New Jersey Institute of Technology
Digital Commons @ NJIT

Dissertations | Electronic Theses and Dissertations

Fall 1-31-2015

Enabling virtualization technologies for enhanced cloud computing

Kashifuddin Qazi
New Jersey Institute of Technology

Follow this and additional works at: https://digitalcommons.njit.edu/dissertations

 Part of the Computer Sciences Commons

Recommended Citation

Qazi, Kashifuddin, "Enabling virtualization technologies for enhanced cloud computing" (2015). *Dissertations*. 106.
https://digitalcommons.njit.edu/dissertations/106

This Dissertation is brought to you for free and open access by the Electronic Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Dissertations by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

NJIT New Jersey's Science & Technology University

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

ENABLING VIRTUALIZATION TECHNOLOGIES FOR ENHANCED CLOUD COMPUTING

by
Kashifuddin Qazi

Cloud Computing is a ubiquitous technology that offers various services for individual users, small businesses, as well as large scale organizations. Data-center owners maintain clusters of thousands of machines and lease out resources like CPU, memory, network bandwidth, and storage to clients. For organizations, cloud computing provides the means to offload server infrastructure and obtain resources on demand, which reduces setup costs as well as maintenance overheads. For individuals, cloud computing offers platforms, resources and services that would otherwise be unavailable to them.

At the core of cloud computing are various virtualization technologies and the resulting Virtual Machines (VMs). Virtualization enables cloud providers to host multiple VMs on a single Physical Machine (PM). The hallmark of VMs is the inability of the end-user to distinguish them from actual PMs. VMs allow cloud owners such essential features as live migration, which is the process of moving a VM from one PM to another while the VM is running, for various reasons.

Features of the cloud such as fault tolerance, geographical server placement, energy management, resource management, big data processing, parallel computing, etc. depend heavily on virtualization technologies. Improvements and breakthroughs in these technologies directly lead to introduction of new possibilities in the cloud. This thesis identifies and proposes innovations for such underlying VM technologies and tests their performance on a cluster of 16 machines with real world benchmarks. Specifically the issues of server load prediction, VM consolidation, live migration, and memory sharing are attempted.

First, a unique VM resource load prediction mechanism based on Chaos Theory is introduced that predicts server workloads with high accuracy. Based on these predictions, VMs are dynamically and autonomously relocated to different PMs in the cluster in an attempt to conserve energy. Experimental evaluations with a prototype on real world data-center load traces show that up to 80% of the unused PMs can be freed up and repurposed, with Service Level Objective (SLO) violations as little as 3%.

Second, issues in live migration of VMs are analyzed, based on which a new distributed approach is presented that allows network-efficient live migration of VMs. The approach amortizes the transfer of memory pages over the life of the VM, thus reducing network traffic during critical live migration. The prototype reduces network usage by up to 45% and lowers required time by up to 40% for live migration on various real-world loads.

Finally, a memory sharing and management approach called ACE-M is demonstrated that enables VMs to share and utilize all the memory available in the cluster remotely. Along with predictions on network and memory, this approach allows VMs to run applications with memory requirements much higher than physically available locally. It is experimentally shown that ACE-M reduces the memory performance degradation by about 75% and achieves a 40% lower network response time for memory intensive VMs.

A combination of these innovations to the virtualization technologies can minimize performance degradation of various VM attributes, which will ultimately lead to a better end-user experience.

ENABLING VIRTUALIZATION TECHNOLOGIES FOR ENHANCED CLOUD COMPUTING

by
Kashifuddin Qazi

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

January 2015

Copyright © 2015 by Kashifuddin Qazi

ALL RIGHTS RESERVED

APPROVAL PAGE

ENABLING VIRTUALIZATION TECHNOLOGIES FOR ENHANCED CLOUD COMPUTING

Kashifuddin Qazi

Dr. Andrew Sohn, Dissertation Advisor — Date
Associate Professor of Computer Science, NJIT

Dr. Alexandros Gerbessiotis, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Cristian Borcea, Committee Member — Date
Associate Professor and Associate Chair of Computer Science, NJIT

Dr. Durgamadhab Misra, Committee Member — Date
Professor of Electrical and Computer Engineering, Associate Chair for Graduate Studies, NJIT

Dr. James McHugh, Committee Member — Date
Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Kashifuddin Qazi

Degree: Doctor of Philosophy

Date: January 2015

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2015
- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2009
- Bachelor of Engineering in Electronics and Telecommunication,
  University of Mumbai, Mumbai, India, 2006

Major: Computer Science

Presentations and Publications:

K. Qazi, Y. Li, and A. Sohn, "Autonomous Elastic Cluster-Wide Memory for Virtual Machines," *ACM Transactions on Computer Science (Submitted)*, 2014.

K. Qazi, Y. Li, and A. Sohn, "Cluster-Wide Shared Memory Pages for Fast Live Migration," *IEEE Transactions on Cloud Computing (Submitted)*, 2014.

K. Qazi, Y. Li, and A. Sohn, "Harnessing Memory Page Distribution for Network-Efficient Amortized Live Migration," in *16th International Conference on High Performance Computing and Communications (HPCC).* Paris, France: IEEE, 2014.

K. Qazi, Y. Li, and A. Sohn, "Workload Prediction of Virtual Machines for Harnessing Data Center Resources," in *7th International Conference on Cloud Computing (CLOUD).* Anchorage, AK, USA: IEEE, 2014.

K. Qazi, Y. Li, and A. Sohn, "PoWER: Prediction of Workload for Energy Efficient Reconfiguration of Virtual Machines," in *4th Symposium on Cloud Computing (SoCC).* San Jose, CA, USA: ACM, 2013.

A. Sohn, K. Qazi, and Y. Li, "Methodically Illustrated Linux Kernel," *Text Book (under contract negotiation)*, 2014.

To my wife and parents - For maintaining their sanity through six years of my PhD, while mine was questionable.

ACKNOWLEDGMENT

I would like to begin by thanking my PhD adviser, Dr. Andrew Sohn for always being my mentor and guide, throughout my pursuit for a PhD. His concern and constant motivation have ensured the best work from me.

My appreciation also extends to the members of my dissertation committee (Dr. Cristian Borcea, Dr. Alexandros Gerbessiotis, Dr. Durgamadhab Misra, and Dr. James McHugh) for their invaluable time, effort, and feedback.

It would be amiss not to mention the department of Computer Science, especially Dr. James Geller, Dr. David Nassimi, and Dr. George Olsen for their constant support and advice. Special thanks are due to the late Dr. Marino Xanthos, Dr. Sotirios Ziavras, and Ms. Clarisa Gonzalez-Lenahan of the Graduate Studies for aiding in my development as a complete individual. Along with the Graduate Student Association and the student community at NJIT, I thank them for allowing me to serve as a student leader. The honor is all mine.

I would also like to appreciate my peers at NJIT, Dr. Ankur Agrawal, Dr. Atreyee Sinha, Mr. Kirtan Shah, and Mr. Siddharth Iyer, for the countless hours of intellectually stimulating discussions. I would be at a loss without them.

Finally, I would like to express deep gratitude to the three most important people in my life. My parents, Dr. Farida Qazi and Dr. Misbahuddin Qazi for instilling in me the thirst for knowledge, and my wife, Ms. Pashma Jumrani Qazi for supporting me and allowing me to pursue my ambitions. I am what I am because of, and for these three individuals.

TABLE OF CONTENTS

Chapter			Page
1	INTRODUCTION		1
	1.1	Internals of Cloud Computing	2
	1.2	Benefits of Cloud Computing	3
		1.2.1 Hardware Costs	4
		1.2.2 Maintenance Costs	4
		1.2.3 Availability	4
		1.2.4 Dynamic Resources	4
		1.2.5 Scalability	4
	1.3	Challenges in Cloud Computing	5
		1.3.1 Resource Management/ Utilization	5
		1.3.2 Fault Tolerance/ Reliability	5
		1.3.3 Security	6
	1.4	New Avenues in Cloud Computing	6
		1.4.1 Geographical Placement	6
		1.4.2 Federated Clouds	7
		1.4.3 Energy Management	7
		1.4.4 Automated Elasticity	7
		1.4.5 Parallel Computing/ Big Data	8
	1.5	Underlying Technologies	8
		1.5.1 VM Load Prediction	8
		1.5.2 VM Consolidation	9
		1.5.3 VM Live Migration	9
		1.5.4 VM Resource Sharing	9
		1.5.5 Nested Virtualization	9
	1.6	Proposed Innovations	10

TABLE OF CONTENTS
(Continued)

Chapter **Page**

1.6.1 Load Prediction and Consolidation for VMs 10
1.6.2 Live Migration of VMs 11
1.6.3 Memory Pooling of VMs 11
1.7 Summary 12
2 BACKGROUND 13
2.1 Load Prediction and Consolidation of VMs 13
2.1.1 Related Work 16
2.2 Live Migration of VMs 20
2.2.1 Related Work 24
2.3 Memory Pooling for VMs 26
2.3.1 Related Work 28
3 PREDICTION AND CONSOLIDATION OF VIRTUAL MACHINES 31
3.1 Chaos Theory for VM Loads 31
3.1.1 Chaotic Loads 32
3.1.2 Prediction using Chaos Theory 32
3.2 The Approach 33
3.2.1 Single VM Resource Load Forecasting 34
3.2.2 Hierarchical Representation 34
3.3 System Design 37
3.3.1 Stats Collector 37
3.3.2 Analyzer 39
3.3.3 Decision Maker 40
3.3.4 Migrator 41
3.4 Experimental Results and Evaluations 41
3.4.1 Prediction Analysis 42

TABLE OF CONTENTS
(Continued)

Chapter **Page**

3.4.2 Consolidation Analysis 45
3.4.3 Overhead Analysis 46
3.5 Summary 48
4 AMORTIZED LIVE MIGRATION OF VIRTUAL MACHINES 49
4.1 The Approach 49
4.1.1 Memory Page Distribution 49
4.1.2 Live Migration 51
4.2 System Design 54
4.2.1 Storing Memory and Mappings 54
4.2.2 The Protocol 56
4.3 Experimental Results and Evaluations 66
4.3.1 Base Approach Evaluation 67
4.3.2 Guided Mode Evaluation 70
4.4 Discussions 74
4.5 Summary 78
5 MEMORY POOLING FOR VIRTUAL MACHINES 79
5.1 The Approach 79
5.2 System Design 82
5.2.1 Prediction Module 82
5.2.2 Decision Module 83
5.2.3 Remote Memory Module 84
5.3 Experimental Results and Evaluations 86
5.3.1 Remote Memory Module Evaluation 88
5.3.2 Prediction Module Evaluation 92
5.3.3 Decision Module Evaluation 92

TABLE OF CONTENTS
(Continued)

Chapter **Page**

5.4 Discussions 96
5.5 Summary 99
6 CONCLUSIONS AND FUTURE WORK 101
6.1 Conclusions 101
6.2 Contributions 102
6.3 Future Work 102
BIBLIOGRAPHY 103

LIST OF TABLES

Table		Page
3.1	Significance of Each Data Point in a Level	39
3.2	Avg. % of PMs Freed Up/ % VMs Violating SLO	48
4.1	Avg. % of Pages Shared at Time of Migration (Pre-Copy)	67
4.2	Distribution of Pages during Migration (Post-Copy)	67
4.3	Avg. Pages in MB Transferred during Migration	68
4.4	Pre-Copy: Migrate Times (ms) for Different Network Bandwidths	71
4.5	Post-Copy: Times (ms) to Free Up Source for Different Network Bandwidths	72
4.6	Impact of Guidance Mode - Avg. Migration and Response Times in ms	73
5.1	Comparison of Time to Complete for Different Schemes with Varying Memory Splits - Intel LINPACK	88
5.2	Comparison of Time to Complete for Different Schemes with Varying Memory Splits - NASPB-IS	90

LIST OF FIGURES

Figure		Page
1.1	Virtualization.	3
2.1	Missing secondary cycles in load patterns.	19
2.2	Sample Google data trace.	19
2.3	Pre-Copy live migration	22
2.4	Post-Copy live migration.	22
2.5	Pre-Copy/ Post-Copy trade-off.	23
3.1	Illustration of hierachical analysis reducing data points.	35
3.2	Logical flow of Stats-Collector and Analyzer.	38
3.3	Different buffer padding overload and ffficiency.	40
3.4	Single VM resource load prediction sample.	43
3.5	Mean Square Error.	43
3.6	Cumulative distribution of MSE.	44
3.7	Average % VMs with error $>$ threshold.	44
3.8	Average physical machines freed up.	45
3.9	% VMs violating SLO.	45
3.10	Comparison of # data points.	47
3.11	Comparison of time to predict.	47
4.1	System overview.	50
4.2	Periodical.	51
4.3	During Pre-Copy.	52
4.4	During Post-Copy.	53
4.5	Collection and Push Phase details.	57
4.6	Pre-Copy: Migration Phase details.	62
4.7	Post-Copy: Migration Phase details.	64
4.8	Pre-Copy: Time comparison in % of base case.	74

LIST OF FIGURES
(Continued)

Figure **Page**

4.9 Post-Copy: Time to free source comparison in % of base case. 74

4.10 Average data over N/W during migration (MB) (For Post-Copy, lower portion is for Pri. Src. while upper portion is for Sec. Src.). 75

4.11 Migration Time CDF - with and without Guidance Mode. 77

4.12 Response Time CDF - with and without Guidance Mode. 77

5.1 ACE-M overview. 80

5.2 The remote memory subsystem. 85

5.3 Remote Memory - Transition between different modes. 85

5.4 Performance degradation in log scale - Intel LINPACK. 87

5.5 Performance degradation in log scale - NASPB-IS. 87

5.6 Time to complete vs available bandwidth. 89

5.7 MSE comparison for Chaos Theory vs Markov Chains predictions. 91

5.8 Cumulative distribution of workload memory size. 93

5.9 Base case. 94

5.10 Performance Degradation Factor comparison for various workload sizes. . . . 95

5.11 Average response time comparison. 96

5.12 Google trace - PMs with memory overload over time. 97

5.13 Cumulative distribution of performance with and without prediction. 99

5.14 Cumulative distribution of response times with and without prediction. 99

CHAPTER 1

INTRODUCTION

Cloud Computing is increasingly becoming an alternative to computing these days, offering multiple advantages over traditional models of computing. From simple remote data storage, to maintaining traffic intensive servers, clouds prove advantageous at a variety of different levels of deployment and use. In general cloud computing, by definition, delivers some services remotely over the Internet. Services such as Amazon Web Services, Microsoft Azure, Google Cloud Platform, etc. provide various options that are designed to satisfy computing needs for organizations with large infrastructures to individuals looking to host a website or store their data.

Cloud Computing can often be classified into three categories of services - IaaS, SaaS, and PaaS according to [1], [2], [3], and [4]. Infrastructure-as-a-Service (IaaS) is designed to offer computing infrastructure, based on which organizations are offloading their infrastructure to the cloud. The benefits of IaaS include less maintenance overheads, and more importantly the ability to dynamically increase or decrease infrastructure (and correspondingly costs) according to needs. For individual users and developers, clouds provide Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS).

SaaS uses the web to deliver applications, thus eliminating the need for individuals to install and run applications on individual computers. Popular SaaS application types include email and collaboration, customer relationship management, and healthcare-related applications.

PaaS offers developers a framework to build upon and develop or customize applications. PaaS eases the development, testing, and deployment of applications, and makes it cost-effective. With this technology, third-party providers can manage O.S.'es, virtualization, servers, storage, networking, and the PaaS software itself. However,

developers still manage and control the applications. For enterprises and other organizations, one example of PaaS could be a private cloud for .NET and Java.

Though all the described ideas can be applied to PaaS and SaaS as well, this thesis mainly focuses on enabling cloud providers to build their infrastructure for IaaS. The term 'user' refers to organizations which are clients of the cloud providers.

In this chapter some underlying technologies and features of cloud computing as well as their benefits, challenges, and current direction are discussed.

1.1 Internals of Cloud Computing

The basis of cloud computing is virtualization technology and the resulting Virtual Machines (VM). Virtualization allows the existence of multiple independent VMs, sharing the same resources (CPU, Memory, Network, Storage) on one Physical Machine (PM). For the end-user, the VM, from outside the data-center's network appears as an individual, stand-alone PM. Where, in the cluster, the VM resides, and how many other VMs share its host PM, is not apparent to the user.

These days cloud providers own data centers that service end-users all-round the globe. A typical data center has thousands of PMs, each of which runs multiple VMs [5]. Typically users are provided resources in the form of individual VMs.

Virtualization works with the help of an added layer in the PM called the Hypervisor. This hypervisor resides on top of the PM's Operating System and is responsible for creating, maintaining and destroying VMs on that PM. It performs a variety of duties to ensure that the requirements of the Guest O.S. in the VM in terms of memory, CPU, networking, and I/O are met. In most cases it works in a way to keep this process transparent from the Guest O.S., which behaves as if it is on its own dedicated PM. Figure 1.1 illustrates this structure.

Several hypervisors like KVM, Xen, VMWare, and VirtualBox exist today. Each has its own advantages and disadvantages. For example, KVM (Open Source) has been

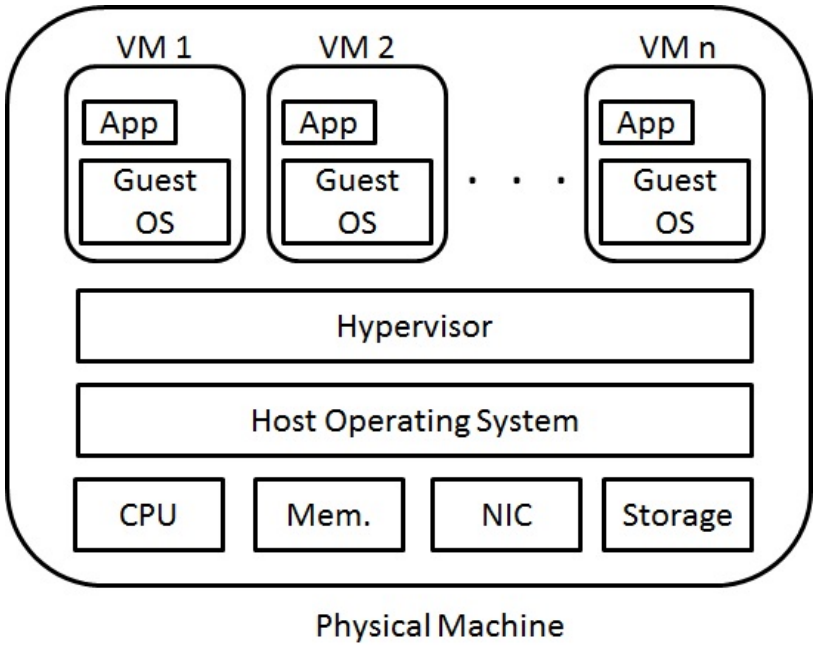


Figure 1.1 Virtualization.

included in the Linux Kernel which means it runs out-of-the-box for Linux based systems. Xen utilizes a method called para-virtualization [6], where the Guest O.S. closely works with the Host O.S. The close relationship with the Host O.S. helps in optimizing various VM features since the Guest O.S. can now work intelligently to improve virtualization from within the VM. Currently, Xen and KVM are two of the most popular hypervisors.

Apart from the basic feature of virtualization, which is multiple VM residents on a single PM, certain other benefits become available due to the fact that VMs are technically just processes on the PM. For example, one such feature is migration, where VMs can be moved around from one PM to another. Migration is further enhanced by using live migration where the VMs are moved around while they are running.

1.2 Benefits of Cloud Computing

Cloud computing offers several benefits to the users as discussed by [4], [7], [8], [9]. Some of these benefits are actually features available to the cloud service providers that affect the users.

1.2.1 Hardware Costs

The biggest benefit to organizations moving their infrastructure to the cloud, is the savings in hardware costs. Organizations no longer have to buy expensive servers and keep updating them every time a newer version of the hardware is available. Also, the structure of the cloud allows cloud providers to offer services at a low cost.

1.2.2 Maintenance Costs

While maintenance costs are not completely eliminated, they are greatly reduced. Moving to the cloud means not having to troubleshoot hardware failures and load balancing. System administrators only have to ensure that the installed server software works as required.

1.2.3 Availability

Typically, data-center and cloud owners maintain large amounts of redundancy and backup of the clients' data. This, along with other tools and techniques allows them to provide high guarantees of availability and SLO. Again, the task of maintaining backups and ensuring up-time of the servers is offloaded to the cloud owners.

1.2.4 Dynamic Resources

Since clouds rely on the principal of VMs, which are relatively easy to create, destroy, and recreate, infrastructures on the cloud are highly dynamic. In other words, organizations do not have to own a fixed number of resources any more. They can increase or decrease the number of servers (correspondingly the VMs) that they lease based on their resource requirements.

1.2.5 Scalability

Cloud offloading typically enables organizations to procure a large number of servers, which would be impractical and cost-inefficient otherwise. This ability to lease large

amounts of resources along with the aforementioned dynamic nature of these resources, means that infrastructures in the cloud can scale suitably as the needs of the organization increase.

1.3 Challenges in Cloud Computing

Despite all the benefits of cloud computing certain challenges still remain in their implementation that might deter blanket use ([10], [11], [8], [9]). Some of these challenges are discussed in this section.

1.3.1 Resource Management/ Utilization

From the point of view of data-center owners, resource management is an important issue. Since multiple VMs contend for limited resources in the cluster, it becomes difficult to balance QoS guarantees and minimum resource wastage. Consider that while servers have a maximum resource requirement, they do not need all the resources at all times. This leads to several physical machines in the data-center not being completely utilized, while others work at full load. In this scenario, it is important to move the VMs around the cluster and manage resources so that free resources can be used for other purposes. However, it is not feasible to simply reassign currently free resources, since existing latent server requirements might suddenly increase in the near future. Being able to manage these allocated, but free, resources would be a substantial benefit to the data-center owners/ operators. This improved usage would directly translate into higher capacities in the data-center and thus lower costs for the end-users.

1.3.2 Fault Tolerance/ Reliability

Even though data-centers offer high SLOs, it is not trivial to maintain these. For some organizations, even minor disruption of services could lead to major losses. Services to end-users could be disrupted for a variety of reasons, including the host physical machine

crashing, the VM crashing, resource overload in the cluster, etc. Various steps have to be taken to ensure minimum damage during such situations. Even then, it is not realistic to completely eliminate these potential mishaps.

1.3.3 Security

Since by definition, clouds entail organizations moving their infrastructure outside of their physical space, security is of paramount concern. The fact that VMs share space with other, potentially malicious VMs makes matters worse. A number of security holes and side channel communications have been shown possible that can indirectly compromise a VM and make it vulnerable.

1.4 New Avenues in Cloud Computing

Apart from the challenges that cloud computing currently faces, there are several other avenues that are being explored. These areas of research, while not strictly required for the basic working of clouds, can introduce a variety of new possibilities and features.

1.4.1 Geographical Placement

Cloud providers such as Amazon divide their data-centers into geographically separate clusters. The placement of a VM in one of these clusters can be chosen based on the proximity of the end-user. However, currently these decisions are static and can be made only at the time of creation of the VM. For applications like local web servers this suffices, though in situations like game servers or personal VMs, this falls short. For example, game servers are joined by gamers all across the globe to form a large league. The VM hosting such servers should automatically relocate itself to a physical machine that is geographically beneficial to the maximum number of players. In case of a personal VM, the VM should relocate itself based on the current geographical location of the user remotely logging in. Literature such as [12] and [13] discuss this idea in depth.

1.4.2 Federated Clouds

The various different hypervisors offer a variety of choices to the cloud providers. While in most cases this is advantageous, it has drawbacks. As mentioned before, different hypervisors have different features and methods of handling VM requirements. This difference in hypervisors indicates that it is currently not possible to simply move a VM from one hypervisor and resume it as is on another hypervisor. The lack of interoperability, in turn makes the clouds incompatible with each other. For example live migration of a VM from Amazon's EC2 to Microsoft's Azure is not possible. Initial selection of a cloud provider may lead to limited or no choice for the users, even if their requirements change in the future. Federated clouds [13] attempt to address this interoperability problem by creating additional layers on top of the hypervisors. These additional layers can work towards unifying the various underlying technologies and make it possible to seamlessly migrate a VM from a cloud provider using a certain hypervisor to another provider using a completely different hypervisor. In general federated clouds could lead to lower pricing, better services and more options for the user [14].

1.4.3 Energy Management

As mentioned, data-centers maintain thousands of physical machines to host the VMs that form the cloud. A big concern for data-center owners is the amount of energy spent in keeping the required physical machines up and running [11]. Expenses in energy are directly reflected in the costs of resources for the users. Efficient energy management will not only ensure reduction in user costs, but will also result in environment friendly infrastructures.

1.4.4 Automated Elasticity

Currently, at the time of VM creation the maximum allowable resources to that VM have to be specified. Once the VM is created however, it is generally not possible to change these

maximum limits while the VM is running. Typically, VM costs depend on the maximum resources specified for the VM. Enabling VMs to increase or decrease the maximum resources on demand can be cost-effective for both the cloud owners and the users.

1.4.5 Parallel Computing/ Big Data

Big Data and parallel computations are two applications that can take advantage of the cloud computing design as shown by [15]. The ability to run multiple VMs is conducive to high task-level parallel computing applications like MapReduce. However, using VMs is different than using actual physical nodes. Thus, substantial research ([16], [17], [18]) is underway to efficiently port these applications to cloud computing.

1.5 Underlying Technologies

Since virtualization is at the core of cloud computing, innovations in virtualization technologies enhance the benefits of clouds, mitigate their challenges and enable exploration of the avenues mentioned before. Some of these enabling technologies and the affected features of cloud computing are discussed in this section. This thesis works towards the innovation of four of these technologies - load prediction, consolidation, live migration, and resource sharing.

1.5.1 VM Load Prediction

Clouds are extensively used to host a variety of servers. These servers running on VMs could be Web servers or internal servers for organizations. Accurate prediction of the constantly changing workloads on these servers is important. These predictions are valuable in ensuring availability, scalability, resource management, fault tolerance, energy management, and automatic elasticity in the cloud.

1.5.2 VM Consolidation

Since these servers exist as VMs, it becomes easy to move them within the cluster and achieve a number of features mentioned above. This ease of migration directly affects energy management.

1.5.3 VM Live Migration

Live migration enables the VM to be moved from one physical machine to another without stopping or pausing the VM in any way. While this is a useful technology, it is still not widely used in data-centers due to various issues such as heavy network usage. Most of the above mentioned cloud advantages, challenges, and avenues will benefit from efficient live migration.

1.5.4 VM Resource Sharing

Multiple VMs contending for limited number of resources on a single PM can lead to problems such as starvation, wastage of resources, etc. Thus it is imperative to ensure that all VMs on a PM use available resources as conservatively and efficiently as they can. On the other hand, the large amount of resources available in the cluster as a whole could be utilized to enable VMs to perform more resource intensive tasks if required. This distribution of resources to the VMs is the responsibility of the hypervisor and the kernel. New, efficient approaches in this area will lead to better resource utilization, higher reliability, resource elasticity, efficient big data computations, etc. in the cloud.

1.5.5 Nested Virtualization

Nested virtualization refers to virtualization within a VM. Realization of nested virtualization requires the ability to run a hypervisor inside a VM. Due to the difference in physical hardware and the virtual hardware offered by VMs, nested virtualization is in a nascent stage. As expected, there is some degradation in performance when running a

nested VM as opposed to a VM on a physical machine. Hypervisors like KVM support limited nested virtualization with relatively low overhead ([19]). However, currently cloud providers do not offer this feature to their users. An advantage of nested virtualization in a cloud scenario would be the ability of users to test and implement virtualization related features from within their VM in the cloud. Nested virtualization can provide users with complete but isolated hypervisor control, without violating the principles of clouds.

1.6 Proposed Innovations

As has been discussed, virtualization technologies are what directly influence cloud computing features. This thesis tackles the following technologies.

1.6.1 Load Prediction and Consolidation for VMs

An important consideration for data-center owners in the cloud computing scenario is the optimal usage of the resources (CPU cycles, Memory, Block I/O and Network Bandwidth) of the PM that make up the cloud or 'machine-farms'. At any given time, the machines should not be overloaded to ensure certain QoS requirements are met and at the same time a minimum number of machines should be running to conserve energy. The loads on individual VMs residing on these machines is not absolutely random. Certain patterns can be found that can help the data center owners arrange the VMs on the PMs such that both of the above competing goals are satisfied (minimum number of machines running without any being overloaded). The current prediction mechanisms are often inaccurate in finding such patterns across the variety of real world loads and often times computation intensive. Most of them also ignore certain characteristics of the loads, which further reduces the accuracy.

The framework that is proposed tries to intelligently predict the behavior of the cluster based on its history and then accordingly distributes VMs in the cluster to free up PMs. These machines can then be re-purposed to accommodate more VMs or turned

off to save energy. Analysis of real world data-center loads shows that they follow a Chaotic time series. Central to the framework are concepts of Chaos Theory with some optimizations that make the framework indifferent to the type of loads and inherent cycles in them. This framework with the modeled Chaos Theory is tested on a testbed cluster and its performance is analyzed. It is demonstrated that the framework performs better than another often used time series method in freeing up PMs for a variety of real-world loads. Additionally, the optimizations added make the framework comparatively faster.

1.6.2 Live Migration of VMs

Live migration of VMs is an important part of cloud and data-center management. Both pre-copy and post-copy algorithms for live migration allow load balancing, physical machine maintenance, and consolidation. Constant endeavors are made to make it as fast and seamless as possible. Maintaining low network usage and fast times during live migration are necessary goals to ensure that the performance of the migrating VM as well as that of other VMs in the cluster is not degraded. Current live migration protocols incur a lot of wasted network bandwidth during migration by transferring redundant memory.

The proposed proactive memory sharing and informing protocol utilizes the common memory pages between VMs to achieve the goals mentioned. A method is proposed to identify and distribute important shared pages in a cluster, to reduce network usage and enable fast times during migration for both pre-copy and post-copy live migration. The protocol is also implemented for both flavors of live migration on a test cluster and the time and bandwidth savings achieved as opposed to current live migration techniques are demonstrated.

1.6.3 Memory Pooling of VMs

While it is possible to create a VM with more memory than physically available (over-committing), just like regular processes, if it demands the extra memory, memory swap

takes place to accommodate this request. Depending on the type of storage used, this swap-in swap-out may not be fast enough and lead to degradation of the VM's performance. On the other hand, network speeds these days are often faster than disk accesses. Also it is often the case that while one VM is memory intensive, other VMs on another physical machine may not require all the memory available to them. These two behaviors can be utilized to extend the physical memory available to a VM outside its host physical machine. Currently, methods to achieve similar goals only exist with special hardware such as RDMA.

The proposed protocol is a method to use all the physical memory in the cluster as a memory pool that can be transparently accessed by all VMs in the cluster without the need for any special hardware. The amount of physical memory available to a VM could then be dynamically adjusted based on a number of factors, like VM memory requirements or even in a pay-as-you-need scenario.

1.7 Summary

This chapter has presented an overview of cloud computing in general, and virtualization in particular. Various virtualization technologies have been identified, which enable various features in the cloud. In this thesis approaches and protocols for these technologies are presented that open up new possibilities in cloud computing by minimizing degradation of VM performance for various attributes. Specifically, innovations are described for load predictions, consolidation, live migration, and memory sharing of VMs.

The rest of the thesis is divided as follows. Chapter 2 elucidates the technologies that the thesis contributes to, as well as other research currently being done on them. Chapters 3 to 5 provide detailed descriptions of the contributions. Chapter 3 discusses a unique framework for load prediction and consolidation of VMs. Chapter 4 introduces a new network-efficient live migration approach. Chapter 5 proposes and demonstrates a novel cluster-wide memory sharing protocol that allows VMs to access all the memory available in a cluster. Finally, Chapter 6 concludes the thesis and suggests future work.

CHAPTER 2

BACKGROUND

The previous chapter highlighted some possible virtualization technologies that directly influence cloud computing features. This chapter discusses three of these areas which this thesis contributes to. Also, other related current research endeavors are described as required.

The virtualization technologies discussed are:

- Load Prediction and Consolidation of VMs
- Live Migration of VMs
- Memory Pooling for VMs

In the chapters that follow, detailed descriptions are given for each of these contributions.

2.1 Load Prediction and Consolidation of VMs

Optimal distribution of the VMs on the data-center cluster is an important problem. The main resources that a VM uses on a PM are CPU cycles, Memory, Block devices and Network. A lot of research has been done in the area of ensuring that VMs are distributed such that in all probability, they do not cause the host PM to overload (in terms of CPU, Memory, Block devices or Network). Even if they do, intelligent software exist to rearrange them and balance the load [20]. This ensures that a certain QoS is maintained for the end-user.

On the other end of this discussion, is an issue opposite, but closely related to the problem of load balancing. It is the concern that we should not be wasting energy and resources by allowing a distribution of VMs that does not cause overload, but uses more

PMs than required to smoothly run the VMs in the cluster. A large body of work [21], [22] reinforces this belief.

VMs in a data center are used for a wide variety of applications and the users are spread throughout the world. The loads on these VMs in terms of usage of CPU, memory, Network, and Block I/O are not absolutely random and correspond to various real world situations like holiday seasons, weekends, workdays, etc. This nature of VMs is well known and it is obvious that effort has been made to utilize this knowledge.

However, simply the knowledge that these patterns exist does not entail that the patterns across all the VMs is similar. Two factors affect the pattern observed on a VM - the type of service that VM provides and the geographic location of the users using that VM. As an example, a VM hosting a shopping website has heavier loads during holiday seasons and lighter loads otherwise. Contrary to this, a VM hosting a database for an office should have higher loads during working days and lesser loads during holidays. Again, this condition changes for different geographic locations based on the off-peak and peak times at that place.

The idea, then, is to look at the past history of the VM and observe if any pattern exists which can be applied to the future.

From the point of view of consolidation of servers, this is an important problem. Simple averages of loads is sub-optimal, since it ignores the fact that two VMs (with the same average load) may be peaking at different times, thus allowing them to co-exist on the same PM without any overload.

Note that even though the sum of their individual maximum loads may overload a PM, the fact that the maximum loads are probably going to occur at different times allows us to collate them on to one PM.

Therefore, it is necessary to try and find patterns in the load of every VM and try to best match VMs that can co-exist peacefully for a long time on the same PM. The major roadblock in this area is the prediction of future behavior of the individual VMs. Other

research deals with a similar idea as above - predicting patterns in the loads of the VMs. Some of these methods are computationally complex. Others are not fine grained and work under the assumption that there is an overall periodic cycle in the load. They do not consider the fact that a VM may have multiple cycles. For example, a daily cycle of high and low usage and then a weekly cycle with lows on weekends.

Initial analysis shows that various real world loads demonstrate Chaotic behavior making them ideal candidates for prediction using Chaos Theory. Chaos Theory has been used to forecast behavior in a variety of situations including power usage [23], dynamic texture synthesis [24], oil prices [25], medical procedures [26], etc. The framework discussed shortly applies similar concepts to VM loads to mitigate the above problems. It is fast, energy-saving and uses concepts from Chaos Theory to allow fine grain predictions, and achieve consolidation at every level of time (hourly, daily, weekly or monthly), with little overhead.

The end result of this is that some PMs are freed up by consolidating VMs (using live migration), and some guarantee or probability is provided of the machines remaining free for a certain period of time. Based on its predictions, the PMs are also automatically reused when chances of overloading increase. Further, the framework makes no assumptions about the kind of applications running on the VMs and requires no modification to the guest operating system, making it application and O.S. independent.

This chapter not only demonstrates the workings of the framework but also provides experimental results to show its efficacy. It shows how the framework manages to free up PMs based on CPU, Memory, Network, and Block I/O loads of the VMs. The framework's efficiency is also compared to a version of Fourier Transform based prediction mechanism, as well as optimal consolidation. Currently the framework works with KVM on Linux machines without any modification to KVM, Libvirt or QEMU. However, conceptually it is hypervisor independent. In the future it is planned to port the framework to other operating systems and also to other Virtual Machine Hypervisors like Xen.

2.1.1 Related Work

The seasonal nature of VMs is well known, and it is obvious that effort has been made to utilize this knowledge.

Considerable work has been done to achieve consolidation of VMs in data centers to reduce power consumption and conserve energy, for example [27], [28]. Consolidation can be performed in two ways, either at the beginning of a VM's life (that is placing it correctly to begin with) or during a VM's life (as the VM's needs change). Various ideas have been suggested to decide the correct placement of VMs in a data center to achieve this. Most of them focus on the four main resources - CPU, Memory, Block I/O and Networking. Some of the methods like [29] need to be aware of the kinds of applications running on the VM, while others like [30] consider parameters like power and CPU frequency to make their decision. Still other ideas like [31] propose modifying underlying operating systems.

This framework attempts to solve the consolidation problem based on the four major resources, without modifications to the guest, and using existing technology in the hypervisor. Its approach can thus be considered to have the following steps:

- Make a prediction for VM behavior
- Use algorithms similar to bin packing to figure a new VM placement configuration
- Migrate VMs to achieve the new configuration
- Re-purpose vacant PMs to accommodate more VMs or turn them off if so desired

Since prediction of VM load is an integral part of the framework, a few VM prediction methods currently being worked on are described. Some of the research mentioned does not directly deal with consolidation of VMs, but their prediction methods are comparable, and hence have been included.

A plethora of techniques like Bayesian Filters, Artificial Neural Networks, SVMs, Fuzzy Logic, PCA, Markov Chains, etc. have been used to aid VM load predictions. Based on [32], they can be classified as:

- Static, threshold-based policies
- Reinforcement Learning
- Queuing theory
- Control theory
- Time-series analysis

Some methods also use a combination of these techniques to figure out a near-optimal method on a case-by-case basis [33]. Describing all the methods currently being researched is out of scope of this thesis, and hence a few that are akin to the approach taken in this chapter are mentioned.

Naive methods like average loads, standard deviations, or threshold-based, though quick and usable, clearly fall short in a number of real world situations. Reinforcement Learning methods are feedback-based systems working on reward and penalty policies. These method have been used in research to adaptively adjust resource allocations based on some performance feedback [34], [35], [36]. Similarly, some works have applied control theory [37], [38], [39], [40]. However, as pointed out by [32], these methods require time to converge to their optimal solutions and have large state-spaces to compute.

Likewise, model-based approaches use Queuing Theory to build models that allow the system to predict the impact of different resource allocation policies on the application performance [41], [42], [43], [44], [45]. The drawback of these methods is that most often, the models need some adjustments beforehand. This implies certain pre-defined knowledge of the application which is difficult to obtain in a data center environment.

Most of these methods are complex and in some cases require some pre-defined idea of the kind of loads on the VM. As opposed to this, the framework being proposed makes no assumptions about the inherent cycles, or behavior of the loads.

Time Series Analysis based method use signal analysis tools like Fourier Transforms, Moving Average, Auto Regression, String Matching, etc. on historical resource loads for load predictions [33]. The proposed framework is most closely related to these methods.

Methods reported in [46] and [47] use autocorrelation to figure out repeating patterns in the VM loads. Gong et al. have done an extensive comparison of Time Series Analysis based techniques for resource scaling [48]. They propose using FFT to identify repeating patterns in resource usage (CPU, memory, I/O and network), and compare it with auto-correlation, auto-regression and histogram. They have demonstrated that auto-regression is computationally intensive and their FFT-based algorithm has lesser overhead. Moreover, they demonstrated that histogram-based methods such as [49] are not as efficient.

The problem with using FFT-like tools is that even though we easily get a general idea of the most dominant cycle in the load of a VM, the methods are less certain while indicating several other cycles that could exist within the load. For example, even after observing a substantial number of weekly cycles, the major cycle calculated is the daily cycle, i.e. the VM has an off-peak every night and a peak throughout the rest of the day. However, this view ignores the fact that the VM has a light load during weekends compared to the rest of the week. Figure 2.1 demonstrates this problem. Missing the weekend cycles (and their periods of low loads) could mean lower level of consolidation where more energy saving could have been potentially possible. Additionally, for loads that do not have an obvious cycle, this method falls short.

As seen by a sample of the Google Data Trace [50] in Figure 2.2, there are two cycles. One cycle is a daily cycle - relative to the daytime, the nighttime load is less. However, the second cycle is weekly. Because of the weekly cycle there are changes in the maximum loads during each day of the week. For e.g., Day 1 is more load intensive than Day 2 and so on.

The figure shows the frequency spectrum calculated by Fourier for this load. As can be seen, it easily identifies periods of 24 hours. The other periods it identifies are

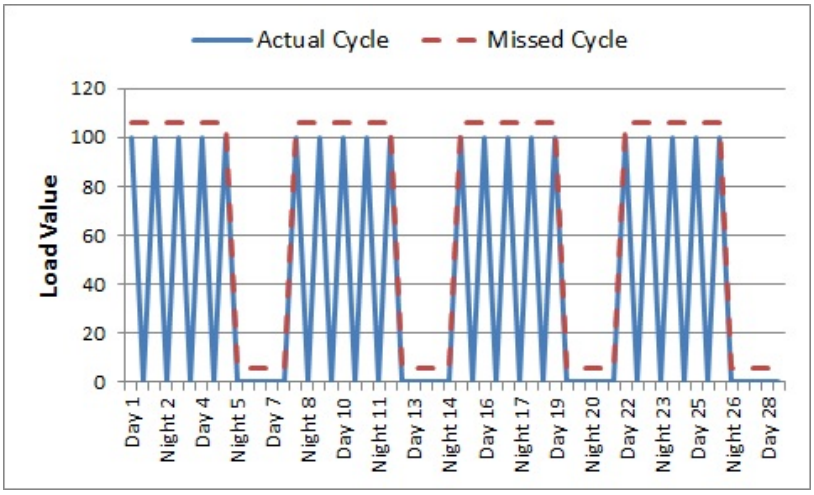


Figure 2.1 Missing secondary cycles in load patterns.

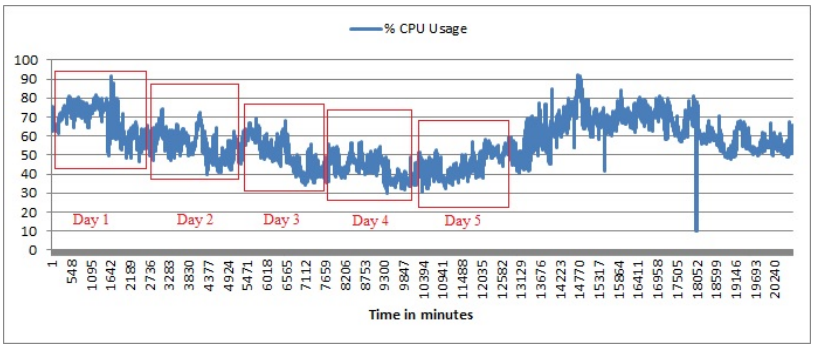


Figure 2.2 Sample Google data trace.

too insignificant in amplitude to be of any use. Thus using FFT, it becomes difficult to predict future loads without the knowledge that the values of these loads is different for each day. Clearly, better prediction mechanisms are needed that are indifferent to such load characteristics while maintaining the advantages of time-series analysis.

The work in [51] extends [48] by implementing a prediction mechanism using Wavelets and Markov Chains. While this approach mitigates the problems with acyclic loads to some extent, it suffers from larger overheads. Also, since the prediction of every step in the future is dependent on the previous value, errors tend to add up when multiple points in the future are predicted at once. Besides, future predictions are made using a fixed-size subset of the historical data. This fixed history size may not hold well for arbitrarily large historical data, where larger patterns exist that may be missed.

Studies with Google Cluster Data, NASA [52] and the World Cup [53] websites' data traces show that the data is in fact chaotic. This makes these loads good candidates for predictions using Chaos Theory. The proposed framework works effectively for multiple levels of cyclic loads as well as non-cyclic loads, as long as the loads are chaotic. It quickly adapts to changes in load patterns and within a short time recognizes new patterns (example weekly patterns).

Moreover, since most other similar Time Series Analysis methods need to go through the entire history of a VM to make a prediction, it becomes computationally costly as the VMs run for a longer time. This framework on the other hand, avoids going through the whole data set, as shall be seen in later sections.

2.2 Live Migration of VMs

Live Migration is a feature of VMs that has shown great promise in recent times. Through live migration, data-center owners can seamlessly move VMs from one physical machine to another with little to no degradation of performance. This is useful in a variety of scenarios

like maintenance of physical machines, load balancing, consolidation for energy savings, etc.

Substantial research is being constantly done to improve live migration. In the real world, live migration can only be extensively adopted if it meets certain Service Level Objectives (SLO)/ Service Level Agreements (SLA) for stability, performance degradation, and speed.

The basic idea of live migration is straightforward and has been extensively discussed in the literature. The two participating physical machines are first identified as source (where the VM currently resides) and destination (where the VM needs to be migrated to). The gist of live migration is that a copy of the VM is initiated at the destination and then memory pages corresponding to the VM are transferred from the source. Once all memory pages have been transferred, CPU states are transferred to the destination. The VM is then shut down at the source and resumed at the destination. Usually, the storage is a shared file system in the cluster.

Currently, two types of live migration algorithms are pre-dominantly in use: pre-copy and post-copy. These algorithms only differ in the process they use to transfer memory pages from the source to the destination. In pre-copy (Figure 2.3), while the VM is still running at the source, its memory pages are iteratively transferred to the destination. After every iteration, pages that have been modified or dirtied (since the VM continued running during this time) are re-transferred. This continues until either a pre-defined number of iterations is performed, or the number of dirtied pages does not change from one iteration to the next. At this point, the VM is paused and all the remaining dirtied pages are transferred to the destination. Finally, the VM is resumed at the destination and it continues execution.

In post-copy (Figure 2.4), the VM is initially paused at the source and immediately started at the destination. Note that no memory pages have been transferred so far. At the destination, every time a page is required by the VM it is requested and received from the source, if it has not already been received. Over time, when all the memory pages have

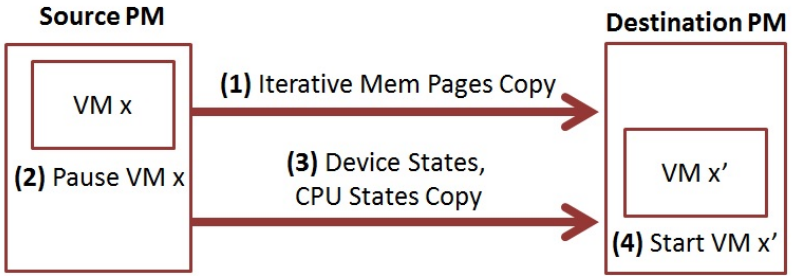


Figure 2.3 Pre-Copy live migration

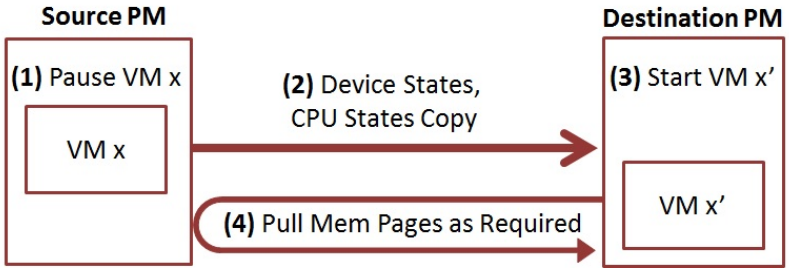


Figure 2.4 Post-Copy live migration.

been requested and received, the source is free to terminate the VM and reclaim all its memory.

Pre-copy migration has the disadvantage of comparatively heavier network usage during migration since the iterative process results in multiple transfers of the same page. This could be a problem if the migration was being performed during times of heavy network usage.

On the other hand, while post-copy puts less burden on the network by sending every page only once, it results in longer times before the whole migration process is complete. Since pages are pulled on demand by the destination, in theory it is possible that migration never completes if some pages are not demanded at all. This is a serious issue if the reason for migration was overloads at the source or if the source needed to be quickly shut down for maintenance. Practically, using a background copy process in post-copy alleviates this problem. However, this diminishes post-copy's advantage of using the network only when required.

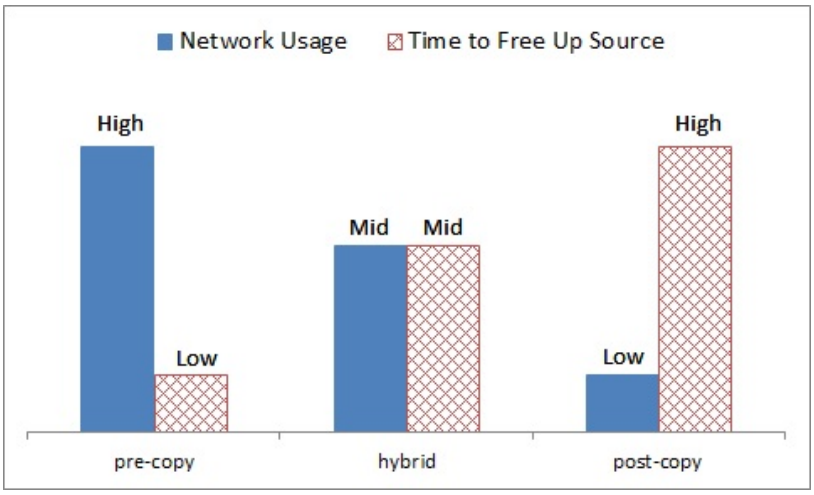


Figure 2.5 Pre-Copy/ Post-Copy trade-off.

In general the choice between pre-copy and post-copy includes trade-offs that have to be considered as shown by Figure 2.5.

In most cases, live migration is performed in times of critical need, like when a physical machine is in urgent need of maintenance, or to reduce load in a physical machine that is overloading. Two major problems arise in situations like these. In general, there is little control regarding the time at which such migrations need to be performed. At that time network usage may already be high. It is therefore imperative to ensure that due to migration, the network is not further congested, adversely affecting performance of other VMs in the cluster. At the same time, the migration time needs to be as fast as possible to quickly free up the source physical machine and hence prevent any performance degradation of the VMs. Thus, *low network usage* and *fast migration times* are two important goals during live migration.

The proposed protocol is designed to attain these two goals for both pre-copy and post-copy migration. It uses the existence of common pages between VMs as well as the fact that periods of low resource usage exist during a VM's life to achieve this. Due to similar applications or operating systems running on VMs, a substantial number of memory pages across a number of the VMs have the same contents. The approach firstly keeps

track of memory pages with same contents that exist between VMs. It then tries to identify important memory pages in the cluster and during low network usage times it distributes these pages in the cluster.

At the time of migration, the protocol ensures that pages that already exist at the destination are not transferred. Moreover, if post-copy migration is used, apart from not transferring pages existent at the destination, the protocol chooses a secondary source that shares a substantial number of memory pages with the primary source. Once the primary source has transferred pages that only it has, it can safely handover responsibility to the secondary source and is free to do other work. Meanwhile, the destination can pull required pages at leisure.

Thus, the approach clearly reduces the use of the network during pre-copy migration and as shall be seen, achieves faster migration times. For post-copy migration, it not only reduces network load during migration, but also allows the source to be relieved of its duties faster. These savings in time and network bandwidth during migration can be crucial in scarce resources at the time of migration. Further, this protocol can also provide suggestions to the source for possible destination candidates that would help achieve the two goals.

2.2.1 Related Work

It is well known that common memory pages exist between VMs. Much work has been done in identifying and using this commonality. VMWare [54] uses this knowledge to make VMs residing on the same physical machine share their common pages. This has been demonstrated to reduce the memory footprint of the VMs in the physical machine by upto 33%. Other research in [55], [56] have discussed the impact of different characteristics such as similar Guest O.S.'es or similar applications on the similarity of pages between VMs.

Effort has also been made to take advantage of this sharing to rearrange VMs in the cluster in order to achieve better local page sharing. The method described in [57] chooses a suitable host for VM placement based on the amount of pages they share based on which live migration is performed.

There has been some research done to avoid transferring redundant pages to improve live migration. Most of these deal with pre-copy migration. Approaches such as [58] consider the self duplicate pages of a VM and avoid transferring these pages multiple times during migration. Most of the other research using this concept deals with group migration or migrating clusters of VMs. This implies that when a number of VMs have to be migrated simultaneously, pages that have already been transferred over the network once are not re-transferred. Approaches such as [59], and [60] utilize this idea. In [61] a similar concept is applied to live migration between racks in a data-center. The central assumption in this is that the link within a rack is faster than the link between racks. Thus, every unique page is transferred only once between racks and then multiplied as required within the rack. Similarly, [62] uses the idea for WANs, since obviously the link between two different LANs is slower than the links inside a LAN. Finally, [63] uses similar features to migrate VMs from one host to multiple hosts.

As opposed to all these approaches, the proposed protocol enhances migration for an individual VM from a source to a destination, although it can also be used for group migrations. Further, it not only considers self duplicate memory pages but also makes use of inter-VM duplicity of pages. While the system performs best in cases where the network speeds are slow (WAN) and disk speeds are fast (Solid State Drives), as opposed to the other methods, it makes no assumption of the existence of two different link speeds. It works within a cluster with a single type of link and as shown by the experiments, improves further in networks with different types of links.

Additionally, the protocol is uniquely applicable to post-copy migration.

2.3 Memory Pooling for VMs

VMs are used for various memory intensive applications. Current big data applications require large amounts of memory that may not be available on a single host Physical Machine (PM). These applications can be executed using a number of different solutions. One solution is using parallel computing. However, not all problems and applications can be suitably parallelized.

Another approach could be to migrate VMs with high memory requirements to PMs that have sufficient physical memory available [64]. This approach fails when the memory requirements are high enough such that no single PM in the cluster has enough physical memory on its own.

The other feasible solution to meet these requirements is to overcommit the VM. In other words, the VM is created with more memory than physically available on the host PM. Since VMs are typically processes on PMs, this is possible. In such cases, normally the host kernel's memory management implements swapping of memory pages to the swap device (generally HDD) to ensure the VM's memory requirements are met when needed.

An alternative to disk swapping is to make sufficient free memory from the cluster available to the VM remotely which is known as remote memory. Memory pages reside in the physical memory of a remote provider PM in the cluster that has enough memory available. These pages are then fetched on-demand, as required by the requesting VM. This remote memory approach is possible since not all of the VMs in a datacenter may be memory intensive.

Both disk swap and remote memory have their advantages in some situations but are prone to underperform in others. The drawback of disk swapping is that it is generally very slow. Since remote pages exist in physical memory of PMs, and network speeds are usually orders of magnitude faster than block access speeds, remote memory outperforms disk swap. Conversely, using Solid State Drives (SSD) could achieve better performance in disk swap as compared to software-based remote memory over a slow network. Hardware-based

RDMA generally performs better than disk swap, but involves the installation of special hardware in the cluster. Similarly, during times of high block I/O usage, disk swap may underperform even compared to a slower network. On the other hand, remote memory with a fast network may underperform compared to disk swap during times of high network usage.

Since remote memory depends on availability of physical memory on a remote provider PM, it is important to select a suitable provider. If the provider PM runs out of physical memory while provisioning a VM, it could fall back to disk swapping. This compounded by the remote fetching overheads could degrade VMs to the point of failing their computations. Moreover, advantages provided by remote memory systems come at the cost of network bandwidth overheads. While current studies in this area assume sufficient network bandwidth available to the VM, this may not be the case in real world scenarios, even if the theoretical bandwidth of the network is high. Apart from memory intensive applications, VMs also run network intensive applications like web or database servers. Besides, VMs are also extensively used as virtual clusters to run parallel jobs like Map-Reduce, which again require intensive communications over the network. Burdening the network with remote page fetching in these cases could not only degrade the requesting VM's performance but also the performance of VMs residing on the provider PM.

It is, hence, important to create a remote memory management system that takes into account these parameters and *dynamically* chooses disk swapping or remote memory as well as correct provider PMs based on the cluster's state during times of high memory usage.

While VM resource usage predictions have been successfully used for a variety of applications including resource management, load balancing, and server consolidation, current implementations of remote memory in clusters do not attempt to predict the behavior of the cluster leading to performance degradation of the VMs. Further, the existence of network intensive VMs and their degradation due to remote memory in

the cluster have not been considered. Besides, most of the existing remote memory management systems either depend on hypervisor features or require installation of expensive, special network components and do not provide for dynamic switching of the remote provider PMs.

This thesis presents the Autonomous Cluster-Wide Elastic Memory (ACE-M) framework that uses prediction mechanisms to predict the behavior of all VMs in the cluster for the Memory, Network and Block I/O resources. It uses these predictions to intelligently decide the local memory/ remote memory split as well as suitable remote provider PMs while avoiding performance degradation of all VMs involved. A new hypervisor agnostic, VM-level, software-based remote memory subsystem is also demonstrated, that mitigates some problems associated with current remote memory implementations as seen in Chapter 5.

Extensive experimental results using real world loads demonstrate first that even using only memory predictions reduces cluster-wide performance degradation by about 55%. Second, using ACE-M that considers memory and network predictions reduces cluster-wide performance degradation by about 75% for the loads.

2.3.1 Related Work

Research related to memory management for VMs typically involves scheduling and allocation policies [65] or VM workload placement in datacenters [66]. These approaches assume the availability of sufficient memory on at least one of the PMs at any given time.

In order to enable applications with memory demands more than physically available on a single PM, some proposed methods suggest over provisioning the system with resources. This includes adding a hybrid memory system [67] [68] [69], or adding dedicated blade servers that stores all data on their main memory [70]. The drawback to these approaches is the additional expense and the requirement to install new hardware.

Cluster-wide or remote memory, has been studied under various names such as Memory Aggregation, Transcendent Memory, and Memory Pooling. Broadly, this idea has been applied to individual processes, clusters as a whole, or VMs. It involves making use of idle memory located at remote nodes. Remote Memory can be categorized into two approaches - remote memory swapping [71] [72] [73], and remote memory mapping [74]. Specifically for VMs, both of these remote memory approaches have been proposed using either software based approaches, or RDMA hardware based solutions [75]. While RDMA hardware based approaches generally perform better than software based solutions, they come at the cost of expensive equipment and special network cards.

In case of VMs, remote swapping can achieve hypervisor agnostic remote memory access using network block devices (NBD) as swap devices [76]. These methods are advantageous since they require little to no modification of the Guest OS as well as the underlying host OS. The disadvantage is that they suffer from significant lack of granularity. Swap is system wide which prevents restricting its usage to certain processes or VMs.

Solutions to replace swapping have been included in the Linux kernel such as using a combination of cleancache, frontswap, and ramster with the transcendent memory patch set. RAMster is one transparent remote memory solution that allows a PM to rely on physical memory of another remote PM. It offers some advantages such as using zcache for local compression, and reducing network usage by off-line transfer of memory contents. Even with these advantages, modifications are needed to the hypervisor that renders these solutions hypervisor dependent. Additionally, reworked backend implementations are needed for any required functionality that already exists in the kernel.

Another promising approach is the use of elastic operating systems that span multiple PMs in the datacenter and hence can view memory on multiple PMs[77]. However, they require major modifications to the OS and their deployment is not trivial.

Other methods for VM remote memory access include those described by MemX [78], TMemCanal [79] and Overdriver [80]. These rely on features of a particular hypervisor (Xen) to work.

The Hecatonchire Project [75] uses the RDMA functionality of the Kernel to support both hardware and software based remote memory access. On further inspection, the project which is currently offline, appears more suited for hardware-based RDMA. It also requires modification of the host kernel as well as the hypervisor and thus, cannot be applied indiscriminately to all hypervisors. In addition, the currently available prototype used for testing, cannot dynamically switch to a new provider PM.

Literature also exists to try and intelligently allocate remote memory in order to avoid overloading the PMs. Overdriver, which is most similar to ACE-M, does not consider Network or Block I/O usage. It is useful to avoid remote swapping due to transient bursts in memory in overcommitted VMs. Also, it only considers migration of the VM or remote swap at the moment of high usage, thus is not proactive. Two other approaches similar to ACE-M are [76] and [81] which use remote swapping with thresholds to select suitable provider PMs. While [76] acknowledges that about 50% network bandwidth is used for remote memory accessing, neither considers network usage in the decision making process.

As opposed to these methods, this thesis discusses a remote memory accessing mechanism that utilizes swap with minor modification in the Linux kernel that makes it both hypervisor agnostic, and granular. The ACE-M framework utilizes VM resource usage predictions based on Chaos Theory to intelligently choose remote provider PMs. This allows VMs to perform computations requiring more memory than physically available on the host PM, while reducing degradation of the rest of the cluster. To the best of our knowledge, no framework exists that utilizes prediction mechanisms to service VM remote memory requirements.

CHAPTER 3

PREDICTION AND CONSOLIDATION OF VIRTUAL MACHINES

This chapter discusses a framework to predict VM loads in servers and consolidate them, allowing the PMs to be switched off to save energy or to be repurposed for other requirements. The central feature of the framework is Chaos Theory modeled for VM loads.

The rest of the chapter is organized as follows. Section 3.1 presents some principles of Chaos Theory from the point of view of VM loads. Section 3.2 describes the approach used to build the whole framework. Section 3.3 illustrates the system design of the framework. Section 3.4 includes the experimental setup and elucidates the efficacy of the protocol through experiments. Section 3.5 summarizes and concludes the chapter.

3.1 Chaos Theory for VM Loads

A feature of chaos is the apparent unpredictability of its future. This feature is known as the 'sensitivity dependence on initial conditions'. For example, if two initial conditions have a small difference, their difference after time will be an exponential separation. While this means that long term predictions are difficult in such systems, short term predictions based on the current state of the system can be made. For example, it might be known from history that a VM's nighttime load is relatively less to its daytime load on Fridays. In this case, while it is not possible to predict a VM's behavior weeks into the future, on a Friday morning, predictions should be possible for its load for the next 12 hours based on its behavior during Fridays in the past and its overall behavior during the current week.

A detailed discussion of Chaos Theory is beyond the scope of this chapter. In this section, some important aspects that are relevant to the problem domain and the mathematical steps to make future predictions are described. Chaos Theory has been extensively described in literature ([82], [83]).

3.1.1 Chaotic Loads

To check whether a time series is chaotic or not, one needs to calculate and analyze the maximum Lyapunov Exponent (λ) of the series. A system is called a chaotic system if its maximum Lyapunov exponent is positive. This task is much easier than the calculation of all the λs. The method described in [84] was used to calculate the maximum λ for sample loads from Google Trace Data, NASA website trace, and Worldcup website trace. The maximum λ for each is positive.

Note that while all perfectly cyclic loads (like sine waves) tend to be chaotic, not all chaotic loads are perfectly cyclic.

In case of VM loads, the patterns that they demonstrate depend on a variety of conditions, mostly predetermined at the time of creation of the VM. These conditions could include the type of service the VM offers (web/ database/ mail/ game server) as well as the geographic location and popularity of the VM. Thus, while every VM will follow certain basic local patterns like daily cycles, there will tend to be huge overall variations, that should nevertheless be consistent for a given VM.

3.1.2 Prediction using Chaos Theory

Assuming the time series of a resource load to be 'T', the major steps in prediction using Chaos Theory are as follows:

- Convert the single dimension time series T to a multi-dimensional phase-space matrix
- Use Nearest Neighbors to determine the next row in the phase-space matrix
- Convert the phase-space matrix back to time series

For the first conversion, methods described by Cao et al [85] and Fraser et al [86] were used.

Let the original time series for a resource on a particular VM be given by:

$$T = (T_1, T_2, ..., T_n) \tag{3.1}$$

Construct a phase-space matrix of the form:

$$Y = [y_1, y_2, ..., y_{n-(d-1)\tau}] \tag{3.2}$$

where y_i is a row in Y and

$$y_i = T_i, T_{i+\tau}, ..., T_{i+(d-1)\tau} \tag{3.3}$$

The terms d (embedding dimension) and τ (delay) are important for the correct working of the Chaos model. Fraser's method is used to first calculate τ and then Cao's method achieves d based on that.

At the current time, assume the last row of the phase-space matrix to be y_t. Find all y_k which are the Nearest Neighbors of y_t. The Nearest Neighbors are simply the rows which are at minimum distance from y_t. The distance between two rows is calculated by subtracting the corresponding elements and averaging the differences.

Then, make a prediction for the next row y_{t+1} using the following prediction model:

$$y_{t+1} = \sum_{j=1}^{NN(y_t)} (y_{j+1} - y_j + y_t) w_j(y_t, y_j) \tag{3.4}$$

where, NN(y_t) denotes the nearest neighbors of y_t, and w_j is simply a weight based on the distance of y_j and y_t

Now, y_{t+1} is the prediction for the next row in phase-space. Append this row to Y and convert back to time series.

The conversion back to time series is quite straightforward. From Y, simply take the first column and then the last τ rows from all the subsequent columns. If the length of the new time series is N', obviously, the last N-N' data points are the future predictions. Calculate y_{t+1}, y_{t+2} and so on until enough future predictions are made.

3.2 The Approach

The basic steps of VM consolidation are as follows:

- Collect relevant VM statistics
- Deduce patterns in the statistics
- Forecast load values for some time in the future
- Rearrange VMs based on these forecast loads using live migration

The VM statistics needed are the loads at equidistant time intervals for the four resources (Memory, CPU, IO, and Network). At any point the framework maintains a history of these loads for each VM up to the current time. It then applies concepts from Chaos Theory to deduce patterns in the loads (based on the history) and also forecast future possible loads.

In this section first the general application of concepts from Chaos Theory to the forecasting of VM loads is discussed. Subsequently, certain optimizations that were applied in order to make the framework work more efficiently are described.

3.2.1 Single VM Resource Load Forecasting

For the historical time-series maintained for each resource of each VM, Chaos Theory as described in the previous section is used to predict the load for some time units in the future. Using this logic, at any time, the framework can calculate the probable future values at a per VM, per resource level.

The following are some optimizations, that while not required, enhance the framework and make it faster, more accurate, and more scalable.

3.2.2 Hierarchical Representation

The previous subsection enabled load forecasting of a single resource on a single VM given its historical data. However, the calculations of d and τ become complex as the number of data points and the number of inherent cycles increase. Also, the computations increase considerably with a large amount of historical data. In order to reduce this complexity an optimization is added to the framework, where forecasting is done at different levels of

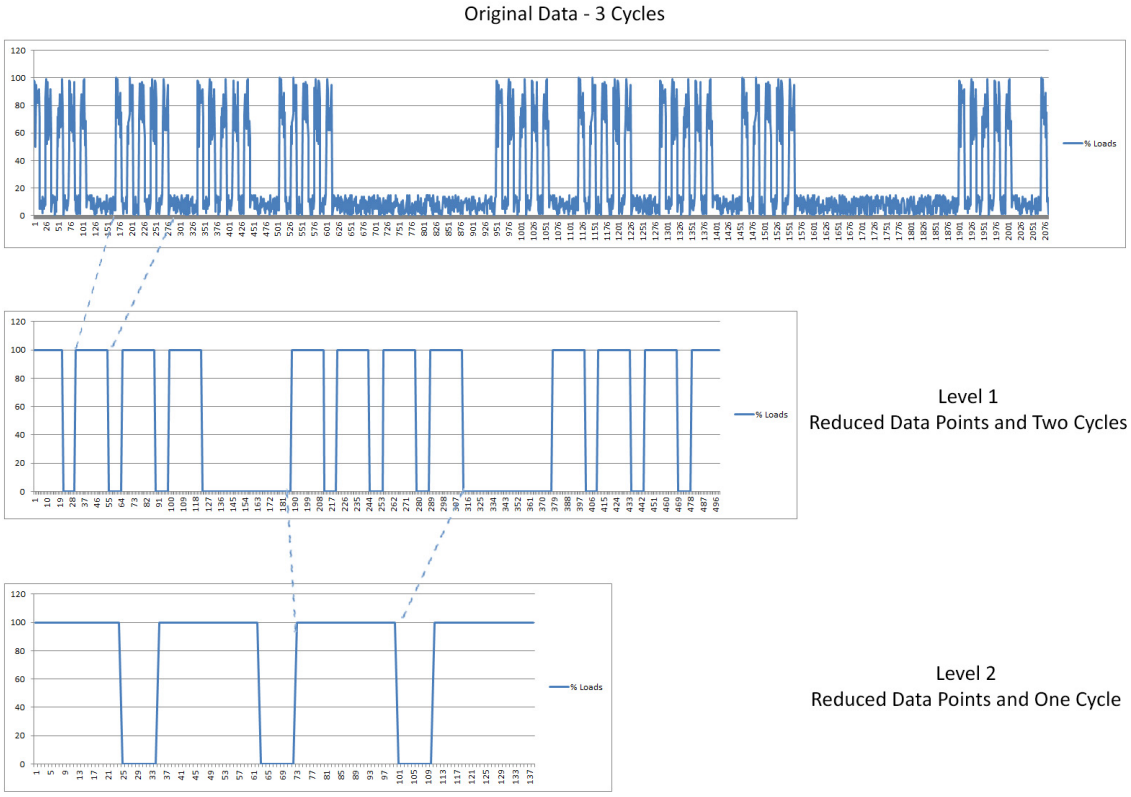


Figure 3.1 Illustration of hierachical analysis reducing data points.

data, instead of the whole data. This simplifies calculations for d and τ and also proves computationally less intensive.

As an over-simplified example of what the hierarchical analysis achieves, the Figure 3.1 shows three levels for a VM. Level 0 is the original data of the loads collected. As can be seen, there are three dominant cycles in the load. Level 1 shows that the same load has been stripped of the smallest cycle and only shows two major cycles. Level 3 again normalizes one cycle to show only one cycle.

The load statistics are still collected at regular, equidistant intervals. However, more levels of statistics are stored now. Every level is an abstracted version of the previous. From the original data, the base level or level0 is is created by normalizing and thresholding the original data. Suppose there are n original data points. Then, every x contiguous values (x defined later) from level0 are averaged out to give n/x data points. Moving averages of

these are taken and thresholded. These n/x data points form level1. The same is applied again to form level2 and so on for a predefined number of levels. New values are added to each level as and when enough statistics accumulate at the tail of the previous level. This requires very few computations at any given time.

In general, for a level L, the time series T is given by

$$T_L = [T_{L_1}, T_{L_2}, ..., T_{L_{n/x}}] \tag{3.5}$$

$$Temp_{L_i} = (\sum_{j=0}^{x-1} T_{(L-1)_{(i*x)+j}})/x \tag{3.6}$$

$$T_{L_i} = (\sum_{j=0}^{x-1} Temp_{L_{i+j}})/x \tag{3.7}$$

Thus, the above equation can be used to calculate a value of time series in a level, based on the previous level.

At every time instance different levels exist that describe the VM's load with different levels of precision, each level with less values than the previous. The idea is to start at the topmost, most abstract level, and begin applying the chaos theory algorithm. Suitable values of τ and d are calculated. This calculation is now less complex since the data points and number of cycles or variations have been reduced. Then the algorithm continues. However, instead of calculating the prediction, simply the Nearest Neighbors in this level's Phase Space Matrix is found. A new Phase Space Matrix of only the Nearest Neighbor rows is formed, with each row replaced by x rows comprising of the corresponding points from the previous level.

Thus for level L, the Phase Space Matrix will have number(Nearest Neighbors in L+1)*x rows. Then Nearest Neighbor comparison is performed on this Phase Space Matrix. This means, in every previous level the framework only searches in the vicinity of rows corresponding to the Nearest Neighbors of the current level. This process continues all the

way down to the base level, where the rest of the forecasting steps are completed and a y_{t+1} calculated.

By intelligently choosing the number of levels, the total comparisons to be done can be greatly reduced. Also, the final forecasting calculations are made in a subset of the base level series.

3.3 System Design

A prototype of the framework was setup in Python with some portions of Chaos Theory calculations written in C. All machines in the cluster run Ubuntu 12.04 and have kvm, libvirt 0.10.1 (with virsh) and virt-top 1.0.6 installed on them. Based on the logic described in the previous section, the framework consists of the following components:

- Stats Collector
- Analyzer
- Decision Maker
- Migrator

One of the PMs in the cluster is chosen as the 'Master'. All the above components reside on this machine. Since there are no special requirements for the Master, it can still be a regular node in the cluster and host virtual machines just like other nodes.

The final goal of the framework is to predict (for sometime in the future) how the VMs are going to behave and based on that re-arrange them (if necessary) to occupy as few a PMs as possible. In this section, each of these components shall be described.

3.3.1 Stats Collector

The Stats Collector component uses the remote capabilities of virt-top and virsh (domblkstat and domifstat) to query each PM in the cluster through ssh. Each PM then provides the master with statistics for CPU, Memory, Block I/O, and Network usages for each VM

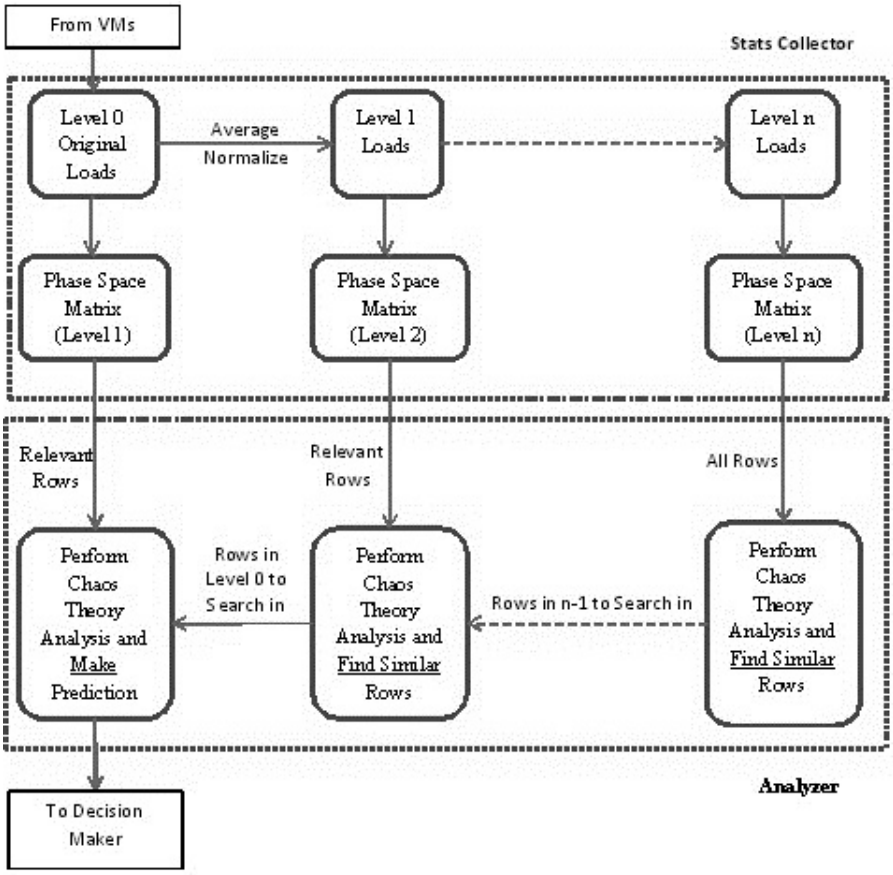


Figure 3.2 Logical flow of Stats-Collector and Analyzer.

residing on it. This information is conveniently provided as a csv file. The statistics are collected every 5 seconds.

Using the equation mentioned in the previous section, the latest values for each hierarchical level are calculated. In this thesis, the value of x is chosen to be 5. This value has been chosen simply because it divides up the collected time series into approximate groups of 10 mins, an hour, a day and so on in each corresponding level. Since in the base level (level 0) each value is the load at intervals of 5 seconds, the values in higher levels approximately represent loads of the durations shown in Table 3.1. In the experiment test-bed, since the experiments are run for 2 weeks at a time, the framework goes no more than 3 levels, since not enough data points accumulate in the higher levels.

Table 3.1 Significance of Each Data Point in a Level

Level	1	2	3	4	5	6	...
Time Unit	25 secs	2 mins	10 mins	1 hour	5 hours	1 day	...

Once the latest values for each level are calculated, the encoding is applied and the final encoded integer for each level is appended to a file corresponding to that level. Technically, any value of x can be chosen that suitably creates the hierarchical structure.

Pre-knowledge of the cluster behavior (if any) can be used to make an even more intelligent decision of x.

3.3.2 Analyzer

The Analyzer goes through the hierarchical files and uses the modified Chaos Theory algorithm described in the previous section to come up with predictions for some time t in the future. This time t has to be significantly greater than the average time it takes to migrate a VM plus the average turn-off and turn-on times of PMs. The framework automatically adjusts and settles on a value of t after a few migrations. In the experimental testbed, t was generally about 10 mins. Thus, the framework consolidates VMs only if they can coexist for at least 10 minutes. Otherwise, it runs the risk of migrating VMs and shutting down PMs, only to turn them on a few minutes later.

A particular resource on a VM can have a load between 0 and 100. Based on initial experiments, the average error in prediction is +/-10. This error is bearable if the predicted value is greater than the actual value. However, if the predicted value is less than the actual value, it might result in a VM placement such that the host PM overloads, thus providing a VM less resources than it needs. This means the VM has to contend with other VMs for its share of at least one resource. This is defined as a Service Level Objective (SLO) violation for that VM. In order to avoid this, some buffer value is added to each predicted

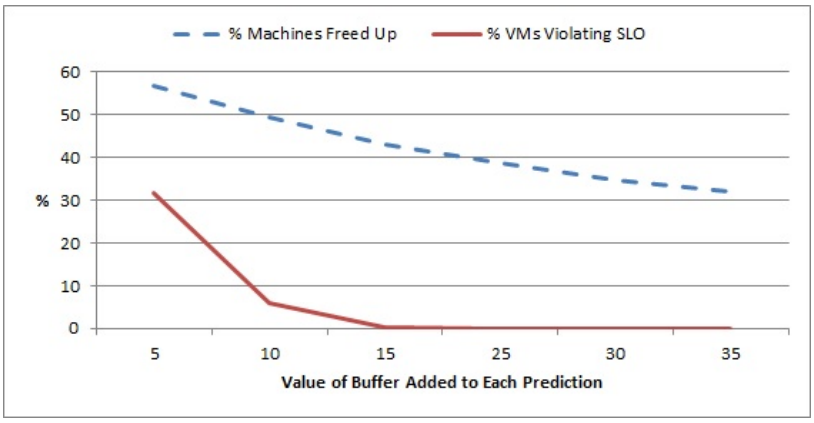


Figure 3.3 Different buffer padding overload and ffficiency.

value. This obviously has a trade-off that the consolidation is slightly less than optimal, but it drastically reduces possible PM overloads.

To decide on a buffer value to add, a few sets of experiments were run with different values of the buffer. Figure 3.3 shows the effect of adding some buffer to the predicted value on the % of VMs violating SLO and the % of PMs freed up.

Obviously, adding a very low value as buffer means better consolidation but more overloads. On the other hand, adding a very high value means less overloads but worse consolidation. Therefore, it was chosen to add an extra 15 to each predicted value since it results in almost no overloads. While the framework currently doesn't do so, this value can be fine-tuned on the run by comparing errors in predicted and actual values every time.

Finally, the framework goes through the predicated values and come up with one number for each resource that is indicative of the resource's behavior in the next t minutes. This is simply the average of the resource's predicted values for the next t minutes. These values for all four resources for all VMs are then input to the Decision Maker.

3.3.3 Decision Maker

The Decision Maker goes through the average predicted values and tries to rearrange the VMs so that the least number of PMs are used with some guarantee that none of the PMs will be overloaded for at least t minutes.

The framework at this point knows the capacity of each PM and the predicted values of the VM resources. This is treated similar to a multi-dimensional bin-packing problem. The problem is simplified by clipping all load predictions to one of four values (high, moderate, fair, low).Thus the VMs are packed into the PMs based on the knowledge that one PM can have a certain number of high VMs.

3.3.4 Migrator

The Migrator receives the Decision Maker's suggested configuration and then remotely live-migrates the VMs over ssh to achieve this configuration. It takes into consideration a variety of factors including the fact that no PM should even temporarily overload during migration. In case the suggested configuration requires more PMs than the ones already running, the Migrator remotely turns on suitable machines. Once the migrations are done, all PMs that are not being used are remotely switched off or prepped for other purposes as required.

3.4 Experimental Results and Evaluations

The framework was tested on a test-bed of 12 PMs. Each PM had an i-3 Processor, with 4 GB of RAM running Ubuntu 12.04. Each VM was created with 1 GB of RAM, running a server flavor of Ubuntu 12.04. This implies that each PM could host a maximum of 4 VMs at full load without overloading. Thus the total number of Virtual Machines hosted in the cluster was 48. The load patterns were based on trace-data collected from Google Data Trace [50], NASA's website, Clarknet Data, and the World Cup website [87] over maximum 2 months. The data was scaled-down so that it fit within 2 weeks. Also since the loads applied were from the same time-zone, each VMs load was randomly time-shifted to demonstrate usage in different time zones. For the Google Cluster Data, traces for 150 random machines were separated from the total machine traces available. For each

machine, the sum of the usages of all the tasks at each time instant was used as the load value.

The NASA, Clarknet and World Cup traces with sampling rates of 1 minute were used to drive RUBiS web servers (PHP) [88].

For the Google Data Traces with sampling rate of 5 minutes, to generate the actual loads on the VM, the following tools were used, driven by the traces:

- Memory: LAME encoder [89], Stress utility, IOzone [90]
- CPU: LAME encoder, Stress utility
- Block IO: IOzone
- Network: netperf [91]

To compare the framework, versions of predictors using Fourier Transform and Sliding Windows (FFT w. SW) based partly on [48] and Wavelets with Markov Chains (WMC) using Haar and Daubechies functions based on [51] were built. At every prediction stage, predictions were made for the next 60 steps. For FFT and WMC, 3000 preceding historical points were used to make the predictions, while for the framework all of the preceding historical points were used. The actual maximum resource usage was 100, though for easy comparisons, they have been normalized to 1.

3.4.1 Prediction Analysis

At a given time, each VM resource can have a maximum load of 100. Figure 3.4 shows the actual and predicted loads of a single CPU resource of a VM for one run of the experiment. The times are in 5 minutes.

The prediction accuracy of the framework is analyzed on three aspects. Figure 3.5 shows the Mean Square Error (MSE) of the framework, WMC, and FFT w. SW for all the data traces used. It can be seen that the framework outperforms the other two methods for all data traces with MSE between 0.005 and 0.015.

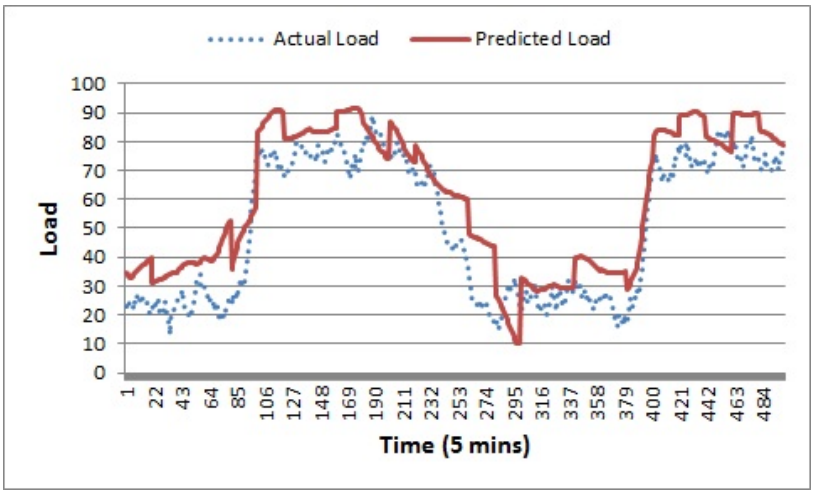


Figure 3.4 Single VM resource load prediction sample.

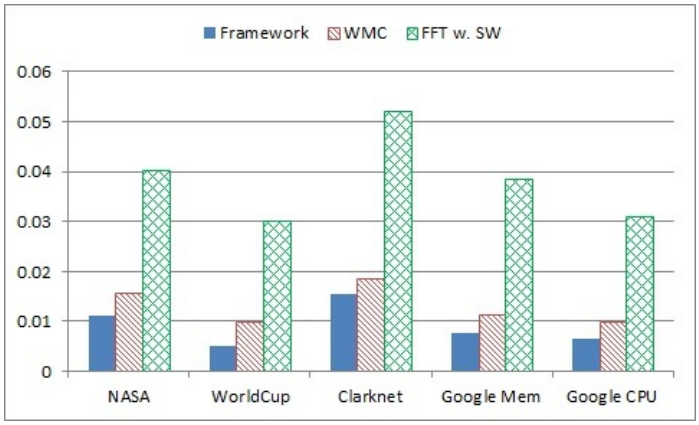


Figure 3.5 Mean Square Error.

$$MSE = 1/n \sum_{j=1}^{n} (predicted - actual)^2 \tag{3.8}$$

Figure 3.6 is the cumulative distribution for the MSE for the WorldCup data trace. With respect to the other methods, the framework converges quickly and nearly 80% predicted values are below 0.004.

Since the framework relies on co-existence and dynamic nature of VMs in a cluster, it is important to analyze the total number of VMs at any time that are predicted wrong. From the point of view of consolidation, the conditions for wrong predictions can be relaxed. In

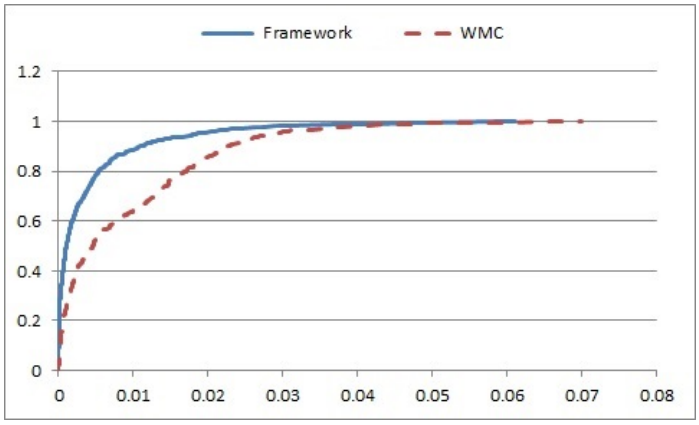


Figure 3.6 Cumulative distribution of MSE.

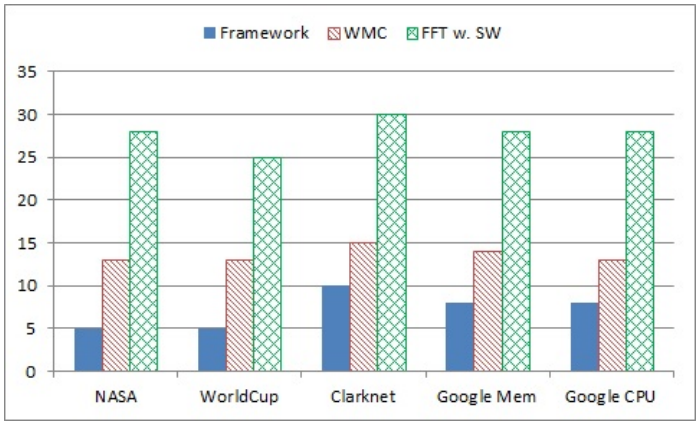


Figure 3.7 Average % VMs with error > threshold.

Figure 3.8 Average physical machines freed up.

Figure 3.9 % VMs violating SLO.

general, if the absolute error of a prediction is below a threshold, the prediction is still useful and results in negligible SLO violations and/or under-consolidation. In all the experiments, since the maximum resource usage of a VM can be 100, the threshold is set to 10. Figure 3.7 demonstrates % VMs in the cluster predicted wrong on an average by this definition for the entire experiment run. Again, the framework has only 5% to 10% wrongly predicted VMs on an average.

3.4.2 Consolidation Analysis

Three sets of experiments were run. Each time, the experiment was run for two weeks and the number of PMs freed up and the number of VMs violating SLO was recorded. A

VM (v) violates SLO when, for any resource, *(total capacity of the host PM) - (sum of requirements of other VMs on that host) < (requirement of VM v)*

Since the RUBiS webservers did not generate enough memory loads, for the consolidation set of experiments, only the Google Memory and CPU data traces were used.

Figure 3.8 shows a comparison of average PMs freed-up for the framework against an optimal solution (retroactively calculated) as well as versions of FFT w. SW and WMC for the three runs of the experiment. Figure 3.9 displays the average number of VMs violating SLO at any given time during the experiments. With the framework, on an average 4.8 of 12 Machines remained freed up during the period, with about 0.44 of 48 VMs violating SLO at any time.

3.4.3 Overhead Analysis

The time to compute for the framework without the hierarchical optimization and WMC is similar. As the historical data considered for prediction increases, the number of points to analyze increases linearly. This results in a sharp increase in per prediction computation time. On the other hand, using hierarchical levels, the number of data points to analyze increase slowly and so does the computation time as history becomes large. Figure 3.10 shows the data points to analyze *with* and *without* using hierarchical levels as the historical data increases. Based on this, Figure 3.11 shows the time taken to calculate 60 predictions points in the future *with* and *without* using hierarchical levels as the historical data considered increases.

Finally, Table 3.2 shows the statistics in terms of % of PMs freed up (out of 12). It also shows the average % of VMs (out of 48) with SLO violations for the framework as well as FFT w. SW and WMC.

When compared to the *optimal* configuration, the framework under-performs as expected. On an average, the framework can turn off 80% of the PMs that could be turned off optimally while keeping the average SLO Violating VMs to about 3%. This is based

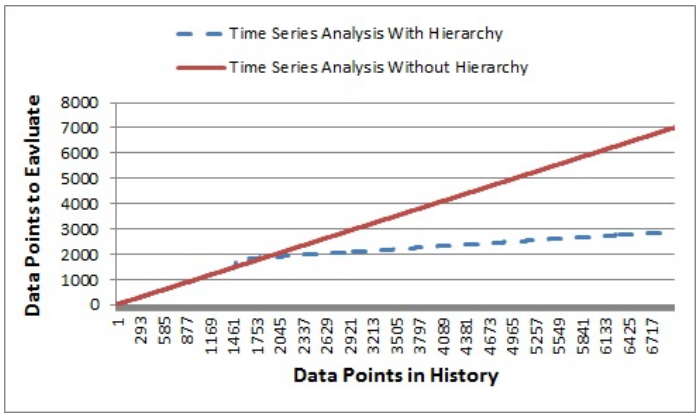


Figure 3.10 Comparison of # data points.

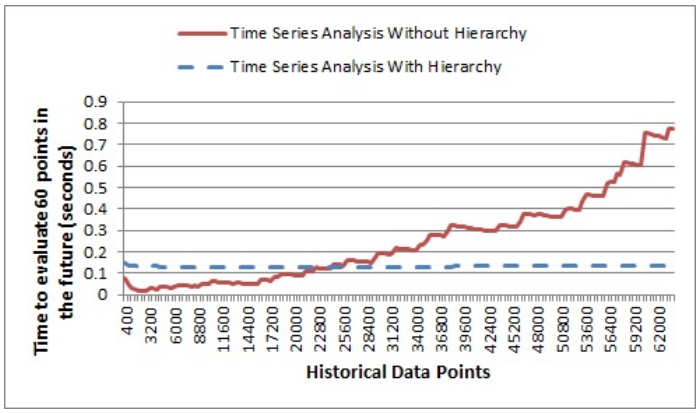


Figure 3.11 Comparison of time to predict.

in two aspects of the framework. First, the prediction errors inherent in Chaos Theory and second, the padding or compensation that were added to the predicted loads to offset under-prediction errors.

As seen in Table 3.2, for various real world loads, the framework outperforms both FFT w. SW and WMC in terms of consolidating VMs in the least number of PMs. Further, it also keeps the % of SLO Violating VMs much smaller than the other methods. This is expected since the framework is unaffected by multiple cycles. In comparison to WMC, due to the nature of chaos theory, every predicted data point does not depend solely on the previous data point. This helps the framework make better multi-step predictions at a time. Moreover, the hierarchical analysis makes it feasible for the framework to quickly consider all of the past history rather than just a window of historical data.

Table 3.2 Avg. % of PMs Freed Up/ % VMs Violating SLO

	Optimal	Framework	WMC	FFT w. SW
Free-Up	50	40	35	24
Violations	NA	3	7	20

3.5 Summary

This chapter discussed that VM loads in data-centers follow a chaotic pattern and hence, Chaos Theory is a viable option for their prediction. Based on the experiments, a framework was demonstrated that uses some concepts from Chaos Theory along with some optimizations to analyze and predict loads of the four resources of Virtual Machines with some precision. The final goal of the framework is to use the predictions to re-arrange the VMs in the cluster to reduce the number of PMs being used, while guaranteeing some level of QoS. It was shown that the framework works suitably for loads that have multiple inherent cycles and is scalable for large historical data. No assumptions need to be made on the type of load cycles or their periods.

Finally, the framework was compared to an optimal case, as well as other time series methods (FFT with Sliding Windows and Wavelets with Markov Chains). The results show that using the framework frees up about 80% of the under-utilized PMs that could have been optimally freed up, with an SLO violation of about 3%. Further, with respect to an optimal configuration the framework is able to free up about 30% more PMs compared to the other methods, with significantly less VM SLO violations. Moreover, it was analyzed that the framework is faster than other time series methods as the size of historical data increases.

As further optimizations to the framework, there are plans to port it to other hypervisors including Xen. In the future, the decision making process could include one more parameter - pre-migration or post-migration. Based on the VMs' predicted loads in the future and the characteristics of pre and post migration, the framework could determine the desired form of migration for each VM.

CHAPTER 4

AMORTIZED LIVE MIGRATION OF VIRTUAL MACHINES

This chapter discusses a network efficient, amortized live migration protocol that utilizes the inter-VM similarity between pages.

The rest of the chapter is organized as follows. Section 4.1 presents the approach to solve the two issues of network efficiency and fast migration times for live migration. Section 4.2 discusses the design considerations and implementation of a prototype. Section 4.3 includes the experimental setup and elucidates the efficacy of the protocol through experiments. Section 4.4 summarizes and concludes the chapter.

4.1 The Approach

Figure 4.1 depicts an overview of the framework. The framework applies to a subset of physical nodes from the cluster that choose to participate and among which migration is desired. In a datacenter, since VMs are not migrated to random nodes, this subset could be a set of nodes contributing to a private cloud, or a group of nodes servicing the same user. From this subset, one node is chosen at random as the Directory Server while the others are denoted as Hosts. The Directory Server is only responsible for syncing the various hosts while the remainder of the protocol is distributed. The Directory Server, which has no special requirements, can also function as a regular host.

4.1.1 Memory Page Distribution

Each Host in the cluster runs a number of VMs and is responsible for maintaining a list of memory pages that are important to the VMs residing on it. These pages are called Locally Significant Pages. This significance depends on the continued existence and use of the page by VMs in the Host. The hosts inform the Directory Server of their respective Locally Significant Pages by sending hashes of the pages. The Directory Server on its part analyzes

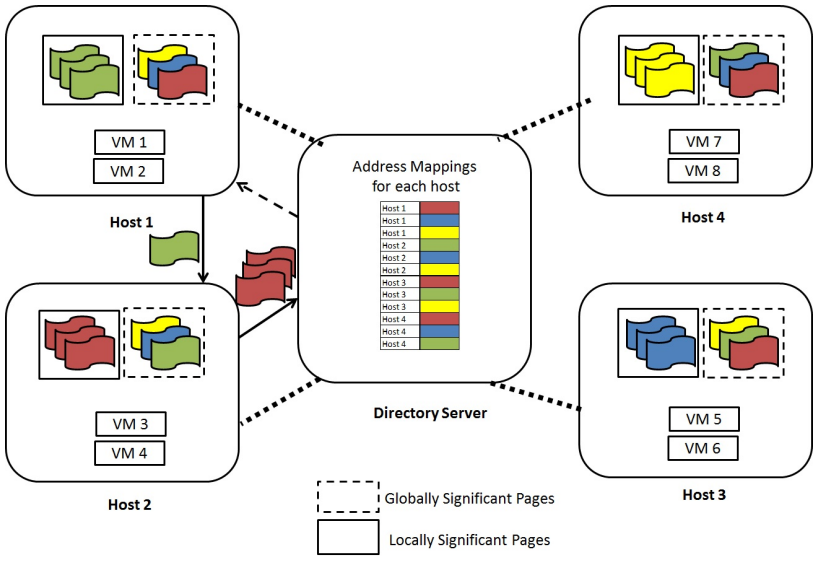


Figure 4.1 System overview.

these hashes and maintains a list of memory pages that are important throughout all the VMs in the cluster. These are called Globally Significant Pages. Note that all Locally Significant Pages need not appear in the Globally Significant Pages list. The decision of marking pages significant or insignificant depends on how often that page is used by different VMs, how many different VMs use that page, and how long that page is in use. This is covered in later sections.

The Directory Server then tries to spread the most significant global pages across hosts in the cluster that do not already have them. It does this by periodically identifying free hosts and requesting hosts that have the significant pages to send them to the hosts that do not. Thus each host has a set of locally significant pages and a set of globally significant pages. At any given time, for each host, the Directory Server also maintains a mapping for all pages that the host has from the Globally Significant Pages.

This periodic distribution of significant pages is depicted in Figure 4.2. As can be seen the Host transfers hashes of pages it deems significant to the Directory Server, which in turn requests other Hosts to provide globally significant pages. The significance marking

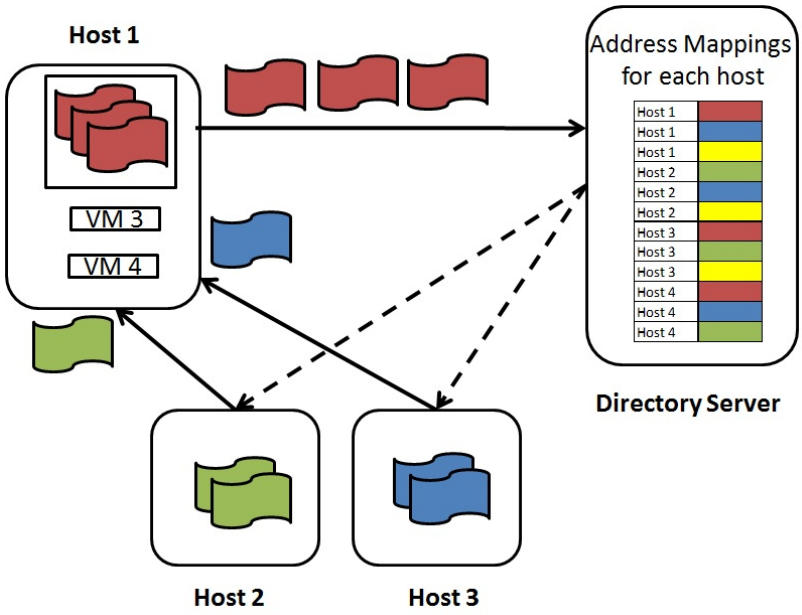


Figure 4.2 Periodical.

of the pages ensures that unnecessary memory transfer does not occur during this periodic phase.

4.1.2 Live Migration

Once this distribution and sharing has taken place, Figures 4.3 and 4.4 show how both pre- and post- copy live migration work with less network usage. In case of pre-copy migration (Figure 4.3), the Directory Server informs the Source of all the pages that it shares with the Destination using a Mapping File. Once migration starts, the Source only needs to transfer those pages that are not already existent at the Destination. For all other pages the Destination fetches them locally. For example, in Figure 4.3, the Source has to transfer pages P1, P2, P3, and P4 to the Destination. However, P2 and P4 already exist on the Destination. Therefore, the Source only transfers pages P1 and P3.

In case of post-copy migration (Figure 4.4), the Directory Server additionally identifies a Secondary Source that shares a lot of pages with the Original or Primary Source. Now the Destination can either pull up pages locally or from the Secondary

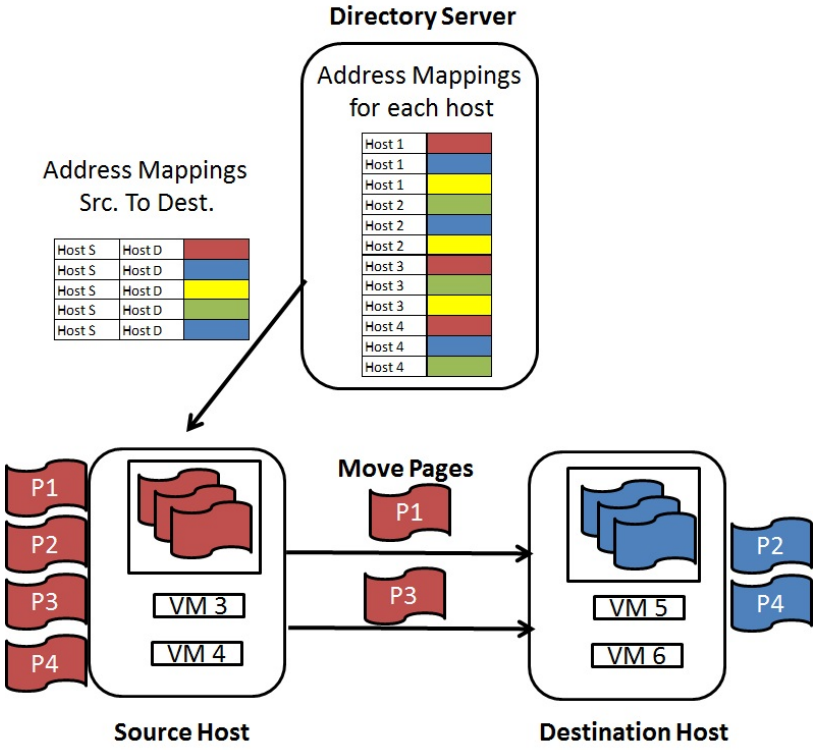


Figure 4.3 During Pre-Copy.

Source on-demand. The Primary Source is only responsible for the pages that are unique to it. Once those subset of pages are transferred to the Destination, the Primary Source is deemed free and is not required in the migration any more. For example, in Figure 4.4, the Destination needs to pull pages P1, P2, P3, and P4. Since the Destination already has P2, it fetches the page locally. P3 and P4 are remotely pulled from the Secondary Source, while the Primary Source is only responsible for P1.

Note that throughout the chapter ‘page’ refers to the contents of the memory page. Thus, similar, shared, or common pages are pages that have the same content, and not that they are the same page frame number in memory.

The details of various aspects of the cluster to achieve the protocol explained in the previous section are now described.

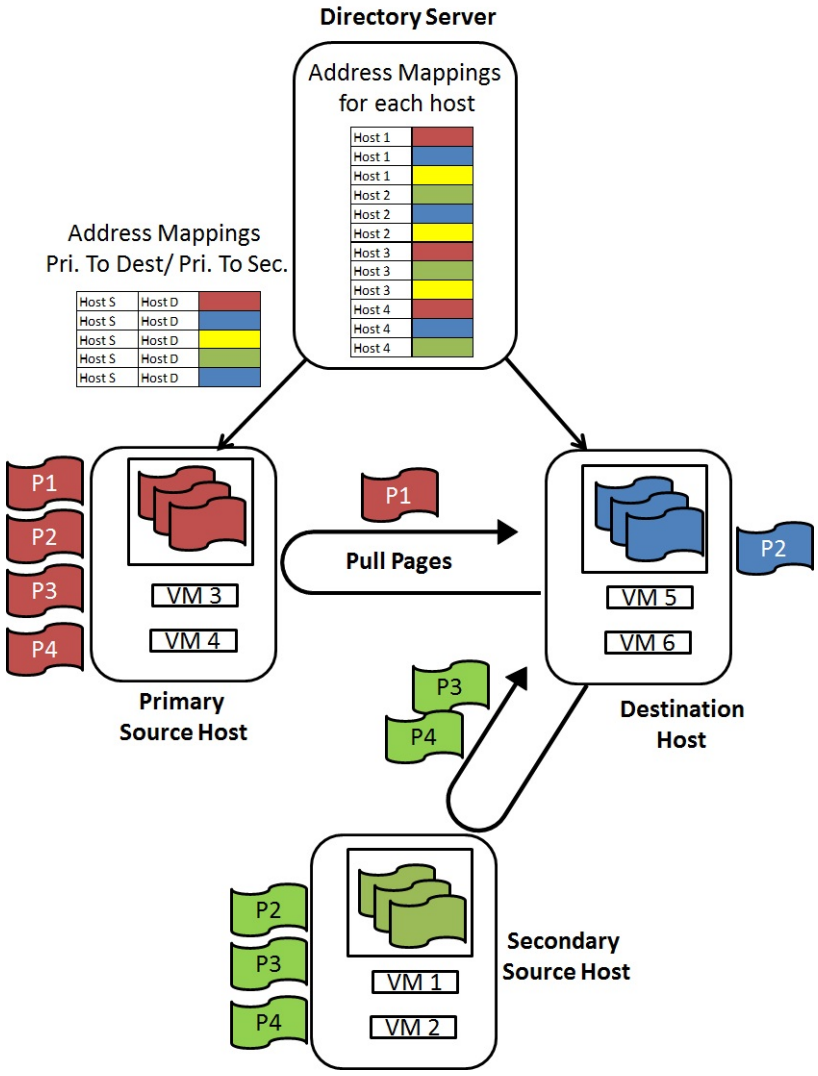


Figure 4.4 During Post-Copy.

4.2 System Design

A prototype of the approach described previously was built using KVM and QEMU. The important considerations for the prototype are how to store the memory and various page mappings, as well as the actual protocol during runtime.

4.2.1 Storing Memory and Mappings

This subsection discusses the various files and formats that are used to maintain a record of the memory pages, their hashes, and their identifiers. Since the Hosts and the Directory Servers store slightly different information, each is dealt with separately.

At the Host Every Host maintains a Memory Pool File (MP) and a Memory Pool Hash File (MPH). The MP is the set of Locally and Globally Significant Pages as mentioned before. It ideally contains all the important pages in the cluster. The MPH contains the hashes of all the pages in the MP.

Every page in the MP is given a unique integer Identifier. This Identifier can be arbitrary and only needs to be unique within the host's MP. It has no implication to or derivation from the page's actual memory address in a particular VM. Since this Identifier should label each page in the MP, for memory of size 4 GB, an integer of 64 bits should be sufficient.

The only constraint on the MP is that at all times it must remain sorted on the Identifiers. At any time, new pages may be added to the MP and /or existing pages may be removed. This can happen in a number of scenarios or phases of the protocol as shall be seen shortly.

The constraint on the MPH is that it should be maintained sorted on the hash values. Each hash value also has the same Identifier as the corresponding page in MP. The hash values for each page will be 20 bytes.

If P_1, P_2, P_3 are page contents, the MP looks like:

$$\begin{aligned} 1 &: P_1 \\ 2 &: P_2 \\ 3 &: P_3 \end{aligned} \tag{4.1}$$

The MPH (assuming $Hash(P_2) < Hash(P_3) < Hash(P_1)$) looks like:

$$\begin{aligned} Hash(P_2) &: 2 \\ Hash(P_3) &: 3 \\ Hash(P_1) &: 1 \end{aligned} \tag{4.2}$$

At the Directory Server The Directory Server maintains the Global Memory Pool Hash File (GMPH), and additionally, for each host, separate Host Mapping Files (HM).

The GMPH is similar to the MPH but is at a global level and contains unique hashes and global Identifiers for all the important pages in the cluster. The Identifiers and hashes are again, 64 bits and 20 bytes respectively.

An example GMPH would be:

$$\begin{aligned} Hash(P_2) &: 1 \\ Hash(P_3) &: 2 \\ Hash(P_4) &: 4 \\ Hash(P_1) &: 3 \end{aligned} \tag{4.3}$$

In this case, the HM for the Host based on the MPH in the previous example (Subsection 4.2.1) would be:

$$\begin{aligned} 1 &: 2 \\ 2 &: 3 \\ 3 &: 1 \end{aligned} \tag{4.4}$$

Notice that for a particular Host, the HM maps its local Identifiers to the pages' global Identifiers as stored by the Directory Server.

The sizes of the MPs and hence, the GMP are not allowed to increases arbitrarily. As will be seen, the protocol applies certain prioritizing algorithms to determine the significance of pages in the cluster and actively discards insignificant pages. This limits the MPs to a predefined size.

4.2.2 The Protocol

The protocol is characterized by 3 Phases.

- Collection Phase - Update MP, MPH, GMPH, HM.
- Push Phase - Clean up Memory Pools by adding significant pages and removing insignificant ones.
- Migration Phase - Use MP, MPH and HM to migrate VM from source host to destination host using either pre-copy or post-copy migration.

Collection Phase The collection takes place at a per Host level whenever the Host is relatively free in terms of its Memory, CPU, Network, and Block I/O usage. The aim of the Collection Phase is to transfer locally significant page Identifiers from the Host to the Directory Server and globally significant pages in the cluster to the Host. The collection phase has to be viewed from the Host's as well as the Directory Server's perspective. Steps 1-7 in Figure 4.5 depict steps in the Collection Phase below.

From the Host's perspective, whenever it is free it initiates the collection phase with the Directory Server and performs the following algorithmic steps to update its MP and MPH.

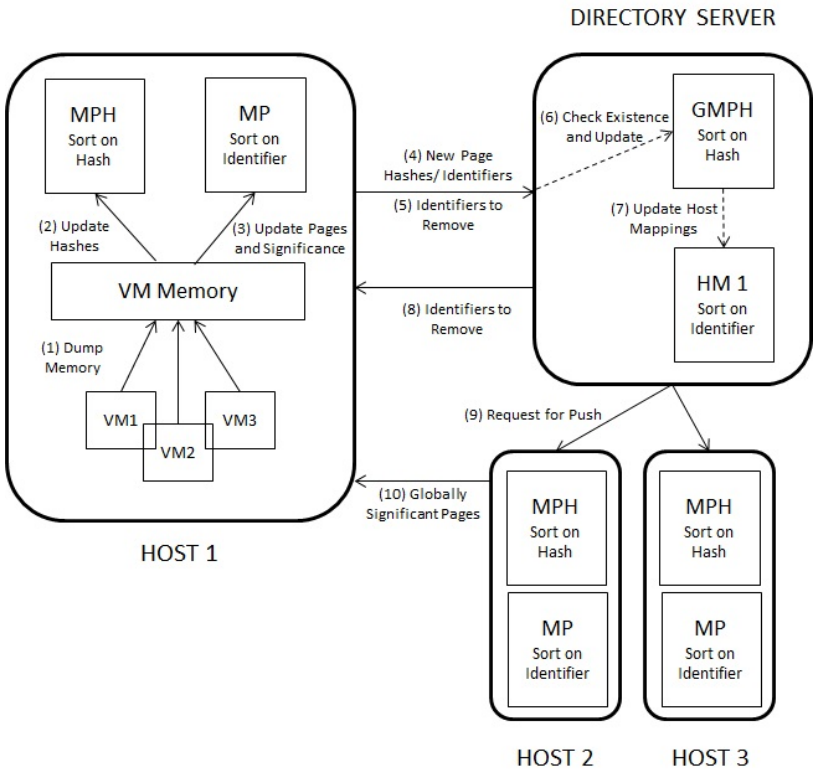


Figure 4.5 Collection and Push Phase details.

```
Step 1
for each VM do
    Dump Memory
end for

Steps 2 and 3
for each New Page do
    PageID = LastIdentifier in MP + 1
    LastIdentifier ++
end for
Sort New Pages on PageID
for each New Page do
    PageHash = SHA1(Page)
end for
Sort on PageHash
for each New Page do
    if Page in MP then
        PageLocalSignificance ++
        Remove New Page and Hash from Memory Dump
    else
        PageLocalSignificance = 0
    end if
end for
MP = MP + MemoryDump
for each New Page Hash do
    Insert into MPH (sorted)
```

```
end for

Step 4
for each New Page do
    if PageLocalSignificance > Threshold then
        Send PageID:PageHash to Directory Server
    end if
end for

Step 5
for each Page NOT in MP do
    Send remove(PageID) to Directory Server
end for
```

The significance value of each page ensures that the Host does not send constantly changing pages during every Collection Phase.

From the Directory Server's perspective, whenever a host initiates the collection phase, the Directory Server performs the following steps to update its GMPH and HM.

```
Step 6
for each received PageHash do
    if PageHash in GMPH then
        GMPH[PageHash] += HostID
    else
        Insert PageHash in GMPH
        GMPH[PageHash] = HostID
```

```
    end if
end for

Step 7
for each received PageHash do
    Update HM with GMPH-PageID
end for
```

Notice that throughout this phase, only the 20 byte hash for each page is sent to the Directory Server and not the whole page. Once the Collection Phase ends, the Directory Server enters the Push Phase and the Host waits to receive any globally significant pages from the other Hosts.

Push Phase The Push Phase is predominantly initiated at the Directory Server and consists of three important steps - calculation of page global significance, the actual requests for pushing of globally significant pages to hosts, and suggesting hosts to get rid of some insignificant pages. Steps 8-10 in Figure 4.5 depict the Collection Phase.

The global significance of a page is assigned as an integer value every time the Directory Server receives some page information from one of the hosts (at the end of the Collection Phase). The significance of a page is dependent on two factors: number of hosts in the cluster using it and average length of time it stays relevant.

Based on the information the Directory Server receives from the Hosts during the Collection Phase, the global significance scores for pages are maintained with the following rules:

- All new pages start off with a negative significance of -10
- Every time a page is in the MP of a 'new' host, the significance is increased by a factor (+1 for the first new host, +10 for the second new host, +20 for the third new host, so on)

- Every time an ‘orphaned’ page exists its significance is decreased by a factor like above
- Every time a host continues to have the page in its MP significance is increased by 1
- Every time a host that had the page no longer uses it, significance is decreased by 1

These rules ensure that pages that are used by a lot of VMs are given high global significance. Also, while continued existence of a page in a Host’s MP is rewarded, it is less rewarded than new Hosts having that page.

At the end of the Collection Phase and after marking global significance for received pages, the Directory Server checks the HM to see if the top most significant pages exist at the Host. If they don’t, the Directory Server requests other Hosts that have those pages in their MP to send the pages to the original Host. Any of these requested Hosts, if free, can inform the Directory Server and push those pages to the original Host. The number of pages pushed depends on the network usage and availability of Hosts at that time. These pages are included by the original Host in its MP with a special marker. Since these pages are not local to the Host their existence at the Host should not artificially inflate their significance. The marker ensures that the Host does not contribute to the significance of the page.

Finally, as the last step of the Push Phase, the Directory Server goes through the Host’s HM and looks for pages with really low significance that exist due to a previous Push Phase (indicated by the marker). If such pages exist, the Directory Server indicates to the host that the pages can be removed from its MP by sending the corresponding Identifiers. The Host then removes the pages and acknowledges the Directory Server, which in turn then updates the Host’s HM.

Migration Phase The migration phase works slightly different in case of Pre-Copy and Post-Copy. Figure 4.6 illustrates details during Pre-Copy migration. Note that steps (4) to (7-a,b) are repeated for each page to be migrated. To begin, the Source first consults the Directory Server and informs it of the Destination. If Guided Mode is used, the Source

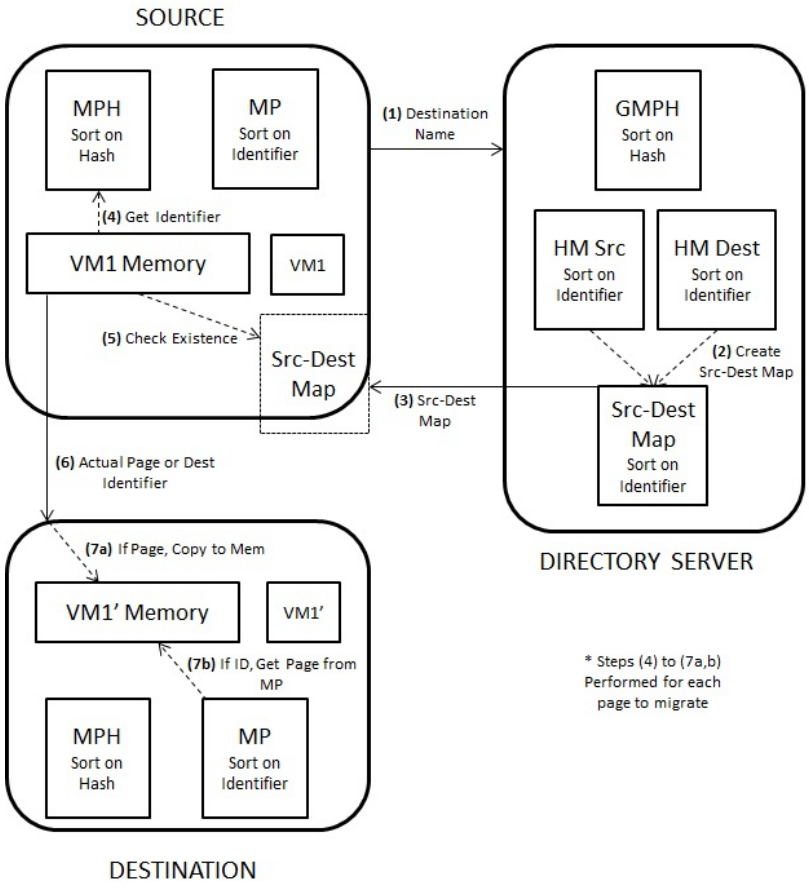


Figure 4.6 Pre-Copy: Migration Phase details.

provides the Directory Server with a list of possible Destinations, which then informs the Source of the most suitable of those based on amount of pages shared. The Directory Server then goes through the two HMs and combines them to generate a mapping of all memory pages that exist both at the Source and the Destination. This mapping is simply:

$$\begin{aligned} &SrcIDa : DestIDx \\ &SrcIDb : DestIDy \\ &SrcIDc : DestIDz \\ &\vdots \end{aligned} \tag{4.5}$$

This mapping is then sent to the Source and migration begins as usual.

For each page to transfer, the Source first hashes the page and obtains its identifier (if any) from the MPH. Since the MPH is kept sorted according to the hashes, this search is quick. It then checks if the identifier exists in the first column of the mapping it received. Again, the mapping received is already sorted on the Source Identifier, making the search quick. If the identifier does exist in the mapping, then the Source simply transfers the Destination Identifier to the Destination. Otherwise it transfers the whole page. Hence if a page exists at the Destination, only 64 bits instead of 4 kB need to be sent.

The Destination checks if the received data is an identifier. In this case, it copies the page corresponding to the identifier from its MP. Otherwise, it copies the received page as usual. Once again, obtaining the page from the MP is trivial, since the MP is already sorted on the Identifier.

This process continues iteratively like Pre-Copy till all the pages or the Identifiers are transferred to the Destination. If the page dirty rate is high, after certain iterations, the remaining pages (or their Identifiers) are transferred via stop-and-copy, just like in regular pre-copy. The Destination VM is then started and the Source VM is paused.

Figure 4.7 illustrates details during Post-Copy migration. For Post-Copy, the Source (now called Primary Source) again consults the Directory Server and informs it of the Destination. The Directory Server prepares the Primary Source to Destination mapping again. However, in this case, the Directory Server also identifies a third host based on the HMs that has the maximum pages shared with the Primary Source. This host is called the Secondary Source. A mapping is also prepared between the Primary Source and the Secondary Source. The Directory Server sends these mappings to the Source. These mappings look like:

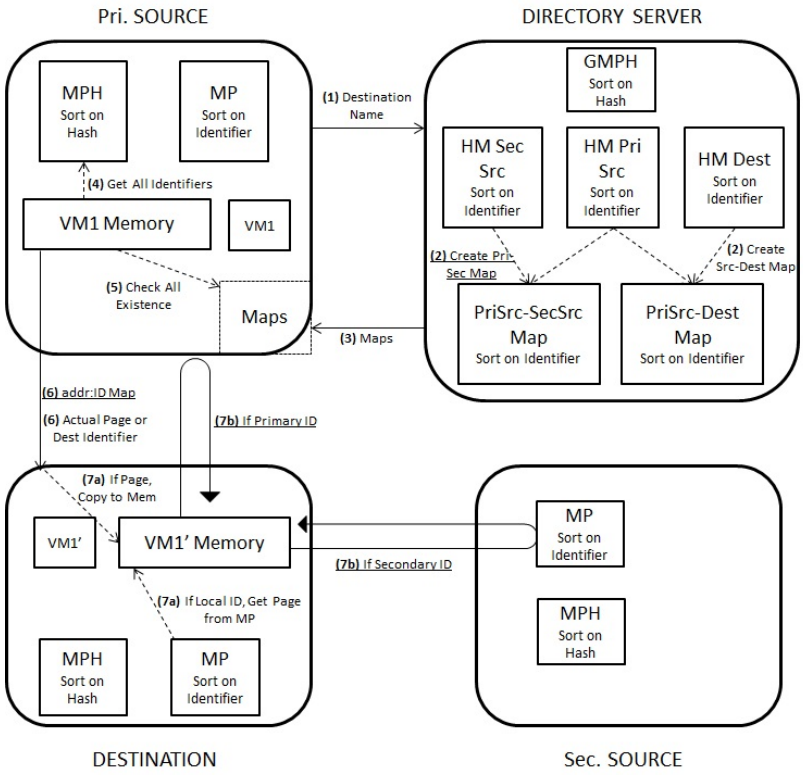


Figure 4.7 Post-Copy: Migration Phase details.

$$\begin{aligned} PriSrcIDa &: DestIDx \\ PriSrcIDb &: DestIDy \\ PriSrcIDc &: DestIDz \\ &: \end{aligned} \tag{4.6}$$

and

$$\begin{aligned} PriSrcIDa &: SecSrcIDx \\ PriSrcIDb &: SecSrcIDy \\ PriSrcIDc &: SecSrcIDz \\ &: \end{aligned} \tag{4.7}$$

The Source then pauses the VM, and creates a mapping of all current page addresses to either an identifier from the Primary Source, Secondary Source, or Destination based on the mappings received. This new address:identifier mapping is then sent to the Destination. Since the mapping does not contain the actual pages, it is relatively small. This mapping looks like:

$$\begin{aligned} PriSrcAddrA &: PriSrcIDa \\ PriSrcAddrB &: SecSrcIDb \\ PriSrcAddrC &: DestIDc \\ &: \end{aligned} \tag{4.8}$$

Post-copy migration is then initiated. Now, every time the Destination needs to pull up a page, it first checks the address in the mapping file and correspondingly either pulls it up from itself, the Secondary Source or the Source, in that order.

In the meanwhile, if background copy has been enabled, the Source background copies only those pages that do not exist either on the Destination or the Secondary Source. Once this is done, the source is free to quit the migration process.

4.3 Experimental Results and Evaluations

To evaluate the prototype system, a testbed cluster of 16 Physical Machines (PM) was used. Each PM has an i-3 Processor, both HDD and SSD storage, with 4 GB of RAM running Ubuntu 13.04. Each Virtual Machine was created with 1 GB of RAM, running different flavors of Linux. This implies that each Physical Machine could host a maximum of 4 VMs at full load without overloading. Thus, the total number of Virtual Machines hosted in the cluster was 64. To demonstrate the effects of Guest O.S. similarity on page sharing, different flavors of Linux were used. The flavors used as Guest O.S. included DSL (D. Small Linux), stress-linux, Ubuntu-server, and a custom built distro bench-linux based on Ubuntu. Only the memory of the VMs was migrated, while the disks were shared.

Since each PM consisted of 4 GB of RAM, the size of the Host Memory Pool File (MP) was restricted to 4 GB. If more significant pages were to be added during the Push Phase, the Directory Server would first suggest removal of some pages to respect this size. In general, the size of a MP is the same as the total amount of physical memory available on the PM.

KVM and QEMU were used to build and test a system based on the protocol. In theory, the protocol can be incorporated in any hypervisor with live migration. For pre-copy, QEMU's default module was modified, while for post-copy we used a modified version of [92]. The only functions modified were in the live migration module of QEMU. The migration:exec feature of QEMU was modified to create MPs as required without pausing or otherwise hampering the VM. Further, both the receiving and sending code in the migration:tcp feature was modified. Before sending a page, the page was first checked in the reduced subset of MPH. If it existed, only an identifier was sent, else the whole page

Table 4.1 Avg. % of Pages Shared at Time of Migration (Pre-Copy)

	VM's Guest O.S. on Dest.	VM's Guest O.S. not on Dest.
Without Push Phase	35%	2%
With Push Phase	40%	20%

Table 4.2 Distribution of Pages during Migration (Post-Copy)

	VM's Guest O.S. on Dest.	VM's Guest O.S. not on Dest.
Primary Src	50%	55%
Secondary Src	10%	25%
Destination	40%	20%

was sent. At the destination, if an identifier was received, the corresponding page from the MP was copied to memory else, as usual the received page was copied to memory.

For the sync-up between the host and the Directory Server, during the various phases, a mix of C, python, and bash script was used.

In order to test the system with real world loads, load patterns obtained from a sample of the Google Data Trace [50] were used. The two month long trace for memory, CPU, block i/o, and derived network was scaled down to fit two weeks. To generate the actual loads matching those patterns the Intel LINPACK [93], and NASA Parallel Benchmarks [94] were used on each VM.

Based on the real world Google data trace used to generate the loads, low resource usage for each Host occurred every 30 minutes on an average. This means that the system could get into the Collection Phase for the Hosts, every 30 minutes on an average without overloading the resources.

4.3.1 Base Approach Evaluation

The first set of experiments were performed without the Guided Mode, using both HDDs and SSDs on the host PMs for comparison. The aim of this set of experiments was to test the network and time savings using the protocol. The experiments were repeated with

Table 4.3 Avg. Pages in MB Transferred during Migration

(a) Pre-Copy

Operating System	Base Case	Guest O.S. on Dest	Guest O.S. not on Dest
DSL	1120	724	947
stress-linux	1139	825	1020
ubuntu-server	1415	1012	1303
bench-linux	1350	971	1201

(b) Post-Copy

Operating System	Base Case	Guest O.S. on Dest	Guest O.S. not on Dest
DSL	950	600	738
stress-linux	1005	620	773
ubuntu-server	1024	672	806
bench-linux	1024	636	785

three different network bandwidths of 1000 Mbps, 100 Mbps, and 16 Mbps (WAN) for comprehensive evaluation.

Experiments were performed in two batches. Each batch consisted of three runs, with each run on a different network speed (1000/100/16 Mbps) assuming full network bandwidth for migration. In each run, the cluster and the prototype was allowed to run for 2 weeks. In those 2 weeks, 4 VMs were migrated at random every 2 hours and statistics recorded at the time of migration. In the first batch (three runs) the migrations were done using pre-copy, while post-copy was used in the second batch (three runs).

Table 4.1 shows the average percentage of pages shared during each migration through the three runs for pre-copy with and without the Push Phase. The term Guest O.S. is used to mean the O.S. installed on a VM. It can be seen that if the migrating VM's Guest O.S. exists as a Guest O.S. on some VM on the Destination, on an average 40% of

the pages already exist on the Destination. For cases where the migrating VM's Guest O.S. does not exist on the Destination, 20% of the pages exist on the Destination.

Table 4.2 shows the average distribution in percentages of pages for Primary Source, Secondary Source and Destination during each migration through three runs for post-copy. Again, when migrating VM's Guest O.S. exists in some VMs at the Destination, only 50% of the pages need to be pulled from the Primary Source. 40% of the pages already exist at the Destination and about 10% of the pages can be pulled from a Secondary Source. For cases with the VM's Guest O.S. not existing at the Destination, only 20% of the pages exist at the Destination. However, 25% of the remaining pages can be pulled from the Secondary Source, leaving the Primary Source responsible for only 55% of the pages.

One important aspect to note here is that VMs also demonstrate self page similarities - that is several pages within the VM itself may have the same contents. In this case as long as the Destination has a copy of those contents, all the similar pages at Source are considered shared. The numbers discussed above include such shared pages.

Tables 4.3a and 4.3b show the average amount of pages transferred over the network during migration for the base case as well as the protocol. Note that while the VMs are created with 1 GB physical memory each, pre-copy needs to transfer more memory due to page dirtying during migration. Post-copy on the other hand, only needs to transfer 1 GB. The values shown for post-copy include transfers from the primary as well as the secondary source.

Table 4.4 displays the average time in milliseconds taken to migrate a VM in a number of different cases using Pre-Copy with HDD and SSD. These are total migration time, since the new protocol does not affect migration downtime. For network speeds of 1000/ 100/ 16 Mbps, the times to migrate VMs with four different flavors of Linux have been recorded separately. The times shown are for the Base Case (current live migration in KVM), and migration using the protocol. In case of using the protocol, the times to migrate

when the migrating VM's Guest O.S. existed on the Destination and when it did not have been separated.

In case of pure Post-Copy, the actual time to migrate is undefined, since it depends on if and when the migrated VM needs the memory pages. In practice some amount of background copying is used. This means, that in addition to pages being pulled, the Source keeps transferring some other pages to the Destination. In order to enable comparing times of migration, post-copy with full background copy from the Primary Source was initiated. This means that the Primary Source transfers all the pages unique to it to the Destination before declaring itself free. The remaining pages are either pulled locally or from the Secondary Source.

Table 4.5, therefore, shows times during migration at which the Primary Source is *freed* of its duties. Note that the migration itself may not be complete, since pages may still be needed from the Destination itself or from the Secondary Source. Again, the times shown are for the Base Case (current live migration in KVM), and migration using the protocol with HDD and SSD. In case of using the protocol, the times to migrate when the migrating VM's Guest O.S. existed on the Destination and when it did not have been separated.

4.3.2 Guided Mode Evaluation

The first set of experiments performed migrations of random VMs to random Destinations. In practice, VMs are not placed randomly with other completely different VMs. They generally reside along with similar VMs in terms of Guest O.S.s and applications running. As was shown in Table 4.1, due to the protocol, hosts with similar VMs share about 40% pages. In such cases, where the Destination host is not random, but within a pre-determined subset of hosts, one other advantage of implementing the protocol is the Guided Mode which is the selection of the Destination itself. Once the Host decides that a VM has to move, it sends the Directory Server a list of possible Destinations it can move to. The

Table 4.4 Pre-Copy: Migrate Times (ms) for Different Network Bandwidths

(a) 1000 Mbps

Operating System	Base Case	Guest O.S. on Dest		Guest O.S. not on Dest	
		HDD	SSD	HDD	SSD
DSL	12931	9038	7845	11095	10012
stress-linux	15241	10536	8631	13263	12140
ubuntu-server	35083	29311	24167	31837	29751
bench-linux	28738	22584	19410	25360	23963

(b) 100 Mbps

Operating System	Base Case	Guest O.S. on Dest		Guest O.S. not on Dest	
		HDD	SSD	HDD	SSD
DSL	90572	65483	52102	75574	71840
stress-linux	94905	70226	56051	80237	76914
ubuntu-server	111628	85145	67889	95834	87110
bench-linux	105027	78418	60053	89401	86391

(c) 16 Mbps

Operating System	Base Case	Guest O.S. on Dest		Guest O.S. not on Dest	
		HDD	SSD	HDD	SSD
DSL	700148	370151	364210	550912	543172
stress-linux	730235	390528	378025	558117	542611
ubuntu-server	781438	421046	391522	610389	616290
bench-linux	740913	402714	377814	584011	594108

Table 4.5 Post-Copy: Times (ms) to Free Up Source for Different Network Bandwidths

(a) 1000 Mbps

Operating System	Base Case	Guest O.S. on Dest		Guest O.S. not on Dest	
		HDD	SSD	HDD	SSD
DSL	10915	7254	7158	8916	8796
stress-linux	12510	9536	8820	10935	10125
ubuntu-server	15732	13104	11501	14121	12647
bench-linux	13264	11149	9435	12003	11314

(b) 100 Mbps

Operating System	Base Case	Guest O.S. on Dest		Guest O.S. not on Dest	
		HDD	SSD	HDD	SSD
DSL	75932	60378	58241	65025	60104
stress-linux	80725	59632	55409	68990	65361
ubuntu-server	91351	61770	61273	80486	73179
bench-linux	89410	51356	50107	70910	68979

(c) 16 Mbps

Operating System	Base Case	Guest O.S. on Dest		Guest O.S. not on Dest	
		HDD	SSD	HDD	SSD
DSL	602325	395308	381501	490136	475136
stress-linux	610813	407181	389794	499520	478637
ubuntu-server	632091	430292	412280	522918	506182
bench-linux	610991	411044	403798	510311	493420

Table 4.6 Impact of Guidance Mode - Avg. Migration and Response Times in ms

	No Protocol	Without Guidance	With Guidance
Migration Time	29568	24993	19825
Response Time	200	160	130

Directory Server then informs the Source of the most suitable of these Destinations based on page sharing between the two Hosts. In this case, a certain guarantee of quick migration can be given.

To test the benefits of using the Guided Mode, a second set of experiments was run with a realistic goal. In this experiment, instead of random migrations, pre-copy migrations were performed with the goal of load balancing and PM consolidation. At any given time, the current memory and CPU requirements of all the VMs are considered and new placements of the VMs are deduced such that the least number of PMs are used, without any overloads. In most cases, it is possible to move a VM to one of multiple PMs to achieve the goal. Without using the Guided Mode, one of these PMs would be chosen at random. However, the Guided Mode can aid this selection and reduce migration times and network bandwidth used.

In addition to the setup mentioned before, the VMs ran RUBiS webservers [88] following the NASA website trace [52] to demonstrate the effect on network usage and degradation with a network bandwidth of 1000 Mbps. Every 2 hours new configurations were calculated to achieve consolidation and load balancing based on the current VM loads. Table 4.6 shows the average times to migrate for traditional pre-copy against the protocol with and without Guided Mode. It also shows the average response times for the RUBiS servers during migrations (the base average response time with sufficient bandwidth is 100 milliseconds).

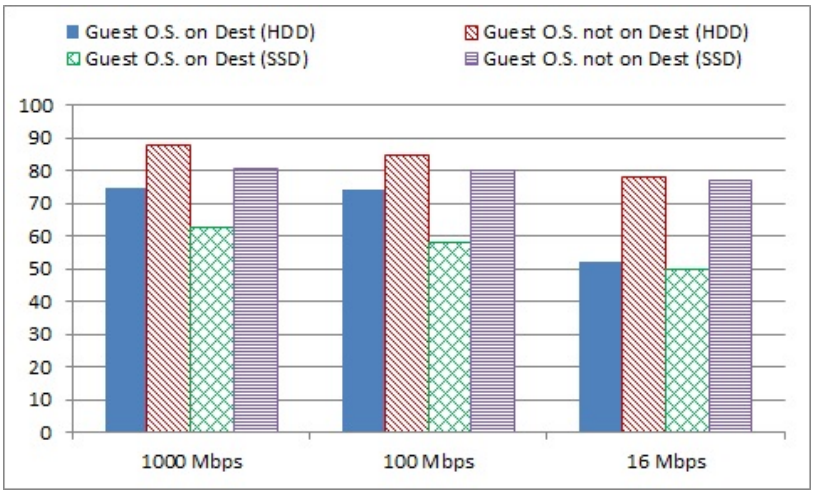


Figure 4.8 Pre-Copy: Time comparison in % of base case.

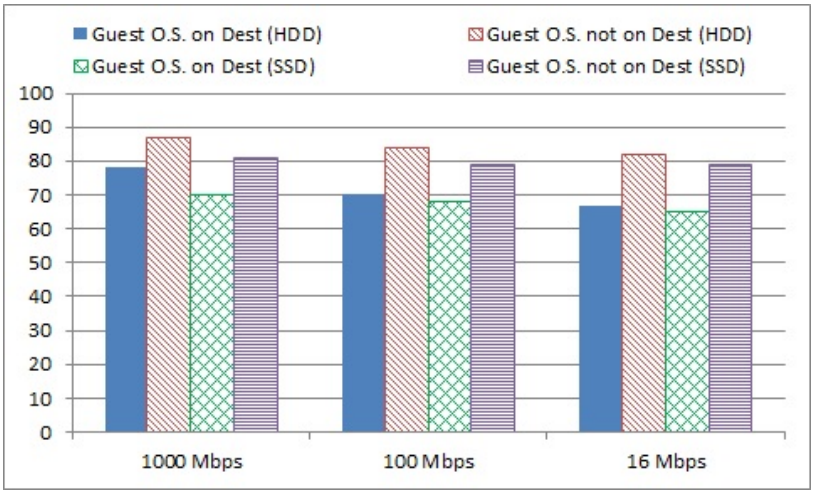


Figure 4.9 Post-Copy: Time to free source comparison in % of base case.

4.4 Discussions

The first set of experimental results show that using the protocol reduces the network bandwidth used in pre-copy and the time for the Source machine to free up in case of post-copy during live migration.

Figure 5.10 shows the improvements in pre-copy migration time using our protocol versus using current live migration. It clearly demonstrates that when migrating a VM to a physical machine that has another VM with the same O.S. and running similar applications, the protocol reduces migration time by up to 40 - 45%.

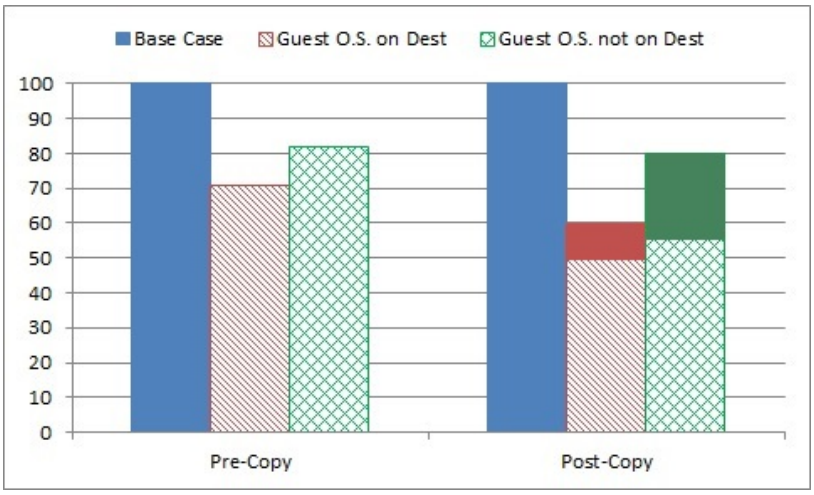


Figure 4.10 Average data over N/W during migration (MB) (For Post-Copy, lower portion is for Pri. Src. while upper portion is for Sec. Src.).

Figure 4.9 shows the improvements in post-copy migration time using the protocol. It demonstrates that when migrating a VM to a physical machine that has another VM with the same O.S., the protocol allows the Source to be free 30% faster.

The improvements in migration times bear a direct relation to the amount of pages transferred during migration, which in turn is dependent on the amount of pages that the migrating VM shares with the Destination. If two Hosts have VMs with the same Guest O.S., they have a lot of common pages by default. Even with the Push Phase, Hosts with all different Guest O.S.s cannot achieve the same amount of common pages. This is why the improvements over base case are greater when a VM is migrating to a Destination that has another VM with the same Guest O.S. and running similar applications.

As is clear, the protocol introduced, in terms of time, works better as the network bandwidth lowers. This is obvious since there is some local overhead involved in the protocol during migrations. With high bandwidth networks like 1000 Mbps, this overhead is more apparent than in the 16 Mbps network. The overhead mainly includes reading from files. As was mentioned in the previous sections, the various files the protocol maintains are sorted in certain ways that make accessing them and searching through them efficient. The

steps could further be optimized to increase the efficiency of the protocol. Since most of the local overhead is file I/O related, use of the faster SSD drives improves the performance.

The network bandwidth savings during migration however, are indifferent to the speed of the network. Figure 4.10 shows the average amount of pages in MB transferred over the network during VM migration for pre-copy and post-copy. It can be seen that the protocol reduces network usage by 30% and 40% for pre-copy and post-copy, respectively.

The trade-off to the improvements is the Collection and Push phases, that require some additional bandwidth periodically. These phases are performed during periods of *low* network usage. It is important to note that there is more control over when these phases are performed than when a live migration needs to be performed. Whether resources are scarce or abundant during live migration is not pre-determined. Intuitively, the need to perform live migration is indication of a load-balancing requirement or imminent machine distress, which means scarcity of resources. On an average, due to the local and global page significance, a Collection and Push phase transferred about 100 MB, which is just 2% of a Host's physical memory.

As mentioned before, the total additional storage required on a Host should at least be the same as the total physical memory on the Host (in our case 4 GB).

Finally, for the second set of experiments, Figure 4.11 demonstrates the effect of using the protocol to aid the selection of the Destination. The CDF is plotted for the time required to complete the migration as a percentage of base case during the second experiment set. It shows that with the protocol in Guided mode, 100% of the migrations complete within 70% of the time of base case as opposed to Non-Guided Mode, where 40% of the migrations require 90% - 100% of base time to complete. Both cases however, still perform better than not using the protocol at all.

Similarly, Figure 4.12 shows the response times for the VMs during migration. Using the protocol with Guided Mode ensures that 80% of the response times stay within 140 ms as opposed to not using the protocol where response times are as high as 220 ms.

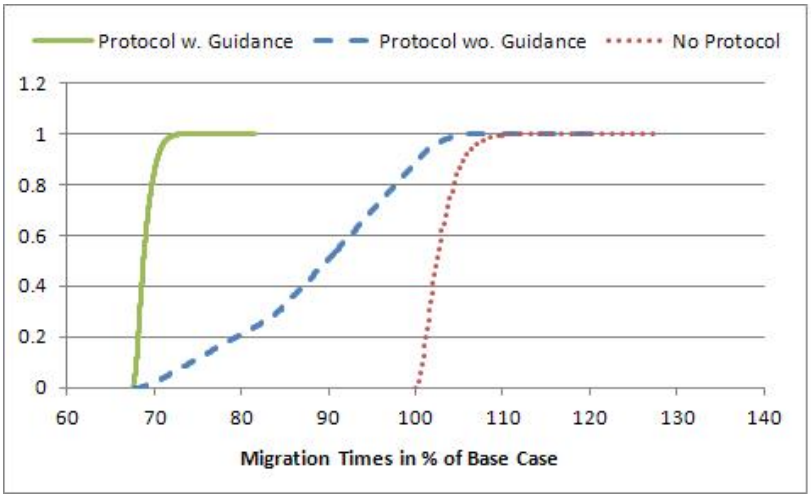


Figure 4.11 Migration Time CDF - with and without Guidance Mode.

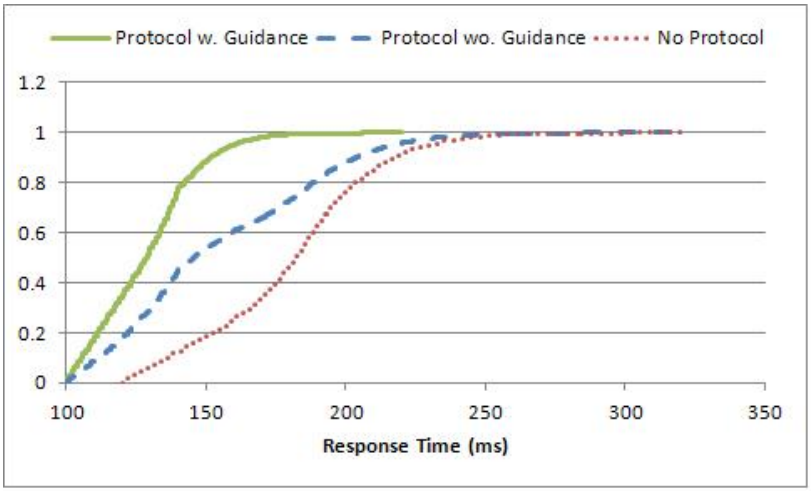


Figure 4.12 Response Time CDF - with and without Guidance Mode.

4.5 Summary

This chapter presented in detail a new live migration protocol that utilizes the commonality of memory pages between VMs along with proactive pushing of significant pages, to help avoid potential performance degradation of VMs during live migration. A prototype of this protocol was implemented and real world loads run through it. The experimental results show that using the protocol works towards the two goals of low network usage and fast times during live migration. Further, the protocol can also be used to identify suitable destination hosts from a cluster that can better achieve these goals.

The problems for both pre-copy and post-copy migrations were tackled. In case of pre-copy live migration, the protocol helps reduce the number of memory pages transferred during the actual migration and thus not only saves network bandwidth but also speeds up the migration. For post-copy, which is less network intensive as is, the protocol further reduces the network load and migration time. Besides, it allows the delegation of a secondary source that reduces the duties of the original source. This allows the original source to be free of the migration earlier than usual, without affecting the benefits of post-copy migration.

A future addition to the protocol is integrating it in other hypervisors like Xen. Also, a look-ahead mechanism could be added that predicts future times of low resource usage and prepares the system in advance for the collection and distribution of pages.

CHAPTER 5

MEMORY POOLING FOR VIRTUAL MACHINES

This chapter discusses a remote memory sharing and management framework called Autonomous Cluster-wide Elastic Memory (ACE-M).

The rest of the chapter is organized as follows. Section 5.1 presents the approach to enable remote memory access at a per VM level and chose suitable remote providers in the cluster. Section 5.2 details the design considerations and prototype for ACE-M. Section 5.3 includes the experimental setup and elucidates the efficacy of ACE-M through experiments. Section 5.4 includes comparative studies and discussions on a few aspects of ACE-M. Section 5.5 summarizes and concludes the chapter.

5.1 The Approach

Figure 5.1 depicts an overview of ACE-M. From the point of view of individual PMs, the approach works in three periodical phases marked by the labels 1, 2a, 2b, and 3 in the figure. In the first phase, a PM uses prediction mechanisms to predict future behavior (memory, network, block usages) of its resident VMs based on their historical usages. It then analyzes the future requirements to decide whether it can sustain the VMs' memory needs. Once the PM decides that the VMs need more memory than locally available, the second phase begins. The PM broadcasts the total amount of memory a VM needs and the amount it can provide to all the PMs in the cluster. PMs that receive this request check their memory and network requirements during the time requested, and if possible, either offer to migrate the VM to themselves, or offer the additionally required memory. In the third phase, the original PM chooses one of these PMs to either migrate the VM (preferable), or attach the VM to the remote PM for additional memory.

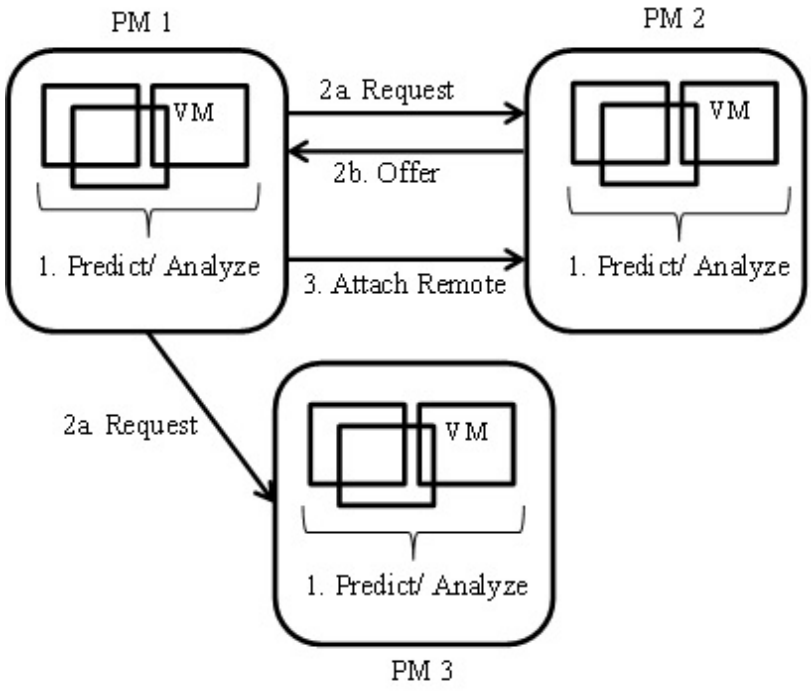


Figure 5.1 ACE-M overview.

The ACE-M framework is therefore composed of three modules or subsystems: Prediction, Decision, and Remote Memory. The algorithm below depicts the pseudo-code for the various modules. Detailed descriptions of the modules follow next.

Prediction Module:

for each VM **do**

$VM_{mem} \leftarrow$ Memory Prediction for t mins

$VM_{net} \leftarrow$ Network Prediction for t mins

end for

Calculate $PM_{mem}, PM_{net}, PM_{blk}$

Decision Module A (Requesting PM):

for each VM **do**

//Enough local memory available?

if $PM_{mem} < VM_{mem}$ **then**

broadcast request

end if

end for

if PMx offered to migrate **then**

migrate VM to PMx

else

if PMx offered to provide **then**

call Remote Memory Module(VM, provider)

end if

end if

Remote Memory Module:

Create nbd RAMDISK on provider

Attach nbd as swap device to VM

Remove other swap devices for VM

Decision Module B (Possible Provider PM):

//Enough local memory on *PMx* available?

if $PMx_{mem} > VM_{mem}$ **then**

Offer to Migrate *VM* to *PMx*

end if

//Enough resources to provide remote memory?

$VM_{memreq} \leftarrow VM_{mem} - HOST_{mem}$

$VM_{netreq} \leftarrow VM_{netreq} + VM_{rnr}$

if $PMx_{mem} > VM_{memreq}$ and $PMx_{net} > VM_{netreq}$ **then**

    Offer to provide remote memory

end if

5.2 System Design

A prototype of ACE-M was built by modifying Linux Kernel 3.16.6 and using KVM and QEMU. This section shall take a closer look at the design specifications for each of the three modules and especially, the modifications required in the Linux Kernel.

5.2.1 Prediction Module

The aim of the Prediction Module is to periodically collect statistics for memory and network. It then uses this history and predicts the future value for both resource usages for the VMs for time t. The time t depends on the accuracy of the prediction mechanism used. The average values over t are taken as the required usage for each resource, for each VM. Since all the predictions are made at a per VM level, the calculated statistics are not marred by external factors like network usage due to remote memory accesses. This is important since network usage due to remote memory is one of the parameters that needs to be optimized by ACE-M and hence should not affect any decisions. Additionally, the future resource usages for each PM are calculated by adding up the usages of all the VMs residing in that PM as well as the required usages for any VMs the PM might be providing for. While any suitable existing prediction mechanism like Markov Chains, Bayesian, Histograms, etc. can be used, ACE-M uses a prediction mechanism based on Chaos Theory. Details of Chaos Theory based predictions for VMs are the same as described in Chapter 3.

Once the future values for the resources have been predicted, the averages of these values are provided to the Decision Module.

5.2.2 Decision Module

The Decision Module consists of two parts (A and B), one from the point of view of the PM requesting additional memory, and the other from the point of view the PM analyzing this request.

Based on the predictions received from the Prediction Module, a PM needs to check a number of scenarios. First it checks for each VM's memory and network prediction, if in the next t minutes, enough local memory and network bandwidth are available. If it is, nothing needs to be done for that VM. In case that is not possible, the PM broadcasts a message to all the PMs in the cluster, specifying the amount of memory the VM needs, the amount of memory that the PM itself can provide, and the duration t for which the additional memory is required. It then waits for offers from other PMs.

A PM that receives the broadcast request (PMx) has to make a decision whether it can fully or partially support the VM. It first checks based on its own predictions for t, if enough local memory and network is available to accommodate the VM for the next t minutes. If it can, it immediately sends a migration offer to the requesting PM. In case this is not possible, the PMx checks to see if it can provide the additional memory required by the VM (i.e. total memory VM needs less memory available on original PM) for the next t minutes. At this point PMx also checks to see if it will have enough network bandwidth during the next t minutes to support the additional network overhead of remote memory. To account for this, every VM has an associated 'remote network requirement' (VM_{rnr}). The VM_{rnr} value is maintained by the host PM and dynamically keeps track of the average network usage per MB of remote memory used by the VM. It starts of with a value of 1 (implying it requires 1 MBps per remote MB) and as the VM actually uses remote memory,

the value is updated. If *PMx* can satisfy the required conditions, *PMx* sends a provider offer to the original VM. Otherwise, *PMx* does not need to respond.

When the original PM receives the offers from other PMs, it needs to choose an offer to take up. If one of the PMs has offered to migrate the VM, this migration is performed, and that PM's future memory values are updated accordingly to incorporate the new VM. If this is not possible, the PM chooses one of the PMs that have offered to provide remote memory and calls the Remote Memory Module for the VM and the provider PM. The provider PM's future memory and network usages are updated accordingly. Finally, if the PM does not receive any offer from the other PMs by the time the memory is required, it falls back to disk swapping for that VM.

5.2.3 Remote Memory Module

ACE-M utilizes the swap functionality of the kernel along with Network Block Devices (NBD) to implement its Remote Memory Subsystem. In order to overcome the lack of granularity when using swap, some modifications are required in the PM's kernel code. Specifically, the get_swap_page function of the Linux kernel where swap device priorities are checked is modified. Instead of choosing a swap device with the highest priority, the modified code chooses a swap device with priority equal to current processes' pid. This implies that multiple swap devices can be attached to the kernel, and each process can associate one or more swap devices with itself.

Figure 5.2 depicts the Remote Memory Subsystem used in ACE-M. Each VM, which is in essence a process, has an associated swap device. This swap device could be a Network Block Device (NBD), or the HDD. In case of an NBD, the NBD corresponds to a RAMDisk in a remote machine. This ensures that all data stored in the NBD is stored in the physical memory of the remote machine. To associate a device with a VM, the device is inserted in the VM's host PM as a swap device with PRIORITY=VM.PID (swapon -p PID NBD).

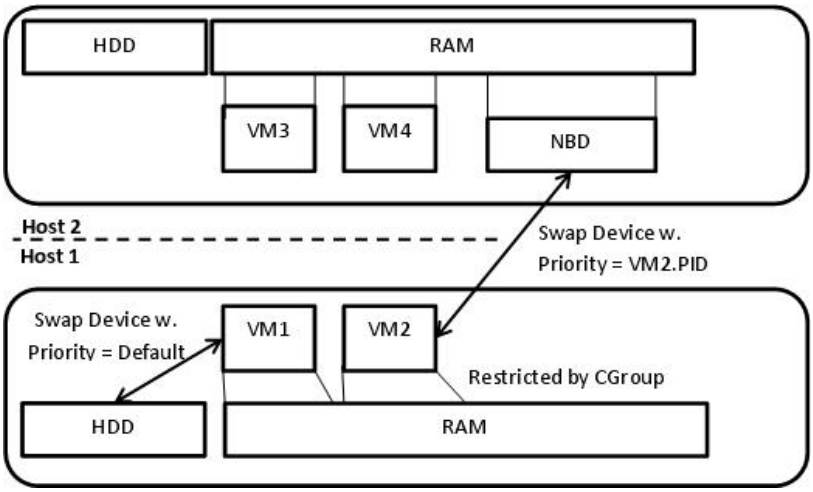


Figure 5.2 The remote memory subsystem.

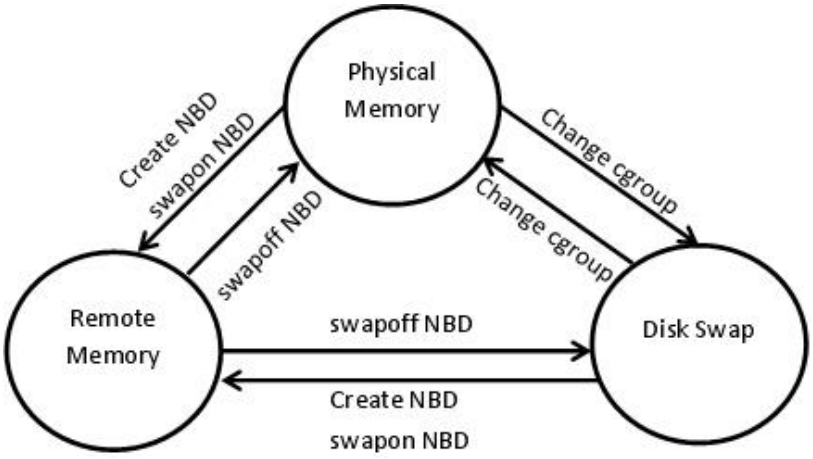


Figure 5.3 Remote Memory - Transition between different modes.

When a VM needs to swap out memory, the modified kernel code checks for a swap device with PRIORITY=VM.PID. If no such device exists, swapping is done to the default swap device (HDD).

Cgroups are used to restrict the physical memory available to a VM, if required. In combination, these two settings allow a VM to access a certain amount of memory from physical memory and the rest from either a remote physical memory, or disk.

The various transitions between different swap modes are shown in Figure 5.3 and discussed below.

Physical Memory to Remote Memory: To transition a VM currently accessing all physical memory to remote memory is straightforward. A new NBD is created on a remote machine. It is associated to the VM as explained above. A new cgroup is created to restrict local physical memory to the desired amount. The VM is added to the cgroup.

Physical Memory to Disk Swap: To transition a VM currently accessing all physical memory to disk swap is similar. No new swap device is created. A new cgroup is created to restrict local physical memory to the desired amount. The VM is added to the cgroup.

Remote Memory to Disk Swap: NBD swap device is disabled with swapoff

Remote Memory to Physical Memory: A new cgroup is created to give the required physical memory access to the VM. The NBD remote swap is then disabled using swapoff.

Disk Swap to Remote Memory: A new NBD is created on a remote machine. It is associated to the VM as explained above. A new cgroup is created to restrict local physical memory to the desired amount. The VM is added to the cgroup.

5.3 Experimental Results and Evaluations

To test the efficacy of ACE-M, a test bed cluster was setup with 16 PMs, each with 4 GB RAM, i-3 processors and 1 Gbps network. Each PM ran Ubuntu 13.10. KVM with QEMU 1.0.0 and libvirt 0.9.8 was used as the hypervisor. A total of 64 VMs were instantiated in this cluster implying 4 VMs per PM. Each VM was created with 7 GB RAM and ran a flavor of Ubuntu Server 12.04. The cluster included both Hard Disk Drives (HDD) and Solid State Drives (SSD) for swapping comparisons.

Experiments were performed to test the framework on two different aspects. First, the remote memory subsystem employed by ACE-M was compared against HDD swapping, SSD swapping, and an implementation of Hecatonchire. Second, the framework as a whole was evaluated for various real world memory intensive and network intensive loads against making no predictions, and predicting memory only.

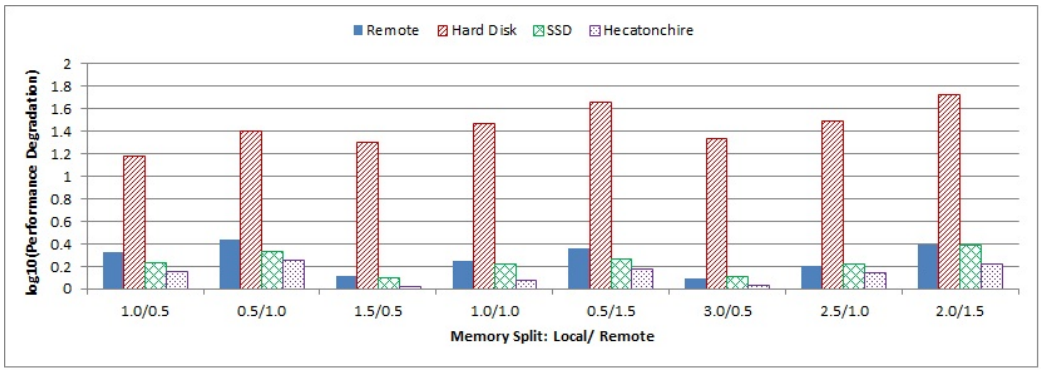


Figure 5.4 Performance degradation in log scale - Intel LINPACK.

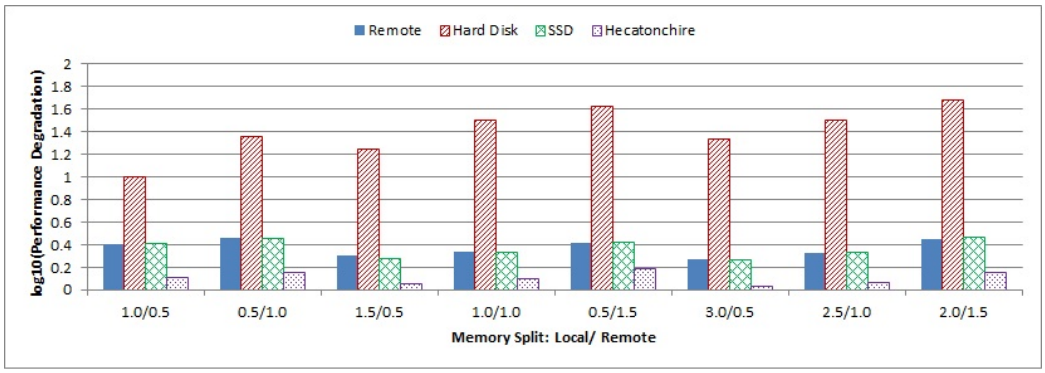


Figure 5.5 Performance degradation in log scale - NASPB-IS.

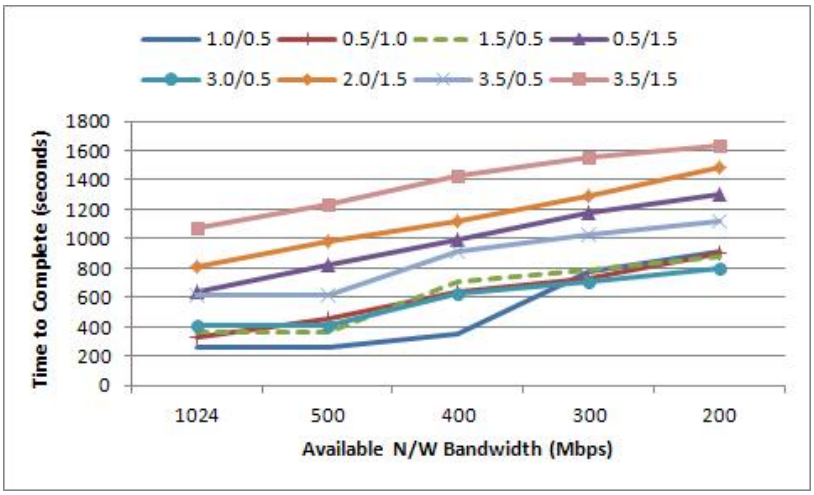
Table 5.1 Comparison of Time to Complete for Different Schemes with Varying Memory Splits - Intel LINPACK

Total Memory Needed (GB)	Base Time (s)	Memory Split Local/ Remote	Time to Complete (seconds)				
			ACE-M Remote	VR Swap	HDD	SSD	Hecatonchire
1.5	122	1.0/0.5	259	247	1864	211	176
		0.5/1.0	331	305	3129	264	220
2.0	280	1.5/0.5	364	322	5583	351	293
		1.0/1.0	500	448	8215	469	333
		0.5/1.5	634	579	12864	520	422
3.5	327	3.0/0.5	407	372	7174	421	351
		2.5/1.0	530	495	10100	549	458
		2.0/1.5	813	762	17290	803	542
4	431	3.5/0.5	718	693	9017	712	NA
4.5	573	3.5/1.0	982	824	29574	965	NA
5	658	3.5/1.5	1275	1189	NA	1210	NA

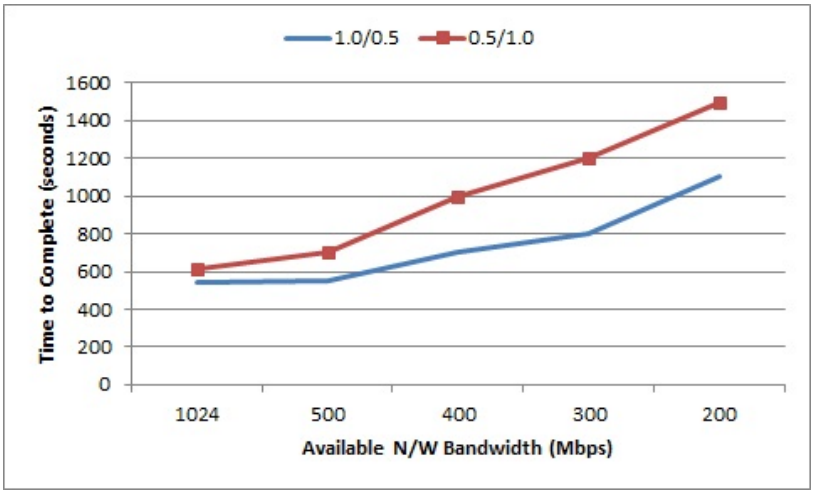
5.3.1 Remote Memory Module Evaluation

In the first set of experiments, ACE-M's remote memory subsystem is compared against HDD and SSD swap schemes as well as Hecatonchire with soft-iWARP [95]. To analyze any overheads due to the modified priority code in the kernel, swap comparisons are also made against 'vanilla remote swap' (VR Swap), which is remote swap without per-VM granularity. This provides a reference for the performance overhead involved in remote memory accessing. Further, the impact of available network bandwidth is evaluated for ACE-M's remote memory subsystem.

Each VM in this evaluation was started with 7 GB of RAM. The Intel LINPACK [93] and NASA Parallel Benchmark Integer Sort (NASPB-IS) [94] were run inside the VMs with varying amounts of memory requirements - 1.5, 2.0, 3.5, 4.0, 4.5, and 5 GB. The base case is obtained by ensuring enough local physical memory to meet the VM requirements. For the other cases, the locally available physical memory was restricted using cgroups, thus forcing the kernel to perform swaps.



(a) Intel LINPACK.



(b) NASPB-IS.

Figure 5.6 Time to complete vs available bandwidth.

Table 5.2 Comparison of Time to Complete for Different Schemes with Varying Memory Splits - NASPB-IS

Total Memory Needed (GB)	Base Time (s)	Memory Split Local/ Remote	Time to Complete (seconds)				
			ACE-M Remote	VR Swap	HDD	SSD	Hecatonchire
1.5	210	1.0/0.5	540	528	2088	548	271
		0.5/1.0	613	572	4780	601	301
2.0	330	1.5/0.5	659	603	5827	632	372
		1.0/1.0	723	651	10462	716	416
		0.5/1.5	851	790	14116	872	508
3.5	424	3.0/0.5	795	769	9176	788	452
		2.5/1.0	902	885	13634	923	492
		2.0/1.5	1204	1134	20523	1230	603
4	576	3.5/0.5	1501	1455	11235	1494	NA
4.5	781	3.5/1.0	1975	1843	35794	1867	NA
5	1018	3.5/1.5	2789	2566	NA	2732	NA

Table 5.1 shows the comparison of all five schemes for different memory requirements in terms of the time required in seconds to complete one run of the Intel LINPACK benchmark. Similarly, Table 5.2 shows the comparisons for the NASPB-IS. Within each memory requirement the various schemes were tested under different restrictions of locally available memory (denoted by the left side of Memory Split). The values marked 'NA' for Hecatonchire failed to be instantiated in the prototype available. For HDD, the values marked 'NA' did not complete within any reasonable time (10+ hours). Clearly, compared to the vanilla remote swap, the kernel additions to ACE-M's remote memory subsystem add some minor overhead.

Figures 5.4 and 5.5 show the Performance Degradation Factor (DF) to complete one NASPB-IS and LINPACK benchmark for HDD swap, SSD swap, ACE-M's Remote Memory, and Hecatonchire, respectively, plotted in the log scale for easy comparison. The DF is defined as

$$DF = Actualtime/Basetime \tag{5.1}$$

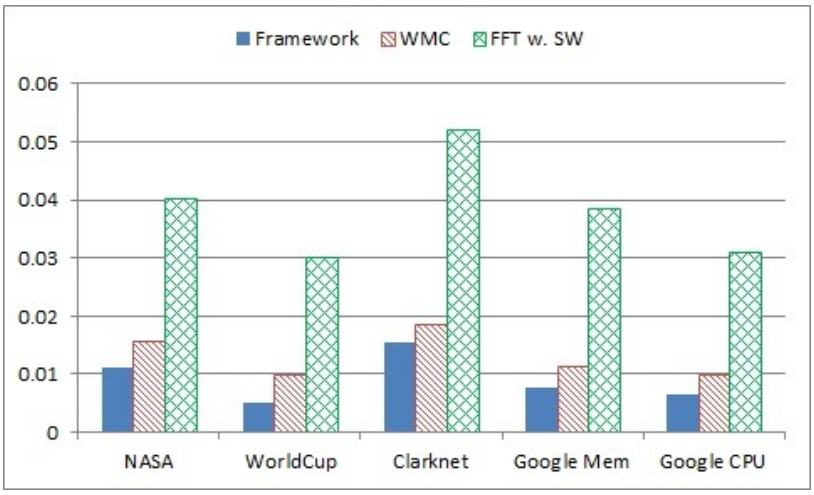


Figure 5.7 MSE comparison for Chaos Theory vs Markov Chains predictions.

It is evident that any method used for remote memory underperforms compared to all memory being available in physical memory. On average ACE-M's remote memory subsystem shows a DF of 4 as opposed to swapping to HDD which has an average DF of 30. It can be seen that ACE-M's Remote Memory outperforms HDD swapping by an order of magnitude. ACE-M is less efficient than Hecatonchire (average DF 2) as expected, but that is the trade-off for applying no modifications to the hypervisor. Similarly, SSD swapping performs almost the same as remote memory in general. However, SSDs involve the installation of new hardware in the cluster which may not be always feasible. As the amount of available physical memory reduces and remote memory used increases, performance degradation increases. This relationship reinforces the hypothesis that remote memory is necessary to make large memory processing of VMs practical, which otherwise would default to HDD swapping.

To test the network bandwidth impact on performance via remote memory, the available bandwidth on the network link between the provider and requester was systematically and actually reduced, again for various splits of local and remote memory. For ACE-M's Remote Memory, Figures 5.6a and 5.6b show the increase in time to complete NASPB-IS and LINPACK as the available network bandwidth reduces for various splits of

local and remote memory. From the graph it is clear that network bandwidth is one of the bottlenecks that degrades performance of the VM using remote memory.

The observation implies that it is necessary to consider both memory and network utilization before allocating remote memory to a VM.

5.3.2 Prediction Module Evaluation

ACE-M's prediction module is built on the Chaos Theory based mechanisms described in Chapter 3 [64]. Predictions on memory and network are made every 10 minutes for the next 10 minutes. As demonstrated, the prediction mechanism predicts real world load traces (Google Cluster Traces [50], and NASA [52], Clarknet [87], and Worldcup [53] websites) with an accuracy of about 0.005 Mean Square Error (MSE). Figure 5.7 shows a comparison of MSE for Chaos Theory based predictions against Markov Chains with Wavelets (MWC) and Fourier Transform with Sliding Windows (FFT w. SW) for a number of different workloads. Chaos theory based predictions outperform MWC that has an average MSE of 0.01 as well as FFT w. SW that has an average MSE of 0.03.

5.3.3 Decision Module Evaluation

A set of experiments is performed to demonstrate the behavior of the cluster under a mix of various memory intensive and network intensive, real world loads. To demonstrate network usage the load patterns were based on trace-data collected from NASA's website [52], Clarknet Data [87], and the Worldcup website [53] over maximum 2 months. For the memory usage Google Cluster Data Trace [50] over 29 days was used. The data was scaled-down so that it fit within 2 weeks.

For the Google Cluster Data, memory traces for 150 random machines were separated from the total 12k machine traces available. For each machine, the sum of the usages of all the tasks at each time instant was used as the load value. Since the Google traces do not specify the actual amounts of memory being used, the maximum usage value

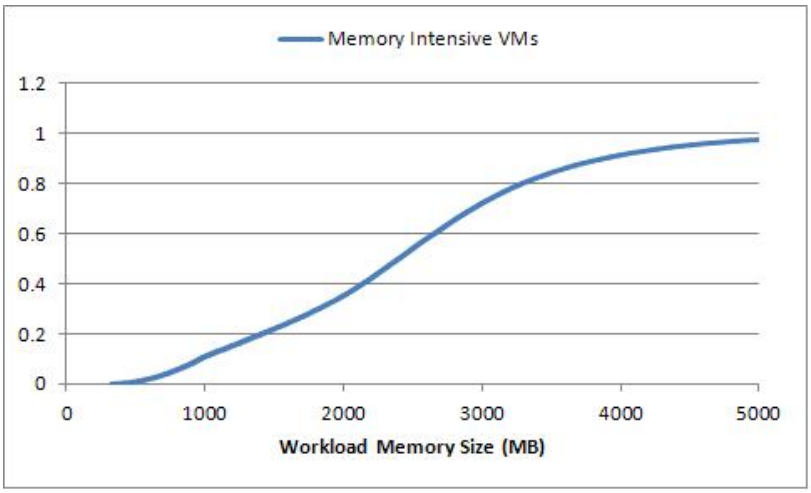
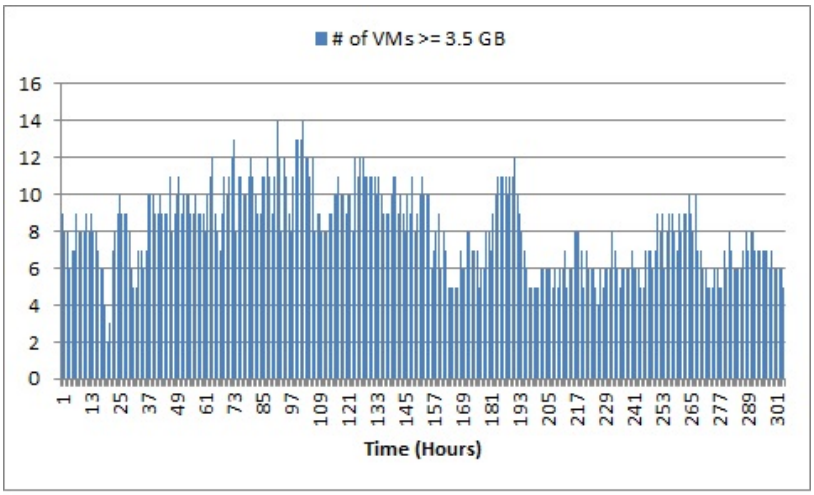


Figure 5.8 Cumulative distribution of workload memory size.

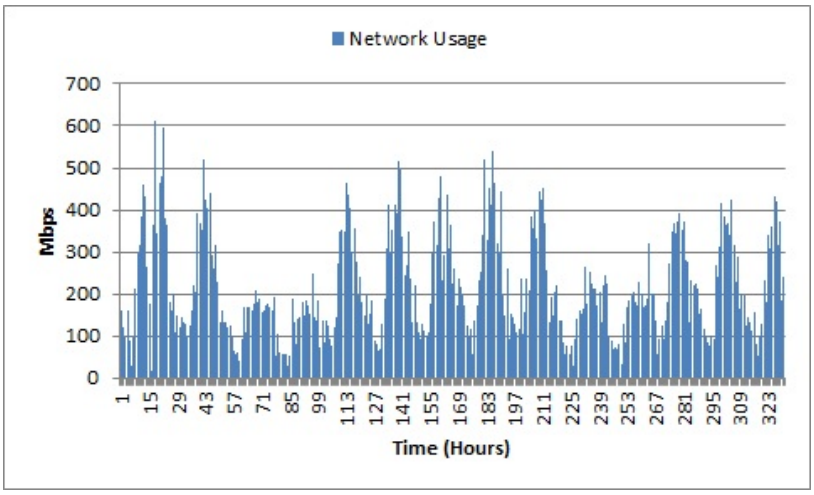
was chosen to be 5 GB and all other values were scaled accordingly. Since all analysis done is relative, without loss of generality, this value was chosen according to the scale of the test-bed cluster.

The NASA, Clarknet and Worldcup traces with sampling rates of 1 minute were used to drive RUBiS web servers (PHP) [88]. Similarly, the Google Data Traces with sampling rates of 5 minutes were used to drive NASPB-IS and Intel LINPACK.

Half of the VMs (32) were driven by the memory intensive Google traces, while the other half were driven by the network intensive Clarknet, NASA website, and Worldcup website traces. For perspective on the memory size of workloads at each instant during the entire experiment, Figure 5.8 shows the average cumulative distribution of all the workload memory sizes in MB for the memory intensive VMs. It can be seen that about 50% of the times, workloads required 2.5 GB or more memory to complete. Since each PM has 4 GB of RAM installed, it should be noted that workloads with sizes more than 3.5 GB (about 20% of the times) definitely required remote memory usage.



(a) Google Trace - VMs out of 64 with workload $>$ 3.5 GB.



(b) RUBiS - Avg. network usage for single VM (Mbps).

Figure 5.9 Base case.

To provide a base reference, the cluster was setup and the experiment performed for each VM one at a time with added RAM to ensure enough local memory availability. Figure 5.9a shows the average number of Google Trace VMs per hour that had workloads requiring more than 3.5 GB of memory over the run of the entire experiment. About 6 VMs every hour belonged to this category. These workloads wouldn't have completed without swapping to disk or remote memory. On an average during the experiment, at any given time the total memory required in the cluster for all 64 VMs was about 60 GB. Figure

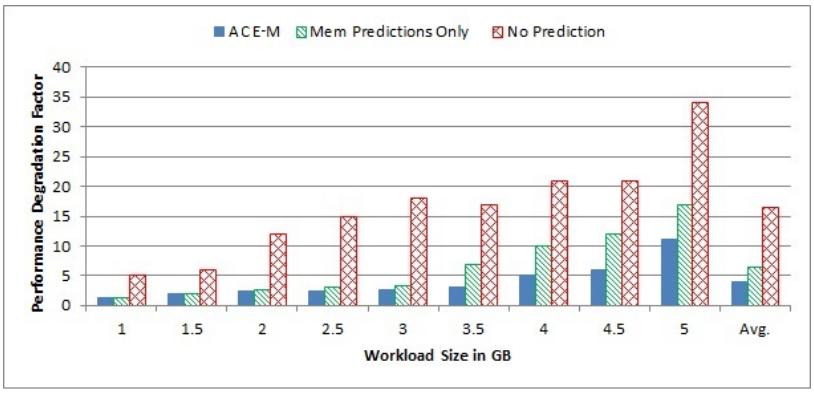


Figure 5.10 Performance Degradation Factor comparison for various workload sizes.

5.9b shows the average hourly network usage in Mbps for each of the VMs running the RUBiS servers over the entire run. The average response time for the RUBiS servers, given sufficient bandwidth was **100 ms**.

ACE-M was then setup and run for the entire cluster over 2 weeks, with the data traces and benchmarking tools mentioned before. In the first run, no predictions were made and decisions were based on the average of memory usage in the last 10 minutes.

The experiment was then repeated with the exact same setup again. This time, predictions were made but decisions were taken based only on the memory predictions taken 10 minutes into the future. Finally, the experiment was repeated again with full ACE-M and decisions were based both on the memory and network usage predictions.

Figure 5.10 shows the average DF to complete the benchmarks separated by the size of the workloads. It is clear that during the entire experiment run ACE-M outperforms making no predictions and making memory predictions only. This performance difference is amplified for instances of larger workload sizes. On an average ACE-M demonstrates DF of 4 as opposed to DF of 16 when predictions are not used which is a 4 times improvement or a 75% reduction in DF.

Figure 5.11 shows the average response times for the RUBiS servers. ACE-M has a 40% lower average response time (150 ms) compared to the other two experiments (250 ms without prediction and with memory predictions only).

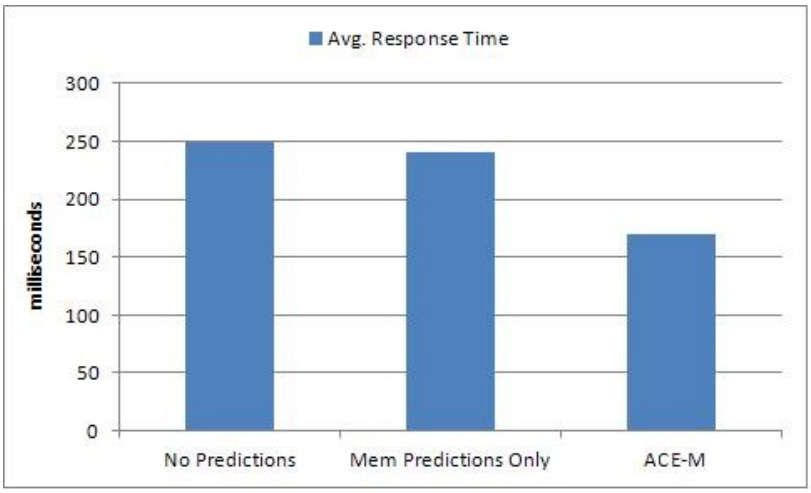


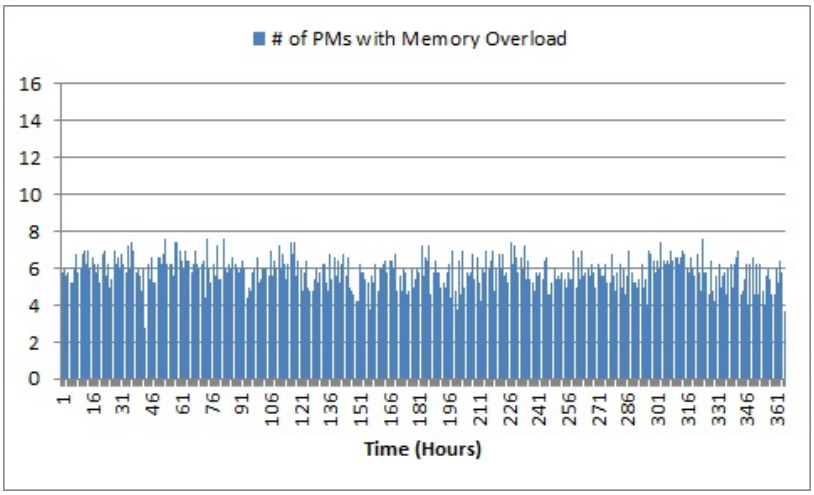
Figure 5.11 Average response time comparison.

5.4 Discussions

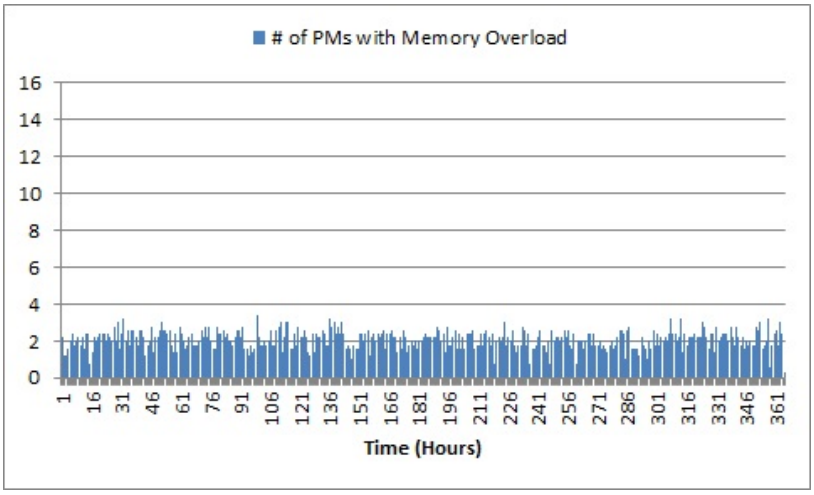
The experiments in the previous section demonstrated that by predicting memory and network usage of the VMs and by utilizing this information to make remote memory allocation decisions, the performance degradation of the participating VM as well as other VMs in the cluster can be controlled.

A comparison of Figures 5.9 and 5.10 shows an improvement in performance by simply allowing remote memory access. Specifically, all the VMs in 5.9a are load runs that would not even complete without remote or disk swapping. In contrast, predicting at least memory loads and utilizing them, steeply improves the performance while also allowing all loads to complete. However, since network usage predictions are not used, the response times for the RUBiS servers still peak often as seen in Figure 5.11. Finally, ACE-M, which utilizes both memory and network predictions demonstrates stable performance as well as improves the response times for the RUBiS servers.

An interesting observation from Figure 5.10 is the impact of network predictions on smaller workload sizes. Since most of the workloads requiring less than 2 GB of memory could be accommodated completely on a single PM somewhere in the cluster, they typically did not require remote memory. They were simply migrated to a relatively free PM. Hence,



(a) No predictions.



(b) With predictions.

Figure 5.12 Google trace - PMs with memory overload over time.

their behavior with or without network predictions is similar. On the contrary, as the workload size increases beyond 2 GB, remote memory requirements become necessary, and predicting both network and memory outperforms predicting memory alone.

The results obtained can be explained more clearly by Figures 5.12a and 5.12b. These figures show the number of PMs over the experiment run that were overloaded on memory. It can be seen that without any predictions being made, on an average 6 PMs encounter memory overloads. As opposed to this, using ACE-M with predictions keeps the average number of PMs with memory overloads down to about 2. This information in itself still does not provide a complete picture since it does not depict the amount of memory overload for each overloaded PM. For the experiments, the amount of memory overload without prediction is about 400 MB on an average. Compared to this, memory overloads with prediction stay at about 100 MB on an average. Thus, using predictions not only reduces the average number of PMs experiencing memory overload, but also reduces the extent of each PM's memory overload.

Figure 5.13 summarizes the improvements achieved when using ACE-M. Without any predictions, the performance degradation factor for the 99th percentile is above 30. With only memory predictions, a sudden steep improvement is seen with a maximum performance degradation factor of about 18. With ACE-M, that also predicts network usage, there is further improvement as the performance degradation factor is 10 at most. Since hard-disk swap contributes most to the performance degradation, ensuring available remote memory using prediction is most important. The impact of network overload is comparatively less than disk-swap. Thus, predictions on network and ensuring sufficient network bandwidth offers some improvement.

It may be argued that current networks offer substantially high bandwidths up to 40 Gbps. However, as mentioned before, it is not the theoretical maximum network bandwidth that is important, but rather the currently available network bandwidth. Networks with high bandwidths may still overload due to network-based applications and VMs running in the

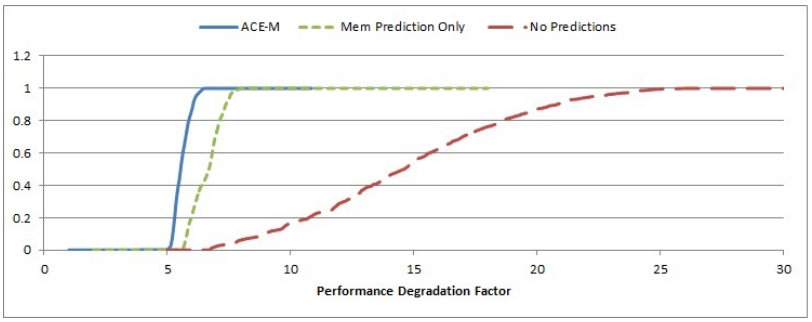


Figure 5.13 Cumulative distribution of performance with and without prediction.

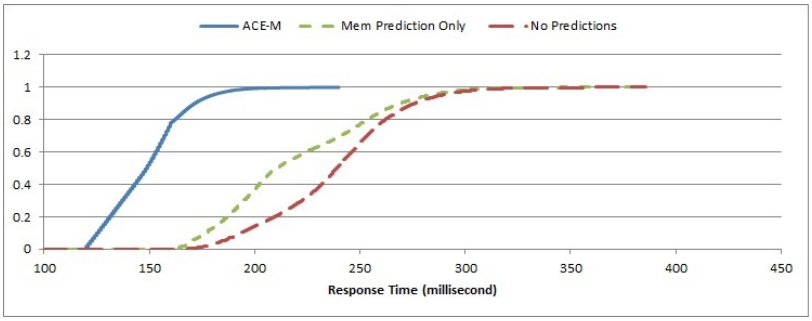


Figure 5.14 Cumulative distribution of response times with and without prediction.

cluster, as shown in the experiments. ACE-M also makes no assumption of the network bandwidth and works similarly with any network speeds.

Figure 5.14 demonstrates the degradation in response times for the various schemes. Again, utilizing both memory and network predictions with ACE-M is considerably more efficient than no predictions or memory predictions only. A slight improvement in response times can be observed when making memory predictions only. This can be attributed to VMs with smaller size workloads being correctly migrated to PMs with enough free memory, thus not requiring any remote memory. Overall, the graph shows that for the 99th percentile ACE-M achieves a faster response time of about 190 ms as opposed to 320 ms when no predictions are made.

5.5 Summary

In this chapter, we presented the ACE-M framework that allows VMs to dynamically utilize large amounts of memory across the entire cluster with minimum degradation of

the cluster. Using this remote memory enables the VMs to run applications requiring more memory than physically available on a single host, where such applications would have been impractical and taken a prohibitively long time to compute otherwise.

It was demonstrated that choosing suitable remote memory configurations in the cluster is not trivial and requires preemptive analysis of the cluster's memory and network behavior, which is missing in current remote memory solutions. ACE-M uses Chaos Theory based prediction mechanisms to preempt memory and network requirements, and chooses suitable PMs in the cluster to provide any additional memory needed. This intelligent configuration minimizes the performance degradation of the cluster from the point of view of both memory and network overloads. Through the framework we also introduced a new software based remote memory subsystem that is hypervisor agnostic and allows a per-VM granularity.

Extensive cluster-wide experiments with real world data traces showed that simply predicting memory usage can reduce memory performance degradation of the cluster by about 55%. ACE-M further reduces the memory performance degradation by about 75% and achieves a 40% lower network response time by utilizing both memory and network predictions.

Further experiments also indicated that ACE-M's built-in remote memory subsystem is comparable to other software based remote memory mechanisms while mitigating problems associated with them such as granularity and dynamic provider switching. The subsystem was shown to perform up to 7 times better than traditional disk swapping.

While for the experimental analysis in this thesis, ACE-M was tested with KVM, the framework and the associated remote memory subsystem can be applied without any modifications to other hypervisors such as Xen. As a future addition to ACE-M, block I/O usage prediction can be added to take a more informed decision between remote memory and disk swapping.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In the preceding chapters, various advantages, challenges, and promising features of clouds were discussed. It was observed that innovations in virtualization technologies lead to new features and possibilities in the cloud. Three such enabling technologies were identified in particular - load prediction and consolidation, live migration, and memory management. Prototypes were built for the proposed methods and their efficacy was demonstrated through real world data-center load traces on a test cluster.

A new VM resource load prediction mechanism based on Chaos Theory was introduced that makes predictions of server workloads with mean square errors as low as 0.004. Based on these predictions, VMs are relocated to different PMs in the cluster in an attempt to conserve energy. Using this framework energy savings of upto 40% were demonstrated with low SLO violations of 3%. This framework was compared to a currently popular prediction tool based on the Fast Fourier Transform, as well as an implementation using Wavelets with Markov Chains.

A new distributed approach for network-efficient, amortized live migration of VMs was presented. This was designed to attain the goals of network bandwidth conservation and fast live migration times to reduce possible VM performance degradation during live migration. As a result of this protocol network bandwidth savings of upto 40% were observed during live migration. Also, migration times were seen to reduce by 45%.

Finally, a novel memory management protocol was discussed that enables VMs to share and utilize all the memory available in the cluster, rather than being restricted to the memory available at the host PM. Through experiments it was demonstrated that by predicting memory usage of the VMs, memory performance degradation of the cluster

could be reduced by about 55%. Further reductions by about 75% and a 40% lower network response time can be achieved by utilizing both memory and network predictions.

6.2 Contributions

To summarize, the main contributions of this thesis are:

1. Chaos Theory based VM load prediction mechanism
2. VM consolidation framework to repurpose lightly used machines
3. Distributed live migration of VMs for network efficiency
4. Remote memory based management framework for practical large memory computations

6.3 Future Work

Future work will include applying a Game Theory based approach to the memory sharing and pooling protocol. Several kernel level modifications are being currently tested to ensure efficient remote swapping. Experimental evaluations will be performed to determine any improvements over Chaos Theory.

Further, a comparative study of various mathematical transforms like the Wigner Transform to the problem of VM load prediction will be performed. Additionally, all the proposed innovations will be incorporated into KVM.

BIBLIOGRAPHY

[1] S. Ullah and Z. Xuefeng, "Cloud computing: a prologue," *arXiv Preprint arXiv:1304.2981*, 2013.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," *NIST special publication*, vol. 800, no. 145, p. 7, 2011.

[3] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[4] K. Scott, "The basics of cloud computing," *Evraz Inc. White Paper*, 2010.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[7] R. Maggiani, "Cloud computing is changing how we communicate," in *International Professional communication conference (IPCC)*. Honolulu, HI, USA: IEEE, 2009, pp. 1–4.

[8] H. G. Miller and J. Veiga, "Cloud computing: Will commodity services benefit users long term?" *IEEE IT professional*, vol. 11, no. 6, pp. 57–59, 2009.

[9] D. Catteddu, *Cloud Computing: Benefits, risks and recommendations for information security*. Springer, 2010.

[10] Z. Zhang and X. Zhang, "Realization of open cloud computing federation based on mobile agent," in *International Conference on Intelligent Computing and Intelligent Systems (ICIS)*, vol. 3. Phoenix, AZ, USA: IEEE, 2009, pp. 642–646.

[11] M. A. Vouk, "Cloud computing - issues, research and implementations," *Journal of Computing and Information Technology*, vol. 16, no. 4, pp. 235–246, 2004.

[12] V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D. Epema, and T. Fahringer, "Efficient management of data center resources for massively multiplayer online games," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Austin, TX, USA: IEEE, 2008, pp. 1–12.

[13] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, 2012.

[14] M. Mihailescu and Y. M. Teo, "Dynamic resource pricing on federated clouds," in *10th International Conference on Cluster, Cloud and Grid Computing (CCGrid)*. Melbourne, Australia: IEEE, 2010, pp. 513–517.

[15] D. Agrawal, S. Das, and A. El Abbadi, "Big data and cloud computing: current state and future opportunities," in *14th International Conference on Extending Database Technology*. Uppsala, Sweden: ACM, 2011, pp. 530–533.

[16] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in *2nd International Conference on Cloud Computing Technology and Science (CloudCom)*. Indianapolis, IN, USA: IEEE, 2010, pp. 17–24.

[17] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for mapreduce in a cloud," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Seattle, WA, USA: ACM, 2011, p. 58.

[18] Y. Geng, S. Chen, Y. Wu, R. Wu, G. Yang, and W. Zheng, "Location-aware mapreduce in virtual cloud," in *International Conference on Parallel Processing (ICPP)*. IEEE, 2011, pp. 275–284.

[19] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour, "The turtles project: Design and implementation of nested virtualization." in *9th Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 10. Vancouver, Canada: USENIX, 2010, pp. 423–436.

[20] F. Wuhib, R. Stadler, and M. Spreitzer, "A gossip protocol for dynamic resource management in large cloud environments," *IEEE Transactions on Network and Service Management*, vol. 9, no. 2, pp. 213–225, 2012.

[21] C. Humphries and P. Ruth, "Towards power efficient consolidation and distribution of virtual machines," in *48th Annual Southeast Regional Conference*. Oxford, MS, USA: ACM, 2010, pp. 75:1–75:6.

[22] F. Y.-K. Oh, H. S. Kim, H. Eom, and H. Y. Yeom, "Enabling consolidation and scaling down to provide power management for cloud computing," in *3rd Conference on Hot Topics in Cloud Computing (HotCloud)*. Berkeley, CA, USA: USENIX, 2011, pp. 14–14.

[23] J. Wang, D. Chi, J. Wu, and H.-y. Lu, "Chaotic time series method combined with particle swarm optimization and trend adjustment for electricity demand forecasting," *Expert Systems with Applications*, vol. 38, no. 7, pp. 8419–8429, 2011.

[24] A. Basharat and M. Shah, "Time series prediction by chaotic modeling of nonlinear dynamical systems," in *12th International Conference on Computer Vision*. Kyoto, Japan: IEEE, 2009, pp. 1941–1948.

[25] I. Agbon and J. Araque, "Predicting oil and gas spot prices using chaos time series analysis and fuzzy neural network model," in *Hydrocarbon Economics and Evaluation Symposium*. Houston, TX, USA: SPE, 2003.

[26] T. A. Denton, G. A. Diamond, R. H. Helfant, S. Khan, and H. Karagueuzian, "Fascinating rhythm: a primer on chaos theory and its application to cardiology," *American Heart Journal*, vol. 120, no. 6, pp. 1419–1440, 1990.

[27] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *8th International Workshop on Middleware for Grids, Clouds and e-Science (MGC)*. Bangalore, India: ACM, 2010, pp. 4:1–4:6.

[28] C. Canali and R. Lancellotti, "Automatic virtual machine clustering based on bhattacharyya distance for multi-cloud systems," in *International Workshop on Multi-cloud Applications and Federated Clouds (MultiCloud)*. Prague, Czech Republic: ACM, 2013, pp. 45–52.

[29] S. Kang, S.-g. Kim, H. Eom, and H. Y. Yeom, "Towards workload-aware virtual machine consolidation on cloud platforms," in *6th International Conference on Ubiquitous Information Management and Communication (ICUIMC)*. Kuala Lumpur, Malaysia: ACM, 2012, pp. 45:1–45:4.

[30] J. Kim, M. Ruggiero, D. Atienza, and M. Lederberger, "Correlation-aware virtual machine allocation for energy-efficient datacenters," in *Conference on Design, Automation and Test in Europe (DATE)*. San Jose, CA, USA: EDA Consortium, 2013, pp. 1345–1350.

[31] H. Lv, Y. Dong, J. Duan, and K. Tian, "Virtualization challenges: a view from server consolidation perspective," in *8th SIGPLAN/SIGOPS conference on Virtual Execution Environments (VEE)*. London, UK: ACM, 2012, pp. 15–26.

[32] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Auto-scaling techniques for elastic applications in cloud environments," Department of Computer Architecture and Technology, UPV/EHU, Tech. Rep., 2012.

[33] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, "Self-adaptive workload classification and forecasting for proactive resource provisioning," in *4th International Conference on Performance Engineering (ICPE)*. Prague, Czech Republic: ACM, 2013, pp. 187–198.

[34] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow," in *7th International Conference on Autonomic and Autonomous Systems (ICAS)*. Venice, Italy: XPS, 2011, pp. 67–74.

[35] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: a reinforcement learning approach to virtual machines auto-configuration," in *6th International Conference on Autonomic Computing (ICAC)*. Barcelona, Spain: ACM, 2009, pp. 137–146.

[36] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, 2012.

[37] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *4th European Conference on Computer Systems*. Nuremberg, Germany: ACM, 2009, pp. 13–26.

[38] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A multi-model framework to implement self-managing control systems for qos management," in *6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Honolulu, HI, USA: ACM, 2011, pp. 218–227.

[39] P. Bodık, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," in *Conference on Hot Topics in Cloud Computing (HotCloud)*. San Diego, CA, USA: USENIX, 2009, pp. 12–12.

[40] L. Wang, J. Xu, M. Zhao, and J. Fortes, "Adaptive virtual resource management with fuzzy model predictive control," in *8th International Conference on Autonomic Computing (ICAC)*. Karlsruhe, Germany: ACM, 2011, pp. 191–192.

[41] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Network Operations and Management Symposium (NOMS)*. Maui, HI, USA: IEEE, 2012, pp. 204–212.

[42] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," in *Twelfth International Workshop on Quality of Service (IWQoS)*. Montreal, Canada: IEEE, 2004, pp. 57–66.

[43] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Transactions on Autonomous Adaptive Systems*, vol. 3, no. 1, pp. 1:1–1:39, 2008.

[44] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *Fourth International Conference on Autonomic Computing (ICAC)*. Jacksonville, Fl, USA: IEEE, 2007, pp. 27–27.

[45] R. P. Doyle, J. S. Chase, O. Asad, W. Jin, and A. Vahdat, "Model-based resource provisioning in a web service utility," in *4th Conference and Symposium on Internet Technologies and Systems*, vol. 4. Seattle, WA, USA: USENIX, 2003, pp. 5–5.

[46] E. S. Buneci and D. A. Reed, "Analysis of application heartbeats: Learning structural and temporal features in time series data for identification of performance problems," in *Conference on Supercomputing*. Austin, TX, USA: IEEE/ACM, 2008, p. 52.

[47] A. Ali-Eldin, J. Tordsson, E. Elmroth, and M. Kihl, "Workload classification for efficient auto-scaling of cloud resources," Department of Computer Science, Umea University, Umea, Sweden, Tech. Rep., 2013.

[48] Z. Gong, X. Gu, and J. Wilkes, "PRESS: Predictive elastic resource scaling for cloud systems," in *International Conference on Network and Service Management (CNSM)*. ON, Canada: IEEE, 2010, pp. 9–16.

[49] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," in *Quality of Service (IWQoS)*. Monterey, CA: ACM, 2003, pp. 381–398.

[50] J. Wilkes, "More Google cluster data," Google research blog, Nov. 2011, posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[51] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: elastic distributed resource scaling for infrastructure-as-a-service," in *International Conference on Automated Computing (ICAC)*. San Jose, CA, USA: USENIX, 2013.

[52] J. Dumoulin and M. Arlitt, "NASA Web Site HTTP Request Logs http://http://ita.ee.lbl.gov/," 1995, accessed: 08-01-2014.

[53] M. Arlitt and T. Jin, "1998 World Cup Web Site Access Logs available at http://www.acm.org/sigcomm/ita/," August 1998, accessed: 08-01-2014.

[54] C. A. Waldspurger, "Memory resource management in VMware ESX server," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 181–194, 2002.

[55] D. Gupta, S. Lee, and M. Vrable, "Difference engine: Harnessing memory redundancy in virtual machines," *Communications of the ACM*, vol. 53, no. 10, pp. 85–93, 2010.

[56] C.-R. Chang, J.-J. Wu, and P. Liu, "An empirical study on memory sharing of virtual machines for server consolidation," in *9th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*. Busan, Korea: IEEE, 2011, pp. 244–249.

[57] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory buddies: exploiting page sharing for smart colocation in virtualized data centers," in *5th SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*. Washington DC, USA: ACM, 2009, pp. 31–40.

[58] J.-H. Chiang, H.-L. Li, and T.-c. Chiueh, "Introspection-based memory de-duplication and migration," in *9th SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*. Houston, TX, USA: ACM, 2013, pp. 51–62.

[59] U. Deshpande, B. Schlinker, E. Adler, and K. Gopalan, "Gang migration of virtual machines using cluster-wide deduplication," in *13th International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. Delft, Netherlands: IEEE/ACM, 2013, pp. 394–401.

[60] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *20th International Symposium on High Performance Distributed Computing (HPCC)*. San Jose, CA, USA: ACM, 2011, pp. 135–146.

[61] U. Deshpande, U. Kulkarni, and K. Gopalan, "Inter-rack live migration of multiple virtual machines," in *6th International Workshop on Virtualization Technologies in Distributed Computing*. Delft, Netherlands: ACM, 2012, pp. 19–26.

[62] P. Riteau, C. Morin, and T. Priol, "Shrinker: improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing," in *Euro-Par Parallel Processing*. Bordeaux, France: Springer, 2011, pp. 431–442.

[63] L. Cui, J. Li, B. Li, J. Huai, C. Hu, T. Wo, H. Al-Aqrabi, and L. Liu, "VMScatter: migrate virtual machines to many hosts," in *9th SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*. Houston, TX, USA: ACM, 2013, pp. 63–72.

[64] K. Qazi, Y. Li, and A. Sohn, "Workload prediction of virtual machines for harnessing data center resources," in *7th International Conference on Cloud Computing (CLOUD)*. Anchorage, AK, USA: IEEE, 2014.

[65] M. Serrano, S. Petit, J. Sahuquillo, R. Ubal, H. Hassan, and J. Duato, "Page-based memory allocation policies of local and remote memory in cluster computers," in *18th International Conference on Parallel and Distributed Systems (ICPADS)*. Nanyang View, Singapore: IEEE, 2012, pp. 612–619.

[66] R. Ghosh and V. K. Naik, "Biting off safely more than you can chew: Predictive analytics for resource over-commit in iaas cloud," in *5th International Conference on Cloud Computing (CLOUD)*. Honolulu, HI, USA: IEEE, 2012, pp. 25–32.

[67] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: a hybrid PRAM and DRAM main memory system," in *46th Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, 2009, pp. 664–669.

[68] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, 2009.

[69] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *International Conference on Supercomputing*. Tucson, AZ, USA: ACM, 2011, pp. 85–95.

[70] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *18th International Symposium on High Performance Computer Architecture (HPCA)*. New Orleans, LA, USA: IEEE, 2012, pp. 1–12.

[71] S. Liang, R. Noronha, and D. K. Panda, "Swapping to remote memory over infiniband: An approach using a high performance network block device," in *International Cluster Computing (ICC)*. Seoul, Korea: IEEE, 2005, pp. 1–10.

[72] T. Newhall, S. Finney, K. Ganchev, and M. Spiegel, "Nswap: A network swapping module for linux clusters," in *Euro-Par Parallel Processing*. Klagenfurt, Austria: Springer, 2003, pp. 1160–1169.

[73] N. Wang, X. Liu, J. He, J. Han, L. Zhang, and Z. Xu, "Collaborative memory pool in cluster system," in *International Conference on Parallel Processing (ICPP)*. XiAn China: IEEE, 2007, pp. 17–17.

[74] H. Midorikawa, M. Kurokawa, R. Himeno, and M. Sato, "DLM: A distributed large memory system using remote memory swapping over cluster nodes," in *International Conference on Cluster Computing*. Tsukuba, Japan: IEEE, 2008, pp. 268–273.

[75] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Hecatonchire: Enabling multi-host virtual machines by resource aggregation and pooling," Department of Computer Science, Umea University, Umea, Sweden, Tech. Rep., 2014.

[76] A. Samih, R. Wang, C. Maciocco, T.-Y. C. Tai, and Y. Solihin, "A collaborative memory system for high-performance and cost-effective clustered architectures," in *1st Workshop on Architectures and Systems for Big Data*. Galveston Island, TX, USA: ACM, 2011, pp. 4–12.

[77] A. Gupta, E. Ababneh, R. Han, and E. Keller, "Towards elastic operating systems," in *14th Conference on Hot Topics in Operating Systems*. Santa Ana Pueblo, NM, USA: USENIX, 2013, pp. 16–16.

[78] U. Deshpande, B. Wang, S. Haque, M. Hines, and K. Gopalan, "Memx: Virtualization of cluster-wide memory," in *39th International Conference on Parallel Processing (ICPP)*. San Diego, CA, USA: IEEE, 2010, pp. 663–672.

[79] Y. Li and Y. Huang, "TMemCanal: A VM-oblivious dynamic memory optimization scheme for virtual machines in cloud computing," in *10th International Conference on Computer and Information Technology (CIT)*. Bradford, United Kingdom: IEEE, 2010, pp. 179–186.

[80] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, "Overdriver: Handling memory overload in an oversubscribed cloud," *ACM SIGPLAN Notices*, vol. 46, no. 7, pp. 205–216, 2011.

[81] T. Okuda, E. Kawai, and S. Yamaguchi, "A mechanism of flexible memory exchange in cloud computing environments," in *2nd International Conference on Cloud Computing Technology and Science (CloudCom)*. Indianapolis, IN, USA: IEEE, 2010, pp. 75–80.

[82] J. D. Farmer and J. J. Sidorowich, "Predicting chaotic time series," *Physical Review Letters*, vol. 59, no. 8, pp. 845–848, 1987.

[83] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis*. Cambridge University Press, 2003, vol. 7.

[84] A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano, "Determining lyapunov exponents from a time series," *Physica D: Nonlinear Phenomena*, vol. 16, no. 3, pp. 285–317, 1985.

[85] L. Cao, "Practical method for determining the minimum embedding dimension of a scalar time series," *Physica D: Nonlinear Phenomena*, vol. 110, no. 1, pp. 43–50, 1997.

[86] A. M. Fraser and H. L. Swinney, "Independent coordinates for strange attractors from mutual information," *Physical Review A*, vol. 33, no. 2, p. 1134, 1986.

[87] "The IRCache Project http://www.ircache.net/," accessed: 08-01-2014.

[88] "RUBiS Online Auction System http://rubis.ow2.org," accessed: 08-01-2014.

[89] "LAME MP3 Encoder http://lame.sourceforge.net/," accessed: 08-01-2014.

[90] "IOzone Filesystem Benchmark http://www.iozone.org/," accessed: 08-01-2014.

[91] "Netperf http://www.netperf.org/netperf/," accessed: 08-01-2014.

[92] T. Hirofuchi and I. Yamahata, "Yabusame: Postcopy live migration for Qemu/KVM," in *KVM Forum*, Vancouver, Canada, 2011.

[93] "Intel LINPACK https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download," accessed: 08-01-2014.

[94] "NASA Parallel Benchmarks https://www.nas.nasa.gov/publications/npb.html," accessed: 08-01-2014.

[95] "Hecatonchire prototype https://github.com/blopeur," accessed: 08-01-2014.