 from PM1 to PM3. Also, VM1 remained on PM1 after the hotspot was mitigated on PM1. This verifies the correctness of our prototyped algorithm. In the interest of time, we tried a few more simple test cases to check that the algorithm triggers migrations correctly, and then moved on with the experimentation.

4.5.2 Verification of Khanna's Algorithm

To verify this algorithm we considered 3 PMs, with PM1 housing 2 VMs say VM1 and VM2. Following are the normalized usage values across cpu, memory and network.

PM1 - 90% 10% 30% has VM1 - 10% 2% 10% and VM2 - 20% 5% 15%
PM2 - 40% 50% 30% Empty- no VMs
PM3 - 60% 35% 42% Empty- no VMs

Thus residual capacity vector of PM1 = [10 90 70]
Mean of PM1 residual capacities say $PM1_{mean} = \frac{10+90+70}{3} = 56.66$
Residual variance of PM1 say $PM1_{res} = \frac{(56.66-10)^2+(56.66-90)^2+(56.66-70)^2}{2} = 1733.33$
residual capacity vector of PM2 = [60 50 70]
Mean of PM2 residual capacities say $PM2_{mean} = \frac{60+50+70}{3} = 60$
Residual variance of PM2 say $PM2_{res} = \frac{(60-60)^2+(60-50)^2+(60-70)^2}{2} = 100$
residual capacity vector of PM3 = [40 65 58]
Mean of PM3 residual capacities say $PM3_{mean} = \frac{40+65+58}{3} = 54.33$
Residual variance of PM3 say $PM3_{res} = \frac{(54.33-40)^2+(54.33-65)^2+(54.33-58)^2}{2} = 166.373$

In ascending order of residual capacity magnitude of PMs we have:$PM2_{res} < PM3_{res} < PM1_{res}$
We considered the migration cost co-efficient across all the three resource dimensions as 1. Hence the resource utilization of VMs is simply the summation of normalized resource usages across cpu, memory and network.

VM1 resource usage $VM1_{use} = 10 + 2 + 10 = 22$
VM2 resource usage $VM2_{use} = 20 + 5 + 15 = 40$
Thus ascending order based on resource utilization of VMs is: $VM1_{use} < VM2_{use}$

Now we set the upper cpu threshold as 75% to trigger hotspot on PM1. Since, VM1 with lowest resource usage is chosen for migration. Next PM2 with least residual capacity is scanned for a potential destination PM. PM2 can hold VM1 as the new resource usage values, if VM1 migrates to PM2, doesnot exceed the threshold.
New PM2 usage - 50% 52% 40% and PM1 usage - 80% 8% 20%.
As we see PM1 is still under hotspot, VM2 is chosen for migration.

Thus new residual capacity vector of PM1 = [20 92 80]
New Mean of PM1 residual capacities $PM1_{mean} = \frac{20+92+80}{3} = 64$
New Residual variance of PM1 $PM1_{res} = \frac{(64-20)^2+(64-92)^2+(64-80)^2}{2} = 1488$
New residual capacity vector of PM2 = [50 48 60]
New Mean of PM2 residual capacities $PM2_{mean} = \frac{50+48+60}{3} = 52.66$
New Residual variance of PM2 $PM2_{res} = \frac{(52.66-50)^2+(52.66-48)^2+(52.66-60)^2}{2} = 41.33$

Since, PM3's resource usage remains the same, its residual capacity variance remains the same which is 166.373.
In ascending order of residual variances PMs we have:$PM2_{res} < PM3_{res} < PM1_{res}$

As PM1 with 80% cpu usage is still under hotspot, VM2 is chosen for migration. PM2 has least residual capacity and hence chosen as potential destination PM for VM2. If VM2 migrates to PM2 from PM1:
New PM2 usage = 70% 57% 55% and PM1 usage - 60% 3% 5%
This mitigates hotspot on PM1 without creating hotspot anywhere else.

Thus new residual capacity vector of PM1 = [40 97 95]
New Mean of PM1 residual capacities $PM1_{mean} = \frac{40+97+95}{3} = 77.33$
New Residual variance of PM1 $PM1_{res} = \frac{(77.33-40)^2+(77.33-97)^2+(77.33-95)^2}{2} = 1046.335$
New residual capacity vector of PM2 = [30 43 45]
New Mean of PM2 residual capacities $PM2_{mean} = \frac{30+43+45}{3} = 39.33$
New Residual variance of PM2 $PM2_{res} = \frac{(39.33-30)^2+(39.33.-43)^2+(39.33-45)^2}{2} = 66.24$

Since, PM3's resource usage remains the same, its residual capacity variance remains the same which is 166.373.
In ascending order of residual variances PMs we have:$PM2_{res} < PM3_{res} < PM1_{res}$

For coldspot detection verification we chose the lower cpu threshold as 45%. Thus PM2 will be detected as coldspot. Lowest cost VM is VM1 which is chosen for migration. Before coldspot detection the l2 norm variance across all the three PMs is:
$var_{old} = \sqrt{1733.33^2 + 100^2 + 166.373^2} = 1744.165$

If VM1 is migrated to PM2 since PM2 has the lowest residual variance, the new l2 norm variance would be:
$var_{new} = \sqrt{1488^2 + 41.33^2 + 166.373^2} = 1497.84$
Since, var_{new} is lower than var_{old} the migration is not triggered. If var_{new} is higher than var_{old} then migration is triggered.
We wanted to get the above outcome from our prototyped algorithm. We fed the following values of resource usages and topology to the algorithm and got the desired output. VM1 and VM2 is migrated from PM1 to PM2. Coldspot mtigation was also separately verified. This verifies the correctness of Khanna's algorithm we prototyped.

4.5.3 Verification of Entropy Algorithm

To verify entropy, we created a simple test case where 4 PMs had 5 VMs running in the following manner:

PM1 - 20% 22% 30% hosted VM1, VM2 and VM3
PM2 - 10% 10% 30% hosted VM4 and
PM3 - 10% 10% 30% hosted VM5
PM4 - Empty - no VMs

All the VMs have 5% cpu, 6% memory and 8% network usage. Since it is not a threshold based approach there is no upper cpu threshold set here, so 1 PM can accomodate all the VMs. The overall resource usages will be less than 100%. So, optimal number of PMs needed is 1.
After VM4 migrates from PM2 to PM1 and VM5 migrates from PM3 to PM1:

$PM1_{use} = 25\%28\%38\%$
$PM2_{use} = 5\%4\%22\%$
$PM3_{use} = 5\%4\%22\%$

As we know, ideally all the VMs should be housed on PM1 as migrating 2 VMs from PM2 and PM3 is less costly than migrating 3 VMs from PM1 to any other PM. Also, the migration cost incurred should be $2 * 256 = 512$ as here we considered VM size as 256 MB RAM for all. In the second case, to verify viable configuration is maintained, we kept the threshold for calling a VM active as 50%. For the purpose of testing, we set the no. of cores of PM1 as 3. And increased the usage of VM1, VM2, VM3 & VM5 to 60% 50% and 55%. VM4 retained its original usage. Expected outcome is VM1, VM2, VM3 and VM4 on PM1 and VM5 on PM4. VM5 cannot be housed on PM1 as PM1 has just 3 cores, it should not house 4 active VMs. Thus outcome expected is all on PM1 except VM5 on PM3, with 1 migration.
We hardcoded and provided the above topology and resource usage values as input to entropy algorithm, and did obtain the desired expected output. Entropy packed all the VMs in PM1 mentioning the optimal number of PMs as 1. It also calculated the migration cost as 512 triggering 2 migrations to procure the new reconfiguration plan, which hosted all the VMs on PM1. In the latter case also, we got the desired output incurring 1 migration. This sanity check verifies the correctness of Entropy algorithm.
Based on our checks with the discussed simple test cases, in the available time we had for the project, we were confident of the correct working of the algorithms. We next did some experiments with synthetic workloads like 'httperf' and obtained some preliminary results, which we

discuss in the subsequent chapters. In the following sections we describe the workload generation, experimentation and evaluation.

Chapter 5

Workload Generation

In this chapter we briefly mention the benchmarking tool used to generate workload on the virtual machines.

5.1 Workload Generation

We have used "httperf" client to generate workload in our experiments. HTTPERF [26] is a cpu intensive workload generating tool used for experimentation purposes. Even in low workloads "httperf" increases the cpu usage of the machines. We have used a two-tier WebCalendar application using Apache web server at frontend and MySQL database server connected at the backend. Apache server hosted the two-tier WebCalendar application which has web and database tiers. Then we used httperf for load generation.
We also used RUBiS workload generator for one of the experiments. We had mysql database installed on all the VMs along with apache and PHP. We created RUBiS database for rubis client and 'intranet' database for WebCalendar. RUBiS is a prototype of an auction site modeled after *eBay.com*. It is used to generate workload or application patterns to evaluate application server's performance. RUBiS benchmark implements selling, browsing and bidding like core functionalities. We have kept the properties of these applications same across all the algorithms to generate same workload patterns.

With httperf we provided http requests which were WebCalendar PHP scripts to be sent to the server. We used the same set of requests across all the experiments. We only varied the rates of sending requests. RUBiS with the same client properties were used invariably across all the algorithms.

The rate parameter of httperf allows us to specify the number of requests to be generated per second. We varied this parameter to create varied load on the virtual servers. In all our experiments we used two types of workload using httperf. They are the following:

- Fixed rate of requests, where the rate of request is constant on a VM all throughout the experimental run.
- Varying rate of requests where we have varied the rates in phases within the same experimentation time window to generate different degree of load on the virtual server.

We have used a combination of the above types in our experiments which means, either VMs receive fixed rate of requests or varying rate of requests from the client. As we change the

topology, we change the workload on the virtual machines as per our requirements to create a certain scenario, like to create hotspot on a PM, we try to make sure that the VMs placed on it receive high rate of requests to increase that PM's cpu utilization and so on. The inter-arrival time of requests for any specific rate is fixed which means we have used deterministic inter-arrival time distribution. Cyclic workloads have not been used where the rates increase and then falls. We have used fixed rates or increasing rates like 10, 20 and 30 requests per second. For experimentation with RUBiS, we have made sure that the *rubis.properties* file at the client is same for all the experiments across the three algorithms. Once we start the emulator, we have not changed anything during the experimentation.

Chapter 6

Evaluation and Results

In this chapter we discuss the experimentation details and evaluate the results.

6.1 Experimentation Test-Bed

Our implementation and evaluation is based on *Xen hypervisor*. Our test-bed in the lab setting comprises of the following:

- Four PMs with Xen 4.0.1 hypervisor installed to serve as the physical hosts- They are all Intel machines out of which four have Quad Core CPU, one has Dual Core and the remaining one has 8 core CPU. Following are the details of the Xen-Installed host PMs:

  - PM IP 10.129.41.180 - Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83 GHz, 728.6 MB RAM, disk - 453 GB. We shall refer to it as *PM180* hereafter, as and when required.
  - PM IP 10.129.41.170 - Intel(R) Core(TM)2 Duo CPU E4500 @ 2.20 GHz, 1.45 GB RAM, disk - 74 GB. We shall refer to it as *PM170* hereafter, as and when required.
  - PM IP 10.129.41.161 - Intel(R) Xeon(R) CPU E5405 @ 2.00 GHz, 6.52 GB RAM, disk - 282 GB. We shall refer to it as *kleinrock* or *PM161* hereafter, as and when required.
  - PM IP 10.129.41.84 - Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83 GHz, 1.84 GB RAM, disk - 457 GB. We shall refer to it as *PM84* hereafter, as and when required.

- One NFS server Set-up - PM IP 10.129.41.70 - Intel(R) Core(TM)2 CPU 6300 @ 1.86 GHz(Dual Core), 1 GB RAM, disk - 150 GB.

- Physical machines which worked as Clients to simultaneously generate load on the Virtual machines hosted on the PMs.

- Seven VMs are created with Ubuntu-10.04, Lucid host operating system. They all have Apache, PHP, Mysql and Muffin configured on them to act as Web and DB servers.

- We have configured, Dell Inspiron-14 laptop to act the Management node, which runs the controller and the Decision Engine.

Python Programming is used to code the tool as well as prototype the algorithms using the APIs. We need the following major softwares for our work.

- **RRD tool** is installed on the Management node running the decision engine, for storage of resource usage data. We have already discussed about RRD in section 3.4.1.
- **Google OR-tools** software for python CSP (constraint programming) solving required for Entropy algorithm. We have discussed that in Section 4.4.1.

6.2 How do we define the goodness of an algorithm?

All the observations and evaluations are done over a fixed time window say T. During this T time frame, we log all the details while the algorithm is running and pre-defined workloads are generated from different clients at the VMs before logging begins. The following points are the **evaluation metrics** for our comparative analysis. In our work, an algorithm (say A) is defined to be better than the other (say B), for a particular scenario,

- The time taken within that time window T to mitigate hotspots or coldspots or reduce the no. of PMs is lesser in A's than B's.
- If the incurred cost of migration in A is lesser than B, then A is better than B else vice-versa. We have used the Migration Cost Metric as the *number of migrations*:-
  - Total no. of migrations triggered in that time window T, we have studied in the research literature that too many frequent migrations makes the system unstable, if lesser no. of migrations are triggered by A than B, to consolidate (and thus mitigate hotspots/coldspots) then A is better than B. Also, more the number of migrations, more the migration overhead.
    Count the no. of migrations triggered by A within T time units = A_count
    Count the no. of migrations triggered by B within T time units = B_count
    If A_count < B_count, A is better than B else vice-versa.
- If an algorithm can reduce more no. of PMs, it is clearly better than the other in consolidating. Again this will be observed within that T time.
  At the start when t=0, no. of PMs = 4
  At the end when t=T, no. of PMs = N_A (say for A)
  If N_A for A is lesser than N_B for B then, A is better than B.
- If the application performance of VMs while running A is found better than B. We can capture this application performance metric using: average response time, lesser the average response time of a VM, better the application performance is and higher the values less good is the performance.

However, in this work we have primarily discussed about the first three points with lesser detailed evaluation of application performance. In our work, application performance evaluation seemed less important for a few experiments due to their design goal itself, in the remaining experiments, due to some unexpected implementation issues, we couldnot correctly log the response time values. Hence, we have restricted our discussion of application performance analysis in the context of comparative study of algorithms. Thus, no. of PMs reduced, no. of migration triggered in some fixed time interval and the time taken to perform consolidation are the evaluation metrics here.

6.3 Experimentation and Evaluation

In this section we discuss the various factors that can affect the VM migration decision making process. Then we emphasize on the factors which we have considered for our purpose of evaluation. Finally, we present the scenarios we considered for experimentation and analyze the results obtained. The primary factors that can affect a consolidation algorithm are the following:

- **Topology** - The initial topology we start with affect the outcome of the algorithm. Varying the topology will affect the decision making process of the VM migration algorithm. We have considered this in our evaluation. We have changed the topology to see the behaviour of each of the algorithms in presence of some pre-defined workload.
- **Workload characteristics** - The type of workload used and the nature of the workload variation will also affect the events triggered by the algorithm. Firstly what kind of application we use will affect the outcome, for example, a cpu intensive workload will be more sensitive towards a threshold based consolidation algorithm where a upper cpu threshold is set. Secondly, when we use a particular kind of workload the way we vary the workload will affect the migration decision making process. Either, a fixed rate of requests can be used or varying the rates within a fixed interval of time can be done to trigger fluctuating resource usage etc. These workload variations are going to determine the final migration decisions taken by the algorithm. We have used two kinds of workloads as discussed in section 5.1. We have considered a case where the workload is generated at a fixed rate and a case where the workload is varied by changing the rates of the workload.
- **Thresholds** - If the algorithms use a threshold based approach, the threshold we set will affect the reactive nature of the algorithm towards detecting hotspots or coldspots and hence the number of migrations. Both Khanna's algorithm and sandpiper use a threshold based technique to mitigate hotspots and/or coldspots. We have used the same lower cpu threshold across all the experiments which is 20% and used two different upper cpu thresholds.
- Other factors like VM memory footprint, the amount of cpu allocated to a VM, the resources considered, the SLA violation metric considered etc also determine the migration decision making process. The migration cost metric is dependent on the memory and cpu allocations and accordingly the algorithms will behave towards a scenario. We have used 256 MB RAM for all the VMs in our experiments.
- **Algorithm/Heuristic** Used - Finally needless to mention, the specific algorithm we use to trigger migration will determine the criteria for triggering migrations and generating a new configuration plan. We have run our experiments on all the three algorithms ensuring that all the other physical parameters when we start the experiments are fixed.

As described in section 6.2 we have shown the variation of CPU usage with time across all the 4 host PMs we used for our work. For every experiment we have shown the number of migrations triggered and the number of PMs used. In certain experiments we also show the time taken to consolidate in lesser number of PMs by different algorithms within the fixed duration of time. Application response time is measured and is shown for just 1 experiment. Due to some unexpected implementation issue we couldnot log accurately the application response time for some experiments. However our preliminary results throw some light on the behaviour of the chosen algorithms.
Table 6.1 briefly summarizes the above discussion:

Table 6.1: Experimentation Design

Factors affecting Migration decision	Topology	Workload characteristics	Algorithm Used	Thresholds
Measured parameters	No.of PMs used	No.of Migrations Triggered	Time taken to achieve consolidation	CPU usage variations

We had design plans to log average response times for all the experiments but due to some discrepancy in logging we could'not get valid logs for some experiments. Unfortunately we shall present the application performance for just 1 experiment. However, we have got some insight into the nature of the chosen algorithms in some pre-defined scenarios we generated in our lab-setup. We have varied the topology or the workload to create different scenarios to find out how the different algorithms react to different scenarios. We will describe each of these scenarios now, with discussion on the subsequent outcome.

6.3.1 Scenario-1 - With Idle VMs

We performed an experiment with idle VMs as the base case, to check how reactively the algorithms behave towards consolidation when no workload is generated. We created the following topology:

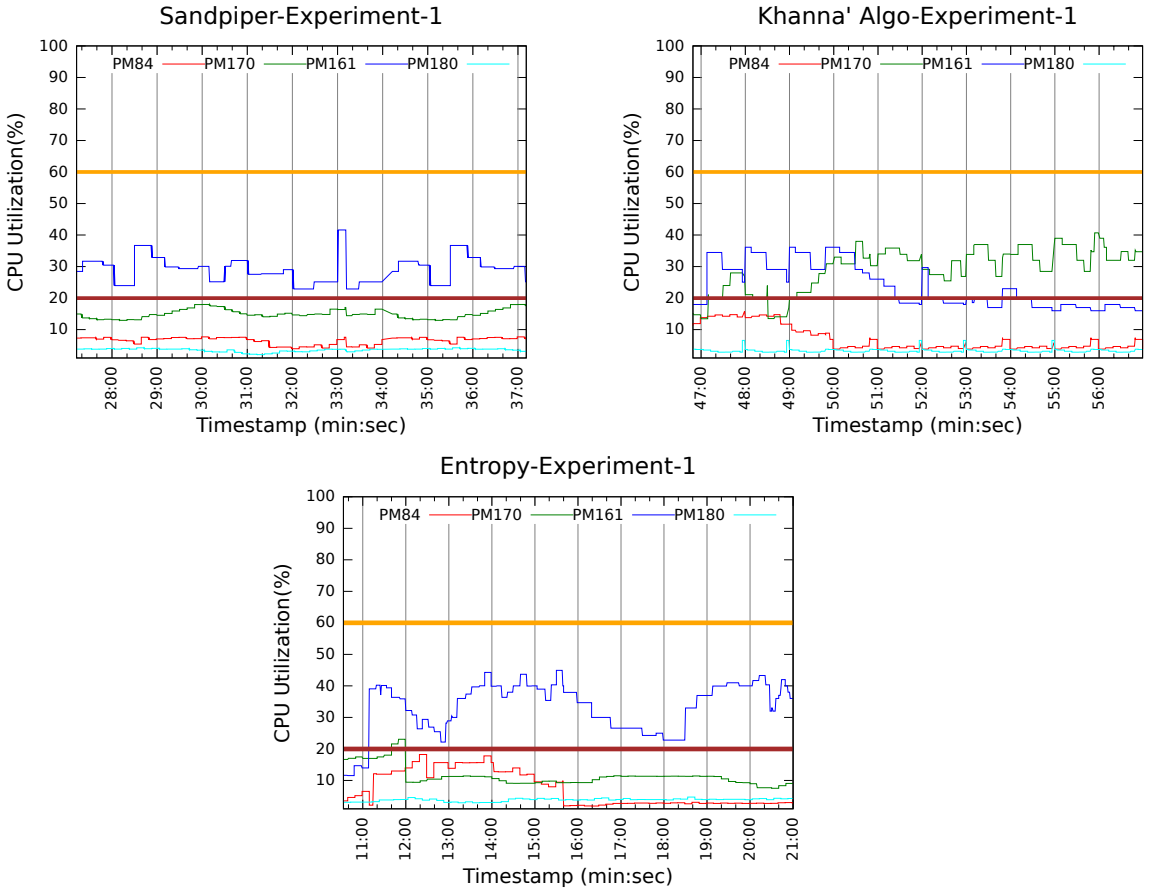
PM84 contained 1 VM - 10.129.41.252 lucid08
PM170 contained 1 VM -10.129.41.214 lucid09
PM161 contained 2 VMs -10.129.41.210 lucid12 and (10.129.41.246) lucid10
10.129.41.180 - empty

We have named the VMs as lucid08, lucid09 and so on. This was the topology we started with. We ran all the three algorithms for **10 minutes** duration. We have the results in 6.1, 6.2 and 6.3. As seen from Figure 6.1 the cpu utilizations of PMs in case of sandpiper is more or less stable will very less fluctations. Since there were no threshold violations, no possibility of migrations in case of Sandpiper. We can see the cpu utilization of PM161 slightly higher, although just two idle VMs are running, remote logging through "sshd" commands from PM161 to other machines for running around 6 scripts per machine might have contributed to some extent for a slightly higher cpu usage. Khanna's Algorithm has lot of variations because of instant detection of coldspots and hence the utilizations had some variations due to migrations. Entropy also triggers migrations due to which there are fluctuations in cpu usage levels, more for PM161 than the others which we will discuss below. After the experimental run the new topology generated are:

- Sandpiper - no change
- Khanna's Algo - all the VMs in PM 170
- Entropy - all the VMs in PM 161

From Figure 6.2 and 6.3 we observe that Khanna's Algo triggers migration of lucid08 from PM84 to PM161 very soon, because it detected a coldspot on PM84. Its coldspot mitigation heuristic sorted all four VMs as per their current utilizations and initiated a series of migrations, trying to increase the overall system variance. It frees up PM 84 and PM 161 in approximately 4 minutes from the start of the algorithm. Since PM 180 was empty at the onset, it was asked to be

Figure 6.1: Experiment-1 cpu usage versus time across 4 PMs



turned off by the algorithm. Thus, all the VMs were packed on PM 170 reducing 2 machines. We can very well observe the gradual increase in CPU usage for PM 170 after it recieved VMs from PM 161. Also, the fact that PM 161 was not chosen as the PM for packing all the VMs verified the correctness of khanna's algorithm. It uses the PM with least residual capacity big enough to hold the VM, PM 161 had more capacity than PM 170, hence PM 161 is not expected for selection. Entropy on the other hand, finds out that only 1 PM is needed to house all the VMs, its configuration plan entailed a set of two migrations, lucid09 from PM 170 and lucid08 from PM84. Using 1 PM, atleast 3 migrations are needed if two VMs from PM 161 were migrated. Since entropy optimizes the migration cost, it chose PM161 as the final destination for all the VMs by reducing 1 extra migration. This packing was done in little more than 1 minute time. We can observe distinctly the increasing cpu utilization at PM161 due to VM reception from other PMs, and decreasing cpu usage at the other PMs.

The following table enlists the measured statistics which we mentioned already in 6.2 section. Clearly Khanna's algo issues more migration increasing the migration overhead over entropy. But it looks up a for a better fit of PMs for all the VMs over entropy.

Table 6.2: Experiment-1 with Idle VMs - Measured Evaluation Metrics

Algorithm	**No.of PMs**	**No.of Migrations**	**Time Taken(mins)**
Sandpiper	3	0	N/A
Khanna's Algo	1	6	4
Entropy	1	2	1.33

From this simple, naive experiment to start with, we takeaway the following points:

- In the base case, while dealing with idle VMs, khanna's algo takes more time to consolidate VMs over entropy, because it detects a coldspot immediately and triggers migrations to increase the overall system variance. Entropy globally finds out an optimal solution to stuff the VMs minimizing the memory transfer of VMs.
- Since Khanna's algo tries to find a best fit solution, it packs all the VMs on a PM which is just sufficient to house all the VMs, but entropy on the other hand, donot enlist any explicit way of checking the best fit solution, as its primary emphasis is on reducing the migration overhead.

Next we move on to a scenario where we generate workload on the VMs.

6.3.2 Scenario-2 with low workload

To check the effect of coldspots we created a scenario of moderate load. We performed an experiment with four VMs two of which were idle and two of moderate workload. We used "httperf" to inject workload using Web Calendar PHP scripts. The workload rate was set to 20 requests per second for lucid 12 and lucid10. We created the following scenario:

PM84 - empty
PM161 - lucid12 and lucid13 - load with rate of 20 req/sec on lucid12
PM170 - lucid14 - no load

Figure 6.2: Experiment-1 VM Migration sequence in 4 PMs for Sandpiper and Khanna's Algo

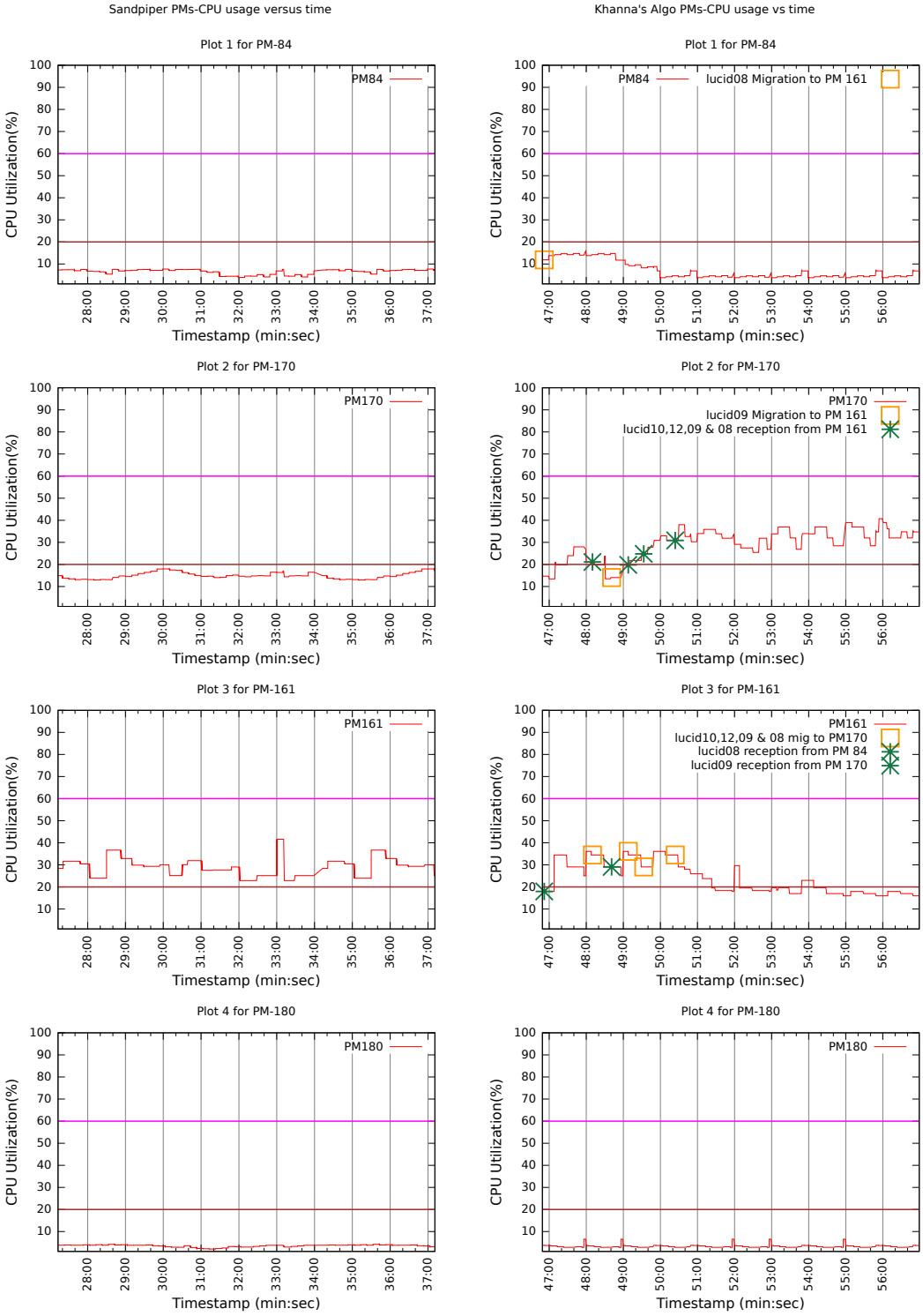
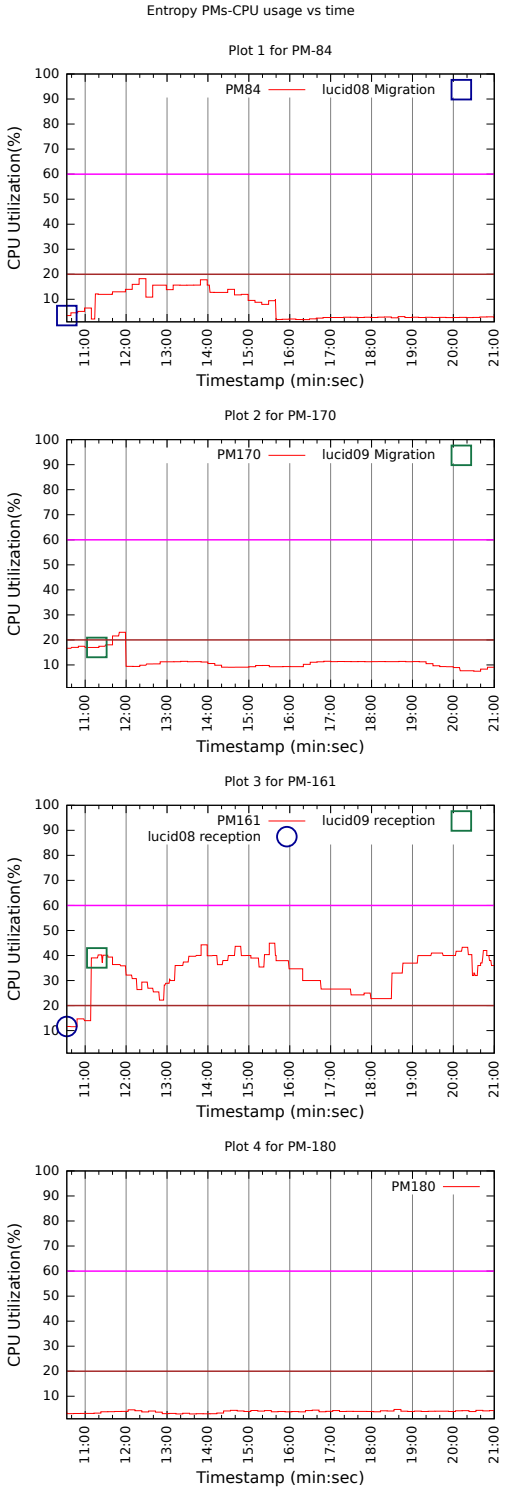


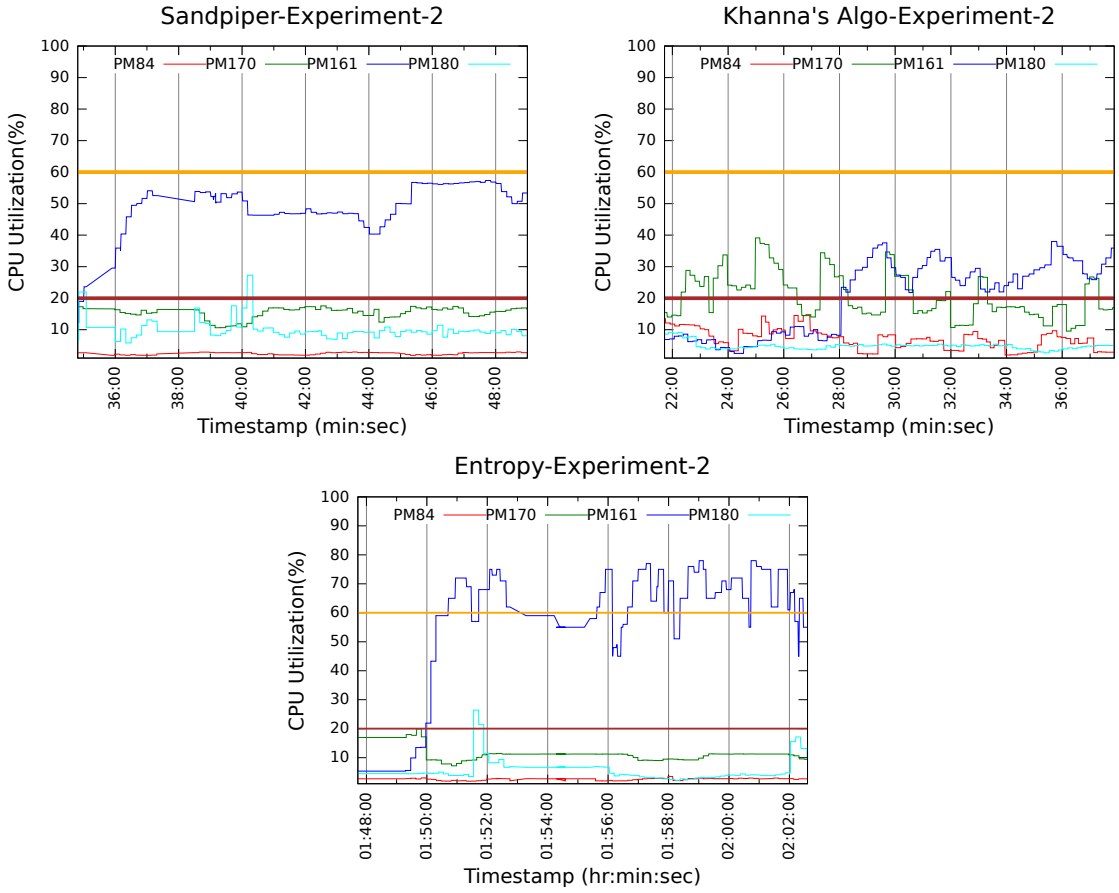
Figure 6.3: Exp-1 Entropy Algo - VM Migration sequence across 4 PMs



PM180 - lucid10 - load with rate 20 req/sec

Ran the experiment for **15 minutes** duration for all the algorithms. The upper cpu threshold was set to 60%. We are moving from an idle VM case gradually towards increasing load situation. Under moderate levels of load what happens is of interest to us, because it is not obvious that if in the naive case Khanna's algo takes 3 minutes more time than entropy to consolidate, the same will happen in the case of low levels of workload as well. We feel that this experiment is interesting because we want to answer questions like - Which VM does they choose to pack the VMs in low workload case? Is it same as the base case? What happens to the time taken to achieve consolidation? By how much does it increase/decrease?".

Figure 6.4: Experiment-2 cpu usage versus time across 4 PMs



From Figure 6.4 we observe that, since upper cpu threshold was not violated which was our design goal, there were no events triggered by sandpiper. Figure 6.4 depicts that Khanna's algo migrates lucid13 from PM 161 to PM84, after detecting a coldspot on PM170, and sorting all four VMs based on their utilizations, clearly lucid13 and lucid14 has lower utilization than lucid12 and lucid10. Next lucid14 is migrated from PM170 to PM84, lucid10 from PM180 to PM170

and finally lucid12 from PM161 to PM170. This series of migrations were triggered based on the heuristic as a part of coldspot mitigation. After such migrations, the cpu usage levels were within the thresholds and hence no more coldspots were detected. Entropy again packs all the VMs in 1 PM by triggering just 2 migrations. The new topology generated were:

- Sandpiper - No change
- Khanna's Algo - PM170 hosts lucid10, lucid12 & lucid14 and PM84 hosts lucid13
- Entropy - all the VMs are moved to PM161.

From Figure 6.5 and 6.6 we see that whenever there has been a VM reception at any of the PMs, the cpu usage levels have increased. Also, since PM 170 houses 3 VMs its usage increases to as high as 37%. But even though PM 161 migrated its VMs, we can observe a rise in its cpu usage in the Figure. We conjecture that sometimes the PM cpu usages are reported higher than we expect them to be because of variation in migration time of VM from/to PMs, in presence of several scripts running on the PMs. The remote logging in PMs where the cpu usage is fluctuating might also have some minor effects on the host from where remote logging is done. Entropy again finds that the optimal number needed to house all the VMs is 1, and puts all the VMs on PM 161, as cpu usage is not considered as migration cost metric here. So, even though cpu usage of lucid10 is expected to be higher than lucid13, it migrates lucid10 on PM161. It is easy to observe that since PM161 has already 2 VMs running on it, moving lucid14 from PM170 and lucid10 from PM180 will incur just 2 migrations to consolidate all the VMs on 1 PM. This increases the cpu usage of PM161 to as high as 70% exceeding the threshold of 60% when all the other PMs have low utilization levels.

Table 6.3 enlists the parameters of comparison, Khanna's algo reduces the number of PMs by 1, PM 84 was already devoid of VMs when we started. Entropy reduces the number of PMs used by 2, triggering just 2 migrations in 2.83 minutes. Khanna's algo triggers 5 migrations in total and uses 2 PMs by 3.28 minutes from the start of the algorithm.

Table 6.3: Experiment-2 with low workload - Measured Evaluation Metrics

Algorithm	**No.of PMs**	**No.of Migrations**	**Time Taken(mins)**
Sandpiper	3	0	N/A
Khanna's Algo	2	5	3.28
Entropy	1	2	2.33

Some takeaway from this experimental analysis are as follows:

- We can see from the graph that the variance of utilization levels across Khanna's algo is less than Entropy. Although Khanna's algo tries to maximize variance, it ensures that the cpu usage levels are within the defined bounds, hence the requirement of defined bounds, affects the migration decision making process keeping the overall utilization levels across the PMs higher than they were observed in the case of Entropy. In entropy, at the cost of 1 PM having very high usage, we have other PMs which can be freed up because of under-utilization. The risk in entropy is that, in face of sudden bursty data after such rigorous consolidation, it might trigger many migrations to other destination PMs or it might need some new PMs to be started to house the VMs, since very less residual capacity will be

Figure 6.5: Experiment-2 VM Migration Sequence in 4 PMs for Sandpiper and Khanna's Algo

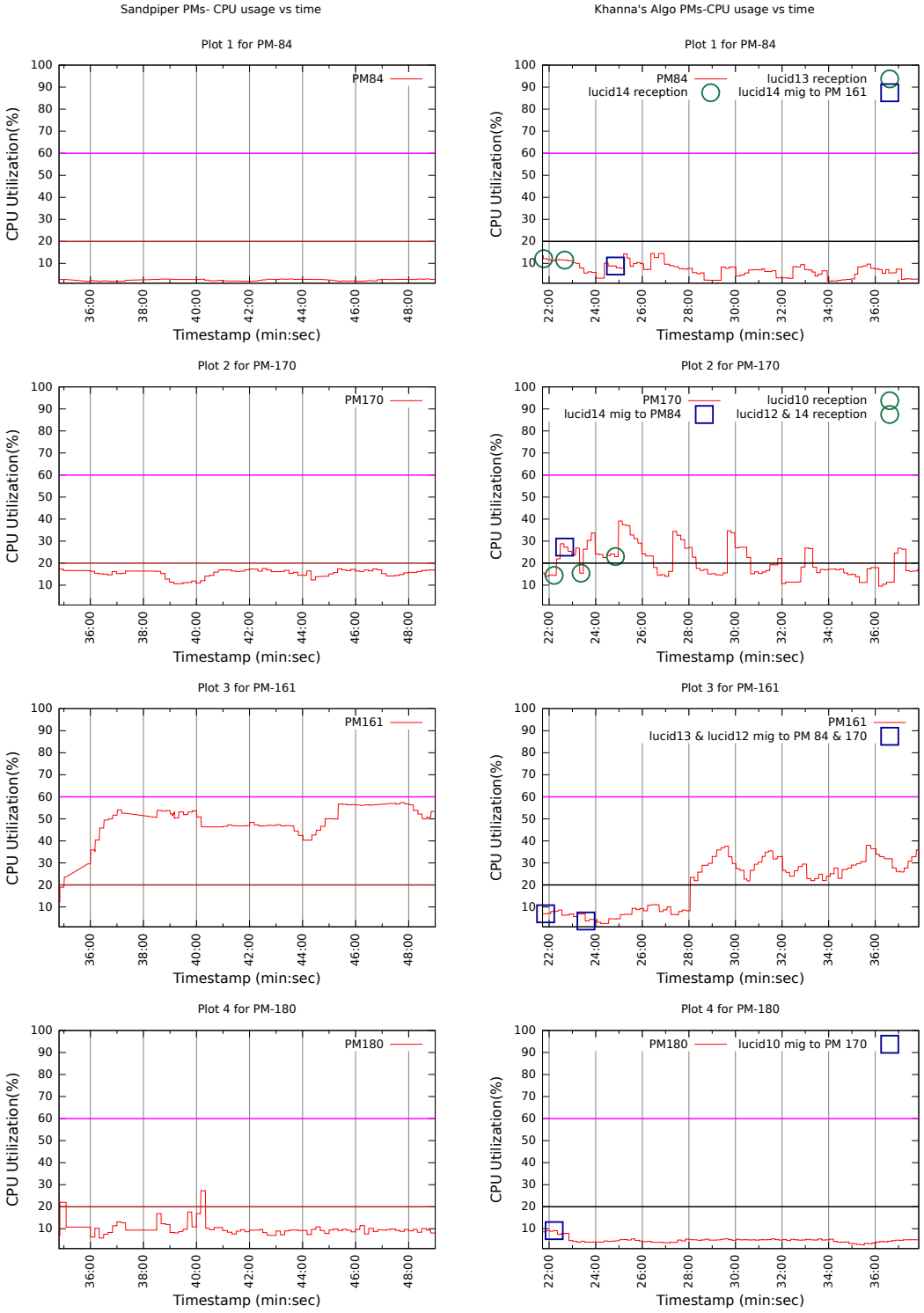
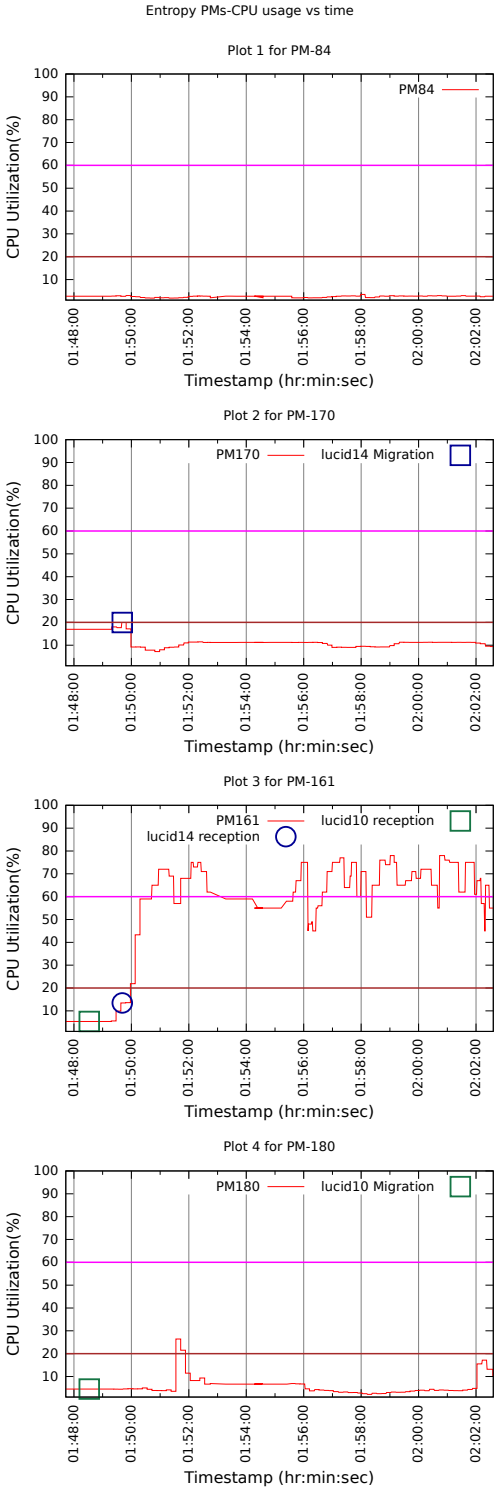


Figure 6.6: Exp 2 -Entropy Algo - VM Migration sequence across 4 PMs



left in the PM where all the VMs are stuffed. In case of Khanna's algo, such a crisis may not arise as there are residual capacities left in the PMs which may accomodate VMs in future.

- If we know the nature of workload in data center beforehand, we can suggest that, Entropy may not be a good choice for too bursty workloads, as everytime the algorithm tightly packs the VMs in a PM, after some discrete interval of time, if there is a sudden surge of high workload, there may arise a situation where new PMs need to be added to the system, if they had been shutdown earlier. However, khanna's algo takes care of the future possibility of VM accomodation in a PM, for which it may be able to accomodate VMs inface of bursty data.

Next we use varying workloads in VMs to fluctuate the cpu usage levels and see the behaviour of the algorithms.

6.3.3 Scenario-3 with changing rates of workload

In the previous experiment the rate of workload being generated was fixed, here we try to change the rate of workload within the duration of experimental run to further fluctuate the VM resource usage levels. This is also an interesting experiment to do since if the rate is fixed, the cpu usage levels will not vary drastically within the duration of experiment and hence after some reconfiguration in the initial topology we started, the PMs are expected to stabilize without further threshold violation. If the rates change, the usage levels will vary more and in such a scenario, we want to have an idea how reactively the algorithms do consolidation. We used "httperf" with Webcalendar PHP scripts for this experiment to inject varying workload. We created the following scenario:

PM84 - lucid08 - no load
PM161 - lucid12 and lucid13 - rates varying as 10 req/sec, 20 req/sec and 30 req/sec on both the VMs
PM170 - lucid09 - rates varying as 10 req/sec, 20 req/sec and 30 req/sec
PM180 - lucid14 - no load
The experiment was run for **10 minutes** duration for all the algorithms. The upper cpu threshold was kept as 60%.

From Figure 6.7 we notice that there is too much of cpu usage level variations compared to the previous experiments which is expected due to variation of rates. Amongst the three, variation in Entropy has been slightly lesser amongst the PMs, not only in this experiment but in the previous experiments as well. Sandpiper mitigates 1 hotspot and triggers 1 migration from PM 170 to PM 180. Khanna's algo performs 5 migrations in total and uses 3 PMs same as sandpiper. Entropy reduces the number of PMs by 3 (from 4) and triggers 3 migrations. As we see PM84 has comparatively higher usages than the others because of VM receptions from other PMs.

From Figure 6.8 and 6.9 we can see that Sandpiper detects a hotspot on PM170, migrates lucid09 to PM180 and reduces the number of PMs by 1. Khanna's Algo detects a coldspot at PM 180, does coldspot mitigation. It triggers a series of migrations, and finally uses 3 PMs at the end of the experimental run. Lucid14 was migrated from PM 180 to PM 84. Lucid13 was migrated from PM161 to PM 170. Lucid 14 from PM84 to PM 161, followed by lucid14 from PM161 to

Figure 6.7: Experiment-3 cpu usage versus time across 4 PMs

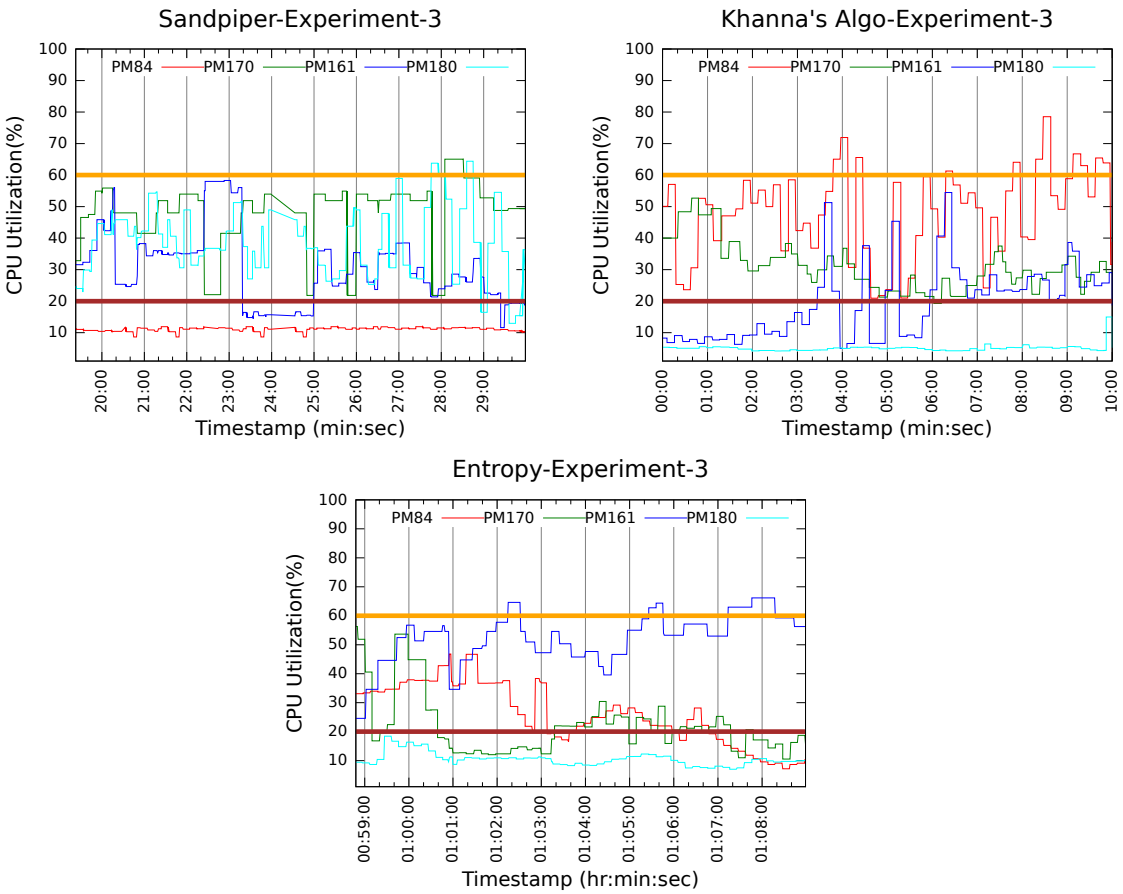


Figure 6.8: Experiment-3- VM Migration sequence in 4 PMs for sandpiper and Khanna's Algo

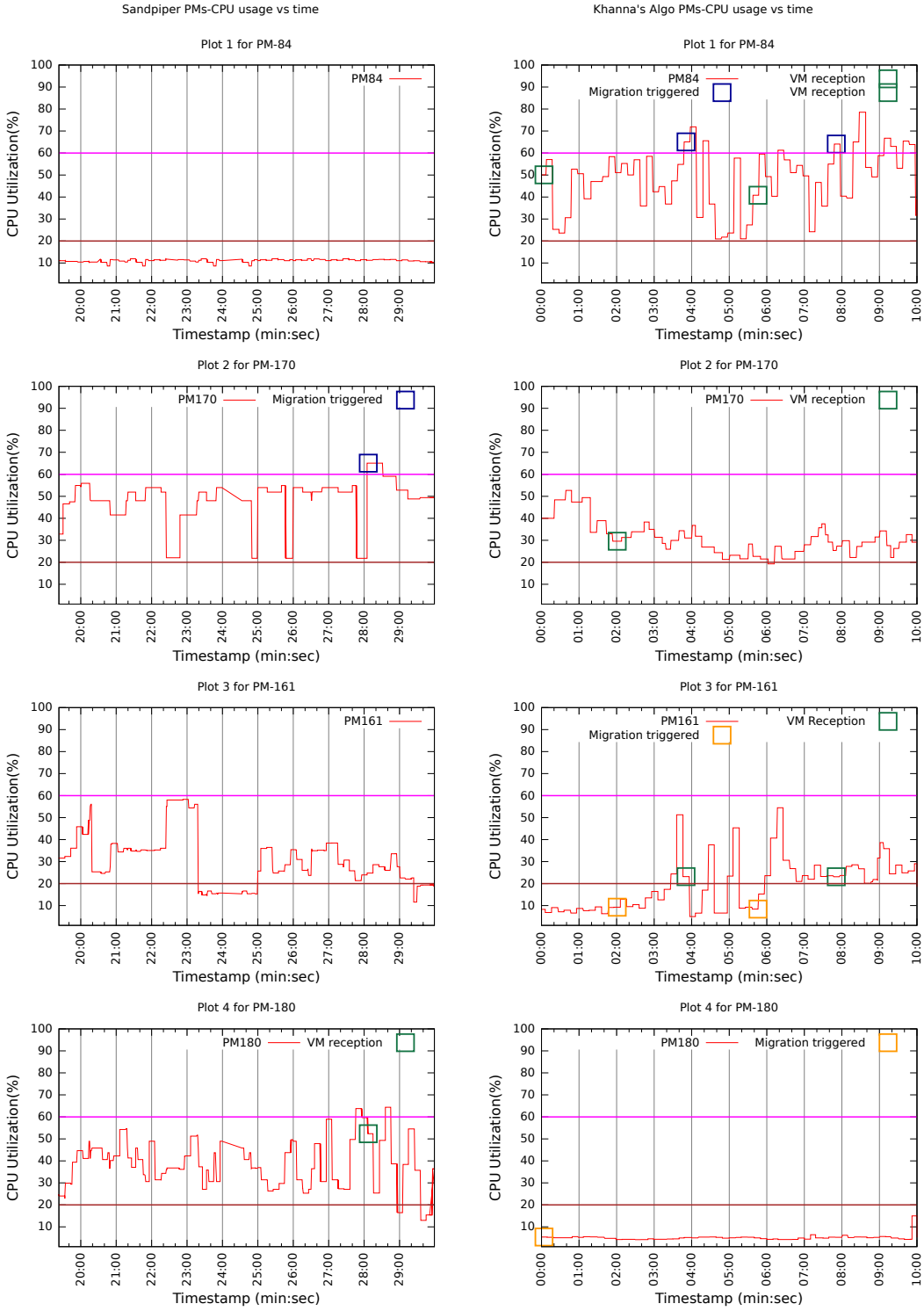
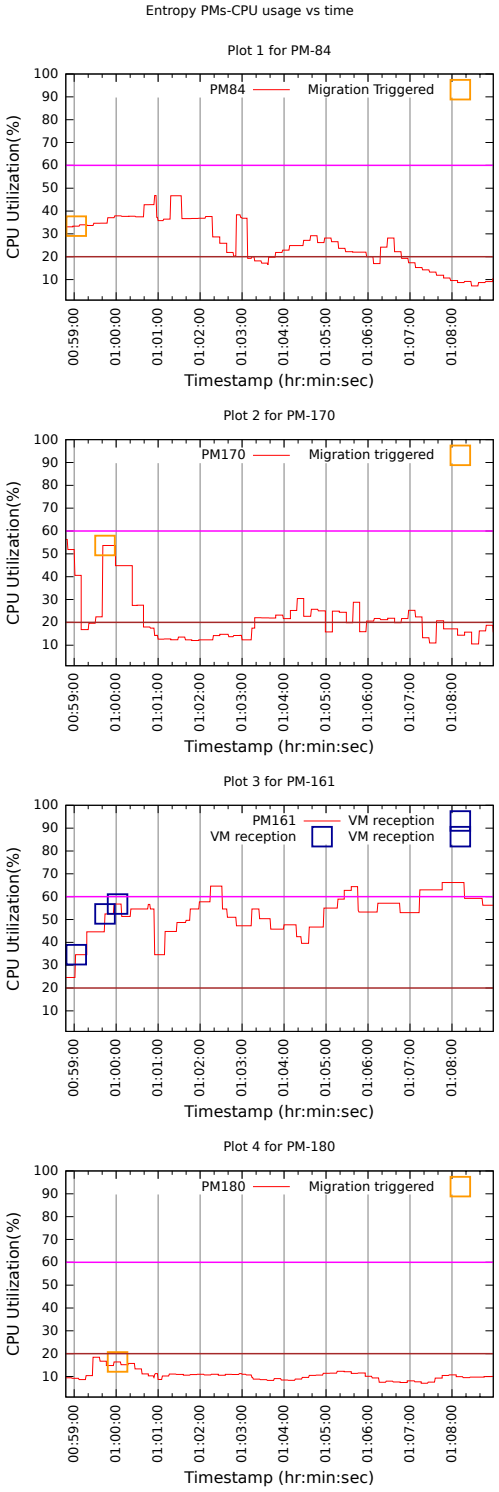


Figure 6.9: Exp-3-Entropy Algo- VM Migration sequence across 4 PMs



PM84, and finally lucid 08 from PM 84 to PM 161. Entropy in this case as well puts all the VMs on PM 161. As we know in entropy, viable configuration is to be satisfied, which means number of active PMs should have access to sufficient available processing units. Here, even though the cpu usage of PM 161 increases, the viability condition might have been satisfied for which all the VMs were put on the PM. Also, we see the decreasing cpu usage trend in the other PMs after entropy migrates all the VMs to PM 161. We expected the cpu usage of PM161 to increase further in this experiments when we use varying workloads over the previous experiment, but comparatively the hike is not that much. We believe that this happened because when the rates are high on a VM, due to increase in processing time of the requests or delay in sending response to the client, the cpu usage of the VMs reduce even if the rate is high from the client. The new topology generated are:

- Sandpiper - PM84 with lucid08, PM161 with lucid12 and lucid13, PM180 with lucid14 and lucid09
- Khanna's Algo - PM84 with lucid14, PM161 with lucid08 and lucid12, PM170 with lucid09 and lucid13
- Entropy - all on PM161

Table 6.4 describes the measured statistics. We see that both sandpiper and Khanna's Algo uses 3 PMs whereas Entropy uses just 1 PM. Sandpiper and Khanna's Algo reacts to hotspots when they are formed, whereas Entropy doesn't wait for hotspots or coldspots to happen, based on the current topology configuration, it searches for an optimal reconfiguration plan, once it finds it, the reconfiguration plan is implemented. That is why in this experiment entropy took only more than 1 minute to generate the new plan whereas, Sandpiper used 3 PMs after approximately 9.34 minutes from the start of the experiment and Khanna's Algo took more than 8 minutes to reduce the number of PMs. Also, Sandpiper triggers just 1 migration over Khanna's Algo and entropy.

Table 6.4: Experiment-3 - with varying rates of workload - Measured Evaluation Metrics

Algorithm	**No.of PMs**	**No.of Migrations**	**Time Taken(mins)**
Sandpiper	3	1	N/A
Khanna's Algo	3	5	8.19
Entropy	1	3	1.5

As mentioned earlier we had design plans to perform evaluation of application performance which is very crucial for our analysis. But due to very unexpected circumstances we could not log valid correct response time values of all the experiments from which we could infer some concrete results. Here, we present the response time variation of 1 VM, lucid12 across all the three algorithms.

From Figure 6.10 we observe that response time for lucid12 in case of sandpiper is higher than Khanna's Algo and entropy. As seen from the graph, in experimenting with Khanna's Algo, the average response time was approximately 200 milliseconds with transient spikes where the response time value increased to as high as 600 millisecs and 1100 millisecs. The reason behind this sudden increase is the fact, that PM161 which hosted lucid12, triggered migrations of lucid13(co-located VM). PM161 had been the source as well destination for three migrations. And we know that the application performance of co-located VMs are affected in such cases. We

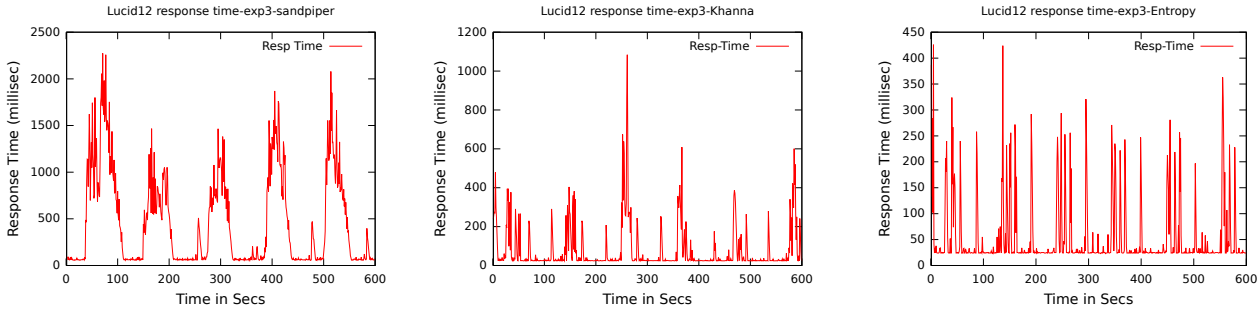


Figure 6.10: Response Time Variation of lucid12

feel reception and migration of VMs from the same host where lucid12 resides must have been the reason of sudden spikes in the response time.

In case of Entropy, the results are in conformity to the events trigggered by the algorithm. PM161 where lucid12 resides was chosen as the destination PM for all the VMs in the topology. As a result, PM161 had been the destination PM for three migrations. This increases the cpu usage of PM161 slightly affecting the response time at the server on lucid12. Inaddition to this, addition of three more VMs in lucid12's host and colocation amongst four other VMs must have affected its request processing, affecting its application performance. In khanna's algo there are infrequent spikes, but in entropy there are no spikes, the increase in response times are persistent, the average is very close to each other.

Unexpectedly, in sandpiper the response time values are more compared to both Khanna's Algo and entropy both of which are comparable. Although in Sandpiper, lucid12 was not migrated neither did its host PM161 undergo VM receptions from other hosts, the response times are higher. This is because of the fact that, the mysql server running at the VM lucid12 got blocked because of too many connections for some time. When the workload was generated with varied rates, since database was inaccessible for some time, the response time values increased and this had an affect on the overall response time values of the entire experimental run.
However, we notice that the application performance of the VM gets affected if the host PM triggers too many migrations or undergoes VM receptions. A migrating VM will undergo degraded application performance. Thus, in this experiment, entropy does best in terms of application performance. Following Figure 6.11 makes the illustration clearer:

From the above experiments we can conclude the following:

- In case of varying workloads, when there are changing cpu utilization levels, we see entropy perform better than Khanna's Algo. It uses less number of PMs and triggers less number of migrations over Khanna's Algo. Moreover, it didnot take much time to procure the reconfiguration plan and initiate the new mapping. It is within acceptable limits to cross the upper cpu threshold, at the benefit of obtaining reduced number of PMs with less migration overhead.

- Too many migrations in the system always induces some amount of instability in the system, also, we can observe the relatively higher variation of cpu utilizations across the PMs with

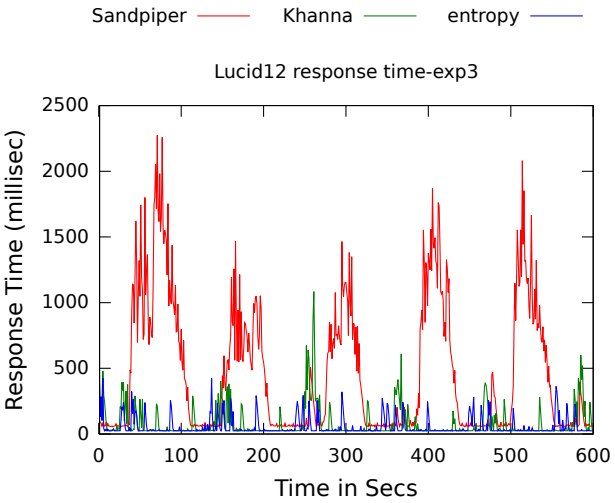


Figure 6.11: Lucid12-Response Times

Khanna's algorithm than the others. Our reasoning is that it is because of frequent VM relocation across PMs which happens in Khanna's Algo.

- Khanna's algorithm seems to be more befitting in an environment where we need cpu usages within specified thresholds as a high priority requirement, in the process of performing consolidation. We do see that better consolidation is possible than what Khanna's Algo does, but the cpu usage at the destination PM may reach too high, beyong acceptance level in some cases.
- The application performance of lucid12 is best in Entropy. But unless we analyze the application performance of all the other VMs present in the topology we cannot concretely say anything about the goodness of the algorithms.

In the next experiment we decrease the upper cpu threshold, increase the number of VMs and see what happens.

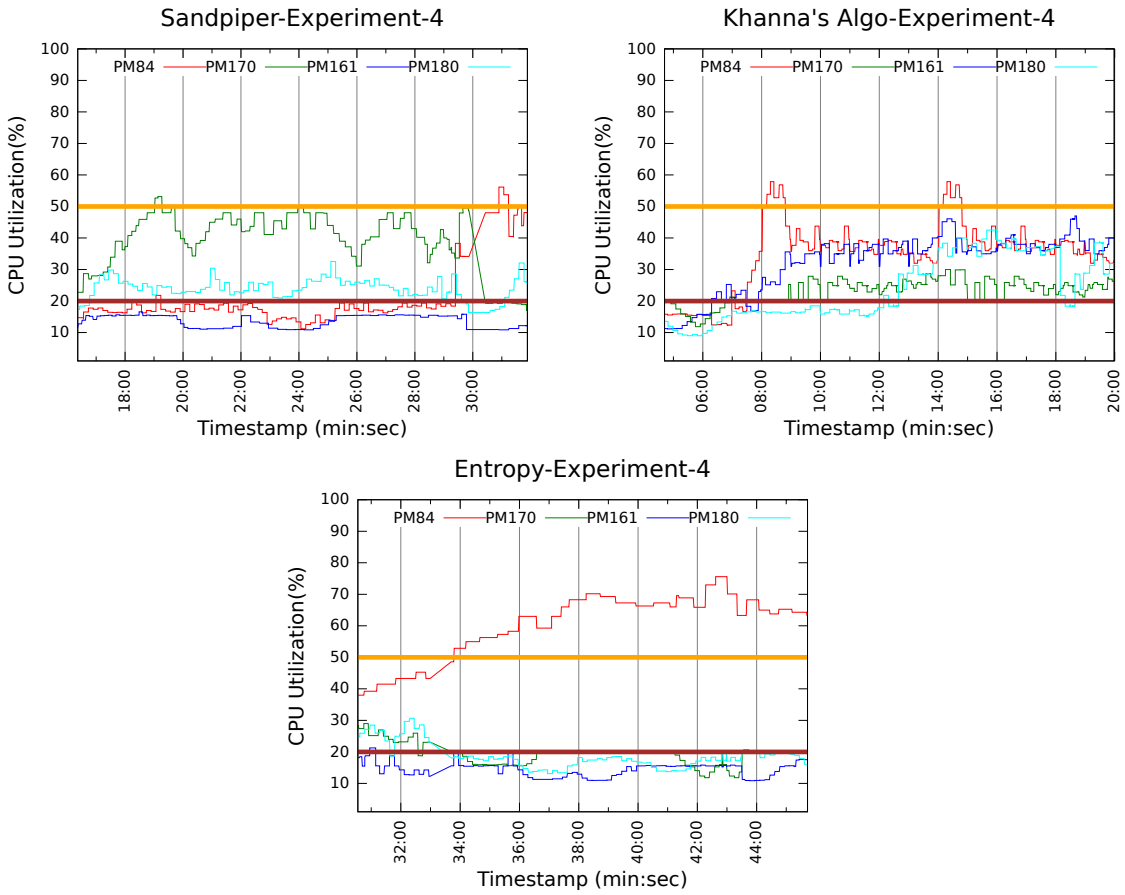
6.3.4 Scenario-4 with both varying and fixed rates of workload

In this experiment, we intended to design a slightly more complex scenario where we have both workload with varying rates and workload with fixed rates. This is different from the previous experiments and will affect the migration decision process. Design was made to avoid coldspots in the PMs, and to keep the cpu usage of the PMs higher than 20%. This could create the possibility of hotspots alone. However, in the beginning, the values of cpu usage reported was low for which coldspot did get detected. We created the following scenario:

PM84 - lucid09 and lucid12 - rate 5req/sec on lucid12 and rate 30 req/sec on lucid09
PM161 - empty
PM170 - lucid14 - rate of 10 req/sec followed by 20 req/sec
PM180 - lucid08 and lucid13 - rate of 10 req/sec on lucid08 & 10req/sec followed by 30req/sec on lucid13

The duration of the experiment was **15 minutes** for all the algorithms. The upper cpu threshold is set to 50 here.

Figure 6.12: Experiment-4 cpu usage versus time across all 4 PMs



From Figure 6.12 we see that Khanna's algorithm tries to bring the cpu utilization of the PMs between the lower and upper thresholds. Since the gap between lower and upper cpu threshold is just 30% the variation of resource utilization across PMs is not much. Clearly, In Entropy, we can easily see that the PM84's cpu usage increases, and the PM's usage falls, which implies that VM from other PMs must have been migrated to PM84. For sandpiper, the cpu usage of PM170 is much higher than the others, with a sudden rise in PM84's cpu usage. Sandpiper detects two hotspots and triggers two migrations to mitigate them. Khanna's Algo triggers five migrations in all and entropy triggers three migrations.

From Figure 6.13 and 6.14, we can observe that Sandpiper detected a hotspot at PM170, migrated lucid14 to PM84, which already had 2 VMs running, after about 10 minutes, a hotspot was detected at PM84 which triggered the migration of lucid09 which had higher utilization, to move from PM84 to PM180. During the remaining duration, no other hotspots were triggered.

Khanna's algorithm, packed all the VMs in PM 161 which had the highest residual capacity. Initial topology had PM161 as empty, at the end of the experiment it had freed up all the 3 PMs filling up just PM161. It emptied PM84, PM180 and PM170 and migrated all the VMs to PM161 for which PM161's usage increased. Here, unexpectedly PM84's cpu usage is high even after the VMs it housed were migrated. We exactly donot know the reason, of such high cpu usage values in case of PM84. Based on our experience with experimentation even after subsequent migrations from a PM, the resource usage values shown by the APIs we have used for fetching data are slightly higher than expected.
Entropy packs all the VMs in PM84. Here an interesting thing to note is that, based on our topology design we can say that at max 3 VMs may be active at a time- lucid12, lucid13 and lucid14. PM84 had a quad core CPU, thus it can hold at max 4 active VMs, we conjecture that because PM84 had sufficient capacity to hold all active and inactive VMs, it was chosen as the destination for consolidation. Apparently, choosing PM84 or PM180 should reduce the migration cost. Our python CSP solver chose PM84. Entropy used 1 PM triggering 3 migrations to pack all 5 VMs in PM84. PM84's utilization increased to as high as 76%. The other PMs were released for which we observe a fall in the cpu usage, consolidating all the VMs in less than 3 minutes. The new topology generated is as follows:

- Sandpiper - PM180 hosted lucid08, lucid09 and lucid13, PM84 hosted lucid12 and lucid14
- Khanna's Algorithm - All VMs on PM161
- Entropy - All VMs on PM84

Table 6.5 summarizes the measured statistics. Sandpiper triggers 2 migrations and uses 2 PMs at the end of the run. Khanna's Algo uses only 1 PM, PM161 which had the highest residual capacity across all the PMs in approximately 7 minutes. Entropy used 1 PM triggering 3 migrations and performed consolidation in less than 3 minutes. This experiment didn't prove to be of much importance in terms of the case where the number of hotspots formed are more and due to fluctuating workloads the residual capacity of PMs would vary.

Table 6.5: Experiment-4 - With both varying & Fixed rates of workload - Measured Evaluation Metrics

Algorithm	**No.of PMs**	**No.of Migrations**	**Time Taken(mins)**
Sandpiper	2	2	N/A
Khanna's Algo	1	5	7.3
Entropy	1	3	2.78

Takeaway points from this experiment:

- Mix of varying and fixed workloads didnot have much impact on the migration decision process. Neither did the resource utilization within a PM fluctuate much nor did the overall resource usage across the PMs increase too much. Depending on the coldspots detected, and the topology given as input, Khanna's Algo may consolidate using lesser number of PMs. Even when the overall PM usage is not varying much, depending on the available resource capacities, Khanna's Algo packs VMs. The variation of PM usage in Entropy increases as all the VMs are migrated to 1 PM.
- As mentioned earlier after migrations are performed from the source PMs, cpu usage reported by the underlying software framework we used are high. Example is PM84's cpu

Figure 6.13: Experiment-4 VM Migration Sequence in 4 PMs for Sandpiper and Khanna's Algo

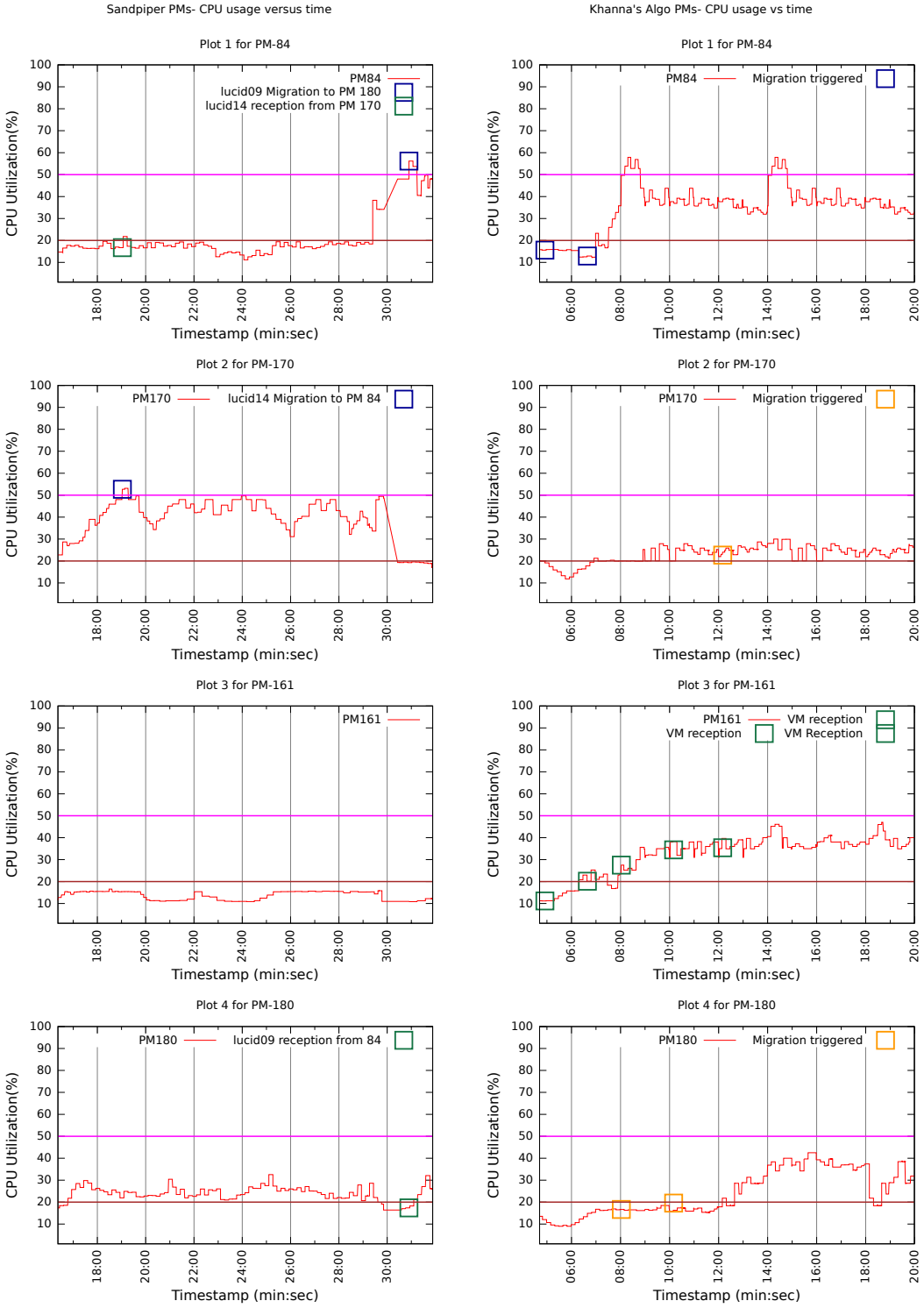
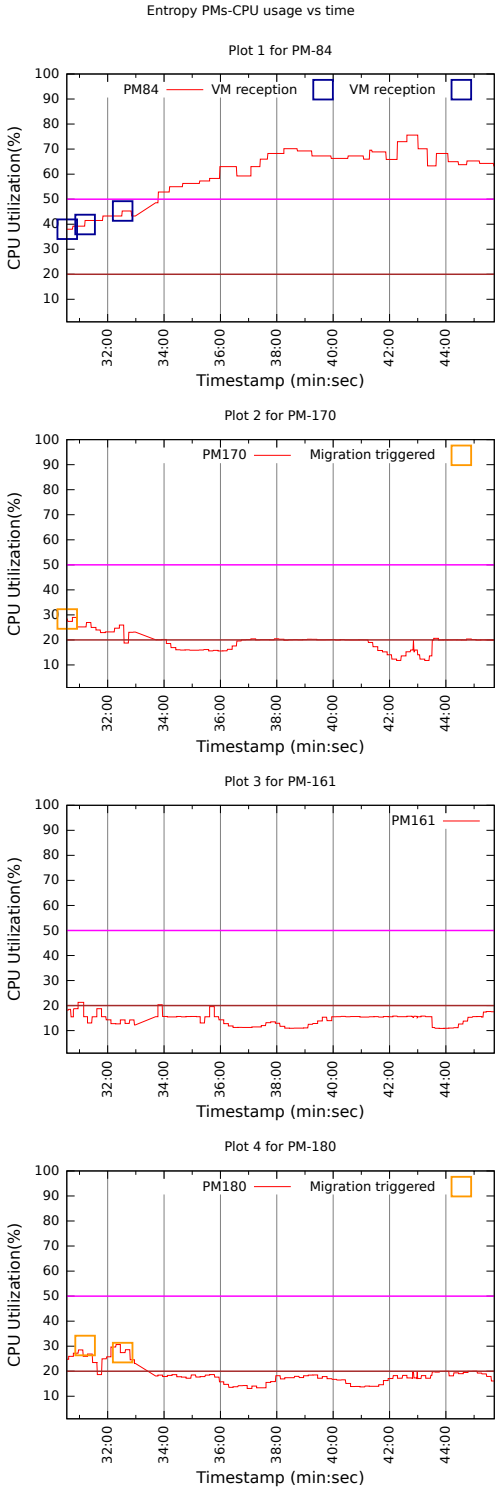


Figure 6.14: Exp-4-Entropy Algo- VM Migration sequence across 4 PMs



usage in case of Khanna's Algo experiment. Although, we donot know the exact reason, but we feel that some minor staggering of values might happen from the actual drop in usage values at the PMs, to the values which are reported during the experimentation and logging. Nevertheless, even if that happens the variation will be too less to cause a major change as far as the comparative analysis is concerned.

6.3.5 Scenario-5 - Only experiment with RUBiS

In this experiment we used RUBiS benchmarking tool to create load on the VMs. The intention was to change the type of application used in this experiment and also observe the cpu usage levels with this application. Since, changing the type of workload affects the migration decision process, we thought we should perform 1 experiment with RUBiS. Due to time constraint we couldnot do multiple experiments with RUBiS with changing request type. We used the same *seed* in the *rubis.properties* file at the client, to generate the same workload for all the experimental runs. The number of clients per node parameter in RUBiS was also kept constant across all the algorithms. The upper cpu threshold was set to 60%. The experimental duration was 15 minutes. We created the following scenario:

PM84 - lucid14 - load with RUBiS Client
PM161 - lucid13 - no load
PM170 - lucid09 -load with RUBiS Client
PM180 - lucid12 -load with RUBiS Client

From Figure 6.15 we notice that for all the PMs the cpu usage levels are high in the beginning and then it slows down. Sandpiper detects two hotspots and triggers two migrations to mitigate the hotspots. Khanna's Algo mitigates one hotspot and triggers four migrations in total. Entropy uses one PM and triggers three migrations to consolidate all the VMs on PM180. The new topology generated are as follows:

- Sandpiper - PM161 has lucid13 and lucid09, PM170 has lucid14, PM180 has lucid12
- Khanna's Algo - PM161 has lucid12, lucid13 and lucid14, PM84 has lucid09
- Entropy - All VMs on PM180

Let us observe Figures 6.16 and 6.17. Sandpiper detects a hotspot on PM84, migrates lucid14 from PM84 to PM170, after 9 minutes detects a hotspot on PM170 and migrates lucid09 from PM170 to PM161. Khanna's Algo migrates all the VMs each from PM170, PM84 and PM180 to PM161 after detecting a coldspot. This consolidates the VMs on 1 PM which is PM161. But, since PM161 exceeded the threshold after about 7 minutes, lucid09 was migrated from PM161 to PM84. Thus, at the end of the run, it uses 2 PMs. Entropy chooses PM180 to pack all the VMs. The overall cpu usage level in Entropy remains low, and doesn't exceed beyond 50%. Also, the number of PMs used by Entropy in all the experiments till now is just one. Designing experiments such that the number of active PMs exceed the total number of cores in the PM using synthetic benchmarks still needs to be done. That will provide a better understanding of the working of entropy algorithm.

Entropy uses 1 PM in this experiment using 3 migrations. Khanna's Algo uses 4 migrations and reduces the number of PMs by 2. Sandpiper uses 3 PMs and triggers two migrations. Thus, entropy does best here. Khanna's Algo uses 2 PMs at the end of 9 minutes of the experimentation

Figure 6.15: Experiment-5 cpu usage versus time across 4 PMs

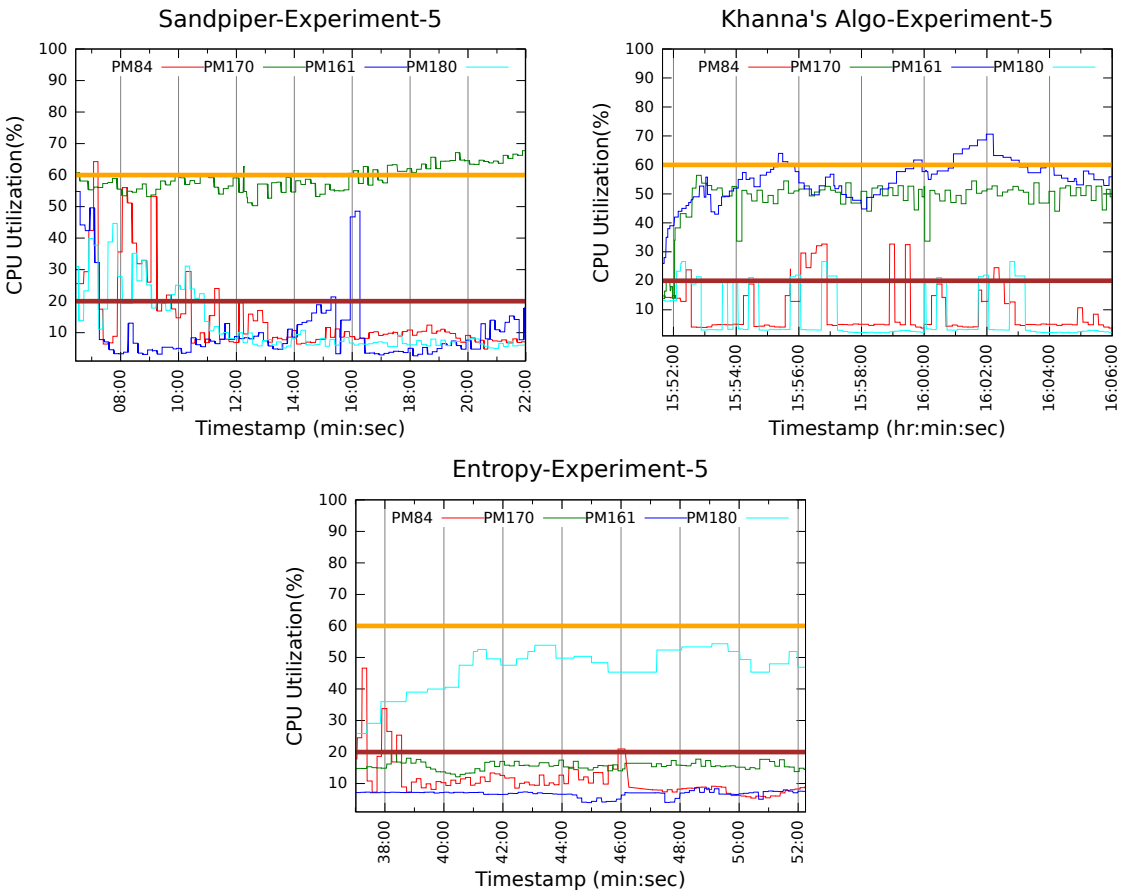


Figure 6.16: Experiment-5- VM migration sequence in 4 PMs for Sandpiper and Khanna's Algo

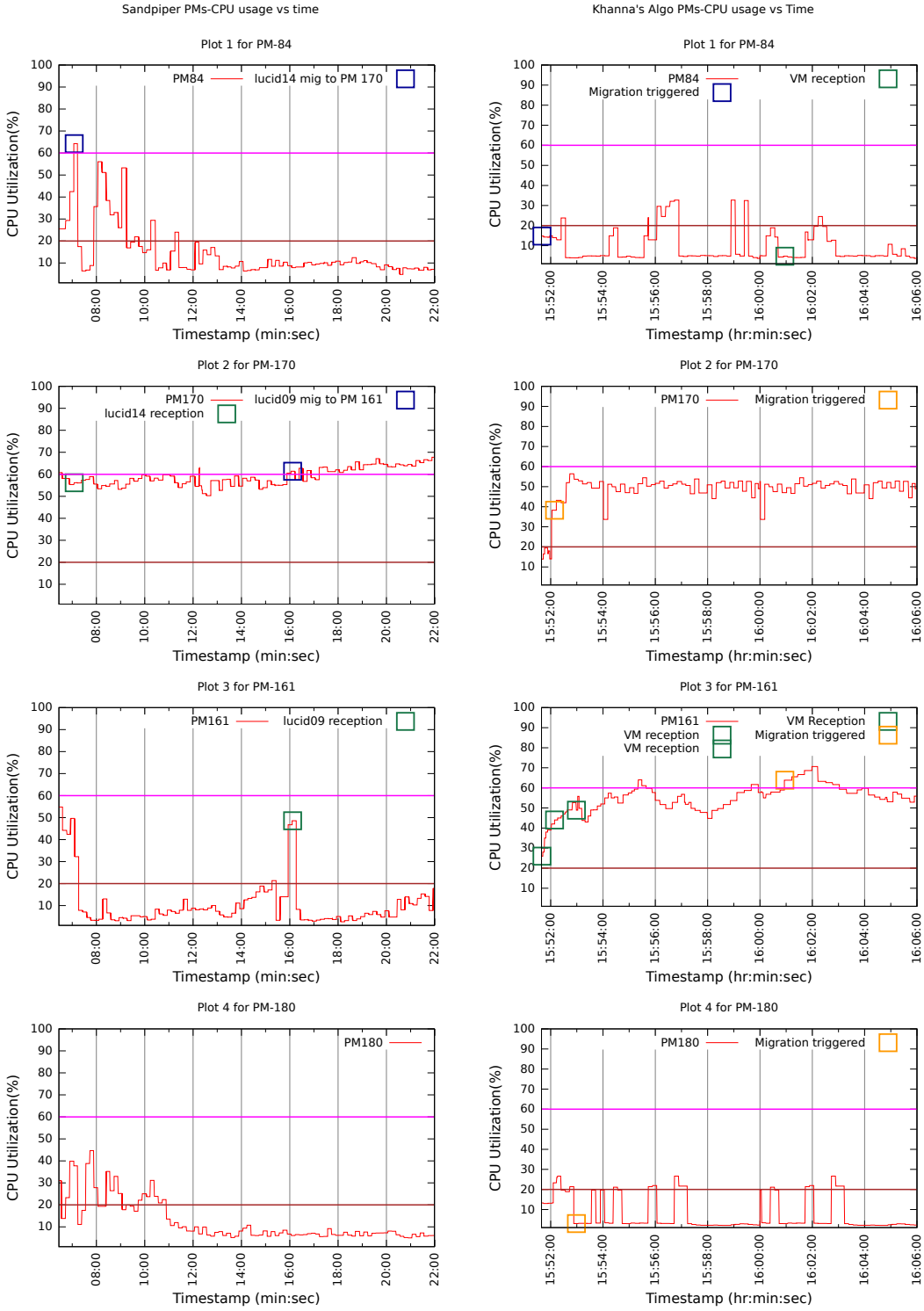
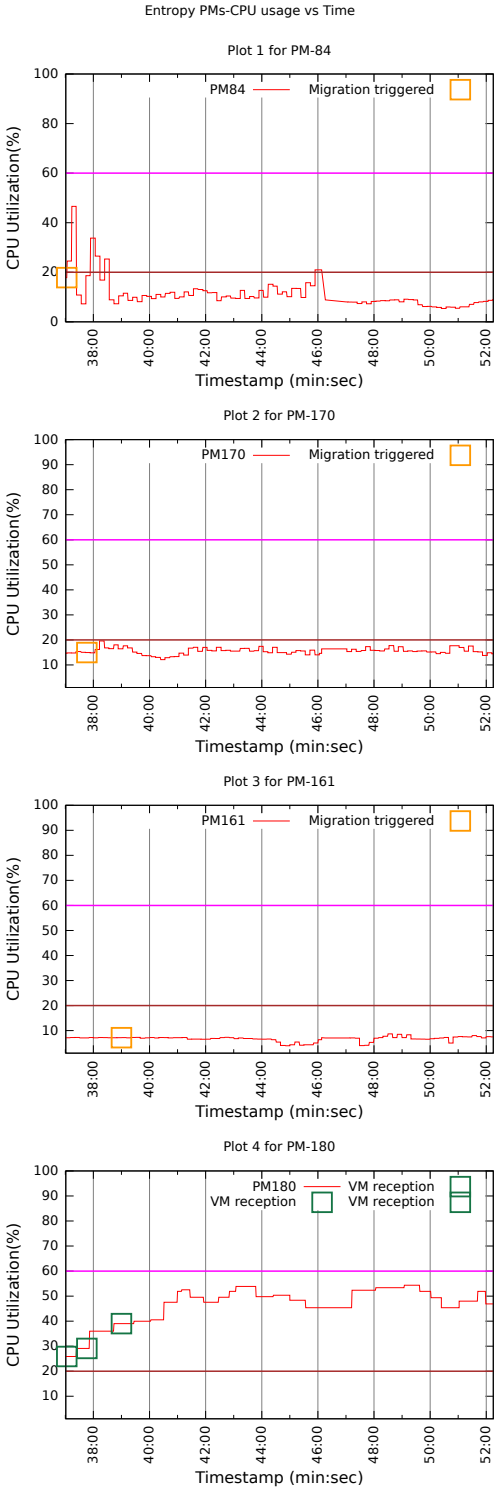


Figure 6.17: Exp-5-Entropy Algo- VM Migration sequence across 4 PMs



and entropy 1 PM by the end of 3 minutes of the experimentation. Table 6.6 lists the results.

Table 6.6: Experiment-5 - With RUBiS application - Measured Evaluation Metrics

Algorithm	**No.of PMs**	**No.of Migrations**	**Time Taken(mins)**
Sandpiper	3	2	N/A
Khanna's Algo	2	4	9
Entropy	1	3	3

Takeaway from this experiment:

- Unlike httperf, moderate RUBiS workload doesnot rapidly increase the cpu usage levels at the VMs. The relative cpu usage level of the VMs are low, compared to httperf which increased cpu usage even at low rates. In our case since, we did a small scale experimentation, the CSP solver didnot take much time to generate the optimal number of PMs or the reconfiguration plan. We noticed that it was in the order of milliseconds. Entropy is more robust for consolidation as in all our experiments it uses just 1 PM, we are aware that considerable cases have not been considered where entropy may take lot of time to generate the reconfiguration plan. Also, as mentioned earlier, inface of infrequent bursty data entropy consolidation algorithm may not have sufficient available resources to house the VMs. Other than these points, as long as workloads are moderately changing and the cluster is not too big (this will impact the time to generate the optimal reconfiguration plan) entropy incurs much less migration overhead than Khanna's Algo. Since Khanna's Algo uses a threshold based approach, it is more sensitive to the thresholds set. Entropy does global optimization which may be a worthwhile choice for server consolidation in a medium sized cluster.

6.3.6 Scenario-6 with varying workloads to create only hotspots

We used again httperf for our experiment. Our experimentation design plan was t0 generate a case where only hotspots occur, such that cpu usage levels are always higher than 20%. This design is important for our experimentation to evaluate the comparative analysis when no coldspot mitigation is triggered by khanna's algorithm. We had created the following scenario:

PM84 - lucid13 and lucid14 - load with varying rates of 20 req/sec, 30 req/sec and 40 req/sec on both
PM161 - lucid10 - Idle
PM170 - lucid12 -load with varying rates 20, 30 and 40 req/sec
PM180 - empty

We ran the experiments with 'httperf' for **10 minutes** duration for all the algorithms. The upper cpu threshold is set to 50 here.
From Figure 6.18 we notice that there is lot of fluctuations in the resource usage levels due to increased rate of workload on all the PMs. on all the PMs. The cpu usage levels do exceed the specified upper cpu threshold during the experimenation. Except PM180 which is empty, the cpu usage levels of all the PMs are more than the lower cpu threshold set, which was our expected design plan.

In Figure 6.19 and 6.20 we observe that Sandpiper detects two hotspots and triggers two migrations. lucid13 from PM84 to PM180 and lucid12 from PM170 is migrated to PM161. The

Figure 6.18: Experiment-6 cpu usage versus time across 4 PMs

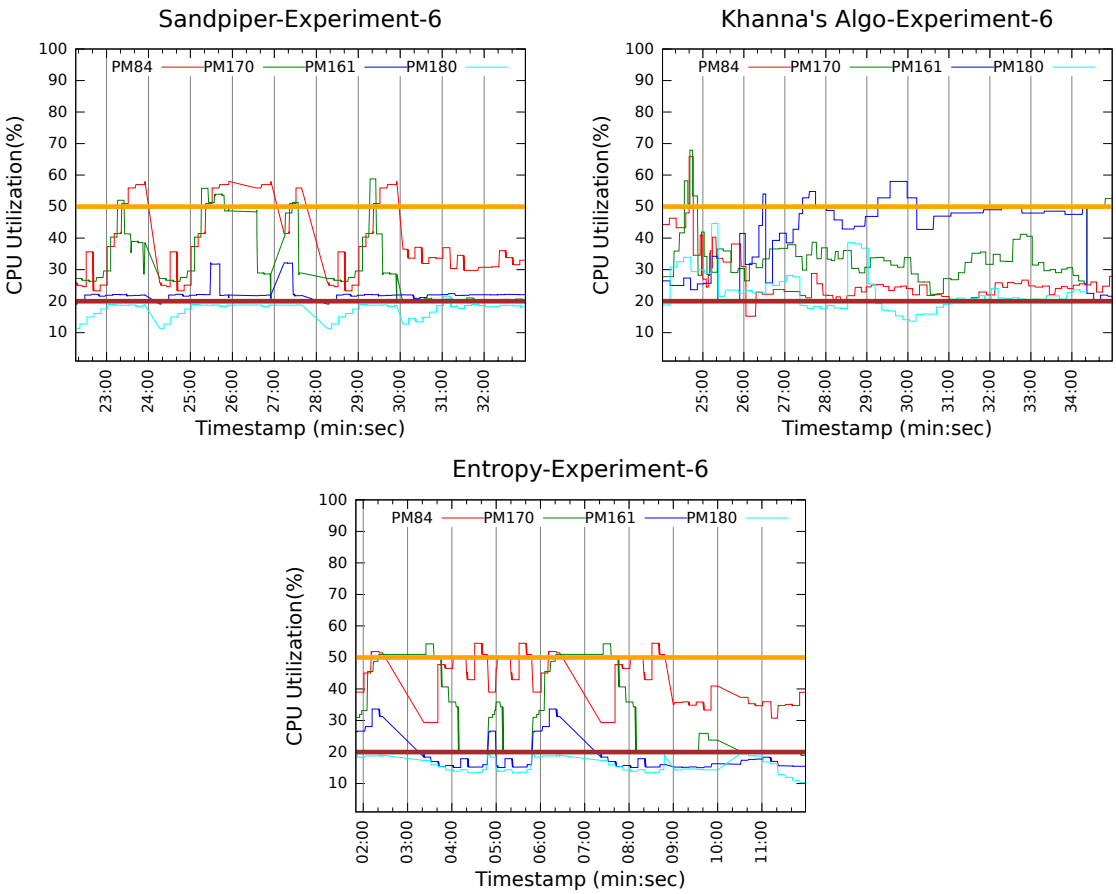


Figure 6.19: Experiment-6 -VM migration sequence in 4 PMs for Sandpiper and Khanna's Algo

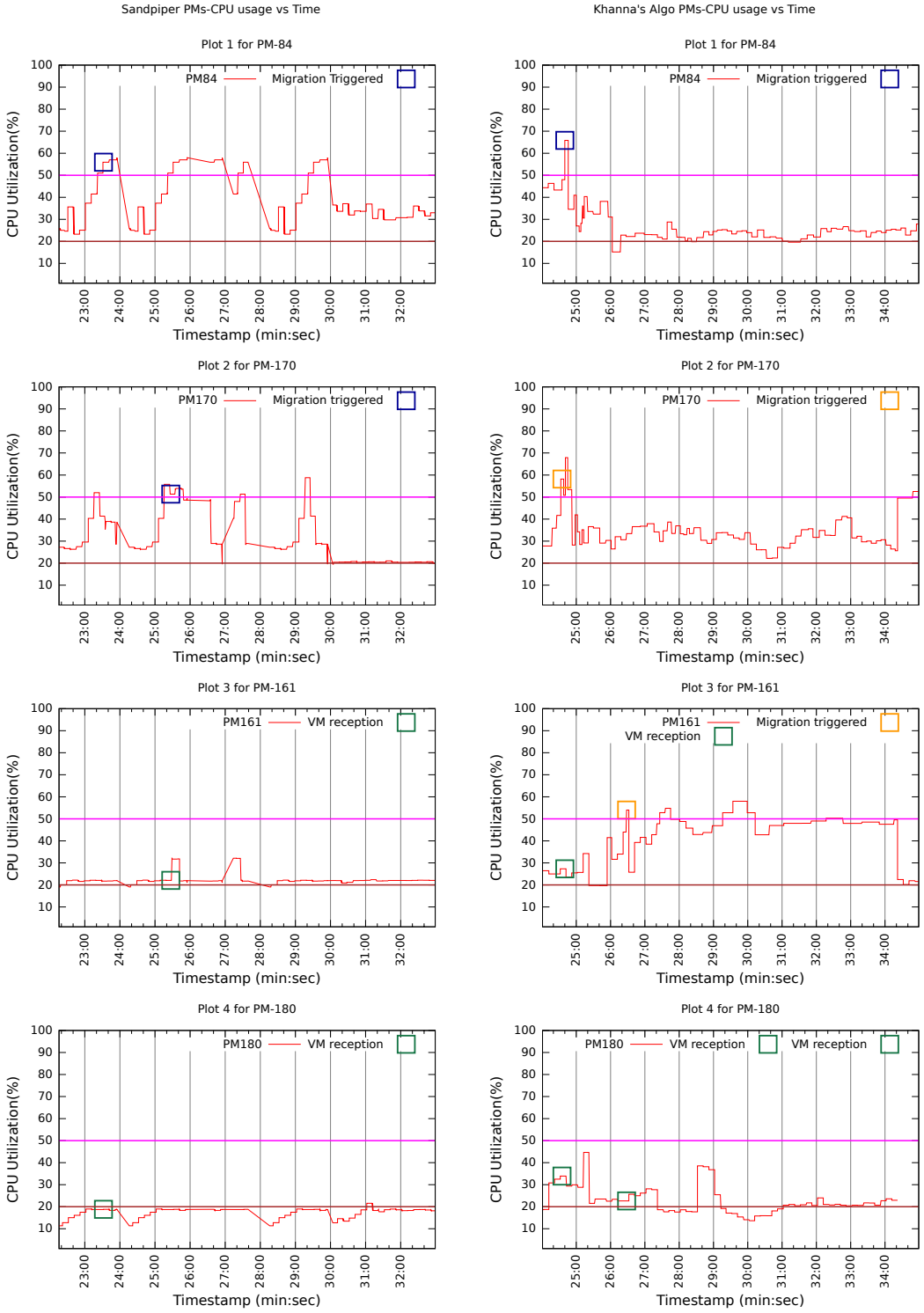
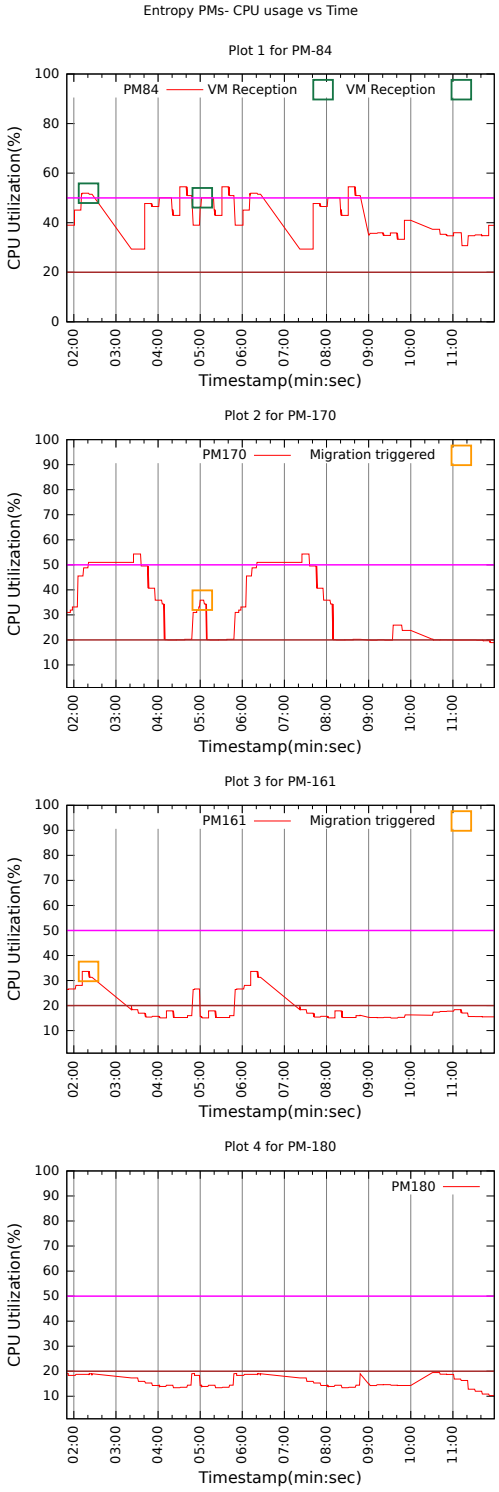


Figure 6.20: Exp-6-Entropy Algo- VM Migration sequence across 4 PMs



first hotspot is detected within first 1.5 minutes and the second hotspot is detected within 2 minutes of the detection of the first hotspot. Khanna's Algo detects three hotspots and mitigates the hotspots by triggering three migrations, lucid14 from PM84 to PM161, lucid12 from PM170 to PM180, and lucid10 from PM161 to PM180. Triggers three migrations and uses three PMs. Entropy uses PM84 to packs all the VMs. Two migrations are triggered by Entropy. Here no coldspots were detected. The new topology generated are as follows:

- Sandpiper - PM84 has lucid14, PM161 has lucid10 and lucid12 PM180 has lucid13
- Khanna's Algo - PM84 has lucid13, PM161 has lucid14, PM180 has lucid12 and lucid10
- Entropy - all VMs on PM84

Table 6.7 summarizes the measured statistics. Sandpiper mitigates two hotspots and uses 3 PMs within 4 minutes, Khanna's Algo mitigates three hotspots in less than 3 minutes, after which it uses 3 PMs, and Entropy consolidates all the VMs in 1 PM in approximately 3 minutes.

Table 6.7: Experiment-6 with varying workloads to check efficiency of hotspot mitigation - Measured Evaluation Metrics

Algorithm	**No.of PMs**	**No.of Migrations**	**Time Taken(mins)**
Sandpiper	3	2	N/A
Khanna	3	3	3
Entropy	1	2	3.12

Takeaway from this experiment:

- In case of hotspot mitigation alone, the no. of migrations triggered by Khanna's Algo is comparable to entropy, unlike Experiments 1 and 2 where the no. of migrations triggered were more. Although just a few experiments donot establish the fact that Khanna's Algo is efficient, Sandpiper which is a hotspot mitigation algorithm also takes around 3 minutes time to mitigate all the hotspots and uses 3 PMs like Khanna's Algo.
- We expected similar resource usage profiles for different algorithms, but as seen from the figures, profiles are slightly different. Also, triggering migration of different VMs with different cpu utilizations varies the PM cpu usage profiles further.
- Entropy again uses 1 PM to pack all the VMs. Entropy evaluation in our work is very restricted, as we donot explicitly make sure that a non-viable configuration is reached, which chooses multiple PMs for consolidation.
- The comparative analysis along with the PM cpu usage profiles, elicits out the fact that entropy is a more robust consolidation algorithm, which rigorously packs VMs on a PM. Since its not a threshold based approach like Khanna's Algo, it hosts VMs on a PM as long as the overall resource usage doesnot shootup to 100%.

6.4 Results

From the above set of experimentations we have got some preliminary results about the nature of algorithms. Following points summarize the results we obtain:

- Entropy is more robust than Khanna's algorithm in terms of efficiency of consolidation. Since it is insensitive to resource thresholds set unlike khanna's Algo, it packs VMs trying to fully utilize its resources, as long as the PM's capacity is not exceeded. More experiments with non-viable configuration would further reveal the efficacy of entropy.

- Khanna's algorithm takes care of the future possibility of VM accomodation in a PM, for which it may be able to accomodate VMs in face of bursty data. It tries to prevent triggering of future migrations from the destination, it chooses at any instant of time. This feature may help in data center environments where sudden surge of workloads occur, and there is no fixed pattern of workload variation.

- In face of infrequent bursts of data entropy may be costly over khanna's algo. Since in most of our experiments we have seen that Entropy uses 1 PM to pack all the VMs, viable condition being satisfied, in case of infrequent unpredictable bursty traffic, when non-viable configuration is created, the system may not have sufficient available resources to provide for the active VMs. It is possible that, the algorithm overpacks the VMs in such a way that too less residual capacity is left, in that case, sudden change of inactive VMs to active VMs may affect the application performance and cost of generating reconfiguration plan.

- Khanna's algorithm is befitting where we prioritize in keeping the resource usages within specified levels in the process of doing consolidation. This algorithm is too sensitive to thresholds, and violation of thresholds trigger series of migrations to ensure that the resource usage values are within pre-defined bounds.

- Sandpiper triggers less number of migrations since it does hotspot mitigation alone. In this context in Exp 3 and Exp 6 both Sandpiper and Khanna's Algo use the same no. of PMs. The implication we get is that coldspot mitigation makes sense when a certain fraction of PMs amongst all present in the topology are underutlized rather than detecting coldspot on just 1 PM and starting the mitigation process. The tradeoff between the migration overhead incurred for running the coldspot mitigation heuristic and the number of PMs reduced have to be analysed further. That can help in using Khanna's Algo more effectively.

- We have got higher number of migrations triggered by Khanna's Algo in most of the experiments amongst the other two, incurring more migration overhead, causing unrest in the system. As we have seen from the cpu usage graphs, in Khanna's algorithm, PMs comparatively undergo more fluctations and variations in cpu usages. Entropy is relatively stable, as it doesn't locally scan and instantly trigger migrations, rather it considers the entire topology and finds the globally optimal solution by procuring the series of migrations to be triggered. This keeps the cpu usage levels amongst the PMs relatively more stable than Khanna's Algo. Hence entropy is a better choice for a stable system over Khanna's algorithm.

- As long as workloads are moderately changing and the cluster is not too big (this will impact the time to generate the optimal reconfiguration plan) entropy incurs much less migration overhead than Khanna's Algo.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Our work attempted to do a comparative analysis of three chosen heuristics for live virtual machine migration based on some empirical evaluation. The three chosen algorithms are sandpiper, Khanna's algorithm and Entropy. We have prototyped these algorithms using our developed software framework which is a mesaurement and monitoring tool for cluster management. The APIs supported by the tool helped in rapid prototyping of algorithms. We have taken the help of Google OR tools to prototype Entropy algorithm which follows a contstraint programming based approach to perform consolidation. To verify the correctness of the prototyped algorithms we have generated test cases to test the algorithms. The algorithms correctly triggered migrations performing the expected events. We have used 'httperf' and 'RUBiS' benchmarking tool to generate workload on the virtual machines for experimentation and evaluation. We designed experiments to check the performance of the algorithms based on some pre-defined metrics like no. of migrations triggered, no. of physical machines used, time taken to reduce the number of PMs. Based on some simple set of experiments, we have obtained some preliminary results which has been discussed in the previous sections. We have got some insight into the nature of the algorithms, and their applicability in certain scenarios. For example, our empirical study illustrates that entropy algorithm may not be a good choice in cases which deals with infrequent bursty traffic, khanna's algo may be better choice etc. This kind of work has not been done by the research community before. This is a first of its kind of work done. We believe that developing the cluster management framework with API support, and doing a comparative study of three chosen algorithms using this framework contributes substantially to the investigation of applicability of a specific algorithm for consolidation or hotspot mitigation.

7.2 Future Work

This project can have lot of extensions. Following are some future work:

- Better evaluation of application performance in terms of throughput, response time can be done to analyze the extent of degradation in face of high workloads.
- No experiments have been done with mixed workloads where different types of applications are used together. This can reveal the impact of chosen application on the effect of consolidation. For instance memory-intensive workload or I/O intensive etc.

- More complex scenarios can be designed for experimentation which creates non-viable configuation on PMs, or creates many hotspots. The behaviour of algorithms can be checked in a wide variety of scanarios which can gain insight into their goodness.
- Increasing the scale of experimentation. Investigations can be done on finding out whether the behaviour of algorithms change when the no. of PMs and VMs are increased in terms of the metrics we defined to compare the goodness of the algorithms.
- Choose more algorithms which does similar job, like consolidation and check their relative behaviour with the already chosen algorithms. Evaluation of more algorithms can facilitate in figuring out the distinct cases where an algorithm will behave well, and hence can be used in those cases only to leverage maximum benefits.
- Deploy the framework which places the decision engine containing the prototyped algorithms in Machines which have server configurations. Our work was on a Lab set-up with less powerful machines. Experimentation in server machines, like IBM blade servers can provide better conviction and validity to the results.

Bibliography

[1] Eucalyptus Cloud. http://open.eucalyptus.com/wiki/IntroducingEucalyptus v2.0.

[2] NFS set-up in Ubuntu. https://help.ubuntu.com/community/NFSv4Howto.

[3] Nmap official website. http://nmap.org/.

[4] RRD Tool. http://oss.oetiker.ch/rrdtool/.

[5] RUBiS Benchmarking. http://rubis.ow2.org/.

[6] Xen Cloud Platform. Xen.org - http://www.xen.org/products/cloudxen.html.

[7] Aameek Singh , Madhukar Korupolu , Dushmanta Mohapatra. Server-Storage Virtualization: Integration and Load Balancing in Data Centers. *Proceedings of the SC 2008 ACM/IEEE conference on Supercomputing*, 2008.

[8] Akshat Verma and Puneet Ahuja and Anindya Neogi. Power-aware dynamic placement of hpc applications. *22nd ACM International Conference on SuperComputing(ICS)*, June 2008.

[9] Apache HTTP server 2.2. The Apache Software Foundation. Website, `http://www.apache.org/`.

[10] Emmanual Arzuaga and David R. Kaeli. Quantifying load imbalance on virtualized enterprise servers. *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 235–242, 2010.

[11] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82:47–111, 2011.

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

[13] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiberg. Live wide-area migration of virtual machines including local persistent state. *International Conference on Virtual Execution Environments(VEE)*, 2007.

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

[15] Umesh Deshpande, Xiaoshuang Wang, and Kartik Gopalan. Live gang migration of virtual machines. *HPDC:High-Performance Parallel and Distributed Computing*, June 2011.

[16] Qingyi Gao, Peng Tang, Ting Deng, and Tianyu Wo. VirtualRank : A Prediction Based Load Balancing Technique In Virtual Computing Environment. *IEEE World Congress on Services*, 2011.

[17] Google OR tools. http://code.google.com/p/or-tools/.

[18] James Greensky, Jason Sonnek, Robert Reutiman, and Abhishek Chandra. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. *39th International Conference on Parallel Processing (ICPP)*, pages 228–237, September 2010.

[19] Laura Grit, David Irwin, Aydan Yumerefendi, and Jeff Chase. Virtual machine hosting for networked clusters:building the foundations for autonomic orchestration. *VTDC:Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, 2006.

[20] Ajay Gulati, Ganesha Shanmuganathan, Irfan Ahmad, and Anne Holler. Cloud Scale Resource Management: Challenges and Techniques. *3rd Usenix Workshop on HotCloud*, June 2011.

[21] Fabien Hermenier, Gilles Muller, Xavier Lorca, Julia Lawall, and Jean-Marc Menaud. Entropy: a consolidation manager for clusters. *Virtual execution environments (VEE)*, Mar. 2009.

[22] Kejiang Ye and Xiaohong Jiang and Dawei Huang and Jianhai Chen and Bei Wang. Live Migration of Multiple Virtual Machines with Resource Reservation in Cloud Computing Environments. *IEEE 4th International Conference on Cloud Computing*, 2011.

[23] G. Keller and H. Lutfiyya. Replication and migration as resource management mechanisms for virtualized environments. *Proceedings of the Sixth International Conference on Autonomic and Autonomous Systems (ICAS)*, pages 137–143 , 7–13, 2010.

[24] Gunjan Khanna, Kirk Beaty, Gautam Dhar, and Andrzej Kochut. Application performance management in virtualized server environments. *Network Operations and Management Symposium NOMS 10th IEEE/IFIP*, pages 373–381, 2006.

[25] Zhen Liu, Mark S. Squillante, Cathy H. Xia, Shun-Zheng Yu, and Li Zhang. Profile-based traffic characterization of commercial web sites. *Proc. of the 18th International Teletraffic Congress (ITC18)*, 2003.

[26] Httperf load generator. http://www.hpl.hp.com/research/linux/httperf/.

[27] Mayank Mishra, Anwesha Das, Purushottam Kulkarni, and Anirudha Sahoo. Dynamic resource management using virtual machine migrations. *Accepted in IEEE Communications Magazine*, 2012.

[28] Muffin proxy 0.9.3a. Muffin project community. Website, `http://muffin.doit.org/`.

[29] MySQL database server 5.0. MySQL community. Website, `http://www.mysql.com/`.

[30] Ripal Nathuji and Karsten Schwan. Virtualpower: Coordinated power management in virtualized enterprise systems. *In Proc. ACM SOSP*, 2007.

[31] Official Xen project site . http://www.cl.cam.ac.uk/research/srg/netos/xen/.

[32] Piyush Masrani. Dynamic CPU Share Allocation and Server Consolidation in Virtualized Data Centers. Technical report, M.Tech. Thesis Indian Institute of Technology, Bombay, Mumbai , India, June 2009.

[33] M Uysal Zhikui Wang Pradeep padala, X.Zhu. Automated control of multiple virtualized resources. *Proceedings of the 4th ACM European conference on Computer systems,Eurosys*, 2009.

[34] T.Wood, G. Tarasuk-Levin, Prashant Shenoy, Peter desnoyers, Emmanuel Ceccheta, and M.D.Corner. Memory buddies : Exploiting page sharing for smart colocation in virtualized data centers. *Proceedings of the ACM SIGPLAN/SIGOPS international conference on Virtual execution environments VEE*, pages 31–40, 2009.

[35] WebCalendar application. WebCalendar developer community. Website, `http://sourceforge.net/projects/webcalendar/`.

[36] Timothy Wood, Prashant Shenoy, and Arun. Black-box and gray-box strategies for virtual machine migration. *4th USENIX Symposium on Networked Systems Design and Implementation*, 2007.

[37] Timothy Wood, Prashant Shenoy, K.K. Ramakrishnan, and Jacobus Van der Merwe. Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines. *International Conference on Virtual Execution Environments(VEE)*, March 2011.