Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing

Anton Beloglazov

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

February 2013

Department of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing

Anton Beloglazov
Supervisor: Prof. Rajkumar Buyya

Abstract

Cloud computing has revolutionized the information technology industry by enabling elastic on-demand provisioning of computing resources. The proliferation of Cloud computing has resulted in the establishment of large-scale data centers around the world containing thousands of compute nodes. However, Cloud data centers consume enormous amounts of electrical energy resulting in high operating costs and carbon dioxide emissions. In 2010, energy consumption by data centers worldwide was estimated to be between 1.1% and 1.5% of the global electricity use and is expected to grow further.

This thesis presents novel techniques, models, algorithms, and software for distributed dynamic consolidation of Virtual Machines (VMs) in Cloud data centers. The goal is to improve the utilization of computing resources and reduce energy consumption under workload independent quality of service constraints. Dynamic VM consolidation leverages fine-grained fluctuations in the application workloads and continuously reallocates VMs using live migration to minimize the number of active physical nodes. Energy consumption is reduced by dynamically deactivating and reactivating physical nodes to meet the current resource demand. The proposed approach is distributed, scalable, and efficient in managing the energy-performance trade-off. The key contributions are:

1. Competitive analysis of dynamic VM consolidation algorithms and proofs of the competitive ratios of optimal online deterministic algorithms for the formulated single VM migration and dynamic VM consolidation problems.
2. A distributed approach to energy-efficient dynamic VM consolidation and several novel heuristics following the proposed approach, which lead to a significant reduction in energy consumption with a limited performance impact, as evaluated by a simulation study using real workload traces.
3. An optimal offline algorithm for the host overload detection problem, as well as a novel Markov chain model that allows a derivation of an optimal randomized control policy under an explicitly specified QoS goal for any known stationary workload and a given state configuration in the online setting.
4. A heuristically adapted host overload detection algorithm for handling unknown non-stationary workloads. The algorithm leads to approximately 88% of the mean inter-migration time produced by the optimal offline algorithm.
5. An open source implementation of a software framework for distributed dynamic VM consolidation called OpenStack Neat. The framework can be applied in both further research on dynamic VM consolidation, and real OpenStack Cloud deployments to improve the utilization of resources and reduce energy consumption.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Anton Beloglazov, 27 February 2013

Acknowledgements

PhD is a once-in-a-lifetime opportunity and experience. It is tough at times and may feel like an eternity, but it teaches you a lot, and I am truly happy that I have had a chance to complete it. It would not have happened without all those people who helped me along the way. First of all, I would like to thank my supervisor, Professor Rajkumar Buyya, who has given me the opportunity to undertake a PhD and provided with invaluable guidance and advice throughout my PhD candidature.

I would like to express my gratitude to the PhD committee members, Professor Chris Leckie, Dr. Saurabh Garg, and Dr. Rodrigo Calheiros, for their constrictive comments and suggestions on improving my work. I would also like to thank all the past and current members of the CLOUDS Laboratory, at the University of Melbourne. In particular, I thank Mukaddim Pathan, Marco Netto, Christian Vecchiola, Suraj Pandey, Marcos dias de Assunao, Kyong Hoon Kim, Srikumar Venugopal, Charity Lourdes, Mustafizur Rahman, Chee Shin Yeo, Xingchen Chu, Rajiv Ranjan, Alexandre di Costanzo, James Broberg, William Voorsluys, Mohsen Amini, Amir Vahid, Dileban Karunamoorthy, Nithiapidary Muthuvelu, Michael Mattess, Adam Barker, Jessie Yi Wei, Javadi Bahman, Linlin Wu, Adel Toosi, Sivaram Yoganathan, Deepak Poola, Mohammed Alrokayan, Atefeh Khosravi, Nikolay Grozev, Sareh Fotuhi, and Yaser Mansouri for their friendship and help during my PhD. I have had a great time with them and my other friends from the CSSE/CIS department – Andrey Kan, Jubaer Arif, Andreas Schutt, Archana Sathivelu, Jason Lee, Sergey Demyanov, Simone Romano, and Goce Ristanoski to name a few. I also thank my other friends in Australia, USA, France, and back in Russia.

I thank my previous supervisors, Dr. Sergey Piskunov and Dr. Valery Mishchenko, for their guidance and help during the work on my Master's and Bachelor's theses. I also thank my collaborators, Prof. Albert Zomaya, Prof. Jemal Abawajy, and Dr. Young Choon Lee. I acknowledge the University of Melbourne and Australian Federal Government for providing me with scholarships to pursue my doctoral studies. I thank the external examiners for their excellent reviews and suggestions on improving this thesis.

I am heartily thankful to my parents and sister for their support and encouragement at all times. Finally, I thank my wife Kseniya for her love, inspiration, patience, and for making my life filled with joy and happiness.

Anton Beloglazov
Melbourne, Australia
27 February 2013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

CLOUD computing has revolutionized the Information and Communication Technology (ICT) industry by enabling on-demand provisioning of elastic computing resources on a pay-as-you-go basis. An organization can either outsource its computational needs to the Cloud avoiding high up-front investments in a private computing infrastructure and consequent costs of maintenance and upgrades, or build a private Cloud data center to improve the resource management and provisioning processes.

The proliferation of Cloud computing has resulted in the establishment of large-scale data centers around the world containing thousands of compute nodes. However, Cloud data centers consume huge amounts of electrical energy resulting in high operating costs and carbon dioxide (CO_2) emissions to the environment. As shown in Figure 1.1, energy consumption by data centers worldwide has risen by 56% from 2005 to 2010, and in 2010 is accounted to be between 1.1% and 1.5% of the total electricity use [70]. Furthermore, carbon dioxide emissions of the ICT industry are currently estimated to be 2% of the global emissions, which is equivalent to the emissions of the aviation industry [52] and significantly contributes to the greenhouse effect. As projected by Koomey [69], energy consumption in data centers will continue to grow rapidly unless advanced energy-efficient resource management solutions are developed and applied.

To address the problem of high energy use, it is necessary to eliminate inefficiencies and waste in the way electricity is delivered to computing resources, and in the way these resources are utilized to serve application workloads. This can be done by improving both the physical infrastructure of data centers, and the resource allocation and management algorithms. Recent advancement in the data center design have resulted in a significant increase of the infrastructure efficiency. As reported by the Open Compute

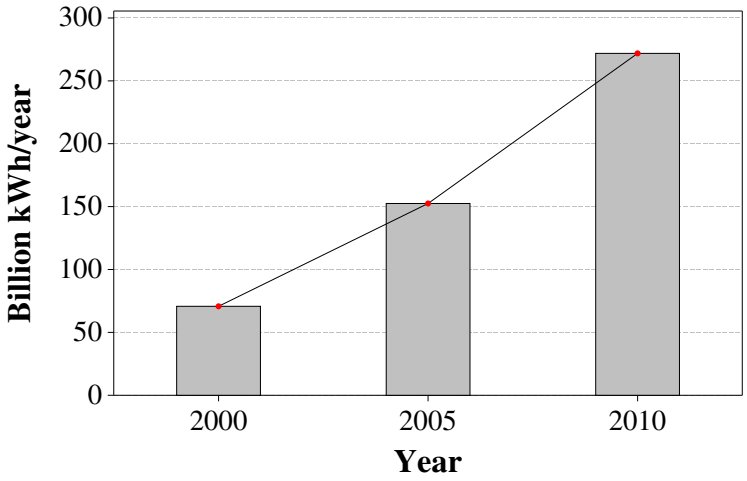


Figure 1.1: The worldwide data center energy consumption 2000-2010 [70]

project, Facebook's Oregon data center achieved a Power Usage Effectiveness (PUE) of 1.08 [89], which means that approximately 91% of the data center's energy consumption is consumed by the computing resources.

A source of energy waste lies in the inefficient usage of computing resources. Data collected from more than 5000 production servers over a six-month period have shown that although servers are usually not idle, the utilization rarely approaches 100% [13]. Most of the time servers operate at 10-50% of their full capacity, leading to extra expenses on over-provisioning, and thus extra Total Cost of Acquisition (TCA) [13]. Moreover, managing and maintaining over-provisioned resources results in the increased Total Cost of Ownership (TCO). In addition, the problem of low server utilization is exacerbated by narrow dynamic power ranges of servers: even completely idle servers still consume up to 70% of their peak power [44]. Therefore, keeping servers underutilized is highly inefficient from the energy consumption perspective. This thesis focuses on the problem of energy-efficient resource management in Cloud data centers, i.e., ensuring that computing resources are efficiently utilized to serve application workloads to minimize energy consumption, while maintaining the required Quality of Service (QoS).

1.1 Energy-Efficient Dynamic Consolidation of Virtual Machines

An ideal way of addressing the energy inefficiency problem is implementing an energy-proportional computing system, where energy consumption of the computing resources

is proportional to the application workload [13]. This approach is partially implemented through the widely adopted Dynamic Voltage and Frequency Scaling (DVFS) technique. DVFS allows the dynamic adjustment of the voltage and frequency of the CPU based on the current resource demand. As a result, current desktop and server CPUs can consume less than 30% of their peak power in low-activity modes, leading to dynamic power ranges of more than 70% [13].

In contrast, dynamic power ranges of all the other server components are much narrower: less than 50% for Dynamic Random Access Memory (DRAM), 25% for disk drives, 15% for network switches, and negligible for other components [44]. The reason is that only the CPU supports active low-power modes, whereas other components can only be completely or partially switched off. However, the performance overhead of a transition between the active and inactive modes is substantial. For example, a disk drive in the deep-sleep mode consumes negligible power, but a transition to the active mode incurs a latency 1000 times higher than the regular access latency. Power inefficiency of the server components in the idle state leads to a narrow overall dynamic power range of a server on the order of 30%. This means that even if a server is completely idle, it still consumes up to 70% of its peak power.

One method to improve the utilization of resources and reduce energy consumption, which has been shown to be efficient is dynamic consolidation of Virtual Machines (VMs) [59, 72, 86, 100, 119] enabled by the virtualization technology. Virtualization allows Cloud providers to create multiple VM instances on a single physical server, thus improving the utilization of resources and increasing the Return On Investment (ROI). The reduction in energy consumption can be achieved by switching idle nodes to low-power modes (i.e., sleep, hibernation), thus eliminating the idle power consumption (Figure 1.2).

Another capability provided by virtualization is *live migration*, which is the ability to transfer a VM between physical servers (referred to as hosts, or nodes) with a close to zero downtime. Using live migration [35] VMs can be *dynamically* consolidated to leverage fine-grained fluctuations in the workload and keep the number of active physical servers at the minimum at all times. Dynamic VM consolidation consists of two basic processes: migrating VMs from underutilized hosts to minimize the number of active hosts; and offloading VMs from hosts when those become overloaded to avoid perfor-

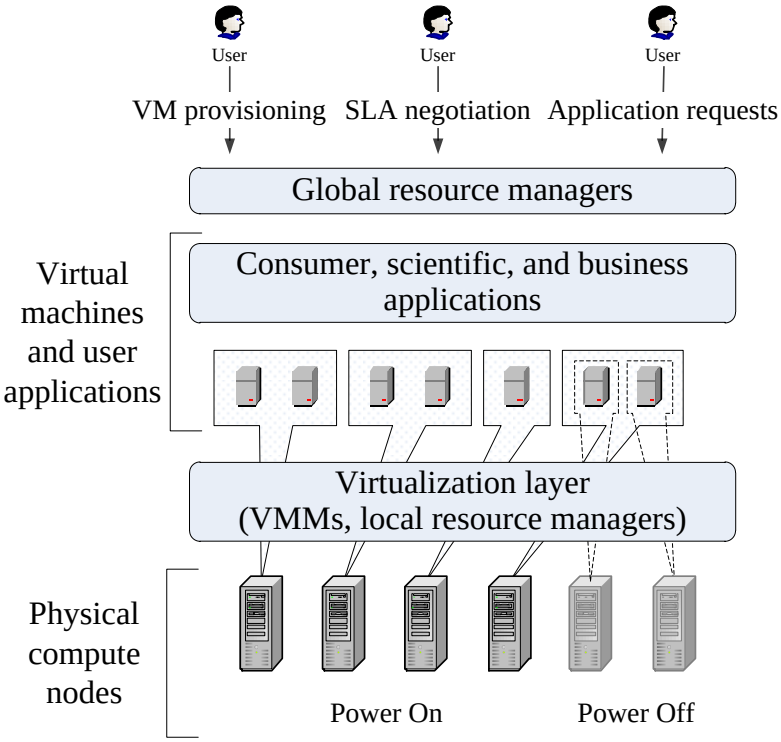


Figure 1.2: The high-level system view

mance degradation experienced by the VMs, which could lead to a violation of the QoS requirements. Idle hosts are automatically switched to a low-power mode to eliminate the static power and reduce the overall energy consumption by the system. When required, hosts are reactivated to accommodate new VMs or VMs being migrated.

However, dynamic VM consolidation in Clouds is not trivial since modern service applications often experience highly variable workloads causing dynamic resource usage patterns. Therefore, unconstrained VM consolidation may lead to performance degradation when an application encounters an increasing demand resulting in an unexpected rise of the resource usage. If the resource requirements of an application are not fulfilled, the application may face increased response times, time-outs or failures. Ensuring QoS defined via Service Level Agreements (SLAs) established between Cloud providers and their customers is essential for Cloud computing environments. Therefore, Cloud providers have to deal with the energy-performance trade-off – minimizing energy consumption, while meeting QoS requirements.

The scope of this thesis is energy-efficient dynamic VM consolidation in Infrastructure as a Service (IaaS) Cloud data centers under QoS constraints. The focus on IaaS en-

vironments, e.g., Amazon Elastic Compute Cloud (EC2)[1], imposes several requirements stipulated by the properties of the environment. Addressing them distinguishes the approach presented in this thesis from the related work, as discussed in Chapter 2.

In particular, it is necessary to handle mixed heterogeneous workloads since multiple independent users dynamically provision VMs and deploy various types of applications on shared physical resources. This means that the resource provider is unaware of the application types deployed in the system; therefore, the system must be application-agnostic, i.e., able to efficiently handle unknown mixed workloads. Another requirement is providing QoS guarantees specified in the SLAs negotiated between the Cloud provider and consumers. Since multiple different applications can coexist in the system, it is necessary to apply a workload independent QoS metric to quantify the performance delivered to the applications. Such a QoS metric can be used to specify system-wide workload independent QoS requirements.

Another aspect distinguishing the work presented in this thesis from the related research is the distributed architecture of the VM management system. A distributed VM management system architecture is essential for large-scale Cloud providers, as it enables the natural scaling of the system to thousands of compute nodes. An illustration of the importance of scalability of a VM management system is the fact that Rackspace[2], a well-known IaaS provider, currently manages tens of thousands of servers. Moreover, the number of servers continuously grows: Rackspace has increased the total server count in the second quarter of 2012 to 84,978 up from 82,438 servers at the end of the first quarter [99]. Another benefit of making the VM management system distributed is the improved fault tolerance by eliminating single points of failure: even if a compute or controller node fails, it would not render the whole system inoperable.

This thesis presents a complete solution to energy-efficient distributed dynamic VM consolidation under QoS constraints. It is evaluated by simulations using workload traces from more than a thousand PlanetLab VMs[3], as well as real experiments on a multi-node testbed using OpenStack Neat[4] – a prototype system implemented as a transparent add-on to the OpenStack Cloud platform.

[1] Amazon EC2. http://aws.amazon.com/ec2/
[2] The Rackspace Cloud. http://www.rackspace.com/
[3] The PlanetLab platform. http://www.planet-lab.org/
[4] The OpenStack Neat framework. http://openstack-neat.org/

1.2 Research Problems and Objectives

This thesis tackles the research challenges in relation to energy-efficient distributed dynamic VM consolidation in IaaS environments under QoS constraints. In particular, the following research problems are investigated:

- **How to define workload-independent QoS requirements.** Since multiple applications of various types can coexist in an IaaS environment and share physical resources, it is necessary to derive a workload independent QoS metric that can be used to define system-wide QoS constraints.
- **When to migrate VMs.** Dynamic VM consolidation comprises two basic processes: (1) migrating VMs from overloaded servers to avoid performance degradation; and (2) migrating VMs from underloaded servers to improve the utilization of resources and minimize energy consumption. A crucial decision that must be made in both situations is determining the best time to migrate VMs to minimize energy consumption, while satisfying the defined QoS constraints.
- **Which VMs to migrate.** Once a decision to migrate VMs from a server is made, it is required to select one or more VMs from the full set of VMs allocated to the server, which must be reallocated to other servers. The problem consists in determining the best subset of VMs to migrate that will provide the most beneficial system reconfiguration.
- **Where to migrate the VMs selected for migration.** Determining the best placement of new VMs or the VMs selected for migration to other servers is another essential aspect that influences the quality of VM consolidation and energy consumption by the system.
- **When and which physical nodes to switch on/off.** To save energy, VM consolidation should be combined with dynamic switching of the power states of nodes, which addresses the problem of narrow power ranges of servers by eliminating power consumption in the idle state. To optimize energy consumption by the system and avoid violations of the QoS requirements, it is necessary to efficiently determine when and which physical nodes should be deactivated to save energy, or reactivated to handle increases in the demand for resources.
- **How to design distributed dynamic VM consolidation algorithms.** To provide

scalability and eliminate single points of failure, it is necessary to design dynamic VM consolidation algorithms in a distributed manner. The problem is that traditionally global resources management algorithms are centralized. Therefore, a new approach is required to make the dynamic VM consolidation system distributed.

To deal with the challenges associated with the above research problems, the following objectives have been delineated:

- Explore, analyze, and classify the research in the area of energy-efficient computing to gain a systematic understanding of the existing techniques and approaches.
- Conduct competitive analysis of algorithms for dynamic VM consolidation to obtain theoretical performance estimates and insights into the design of online algorithms for dynamic VM consolidation.
- Propose a workload independent QoS metric that can be used in defining system-wide QoS constraints in IaaS environments.
- Propose an approach to designing a dynamic VM consolidation system in a distributed manner.
- Develop online algorithms for energy-efficient distributed dynamic VM consolidation for IaaS environments satisfying workload-independent QoS constraints.
- Design and implement a distributed dynamic VM consolidation system that can be used to evaluate the proposed algorithms on a multi-node IaaS testbed.

1.3 Methodology

The research methodology followed in this thesis consists of several consecutive steps summarized below:

1. Conduct theoretical analysis of dynamic VM consolidation algorithms to obtain theoretical performance estimates and insights into designing such algorithms. Since dynamic VM consolidation algorithms are online, the framework of competitive analysis has been applied. In this framework, the theoretical performance of an online algorithm is evaluated in comparison with the performance of an optimal offline algorithm designed for the same problem.
2. Develop distributed dynamic VM consolidation algorithms based on the insights

from the conducted competitive analysis and derived system model. Since the problem of dynamic VM consolidation involves a set of algorithms as discussed in the following chapters, multiple algorithms have been developed based on prior work on approximation algorithms for the bin packing problem, statistical analysis of historical data, and Markov chain modeling.

3. Evaluate the proposed algorithms through discrete-event simulation using the CloudSim simulation toolkit[5] extended to support power and energy-aware simulations. As the target system is an IaaS, a Cloud computing environment that is intended to create a view of infinite computing resources to the users, it is essential to evaluate the proposed algorithms on a large-scale virtualized data center infrastructure. However, conducting repeatable large-scale experiments on a real infrastructure is extremely difficult. Therefore, to ensure the repeatability and reproducibility of experiments, as well as carry out large-scale experiments, discrete-event simulation has been chosen as the initial way to evaluate the performance of the proposed algorithms.
4. Simulate application workloads using real-world workload traces from the PlanetLab platform. The CPU utilization of the VMs has been simulated based on the data provided as a part of the CoMon project, a monitoring infrastructure for PlanetLab [92]. The project provides data collected every 5 minutes from more than a thousand VMs instantiated in more than 500 locations around the world. The workload from PlanetLab VMs is representative of an IaaS Cloud environment, such as Amazon EC2, as the VMs are created and managed by multiple independent users, and the infrastructure provider is not aware of application workloads in the VMs. Furthermore, the overall system workload is composed of multiple independent heterogeneous applications, which also corresponds to an IaaS environment. The difference from a public Cloud provider, such as Amazon EC2, is that PlanetLab is an infrastructure mainly used for research purposes; therefore, the applications are potentially closer to the HPC type, which are typically CPU-intensive with lower dynamics in the resource utilization compared with web services. HPC workload is easier to handle for a VM consolidation system due to

[5]The CloudSim simulation toolkit. http://code.google.com/p/cloudsim/

infrequent variation in the resource utilization. Therefore, to stress the system, the original workload traces are filtered to select the ones that exhibit high variability.

5. Implement a system prototype as an add-on to an open source Cloud platform, which has been chosen to be OpenStack[6]. Apart from simulations, it is important to evaluate the proposed algorithms on a real infrastructure. A prototype distributed dynamic VM consolidation system has been implemented and used to experimentally evaluate the proposed algorithms on a real multi-node IaaS testbed.

1.4 Contributions

The contributions of this thesis can be broadly divided into 4 categories: classification and analysis of the area, competitive analysis of dynamic VM consolidation algorithms, novel algorithms for distributed dynamic VM consolidation, and implementation of a framework for distributed dynamic VM consolidation. The **key contributions** are:

1. A taxonomy and survey of the state-of-the-art in energy-efficient computing.

2. Competitive analysis of dynamic VM consolidation algorithms:
   - Formal definitions of the single VM migration and dynamic VM consolidation problems.
   - A proof of the cost incurred by an optimal offline algorithm for the single VM migration problem.
   - Competitive analysis and proofs of the competitive ratios of optimal online deterministic algorithms for the single VM migration and dynamic VM consolidation problems.

3. Novel algorithms for distributed dynamic VM consolidation:
   - The introduction of a distributed approach to energy and performance efficient dynamic VM consolidation.
   - Novel heuristics for the problem of energy and performance efficient dynamic VM consolidation following the introduced distributed approach.
   - A simulation-based evaluation and performance analysis of the algorithms.

[6]The OpenStack Cloud platform. http://openstack.org/

- An analytical model showing that to improve the quality of VM consolidation, it is necessary to maximize the mean time between VM migrations initiated by the host overload detection algorithm.
- An optimal offline algorithm for the host overload detection problem, and proof of its optimality.
- A novel Markov chain model that allows a derivation of a randomized control policy that optimally solves the problem of maximizing the mean time between VM migrations under an explicitly specified QoS goal for any known stationary workload and a given state configuration in the online setting.
- A heuristically adapted algorithm for handling unknown non-stationary workloads using the Multisize Sliding Window workload estimation approach [80], which leads to comparable to the best benchmark algorithm performance in terms of the inter-migration time, while providing the advantage of explicit specification of a QoS goal. The adapted algorithm leads to approximately 88% of the mean inter-migration time produced by the optimal offline algorithm for the input workload traces used in the experiments.

4. Software implementation of OpenStack Neat[7], a framework for distributed dynamic VM consolidation as an add-on to the OpenStack Cloud platform:
   - An architecture of an extensible software framework for dynamic VM consolidation designed to transparently integrate with OpenStack installations and allowing configuration-based substitution of multiple implementations of algorithms for each of the sub-problems of dynamic VM consolidation.
   - An open source software implementation of the framework in Python released under the Apache 2.0 license and publicly available online.
   - An implementation of several algorithms for dynamic VM consolidation proposed and evaluated by simulations in the previous chapters.
   - An initial version of a benchmark suite comprising the software framework, workload traces, performance metrics, and methodology for evaluating and comparing distributed dynamic VM consolidation solutions.
   - Experimental evaluation of the framework on a 5-node OpenStack deploy-

[7] The OpenStack Neat framework. http://openstack-neat.org/

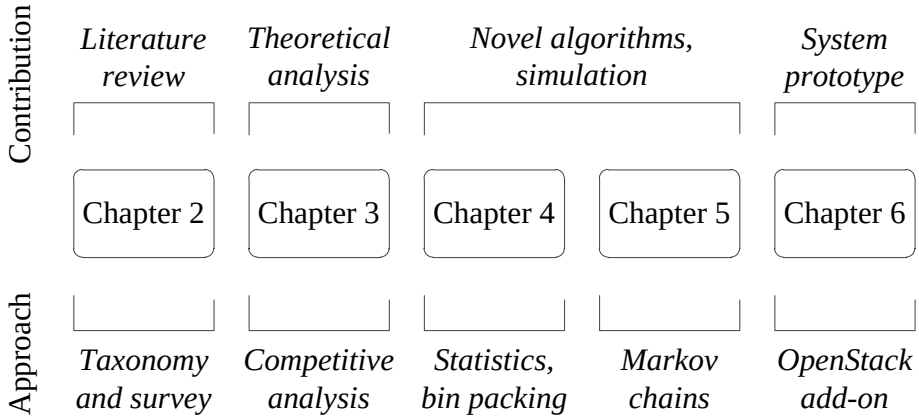


Figure 1.3: The thesis organization

ment using real-world application workload traces collected from more than a thousand PlanetLab VMs [92]. The experiments have shown that dynamic VM consolidation is able to reduce energy consumption by the compute nodes by up to 30% with a limited performance impact.

1.5 Thesis Organization

The core chapters of this thesis are structured as shown in Figure 1.3 and are derived from several journal papers published during the PhD candidature. The remainder of the thesis is organized as follows:

- Chapter 2 presents a taxonomy and survey of energy-efficient computing systems, as well as the scope of this thesis and its positioning in the area. This chapter is derived from [18]:
  - **Anton Beloglazov**, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya, "A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems," *Advances in Computers, Marvin V. Zelkowitz (editor)*, Volume 82, Pages: 47-111, Academic Press, USA, 2011.
- Chapter 3 presents competitive analysis of dynamic VM consolidation algorithms. This chapter is derived from [16]:
  - **Anton Beloglazov** and Rajkumar Buyya, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dy-

namic Consolidation of Virtual Machines in Cloud Data Centers," *Concurrency and Computation: Practice and Experience (CCPE)*, Volume 24, Issue 13, Pages: 1397-1420, John Wiley & Sons, Ltd, USA, 2012.

- Chapter 4 proposes novel algorithms for distributed dynamic VM consolidation. This chapter is derived from [15,16]:
  - **Anton Beloglazov**, Jemal Abawajy, and Rajkumar Buyya, "Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing," *Future Generation Computer Systems (FGCS)*, Volume 28, Issue 5, Pages: 755-768, Elsevier Science, The Netherlands, 2012.
  - **Anton Beloglazov** and Rajkumar Buyya, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers," *Concurrency and Computation: Practice and Experience (CCPE)*, Volume 24, Issue 13, Pages: 1397-1420, John Wiley & Sons, Ltd, USA, 2012.
- Chapter 5 proposes a novel host overload detection algorithm based on a Markov chain model. This chapter is derived from [17]:
  - **Anton Beloglazov** and Rajkumar Buyya, "Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers Under Quality of Service Constraints," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Volume 24, Issue 7, Pages: 1366-1379, IEEE CS Press, USA, 2013.
- Chapter 6 describes the architecture and implementation of OpenStack Neat, a framework for distributed dynamic VM consolidation in OpenStack Clouds. This chapter is derived from:
  - **Anton Beloglazov** and Rajkumar Buyya, "OpenStack Neat: A Framework for Dynamic Consolidation of Virtual Machines in OpenStack Clouds," *Software: Practice and Experience (SPE)*, John Wiley & Sons, Ltd, USA, 2013 (in review).
- Chapter 7 concludes the thesis with a summary of the main findings, discussion of future research directions, and final remarks.

Chapter 2

A Taxonomy and Survey of Energy-Efficient Computing Systems

Traditionally, the development of computing systems has been focused on performance improvements driven by the demand of applications from consumer, scientific, and business domains. However, the ever-increasing energy consumption of computing systems has started to limit further performance growth due to overwhelming electricity bills and carbon dioxide footprints. Therefore, the goal of the computer system design has been shifted to power and energy efficiency. To identify open challenges in the area and facilitate further advancements, it is essential to synthesize and classify the research on power- and energy-efficient design conducted to date. This chapter discusses the causes and problems of high power / energy consumption, and presents a taxonomy of energy-efficient design of computing systems covering the hardware, operating system, virtualization, and data center levels. The key works in the area are surveyed and mapped onto the taxonomy to guide future design and development efforts. This chapter concludes with a discussion of the scope of the current thesis and its positioning within the research area.

2.1 Introduction

THE primary focus of designers of computing systems and the industry has been on the improvement of the system performance. According to this objective, the performance has been steadily growing driven by more efficient system design and increasing density of the components described by Moore's law [84]. Although the performance per watt ratio has been constantly rising, the total power drawn by computing systems has hardly decreased. Oppositely, it has been increasing every year that can be illustrated by the estimated average power use across three classes of servers presented in Table 2.1 [69]. If this trend continues, the cost of energy consumed by a server during its lifetime will exceed the hardware cost [12]. The problem is even worse for large-scale

Table 2.1: Estimated average consumption per server class (W/U) 2000–2006 [69]

Server class	2000	2001	2002	2003	2004	2005	2006
Volume	186	193	200	207	213	219	225
Mid-range	424	457	491	524	574	625	675
High-end	5,534	5,832	6,130	6,428	6,973	7,651	8,163

compute infrastructures, such as clusters and data centers. It was estimated that energy consumption by IT infrastructures has risen by 56% from 2005 to 2010, and in 2010 accounted to be between 1.1% and 1.5% of the global electricity use [70]. Apart from high operating costs, this results in substantial carbon dioxide (CO_2) emissions, which were estimated to be 2% of the global emissions [52].

Energy consumption is not only determined by hardware efficiency, but also by the resource management system deployed on the infrastructure and the efficiency of applications running in the system. The interdependence of different levels of computing systems in regard to energy consumption is shown in Figure 2.1. Energy efficiency impacts end-users in terms of resource usage costs, which are typically determined by the Total Cost of Ownership (TCO) incurred by the resource provider. Higher power consumption results not only in boosted electricity bills but also in additional requirements to the cooling system and power delivery infrastructure, i.e., Uninterruptible Power Supplies (UPS), Power Distribution Units (PDU), and so on.

With the growth of the density of computer components, the cooling problem becomes crucial, as more heat has to be dissipated for a square meter. The problem is especially important for 1U and blade servers. These types of servers are the most difficult to cool because of the high density of components, and thus, the lack of space for the air flow. Blade servers bring the advantage of higher computational power in less rack space. For example, 60 blade servers can be installed into a standard 42U rack [83]. However, such a system requires more than 4000 W to supply the resources and cooling system. This is significantly higher compared with the same rack filled by 1U servers consuming 2500 W. Moreover, peak power consumption tends to limit further performance improvements due to constraints of the power distribution facilities. For example, to power a server rack in a typical data center, it is necessary to provide about 60 A [102]. Even if the cooling problem can be addressed in future systems, it is likely that delivering

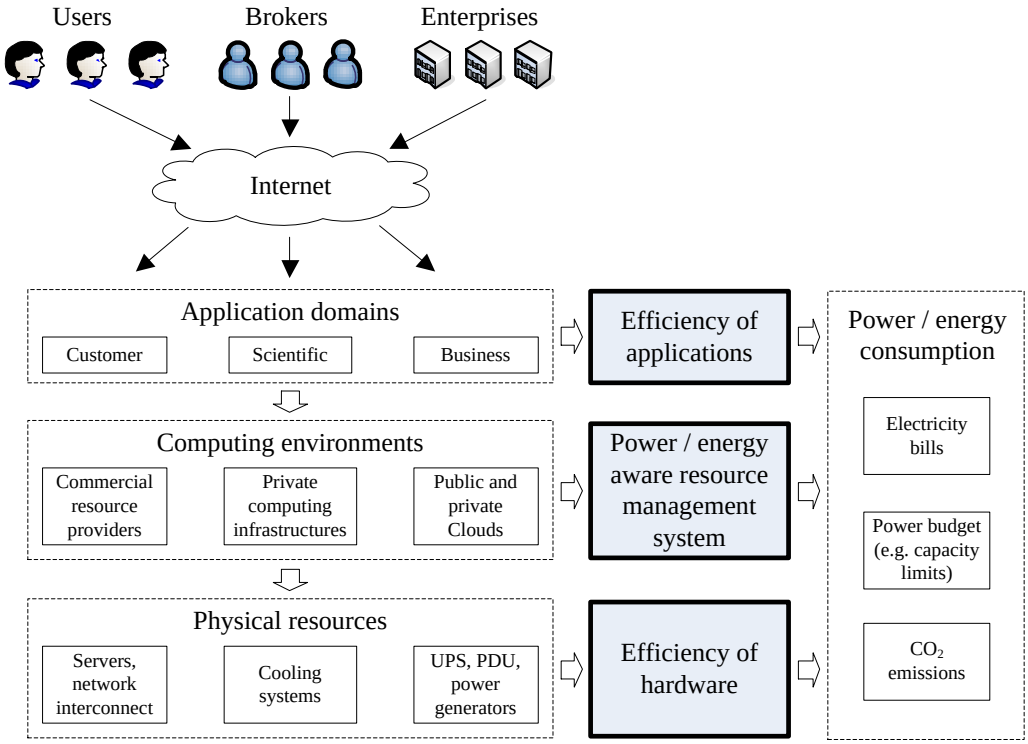


Figure 2.1: Energy consumption at different levels in computing systems

current in such data centers will reach the power delivery limits.

Apart from the overwhelming operating costs and the total cost of acquisition (TCA), another rising concern is the environmental impact in terms of carbon dioxide (CO_2) emissions caused by high energy consumption. Therefore, the reduction of power and energy consumption has become a first-order objective in the design of modern computing systems. The roots of energy-efficient computing, or Green IT, practices can be traced back to 1992, when the U.S. Environmental Protection Agency launched Energy Star, a voluntary labeling program designed to identify and promote energy-efficient products to reduce the greenhouse gas emissions. Computers and monitors were the first labeled products. This led to the widespread adoption of the sleep mode in electronic devices.

At that time, the term "green computing" was introduced to refer to energy-efficient personal computers [103]. At the same time, the Swedish confederation of professional employees has developed the TCO certification program – a series of end-user and environmental requirements for IT equipment including video adapters, monitors, keyboards, computers, peripherals, IT systems, and even mobile phones. Later, this program has been extended to include requirements on ergonomics, magnetic and electrical

field emission levels, energy consumption, noise level, and use of hazardous compounds in hardware. The Energy Star program was revised in October 2006 to include stricter efficiency requirements for computer equipment a tiered ranking system.

There are a number of industry initiatives aiming at the development of standardized methods and techniques for the reduction of energy consumption and carbon dioxide emissions in computing environments. They include Climate Savers Computing Initiative (CSCI), Green Computing Impact Organization, Inc. (GCIO), Green Electronics Council, The Green Grid, FIT4Green, ECO2Clouds, Eco4Cloud, International Professional Practice Partnership (IP3), with the membership of large companies, such as AMD, Dell, HP, IBM, Intel, Microsoft, Sun Microsystems, and VMware.

Energy-efficient resource management has been first introduced in the context of battery-powered mobile devices, where energy consumption has to be reduced to improve the battery lifetime. Although techniques developed for mobile devices can be applied or adapted for servers and data centers, this kind of systems requires specific methods. This chapter discusses various ways of reducing power and energy consumption in computing systems, as well as recent research that deals with power and energy efficiency at the hardware and firmware, Operating System (OS), virtualization, and data center levels. The objectives of this chapter is to give an overview of the recent research advancements in energy-efficient computing, classify the approaches, discuss open research challenges, and position the current thesis within the research area.

The reminder of this chapter is organized as follows. In the next section, power and energy models are introduced. Section 2.3 discusses problems caused by high power and energy consumption. Sections 2.4.1-2.4.4 present the taxonomy and survey of the research in energy-efficient design of computing systems. The chapter is concluded with the positioning of the current thesis within the research area in Section 2.5, followed by a summary and directions for future work in Section 2.6.

2.2 Power and Energy Models

To understand power and energy management mechanisms, it is essential to clarify the terminology. Electric current is the flow of electric charge measured in amperes. Am-

peres define the amount of electric charge transferred by a circuit per second. Power and energy can be defined in terms of work that a system performs. Power is the rate at which the system performs the work, while energy is the total amount of work performed over a period of time. Power and energy are measured in watts (W) and watt-hour (Wh), respectively. Work is done at the rate of 1 W when 1 A is transferred through a potential difference of 1 V. A kilowatt-hour (kWh) is the amount of energy equivalent to a power of 1 kW (1000 W) being applied for one hour. Formally, power and energy can be defined as shown in (2.1).

$$\begin{aligned} P &= \frac{W}{T}, \\ E &= PT, \end{aligned} \tag{2.1}$$

where P is power, T is a period of time, W is the total work performed during that period of time, and E is energy. The difference between power and energy is very important since a reduction of power consumption does not always reduce the consumed energy. For example, power consumption can be decreased by lowering the CPU performance. However, in this case, a program may take longer to complete its execution consuming the same amount of energy. On one hand, a reduction of peak power consumption results in decreased costs of the infrastructure provisioning, such as costs associated with capacities of UPS, PDU, power generators, cooling system, and power distribution equipment. On the other hand, decreased energy consumption reduces the electricity bills.

Energy consumption can be reduced temporarily via Dynamic Power Management (DPM) techniques, or permanently applying Static Power Management (SPM). DPM utilizes the knowledge of the real-time resource usage and application workloads to optimize energy consumption. However, it does not necessarily decrease peak power consumption. In contrast, SPM prescribes the usage of highly efficient hardware components, such as CPUs, disk storage, network devices, UPS, and power supplies. These structural changes usually reduce both energy and peak power consumption.

2.2.1 Static and Dynamic Power Consumption

The major part of power consumption in complementary metal-oxide-semiconductor (CMOS) circuits comprises static and dynamic power. Static power consumption, or

leakage power, is caused by leakage currents that are present in any active circuit, independently of clock rates and usage scenarios. This static power is mainly determined by the type of transistors and process technology. The reduction of static power requires improvements of the low-level system design; therefore, it is not within the scope of this chapter. More details regarding possible ways to improve energy efficiency at this level can be found in the survey by Venkatachalam and Franz [117].

Dynamic power consumption is created by circuit activity (i.e., transistor switches, changes of values in registers, etc.) and depends mainly on a specific usage scenario, clock rates, and I/O activity. The sources of dynamic power consumption are the short-circuit current and switched capacitance. Short-circuit current causes only 10-15% of the total power consumption and so far no way has been found to reduce this value without compromising the performance. Switched capacitance is the primary source of dynamic power consumption; therefore, dynamic power consumption can be defined as (2.2).

$$P_d = aCV^2f, \tag{2.2}$$

where a is the switching activity, C is the physical capacitance, V is the supply voltage, and f is the clock frequency. The values of switching activity and capacitance are determined by the low-level system design. The combined reduction of the supply voltage and clock frequency lies in the roots of the widely adopted DPM technique called Dynamic Voltage and Frequency Scaling (DVFS). The main idea of this technique is to intentionally scale down the CPU performance, when it is not fully utilized, by decreasing the voltage and frequency of the CPU. In the ideal case, this should result in a cubic reduction of dynamic power consumption. DVFS is supported by most modern CPUs including mobile, desktop, and server systems. This technique is discussed in detail in Section 2.4.1.

2.2.2 Sources of Power Consumption

According to data provided by Intel Labs [83], the main part of power consumed by a server is accounted for the CPU, followed by the memory and losses due to the power supply inefficiency (Figure 2.2). The data show that the CPU no longer dominates power consumption by a server. This resulted from the continuous improvements of the CPU power efficiency combined with power-saving techniques (e.g., DVFS) that enable active

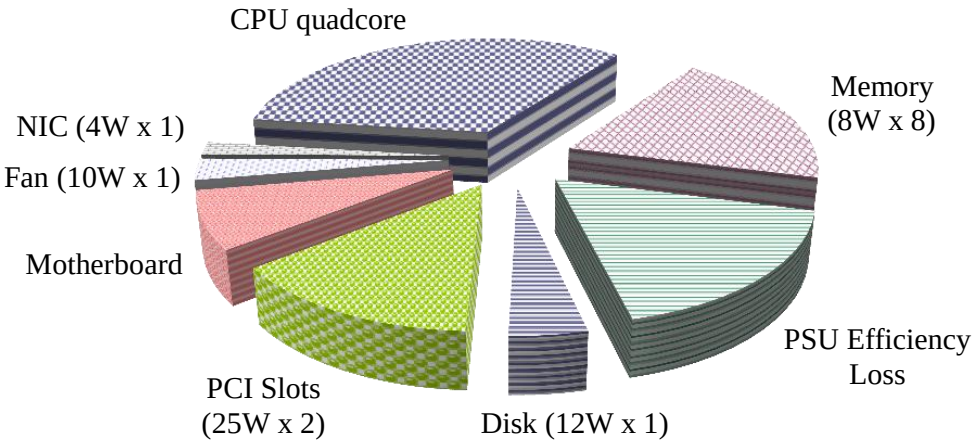


Figure 2.2: Power consumption by server components [83]

low-power modes. In these modes, a CPU consumes a fraction of the total power, while preserving the ability to execute programs. As a result, current desktop and server CPUs can consume less than 30% of their peak power in low-activity modes, leading to dynamic power ranges of more than 70% of the peak power [13].

In contrast, dynamic power ranges of all the other server components are much narrower: less than 50% for Dynamic Random Access Memory (DRAM), 25% for disk drives, 15% for network switches, and negligible for other components [44]. The reason is that only the CPU supports active low-power modes, whereas other components can only be completely or partially switched off. However, the performance overhead of a transition between the active and inactive modes is substantial. For example, a disk drive in the deep-sleep mode consumes almost no power, but a transition to the active mode incurs a latency 1000 times higher than the regular access latency. Power inefficiency of the server components in the idle state leads to a narrow overall dynamic power range of 30%: even if a server is completely idle, it still consumes more than 70% of its peak power.

Another reason for the reduction of the fraction of power consumed by the CPU relatively to the whole system is the adoption of multi-core architectures. Multi-core processors are substantially more efficient than conventional single-core processors. For example, servers built with recent Quad-core Intel Xeon processor can deliver 1.8 teraflops at the peak performance, using less than 10 kW of power. To compare with, Pentium processors in 1998 would consume about 800 kW to achieve the same performance [83].

The adoption of multi-core CPUs along with the increasing use of virtualization and

data-intensive applications resulted in the growing amount of memory in servers. In contrast to the CPU, DRAM has a narrower dynamic power range, and power consumption by memory chips is increasing. Memory is packaged in dual in-line memory modules (DIMMs), and power consumption by these modules varies from 5 to 21 W per DIMM for the DDR3 and fully buffered DIMM (FB-DIMM) memory technologies [83]. Power consumption by a server with eight 1 GB DIMMs is about 80 W. Modern large servers currently use 32 or 64 DIMMs, which leads to power consumption by the memory being higher than by the CPUs. Most of power management techniques are focused on the CPU; however, the constantly increasing frequency and capacity of memory chips in addition to the problem of high energy consumption raise the cooling requirements. These facts make memory one of the most important server components that has to be efficiently managed.

Power supplies transform alternating current (AC) into direct current (DC) to feed the server components. This transformation leads to significant power losses due to the inefficiency of the current technology. The efficiency of power supplies depends on their load. They achieve the highest efficiency at loads within the range of 50-75%. However, most data centers normally create a load of 10-15% wasting the majority of the consumed electricity and leading to the average power losses of 60-80%. As a result, power supplies consume at least 2% of the US electricity production [83]. More efficient power supply design can save more than a half of energy consumption.

The problem of the low average utilization also applies to disk storage, especially when disks are attached to servers in a data center. However, this can be addressed by moving the disks to an external centralized storage array. Nevertheless, intelligent policies are required to efficiently manage a storage system containing thousands of disks.

2.2.3 Modeling Power Consumption

To develop new policies for DPM and understand their impact, it is necessary to create a model of dynamic power consumption. Such a model should be able to predict the actual value of power consumption by a system based on some run-time system characteristics. One of the ways to accomplish this is to utilize power monitoring capabilities that are built into modern computer servers. These capabilities allow the monitoring of power

consumption by a server in real-time and collecting accurate statistics of the power usage. Based on the data, it is possible to derive a power consumption model for a particular system. However, this approach requires collecting data for every target system.

Fan et al. [44] found a strong relationship between the CPU utilization and total power consumption by a server. The idea behind the proposed model is that power consumption by a server grows linearly with the growth of the CPU utilization from the value of power consumption in the idle state up to the power consumed when the server is fully utilized. This relationship can be expressed as shown in (2.3).

$$P(u) = P_{idle} + (P_{busy} - P_{idle})u, \tag{2.3}$$

where P is the estimated power consumption, P_{idle} is power consumption by an idle server, P_{busy} is the power consumed by the server when it is fully utilized, and u is the current CPU utilization. The authors have also proposed an empirical nonlinear model given in (2.4):

$$P(u) = P_{idle} + (P_{busy} - P_{idle})(2u - u^r), \tag{2.4}$$

where r is a calibration parameter that minimizes the square error and has to be obtained experimentally. For each class of machines of interest, a set of calibration experiments must be performed to fine tune the model.

Extensive experiments on several thousand nodes under different types of workloads (Figure 2.3) showed that the derived models accurately predict power consumption by server systems with the error below 5% for the linear model and 1% for the empirical model. The calibration parameter r was set to 1.4 to obtain the presented results. These precise results can be explained by the fact that the CPU is the main power consumer in servers and, in contrast to the CPU, other system components (e.g., I/O, memory) have narrow dynamic power ranges or their activities correlate with the CPU activity. For example, current server processors can reduce power consumption up to 70% by switching to low-power-performance modes [13]. However, dynamic power ranges of other components are much narrower: $< 50\%$ for DRAM, 25% for disk drives, and 15% for network switches.

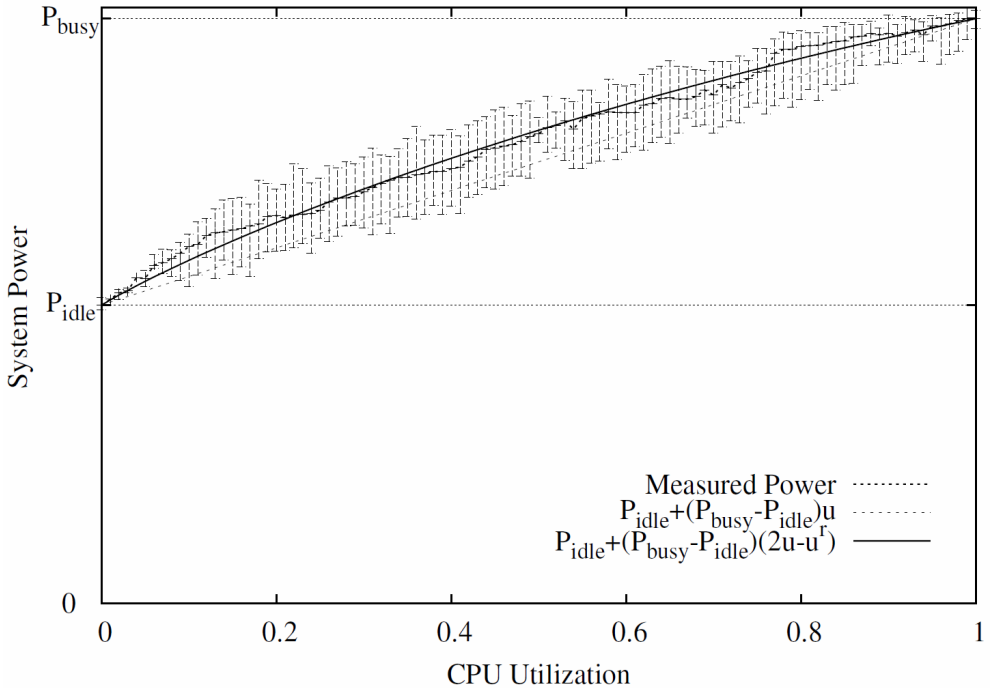


Figure 2.3: The relation between power consumption and CPU utilization of a server [44]

The presented accurate and simple power models enable easy prediction of power consumption by a server supplied with the CPU utilization data and power consumption values at the idle and maximum CPU utilization states. Therefore, it is especially important that the increasing number of server manufactures publish actual power consumption figures for their systems at different utilization levels [22]. This is driven by the adoption of the ASHRAE Thermal Guideline [7] that recommends providing power ratings for the minimum, typical and full CPU utilization.

Dhiman et al. [41] found that although regression models based on just the CPU utilization are able to provide reasonable prediction accuracy for CPU-intensive workloads, they tend to be considerably inaccurate for prediction of power consumption caused by I/O- and memory-intensive applications. The authors proposed a power modeling methodology based on Gaussian mixture models that predicts power consumption by a physical machine running multiple virtual machine (VM) instances. To perform predictions, in addition to the CPU utilization, the model relies on run-time workload characteristics such as the number of instructions per cycle (IPC) and the number of memory accesses per cycle (MPC).

The proposed approach requires a training phase to perceive the relationship between the workload metrics and power consumption. The authors evaluated the proposed model via experimental studies involving different workload types. The obtained experimental results showed that the model predicts power consumption with high accuracy ($< 10\%$ prediction error), which is consistent for all the tested workloads. The proposed model outperforms regression models by a factor of 5 for specific workload types. This proves the importance of architectural metrics like IPC and MPC as compliments to the CPU utilization for power consumption prediction.

2.3 Problems of High Power and Energy Consumption

Energy consumption by computing facilities raises various monetary, environmental, and system performance concerns. A recent study on power consumption of server farms [69] showed that in 2005 the electricity use by servers worldwide – including their associated cooling and auxiliary equipment – cost 7.2 billion dollars. The study also indicates that the electricity consumption in that year had doubled compared to the consumption in 2000. Clearly, there are environmental issues with the generation of electricity. The number of transistors integrated into today's Intel Itanium 2 processor reaches nearly 1 billion. If the transistor density continues to growth, the heat (per cm^2) produced by future processors would exceed that of the surface of the Sun [68]. The scope of energy-efficient design is not limited to main computing components (e.g., processors, storage devices, and visualization facilities), but can expand into a much larger range of resources associated with computing facilities including auxiliary equipment, water used for cooling, and even the floor space occupied by the resources.

While recent advances in hardware technologies including low-power processors, solid state drives, and energy-efficient monitors have alleviated the energy consumption issue to a certain degree, a series of software approaches have significantly contributed to the improvement of energy efficiency. These two approaches (hardware and software) should be seen as complementary rather than competitive. User awareness is another non-negligible factor that should be taken into account when discussing Green IT. User awareness and behavior in general considerably affect computing workload and resource

usage patterns. This, in turn, has a direct relationship with energy consumption of not only core computing resources but also auxiliary equipment, such as cooling/air conditioning systems. For example, a computer program developed without paying much attention to its energy efficiency may lead to excessive energy consumption and may contribute to higher heat emission resulting in increases in the energy consumed for cooling.

Traditionally, power- and energy-efficient resource management techniques were applied to mobile devices. It was dictated by the fact that such devices are usually battery-powered, and it is essential to apply power and energy management to improve their lifetime. However, due to the continuous growth of power and energy consumption by servers and data centers, the focus of power and energy management techniques has been switched to such systems. Even though the problems caused by high power and energy consumption are interconnected, they have their specifics and have to be considered separately. The difference is that peak power consumption determines the cost of the infrastructure required to maintain the system operation, whereas energy consumption accounts for electricity bills. These two issues are discussed in detail in the next sections.

2.3.1 High Power Consumption

The main reason of power inefficiency in data centers is the low average utilization of resources. To show this, the workload trace data provided as a part of the CoMon project[1], a monitoring infrastructure for PlanetLab[2], were analyzed. The data on the CPU utilization by more than a thousand servers located at more than 500 places around the world were used in the analysis. The data were collected in 5 minute intervals during the period from 10 to 19 May 2010. The distribution of the CPU utilization data over 10 days along with the characteristics of the distribution are presented in Figure 2.4.

The data confirm the observation made by Barroso and Holzle [13]: the average CPU utilization is below 50%. The mean value of the CPU utilization is 36.44% with 95% confidence interval from 36.40% to 36.47%. The main run-time reasons of under-utilization in data centers are variability of the workload and statistical effects. Modern service applications cannot be kept on fully utilized servers, as even non-significant workload

[1]The CoMon project. http://comon.cs.princeton.edu/
[2]The PlanetLab platform. http://www.planet-lab.org/

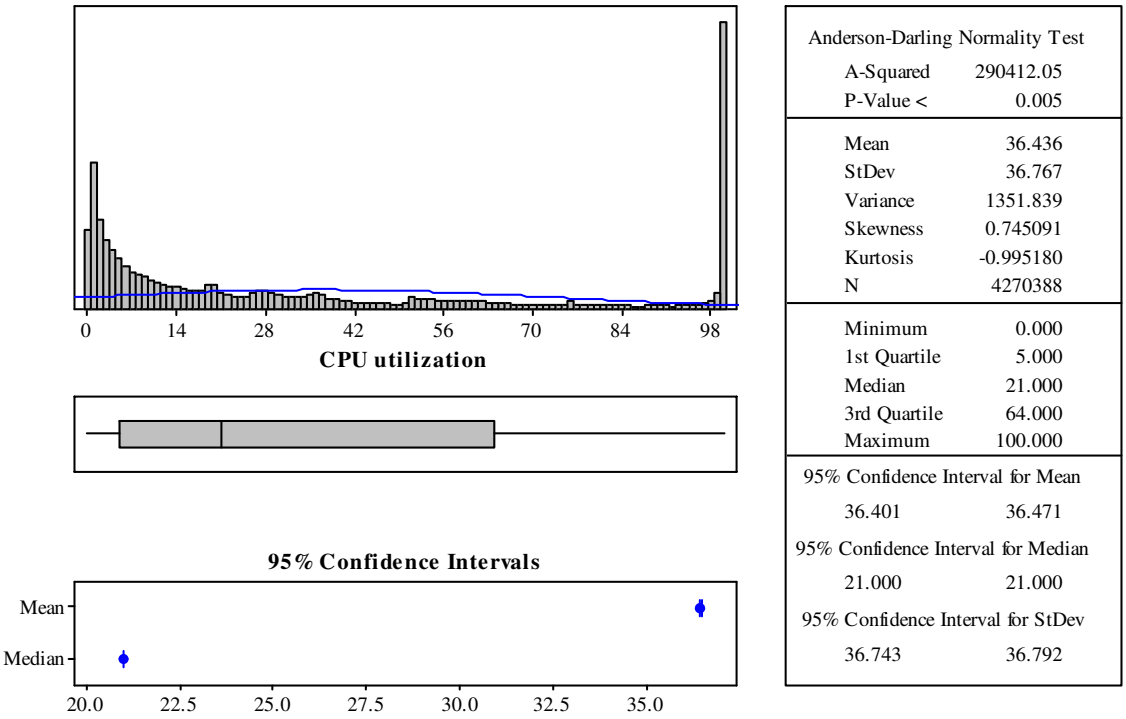


Figure 2.4: The CPU utilization from the PlanetLab nodes over a period of 10 days

fluctuation would lead to performance degradation and failing to provide the expected Quality of Service (QoS). However, servers in a non-virtualized data center are unlikely to be completely idle because of background tasks (e.g., incremental backups), or distributed data bases / file systems. Data distribution helps to tackle load-balancing problems and improves fault tolerance. Furthermore, despite the fact that the resources have to be provisioned to handle theoretical peak loads, it is very unlikely that all the servers of a large-scale data centers will be fully utilized simultaneously.

Systems where the average utilization of resources is less than 50% represent huge inefficiency, as most of the time only a half of the resources are actually in use. Although the resources on average are utilized by less than 50%, the infrastructure has to be built to handle the potential peak load, which rarely occurs in practice. In such systems, the cost of the over-provisioned capacity is very significant and includes expenses on the extra capacity of cooling systems, PDU, generators, power delivery facilities, UPS, and so on. The lower the average resource utilization in a data center, the more expensive the data center becomes as a part of the TCO, as it has to support peak loads and meet the require-

ments of handling peak power consumption [44]. Moreover, peak power consumption can constrain the further growth of power density, as power requirements already reach 60 A for a server rack [102]. If this tendency continues, further performance improvements can be bounded by the power delivery capabilities.

Another problem of high power consumption and increasing density of server components (i.e., 1U, blade servers) is the heat dissipation. Much of the electrical power consumed by computing resources gets turned into heat. The amount of heat produced by an integrated circuit depends on how efficient the component design is, and the voltage and frequency at which the component operates. The heat generated by the resources has to be dissipated to keep them within their safe thermal state. Overheating of the components can lead to a decrease of their lifetime and high error-proneness.

Moreover, power is required to feed the cooling system operation. Although modern cooling systems are more efficient, just a few years ago for each watt of power consumed by computing resources, an additional 0.5-1 W was required for the cooling system [102]. For example, in 2004 to dissipate 1 W consumed by a high-performance computing (HPC) system at the Lawrence Livermore National Laboratory (LLNL), 0.7 W of additional power is needed for the cooling system [95]. Moreover, modern high-density servers, such as 1U and blade servers, further complicate cooling because of the lack of space for airflow within the packages. These facts illustrate the concerns about the efficiency and real-time adaptation of the cooling system operation.

2.3.2 High Energy Consumption

A way to address high power consumption is the minimization of the peak power required to feed a completely utilized system. In contrast, energy consumption is defined by the average power consumption over a period of time. Therefore, the actual energy consumption by a data center does not directly determine the cost of the infrastructure. However, it is reflected in the cost of electricity consumed by the system, which is the main component of data center operating costs. Furthermore, in most data centers, 50% of the consumed energy never reaches the computing resources: it is consumed by the cooling facilities or dissipated in conversions within the UPS and PDU systems. With the current tendency of the continuously growing energy consumption and costs associated

with it, the point when operating costs exceed the cost of computing resources themselves in few years can be reached soon. Therefore, it is crucial to develop and apply energy-efficient resource management strategies in data centers.

In addition to high operating costs, another problem caused by the growing energy consumption is high CO_2 emissions, which contribute to the global warming. According to Gartner [52] in 2007, the Information and Communications Technology (ICT) industry was responsible for about 2% of global CO_2 emissions, which is equivalent to the aviation industry. According to the estimation by the U.S. Environmental Protection Agency (EPA), the current efficiency trends led to the increase of annual CO_2 emissions from 42.8 million metric tons ($MMTCO_2$) in 2007 to 67.9 $MMTCO_2$ in 2011. Intense media coverage has raised the awareness of people around the climate change and greenhouse effect. More and more customers start to consider the "green" aspect in selecting products and services. Besides the environmental concern, companies have begun to face business risks caused by being non-environment friendly. The reduction of CO_2 footprints is an important problem that has to be addressed in order to facilitate further advancements and proliferation of computing systems.

2.4 The State of the Art in Energy-Efficient Computing Systems

A large volume of research has been done in the area of power and energy-efficient resource management in computing systems. As power and energy management techniques are closely connected, in this chapter they are referred to as power management. As shown in Figure 2.5, at the high-level, power management techniques can be divided into static and dynamic. From the hardware point of view, SPM contains all the optimization methods that are applied at the design time at the circuit, logic, architectural, and system levels [40, 117]. The Circuit level optimization is focused on the reduction of the switching activity power of individual logic gates and transistor level combinational circuits by the application of complex gate design and transistor sizing. The optimization at the logic level is aimed at the switching activity power of logic-level combinational and sequential circuits. Architecture level methods include the analysis of system design and subsequent incorporation of power optimization techniques into it. In other words, this

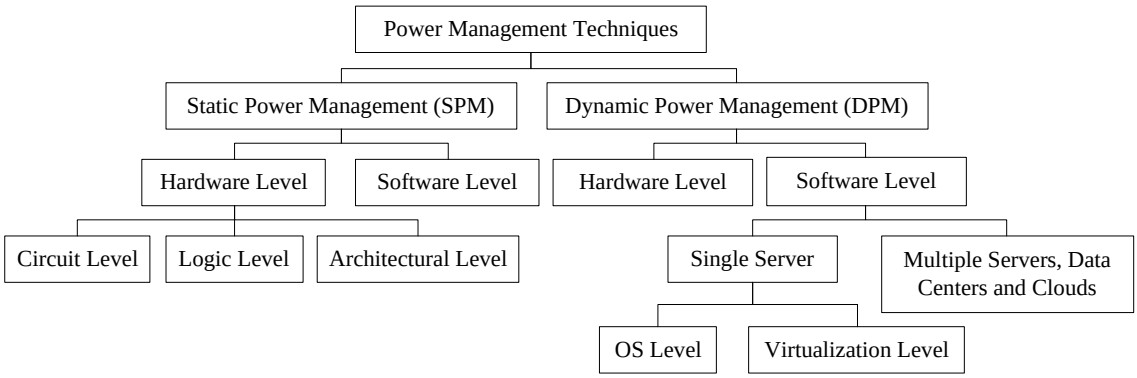


Figure 2.5: A high-level taxonomy of power and energy management

kind of optimization refers to the process of efficient mapping of a high-level problem specification onto a register-transfer level design.

Apart from the optimization of the hardware-level system design, it is extremely important to carefully consider the implementation of programs that are to be executed on the system. Even with perfectly designed hardware, poor software design can lead to dramatic performance and power losses. However, it is impractical or impossible to analyze power consumption caused by large programs at the operator level, as not only the processes of code generation and compilation, but also the order of instructions can have an impact on power consumption [53, 113, 114]. Therefore, indirect estimation methods can be applied. For example, it has been shown that faster code almost always implies lower energy consumption [88, 115]. Since generic methods for synthesizing optimal algorithms are not currently available, algorithm analysis is an important research area.

This chapter focuses on DPM techniques that include methods and strategies for runtime adaptation of the system behavior according to the current resource requirements or any other dynamic characteristic of the system state. A major assumption enabling DPM is that systems experience variable workloads during their operation allowing the dynamic adjustment of power states according to the current performance requirements. Another often made assumption is that the workload can be predicted to a certain degree, which enables the inference of future system states and taking the appropriate actions.

As shown in Figure 2.5, DPM techniques can be distinguished by the level at which they are applied: hardware or software. Hardware DPM varies for different hardware components, but usually can be classified as Dynamic Performance Scaling (DPS), such as

DVFS, and partial or complete Dynamic Component Deactivation (DCD) during periods of inactivity. In contrast, software DPM techniques utilize an interface to the system's power management capabilities and according to their policies apply hardware DPM. The following subsections 2.4.1-2.4.4 detail and survey DPM approaches proposed in the literature focusing on the hardware, OS, virtualization, and data center levels.

The following sections detail different levels of the presented taxonomy: Section 5 discusses power optimization techniques that can be applied at the hardware level. The approaches proposed for power management at the OS level are surveyed in Section 6, followed by the discussion of modern virtualization technologies and their impact on power-aware resource management in Section 7, and the recent approaches applied at the data center level in Section 8.

2.4.1 Hardware and Firmware Level

As shown in Figure 2.5, DPM techniques applied at the hardware and firmware level can be broadly divided into two categories: Dynamic Component Deactivation (DCD) and Dynamic Performance Scaling (DPS). DCD techniques are built upon the idea of clock gating of parts of an electronic component or complete disabling the components during periods of inactivity. In contrast, instead of completely deactivating system components, DPS techniques dynamically adjust the performance of the components according to the real-time demands.

Dynamic Component Deactivation

Computer components that do not support performance scaling and can only be deactivated require techniques that leverage the workload variability and disable the components when they are idle. The problem is trivial in the case of a negligible power and performance overhead of transitions between power states. However, in reality such transitions lead to not only delays, which can degrade performance of the system, but also to additional power draw caused by the reinitialization of the components. For example, if entering a low-power state requires a shutdown of the power supply, returning to the active state will cause a delay consisting of turning on and stabilizing the power

supply and clock, reinitializing the system, and restoring the context [20].

In the case of non-negligible transitions, efficient power management turns into a complex online optimization problem. A transition to low-power state is worthwhile only if the period of inactivity is longer than the aggregated delay of transitions from and to the active state, and the saved power is higher than the power required to reinitialize the component. In most real-world systems, there is a limited or no knowledge of the future workload. Therefore, a prediction of an effective transition has to be done based on an analysis of the historical data or some other model. A large volume of research has been done on the development of efficient methods for solving this problem [2, 20]. As shown in Figure 2.5, DCD techniques can be divided into predictive and stochastic.

Predictive techniques are based on the correlation between the past history of the system behavior and its near future. The efficiency of such techniques is highly dependent on the actual correlation between past and future events and the quality of tuning for a particular workload type. A non-ideal prediction can result in an over- or under-prediction. An over-prediction means that the actual idle period is shorter than the predicted, leading to a performance penalty. On the other hand, an under-prediction means that the actual idle period is longer than the predicted. This case does not have any influence on the performance; however, it results in reduced energy savings.

Predictive techniques can be further split into static and adaptive. Static techniques utilize some threshold of a real-time execution parameter to make predictions of idle periods. The simplest policy is called fixed timeout. The idea is to define the length of time, after which a period of inactivity is considered to be long enough to transition to a low-power state. The reactivation of the component is initiated once the first request to the component is received. The policy has two advantages: it can be applied to any workload type, and over- and under-predictions can be controlled by adjusting the value of the timeout threshold. However, the disadvantages are obvious: the policy requires the tuning of the threshold value for each workload; it always leads to a performance loss on the activation; and the energy consumed from the beginning of an idle period until the timeout is wasted. Two ways to overcome the drawbacks of the fixed timeout policy were proposed: predictive shutdown and predictive wake-up.

Predictive shutdown policies address the problem of the missed opportunity to save

energy before the timeout. These policies utilize the assumption that previous periods of inactivity are highly correlated with the nearest future. According to an analysis of the historical information, they predict the length of the next idle period before it actually begins. These policies are highly dependent on the actual workload and the strength of correlation between past and future events. History-based predictors were shown to be more efficient and less safe than timeouts [109]. Predictive wake-up techniques aim to eliminate the performance penalty on the reactivation. A transition to the active state is predicted based on the past history and performed before an actual user request [61]. This technique increases energy consumption but reduces performance losses on wake-ups.

All the mentioned static techniques are inefficient in cases when the system workload is unknown or can vary over time. To address this problem, adaptive predictive techniques have been introduced. The basic idea is to dynamically adjust the parameters, which are fixed for the static techniques, according to the prediction quality that they have provided in the past. For example, the timeout value can be increased if for the last several intervals the value led to over-predictions. Another way to implement adaptation is to maintain a list of possible values of the parameter of interest and assign weights to the values according to their efficiency at previous intervals. The actual value is obtained as a weighted average over all the values in the list. In general, adaptive techniques are more efficient than static when the type of the workload is unknown *a priori*. Several adaptive techniques are discussed in the paper by Douglis et al. [42].

Another way to deal with non-deterministic system behavior is to formulate the problem as stochastic optimization, which requires building an appropriate probabilistic model of the system. For instance, in such a model, system requests and power state transitions are represented as stochastic processes and can be modeled as Markov processes. At any moment, a request arrives with some probability and a device power state transition occurs with another probability obtained by solving the stochastic optimization problem. It is important to note that the results, obtained using the stochastic approach, are expected values, and there is no guarantee that the solution will be optimal for a particular case. Moreover, constructing a stochastic model of the system in practice may not be straightforward. If the model is inaccurate, the policies may not provide the efficient control.

Dynamic Performance Scaling

DPS includes power management techniques that can be applied to computer components supporting dynamic adjustment of their performance proportionally to power consumption. In addition to complete deactivation, some components, such as the CPU, allow gradual reductions or increases of the clock frequency along with adjustments of the supply voltage. This approach is useful when the resource is not fully utilized. The widely adopted DVFS technique is an example of DPS.

Although the CPU frequency can be adjusted independently, frequency scaling by itself is rarely worthwhile as a way to conserve switching power. Most power savings come from dynamic voltage scaling in addition to frequency scaling because of the V^2 component and the fact that modern CPUs are strongly optimized for low voltage states. Dynamic voltage scaling is usually used in conjunction with frequency scaling, as the frequency that a chip may run at is related to the operating voltage. The efficiency of some electrical components, such as voltage regulators, decreases with a temperature increase, so the power usage may increase with higher temperature.

Since increasing power use may raise the temperature, increases in voltage or frequency may raise the system power demand even faster than the CMOS formula indicates, and vice versa. DVFS reduces the number of instructions a processor can issue in a given amount of time, thus reducing the performance. This, in turn, increases the run-time of program segments that are significantly CPU bound. Hence, it creates a challenge of providing the optimal energy/performance control, which has been extensively investigated by scientists in recent years.

Although the application of DVFS may seem to be straightforward, real-world systems raise many complexities that have to be taken into account. First of all, due to complex architectures of modern CPUs (i.e., pipelining, multilevel cache, etc.), the prediction of the required CPU clock frequency that will meet the application performance requirements is not trivial. Another problem is that in contrast to the theory, power consumption by a CPU may not be quadratic to its supply voltage. For example, some architectures may include several supply voltages that power different parts of the chip, and even if one of them is reduced, the overall power consumption will be dominated by the larger supply voltage [13].

Moreover, the execution time of a program running on the CPU may not be inversely proportional to the clock frequency, and DVFS may result in non-linearities in the execution time [26]. For example, if a program is memory or I/O bounded, the CPU speed will not have a dramatic effect on the execution time. Furthermore, slowing down the CPU may lead to changes in the order, in which the tasks are scheduled [117]. In summary, DVFS can provide substantial energy savings; however, it has to be applied carefully, as the result may vary significantly for different hardware and software systems.

Approaches that apply DVFS to reduce energy consumption by a system can be divided into interval-based, inter- and intratask [26]. Interval-based algorithms are similar to adaptive predictive DCD approaches in that they also utilize the knowledge of the past periods of CPU activity [56, 125]. Depending on the CPU utilization during previous intervals, they predict the utilization in the near future and appropriately adjust the voltage and clock frequency. Wierman et al. [128] and Andrew et al. [5] conducted analytical studies of speed scaling algorithms in processor sharing systems. They proved that no online energy-proportional speed scaling algorithm can be better than 2-competitive compared with an optimal offline algorithm. Moreover, they found that sophistication in the design of speed scaling algorithms does not provide significant performance improvements; however, it dramatically improves robustness to errors in the estimation of workload parameters.

Intertask approaches instead of relying on coarse-grained data on the CPU utilization distinguish different tasks running in the system and assign them different speeds [48, 126]. The problem is easily solvable if the workload is known *a priori* or constant over the whole period of task execution. However, the problem becomes non-trivial when the workload is irregular. In contrast to intertask, intratask approaches leverage fine-grained information about the structure of programs and adjust the processor frequency and voltage within the tasks [73, 79]. Such policies can be implemented by splitting a program execution into time slots and assigning different CPU speeds to each of them. Another way to apply such policies is to implement them at the compiler level. This kind of approaches utilizes the compiler's knowledge of the program structure to make inferences about time periods suitable for clock frequency reduction.

Advanced Configuration and Power Interface

Many DPM algorithms, such as timeout-based as well as other predictive and stochastic policies, can be implemented in hardware as a part of an electronic circuit. However, a hardware implementation highly complicates the modification and reconfiguration of the policies. Therefore, there are strong reasons to shift the implementation to the software level. In 1996 to address this problem, Intel, Microsoft, and Toshiba have published the first version of the Advanced Configuration and Power Interface (ACPI) specification – an open standard defining a unified OS-centric device configuration and power management interface. In contrast to previous basic input/output system (BIOS) central, firmware-based, and platform-specific power management systems, ACPI describes platform-independent interfaces for hardware discovery, configuration, power management, and monitoring.

ACPI is an attempt to unify and improve the existing power and configuration standards for hardware devices. The standard brings DPM into the OS control and requires an ACPI-compatible OS to take over the system and have the exclusive control of all aspects of power management and device configuration. The main goals of ACPI are to enable all computing systems to implement DPM capabilities, and simplify and accelerate the development of power-managed systems. It is important to note that ACPI does not impose any constraints on particular power management policies, but provides an interface that can be used by software developers to leverage the flexibility in adjusting the system's power states.

ACPI defines a number of power states that can be enabled in the system at runtime. The most relevant states in the context of DPM are C-states and P-states. C-states are the CPU power states C0-C3 denoting the *Operating State*, *Halt*, *Stop-Clock*, and *Sleep Mode* respectively. In addition, while the processor operates, it can be in one of several power-performance states (P-state). Each of these states designates a particular combination of DVFS settings. P-states are implementation-dependent, but P_0 is always the highest performance state, with P_1 to P_n being successively lower performance states, up to an implementation-specific limit of n and no greater than 16. P-states have become known as SpeedStep in Intel processors, PowerNow!, or Cool'n'Quiet in AMD processors, and PowerSaver in VIA processors. ACPI is widely used by OSes, middleware, and

application-level software for managing power states of the system.

The introduction of ACPI has drastically simplified software power management and resulted in broad research studies in this area. The problem of power-efficient resource management has been investigated in different contexts of device-specific management, OS-level management of virtualized and non-virtualized servers, followed by multiple-node system such as homogeneous and heterogeneous clusters, data centers, and Clouds, as discussed in the following sections.

2.4.2 Operating System Level

This section discusses research works that deal with power-efficient resource management in computing systems at the OS level. The taxonomy of the characteristics used to classify the proposed approaches is presented in Figure 2.6. In particular, the proposed solutions are classified in regard to the following characteristics:

- Application adaption – whether the system requires modifications of the application-level software to take advantage of power management.
- System resources – whether the system focuses on optimizing a single system resource, such as the CPU; or multiple system resources.
- Target systems – whether the approach is general and can be applied to an arbitrary system; or specializes on mobile devices or server hardware.
- Goal – whether the system minimizes power / energy consumption under performance constraints; or manages a power budget, also referred to as *power capping*.
- Power saving techniques – DPS techniques, such as DVFS; DCD techniques; or just making the resources idle without explicit changes in hardware power states, which is referred to as *resource throttling*.
- Workload – whether the system is transparent to the application workload; or focuses on particular types of applications, such as service or HPC applications.

A classification of the reviewed research in regard to the most significant characteristics is shown in Table 2.2.

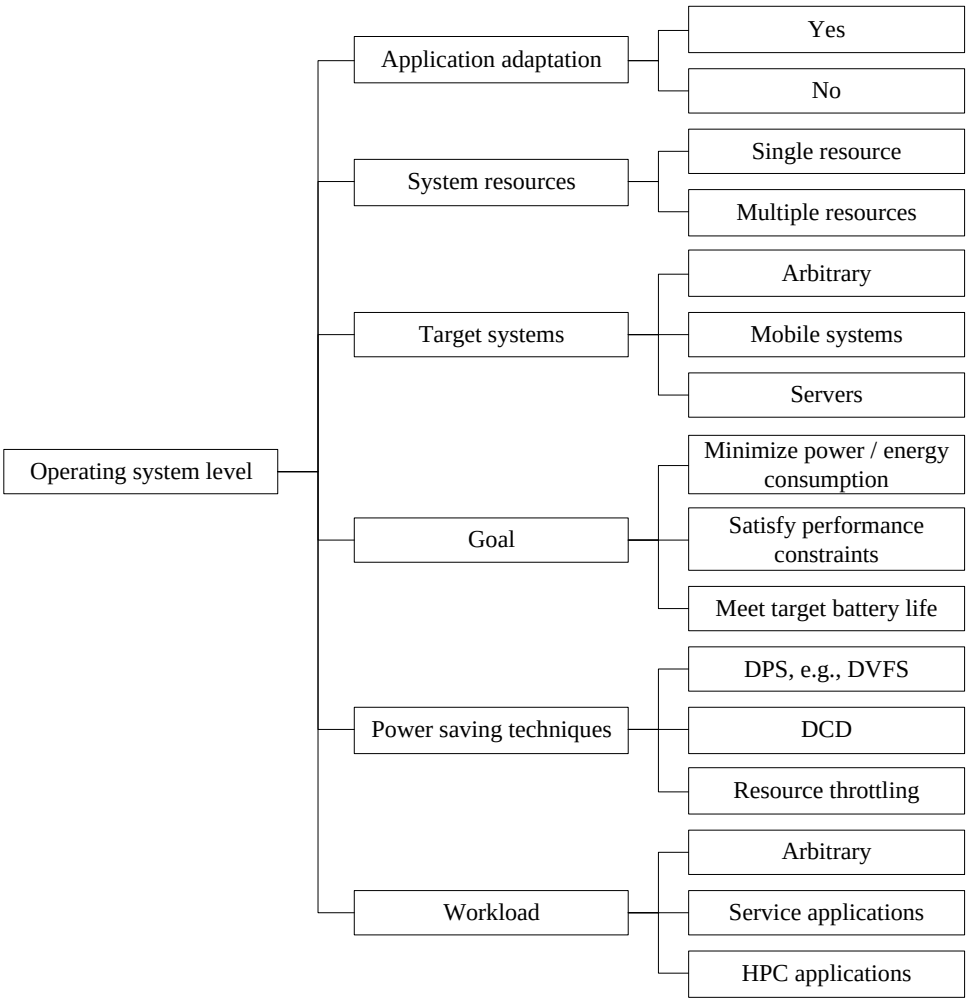


Figure 2.6: The operating system level taxonomy

Table 2.2: Operating system level research

Project name	System resources	Target system	Goal	Power-saving
The on-demand governor [91]	CPU	Arbitrary	Min power under performance constraints	DVFS
ECOsystem [131, 132]	CPU, RAM, disk, network	Mobile systems	Meet the target battery life	Resource throttling
Nemesis OS [87]	CPU, RAM, disk, network	Mobile systems	Meet the target battery life	Resource throttling
GRACE [104, 116]	CPU, network	Mobile systems	Min energy under performance constraints	DVFS, resource throttling
Linux/RK [101]	CPU	Servers	Min energy under performance constraints	DVFS
Coda/Odyssey [49]	CPU, network	Mobile systems	Min energy via application degradation	Resource throttling
PowerNap [81]	CPU, RAM, disk, network	Servers	Min power, min performance loss	DCD
Barely alive memory servers [3]	CPU, disk, network	Servers	Min power, min performance loss	DCD

The On-Demand Governor (Linux Kernel)

Pallipadi and Starikovskiy [91] developed an in-kernel real-time power manager for the Linux OS called the on-demand governor. The manager continuously monitors the CPU utilization multiple times per second and sets a clock frequency and supply voltage pair that corresponds to the current performance requirements keeping the CPU approximately 80% busy to handle fast changes in the workload. The goal of the on-demand governor is to minimize the performance loss due to the reduced CPU frequency.

Modern CPU frequency scaling technologies provide an extremely low transition latency allowing dynamic adjustment of power consumption and performance matching the variable workload demand with almost negligible performance overhead. For example, Enhanced Intel Speedstep Technology enables frequency switching with the latency as low as 10 ms. To various performance requirements, the on-demand governor can be tuned via the specification of the rate, at which the CPU utilization is checked, and the value of the upper utilization threshold, which is by default set to 80%.

The on-demand governor effectively handles multiprocessor SMP systems as well as

multi-core and multi-threading CPU architectures. The governor manages each CPU individually and can manage different cores of the CPU independently if that is supported by the hardware. In cases when different cores of the CPU are dependent on each other in terms of frequency, they are managed together as a single entity. In order to support this design, the on-demand governor sets the frequency of all the cores based on the highest utilization among the cores in the group.

A number of improvements are currently under investigation, including parallel calculation of the utilization and a dedicated work queue. The original governor samples the utilization of all of the processors in the system in a centralized manner, which may become a performance bottleneck with the increase of the number of CPU cores. To overcome this problem, the authors proposed parallel sampling independently for each CPU. Another improvement that can increase the performance for multiprocessor systems is to have dedicated kernel threads for the governor, and implement sampling and frequency adjustment in the context of a particular kernel thread.

ECOsystem

Zeng et al. [131, 132] proposed and developed ECOsystem – a framework for managing energy as a first-class OS resource aimed at battery-powered devices. The fundamental assumption is that applications play an important role in energy distribution opportunities that can be leveraged only at the application level. ECOsystem provides an interface to define a target battery lifetime and application priorities used to determine the amount of energy that will be allocated to the applications at each time frame.

The authors split OS-level energy management into two dimensions. Along the first dimension, there is a variety of system components (e.g., CPU, memory, disk storage, network interface) that consume energy concurrently. The other dimension spans applications that utilize the system components, and thus, cause energy consumption. To address the problem of accounting the energy usage by both components and applications, the authors introduced a new measurement unit called currentcy. One unit of currentcy represents the right to consume a certain amount of energy during a fixed period of time.

When the user sets a target battery lifetime and prioritizes the applications, ECOsystem transforms these data into an appropriate amount of currentcy and determines how

much currentcy should be allocated to each application at each time frame. The length of the timeframe sufficient to achieve smooth energy allocation was empirically determined to be 1 second. An application expends the allocated amount of currentcy by utilizing the CPU, performing disk and memory accesses and consuming other system resources. An application can accumulate currentcy up to a specified limit. When the expenditure of an application exceeds the allocated amount of currentcy, none of the associated processes are scheduled or otherwise serviced. The system is implemented as a modified Linux kernel. The obtained experimental results showed that the proposed model can be effectively used to meet different energy goals, such as achieving a target battery lifetime and proportional energy distribution among competing applications.

Nemesis OS

Neugebauer and McAuley [87] developed the resource-centric Nemesis OS – an OS for battery-powered devices that strives to provide consistent QoS for time-sensitive application, such as multimedia applications. Nemesis provides fine-grained control and accounting for energy usage over all the system resources: CPU, memory, disk, and network bandwidth. To implement per-process resource usage accounting, the OS is vertically structured: most of the system functions, protocol stacks, and device drivers are implemented as user-level shared libraries that execute in application processes. This design allows accurate accounting for energy consumption by individual applications.

The goal of Nemesis is to address the problem of the battery lifetime management. To achieve a target battery lifetime specified by the user, the system relies on the cooperation with applications. If the current energy consumption rate exceeds the threshold that may result in failing to meet the user's expectations, the system charges the applications according to their current energy usage. The applications should interpret the charges as feedback signals and adapt their behavior. The applications are supposed to limit their resource usage according to the data provided by the OS. However, not all application may support adaptation. In this case, the user can prioritize the applications leading to shutting down the low-priority tasks. Nemesis supports a number of platforms including Intel 486, Pentium, Pentium Pro and Pentium II-based PCs, DEC Alpha workstations and evaluation boards, and StrongARM SA-110-based network computers.

The Illinois GRACE Project

Sachs et al. [104, 116] created the Illinois GRACE project (Global Resource Adaptation through CoopEration). They proposed saving energy through coordinated adaptation at multiple system layers according to changes in the application demand for system resources. The authors proposed three levels of adaptation: global, per-application, and internal adaptation. The global adaptation takes into account all the applications running in the system and all the system layers. At this level, the system responds to significant changes in the workload, such as an entry or exit of an application. The per-application adaptation considers each application in isolation and is invoked periodically to adapt all the system resources to the application demand. The internal adaptation focuses on different system resources independently and adapts their states. All the adaptation levels are coordinated in order to ensure adaptation decisions that are efficient.

The framework supports adaptations of the CPU performance (DVFS), applications (frame rate and dithering), and soft CPU scaling (CPU time allocation). The second generation of the framework (GRACE-2) focuses on hierarchical adaptation for mobile multimedia systems. Moreover, it leverages the adaptation of the application behavior depending on the resource constraints. Apart from the CPU adaptation, GRACE-2 enforces network bandwidth constraints and minimizes the network transmission energy. The approach is implemented as a part of the Linux kernel and requires applications to be able to limit their resource usage at run-time in order to leverage the per-application adaptation technique. There is only limited support for legacy applications.

The experimental results showed that the application adaptation provides significant benefits over the global adaptation when the network bandwidth is constrained. Energy savings in a system with the CPU and network adaptations when adding the application adaptation reach 32% (22% on average). Combined CPU and application adaptations were found to result in more than additive energy savings.

Linux/RK

Rajkumar et al. [101] proposed several algorithms for the application of DVFS in real-time systems and implemented a prototype as a modified Linux kernel, Linux/Resource

Kernel (Linux/RK). The objective is to minimize energy consumption, while maintaining the performance isolation between applications. The authors proposed 4 alternative DVFS algorithms that are automatically selected by the system when appropriate.

SystemClock Frequency Assignment (Sys-Clock) is suitable for systems where the overhead of voltage and frequency scaling is too high to be performed at every context switch. A single clock frequency is selected at the admission of an application and kept constant until the set of applications running in the system changes. Priority-Monotonic Clock Frequency Assignment (PM-Clock) is suitable for systems with a low voltage and frequency scaling overhead allowing the adjustment of the voltage and frequency settings at each context switch. Each application is assigned its own constant clock frequency, which is enabled when the application is allocated a CPU time frame. Optimal Clock Frequency Assignment (Opt-Clock) uses a non-linear optimization model to determine the optimal frequency for each application that minimizes energy consumption. Due to high computational complexity, this technique is suitable only for offline usage. Dynamic PMClock (DPM-Clock) suits systems where the average execution time of an application is significantly shorter than the worst case. The experimental results showed that Sys-Clock, PM-Clock, and DPM-Clock provide up to 50% energy savings.

Coda and Odyssey

Flinn and Satyanarayanan [49] explored the problem of managing limited computing resources and battery lifetime in mobile systems, as well as addressing the variability of the network connectivity. They developed two systems: Coda and Odyssey that implement adaptation across multiple system levels. Coda implements application-transparent adaptation in the context of a distributed file system, which does not require any modification of legacy applications.

Odyssey is responsible for initiating and managing application adaptations. This kind of adaptation allows the adjustment of resource consumption by the cost of the output data quality, which is mostly suitable for multimedia applications. For example, in cases of constrained resources video data can be processed or transferred over network in a lower resolution or sound quality can be reduced.

Odyssey introduces a term fidelity that defines the degree to which the output data

corresponds to the original quality. Each application can specify acceptable levels of fidelity that can be requested by Odyssey when the resource usage has to be limited. When Odyssey notifies an application about a change of the resource availability, the application has to adjust its fidelity to match the requested level. The evaluation results showed that this approach allows the extension of the battery lifetime by up to 30%. A limitation of such a system is that all the applications have to be modified in order to support the proposed approach.

PowerNap

Meisner et al. [81] proposed an approach for power conservation in server systems based on fast transitions between active and low-power states. The goal is to minimize power consumption by a server while it is idle. Instead of addressing the problem of achieving energy-proportional computing as proposed by Barroso and Holzle [13], the proposed approach requires only two power states (sleep and fully active) for each system component. The other requirements are fast transitions between the power states and low power consumption in the sleep mode.

To investigate the problem, the authors collected fine-grained utilization traces of several servers serving different workloads. According to the data, the majority of idle periods are shorter than 1 second with the mean length in the order of hundreds of milliseconds, whereas busy periods are even shorter falling below 100 ms for some workloads. The main idea of the proposed approach is to leverage short idle periods that occur due to the workload variability. To estimate the characteristics of the hardware suitable for the proposed technique, the authors constructed a queueing model based on the characteristics of the collected utilization traces. They found that if the transition time is shorter than 1 ms, it becomes negligible and power savings vary linearly with the utilization for all workloads. However, with the growth of the transition time, power savings decrease and the performance penalty becomes higher. When the transition time reaches 100 ms, the relative response time for low utilization can grow up to 3.5 times in comparison to a system without power management, which is unacceptable for real-world systems.

The authors concluded that if the transition time is shorter than 10 ms, power savings are approximately linear to the utilization and significantly outperform the effect

from DVFS for low utilization ($< 40\%$). However, the problem is that the requirement of the transition time being less than 10 ms cannot be satisfied with the current level of technology. According to the data provided by the authors, modern servers can ensure the transition time of 300 ms, which is far from the required 10 ms. The proposed approach is similar to the fixed timeout DCD technique, but adapted to fine-grained application. Therefore, all the disadvantages of the fixed timeout technique are inherited by the approach, that is, a constant performance penalty on wake-ups and an overhead in cases when the idle period is shorter than the transition time to and from the low-power state. The authors reported that if the stated requirements are satisfied, the average server power consumption can be reduced by 74%.

Barely Alive Memory Servers

Anagnostopoulou et al. [3] proposed a family of low-power server states referred to as Barely Alive (BA). The approach is aimed at server environments, where workload consolidation is effective in reducing energy consumption, while fast server reactivation is a necessity. Such systems include web server farms, where the average resource utilization is below 50%; therefore, workload consolidation and switching idle servers off may bring significant energy savings. However, the problem of turning servers off is that the reactivation results in a substantial latency, which is undesirable and may lead to a reduction in the QoS in cases of workload spikes.

With the proposed BA states, instead of completely shutting a server down, most of its components are deactivated, while some are kept on. The BA1-BA5 states range from keeping active only the memory devices, memory controller, and network interface (O(30W) power consumption), to keeping active one or more CPU cores, fans, network interfaces and discs (O(70W) power consumption). The proposed states result in fast transition times back to the active states, i.e., on the order of seconds. Consequently, the lower-power states result in higher transition times and energy consumption. A key advantage of the proposed family of states is that they allow modifications of the data resident in the main memory, while the server is in a low-power state and allowing low-latency reactivation. This enables the system to use all of the cluster's memory, while adjusting the power consumption according to the current workload. The conducted

simulation study revealed that the barely alive states are able to reduce service energy consumption by up to 38% compared with an energy-oblivious system.

2.4.3 Virtualization Level

A technology that is able to improve the utilization of server resources, and thus, reduce power consumption, is virtualization of computing resources. Virtualization introduces an abstraction layer between an OS and hardware. Physical resources can be split into a number of logical slices called Virtual Machines (VMs). Each VM can accommodate an individual OS creating for the user a view of a dedicated physical resource and ensuring the performance and failure isolation between VMs sharing a single physical machine.

Virtualization allows one to create several VMs on a physical server; and therefore, reduce the amount of hardware in use and improve the utilization of resources. The concept originated with the IBM mainframe OSes of the 1960s, but was commercialized for x86-compatible computers only in the 1990s. Several commercial companies and open-source projects now offer software packages to enable the transition to virtual computing. Intel Corporation and AMD have also built proprietary virtualization enhancements to the x86 instruction set to support hardware-assisted virtualization.

Among the benefits of virtualization are improved fault and performance isolation between applications sharing the same compute node (a VM is viewed as a dedicated resource to the user); the ability to relatively easily move VMs from one physical host to another using live or offline migration; and support for hardware and software heterogeneity. The ability to migrate VMs at run-time enables the technique of energy-efficient dynamic VM consolidation applied at the data center level, discussed in the next section.

The proliferation of virtualization has a potential to drive wider adoption of the concept of terminal servers and thin clients, which have also been used in the Green IT practices. In this concept, multiple users connect to a central server over the network using thin clients. While all the computing required by the users is done on the shared physical server, from the user perspective, the interaction is similar to that with a dedicated computing resource. In regard to energy consumption, the advantage of thin clients is that they consume significantly less energy compared with a regular workstation. Thin clients started gaining relevance with the adoption of Software as a Service (SaaS), which

is one of the service models of Cloud computing [28], or Virtual Desktop Infrastructures (VDI) promoted by virtualization software vendors, such as VMware View[3], Citrix XenDesktop[4], and Oracle Virtual Desktop Infrastructure[5].

The virtualization layer lies between the hardware and OS, and is implemented by a Virtual Machine Monitor (VMM). The VMM takes control over the resource multiplexing and manages the allocation of physical resources to the VMs. There are two ways in which a VMM can participate in power management:

1. A VMM can act as a power-aware OS: monitor the overall system performance and appropriately apply DVFS or any DCD techniques to the system components.
2. A VMM can leverage the power management policies applied by the guest OSes using the application-level knowledge, and map power management commands issued by the OSes of different VMs on actual changes in the hardware power state, or enforce system-wide power limits in a coordinated manner.

The following subsections discuss three of the most popular virtualization technology solutions: Xen hypervisor, VMware solutions, and Kernel-based Virtual Machine (KVM). All of these systems support the first described way of power management; however, none allows the coordination of VM specific calls for power state changes. Stoess et al. [111] proposed an approach that utilizes both system-wide power control and fine-grained application-specific power management performed by guest OSes.

The Xen Hypervisor

The Xen hypervisor is an open-source virtualization technology developed collaboratively by the Xen community and engineers from over 20 innovative data center solution vendors [11]. Xen is licensed under the GNU General Public License (GPL2) and available at no charge in both source and object formats. Xen's support for power management is similar to what is provided by the Linux on-demand governor described earlier. Xen supports ACPI P-states implemented in the cpufreq driver [124]. The system periodically measures the CPU utilization, determines the appropriate P-state, and issues a

[3]VMware View. http://www.vmware.com/products/view/
[4]Citrix XenDesktop. http://www.citrix.com/products/xendesktop/
[5]Oracle Virtual Desktop Infrastructure. http://www.oracle.com/us/technologies/virtualization/virtual-desktop-infrastructure/

platform-dependent command to make a change in the hardware power state. Similarly to the Linux power management subsystem, Xen contains four governors:

- Ondemand – chooses the best P-state according to current resource requirements.
- Userspace – sets the CPU frequency specified by the user.
- Performance – sets the highest available clock frequency.
- Powersave – sets the lowest clock frequency.

In addition to P-states, Xen also incorporates the support for C-states (CPU sleep states) [124]. When a physical CPU does not have any task assigned, it is switched to a C-state. When a new request comes, the CPU is switched back to the active state. The issue is to determine which C-state to enter: deeper C-states provide higher energy savings by the cost of a higher transition latency. At the moment, Xen by default switches the CPU into the first C-state, which provides the shortest transition delay. However, the user can specify a C-state to enter. As the CPU wakes up upon receiving load, it always gets an inevitable performance penalty, which corresponds to the fixed timeout DCD policy.

Apart from P- and C-states, Xen also supports offline and live migration of VMs, which can be leveraged by power-aware dynamic VM consolidation algorithms. Migration is used to transfer a VM between physical hosts. Offline migration moves a VM from one host to another by suspending, copying the VM's memory contents, and then resuming the VM on the destination host. Live migration allows transferring a VM without a suspension. From the user side such migration should be inconspicuous. To perform a live migration, both hosts must be running Xen and the destination host must have sufficient resources (e.g., memory capacity) to accommodate the VM after the transmission. At the destination host Xen starts a new VM instance that forms a container for the VM to be migrated. Xen cyclically copies memory pages to the destination host, continuously refreshing the pages that have been updated on the source. When the number of modified pages is not shrinking anymore, it stops the source instance and copies the remaining memory pages. Once the process is completed, the new VM instance is started.

To minimize the migration overhead, the hosts are usually connected to a Network Attached Storage (NAS) or similar storage solution, which eliminates the necessity to copy the disk content. It is claimed that the final phase of live migration (i.e., when both instances are suspended) typically takes approximately 50 ms. Through their ex-

periments on a Xen testbed, Lefèvre and Orgerie [74] have shown that VM live migration has a significant impact on energy consumption. The live migration technology has facilitated the development of various energy conservation dynamic VM consolidation approaches proposed by researchers around the world.

Kernel-based Virtual Machine (KVM)

KVM is open source virtualization software implemented as a module of the Linux kernel [67]. Under this model, Linux works as a hypervisor, while all the VMs are regular processes managed by the Linux scheduler. This approach reduces the complexity of the hypervisor implementation, as scheduling and memory management are handled by the Linux kernel.

KVM supports the S4 (hibernate) and S3 (sleep/stand by) power states[6]. S4 does not require any specific support from KVM: on hibernation, the guest OS dumps the memory state to a hard disk and initiates powering off the computer. The hypervisor translates this signal into termination of the appropriate process. On the next boot, the OS reads the saved memory state from the disk, resumes from the hibernation, and reinitializes all the devices. During the S3 state, memory is kept powered, and thus the content does not need to be saved to a disk. However, the guest OS must save the states of the devices, as they should be restored on a resume. During the next boot, the BIOS should recognize the S3 state, and instead of initializing the devices jump directly to the restoration of the saved device states. Therefore, the BIOS has to provide special support or such behavior.

VMware

VMware ESX Server and VMware ESXi are enterprise-level virtualization solutions offered by VMware, Inc. Similar to Xen, VMware supports host-level power management via DVFS. The system monitors the CPU utilization and continuously applies appropriate ACPI's P-states [120]. VMware VMotion and VMware Distributed Resource Scheduler (DRS) are two other services that operate in conjunction with ESX Server and ESXi [121]. VMware VMotion enables live migration of VMs between physical nodes, which can be

[6]KVM Power Management. http://www.linux-kvm.org/page/PowerManagement

initiated programmatically or manually by system administrators. VMware DRS monitors the resource usage in a pool of servers and uses VMotion to continuously rebalance VMs according to the current workload and load-balancing policy.

VMware DRS contains a subsystem called VMware Distributed Power Management (DPM) to reduce power consumption by a pool of servers by dynamically switching off spare servers [121]. Servers are powered back on when there is a rising demand for resources. VMware DPM utilizes live migration to reallocate VMs keeping the minimal number of servers powered on. VMware ESX Server and VMware ESXi are free for use, whereas other components of VMware Infrastructure have a commercial license.

Energy Management for Hypervisor-based VMs

Stoess et al. [112] proposed a framework for energy management on virtualized servers. Typically, energy-aware OSes assume the full knowledge and control over the underlying hardware, implying device- or application-level accounting for the energy usage. However, in virtualized systems, a hardware resource is shared among multiple VMs. In such an environment, device control and accounting information are distributed across multiple VMs making it infeasible for an OS to take the full control over the hardware. This results in the inability of energy-aware OSes to apply their policies in the system. The authors proposed mechanisms for fine-grained guest OS-level energy accounting and allocation. To encompass the diverse demands for energy management, the authors proposed the use of the notion of energy as the base abstraction in the system, an approach similar to the currentcy model in ECOsystem described earlier.

The prototypical implementation comprises two subsystems: a host-level resource manager and an energy-aware OS. The host-level manager enforces system-wide power limits across VM instances. The power limits can be dictated by a battery or a power generator, or by thermal constraints imposed by the reliability requirements and cooling system capacity. To meet the defined power constraints, the manager determines power limits for each VM and device type that cannot be exceeded. The complementary energy-aware OS is capable of fine-grained application-specific energy management. To enable application-specific energy management, the framework supports accounting and control not only for physical but also for virtual devices. This enables the guest resource

management subsystems to leverage their application-specific knowledge.

The experimental results showed that the prototype is capable of enforcing power limits for energy-aware and energy-unaware guest OSes. Three areas are considered to be important for future work: devices with multiple power states, processors with support for hardware-assisted virtualization, and multi-core CPU architectures.

2.4.4 Data Center Level

This section discusses recent research efforts in the area of power management at the data center level. Although DVFS provides an efficient way of managing power consumption of the CPU, the overall dynamic power range of servers remains narrow. Even if a server is completely idle, it still consumes up to 70% of power, as discussed in Section 2.2.2. Eliminating static power consumption by servers is only possible by switching them off, or to a low-power mode, such as the sleep mode. These circumstances have led to the creation of various data center level solutions aimed at consolidating the workload to fewer physical servers and deactivating the idle servers, which improves the utilization of resources and reduces power / energy consumption.

Workload consolidation is a non-trivial problem since aggressive consolidation may lead to performance degradation of applications. Therefore, consolidation is typically constrained by QoS requirements often defined in terms of Service Level Agreements (SLAs). The following subsections survey various approaches to energy-efficient resource management in non-virtualized and virtualized data centers. The taxonomy of the characteristics used to classify the reviewed approaches is presented in Figure 2.7. In particular, the following characteristics are considered:

- Virtualization – whether the approach leverages virtualization of data center resources. This characteristic is especially important considering the proliferation of Cloud computing.
- System resources – whether the system takes into account the utilization of a single resource, such as the CPU, or multiple system resources.
- Target systems – whether the system is aimed at homogeneous or heterogeneous computing resources. Efficient handling of heterogeneity is crucial in large-scale data centers, as their capacity is typically increased incrementally over time result-

ing in a system of heterogeneous resources.

- Goal – whether the goal of the system is to minimize power / energy consumption under performance constraints; or meet the power budget.
- Power-saving techniques – whether the system applies DPS, such as DVFS; resource throttling; DCD, including switching the power states of whole servers; or workload / VM consolidation to minimize power / energy consumption.
- Workload – whether the system is application-agnostic and able to handle arbitrary workloads; or focuses on particular types of applications, such as services or HPC.
- Architecture – whether the resource management system is centralized requiring a control algorithm to run on a master node, or distributed.

Table 2.3 highlights the most significant characteristics of the reviewed solutions proposed in the literature. The next section discusses implications of Cloud computing, followed by a survey of proposed approaches to energy-efficient resource management in data centers.

Table 2.3: Data center level research

Authors	Virt.	Resources	Goal	Power-saving
Pinheiro et al. [97]	No	CPU, disk, network	Min energy under performance constraints	Workload consolidation, server power switching
Chase et al. [32]	No	CPU	Min energy under performance constraints	Workload consolidation, server power switching
Elnozahy et al. [43]	No	CPU	Min energy under performance constraints	DVFS, server power switching
Chen et al. [33]	No	CPU	Min energy under performance constraints	DVFS, workload consolidation, server power switching
Heath et al. [58]	No	CPU, disk, network	Min energy, max throughput	Workload consolidation, server power switching
Srikantaiah et al. [108]	No	CPU, disk	Min energy under performance constraints	Workload consolidation, server power switching
Gandhi et al. [50]	No	CPU	Meet power budget and min mean execution time	DVFS
Garg et al. [51]	No	CPU	Min energy and CO_2 emissions, max profit	Leveraging heterogeneity, DVFS
Nathuji and Schwan [85,86]	Yes	CPU	Min energy under performance constraints	DFVS, soft scaling, VM consolidation, server power switching
Raghavendra et al. [100]	Yes	CPU	Min power and meet power budget	DVFS, VM consolidation, server power switching
Kusic et al. [72]	Yes	CPU	Min power under performance constraints	VM consolidation, server power switching

Table 2.3: Data center level research (continued)

Authors	Virt.	Resources	Goal	Power-saving
Stillwell et al. [110]	Yes	CPU	Min energy under performance constraints	VM consolidation, resource throttling
Song et al. [106]	Yes	CPU, RAM	Min energy under performance constraints	Resource throttling
Cardosa et al. [31]	Yes	CPU	Min power under performance constraints	DFVS, soft scaling
Verma et al. [119]	Yes	CPU	Min power under performance constraints	DVFS, VM consolidation, server power switching
Gmach et al. [55]	Yes	CPU, RAM	Min energy under performance constraints	VM consolidation, server power switching
Buyya et al. [27, 66]	Yes	CPU	Min energy under performance constraints	Leveraging heterogeneity, DVFS
Kumar et al. [71]	Yes	CPU, RAM, network	Min power under performance and power budget constraints	DVFS, VM consolidation

Implications of Cloud Computing

Traditionally, an organization purchases its own computing resources and deals with the maintenance and upgrades of the hardware and software, resulting in additional expenses. The recently emerged Cloud computing paradigm [28] leverages virtualization and provides the ability to provision resources on-demand on a pay-as-you-go basis. Organizations can outsource their computation needs to the Cloud, thereby eliminating the necessity to maintain their own computing infrastructure. Cloud computing naturally leads to energy efficiency by providing the following characteristics:

- Economy of scale and elimination of redundancies.
- Increased utilization of computing resources.
- Location independence – VMs can be moved to a place where energy is cheaper.
- Scaling up/down and in/out – the resource usage can be adjusted to suit the current requirements.
- Efficient resource management by Cloud providers, which maximize their profit.

Cloud computing has become a very promising paradigm for both consumers and providers in various areas including science, engineering, and not to mention business. A Cloud typically consists of multiple resources possibly distributed and heterogeneous.

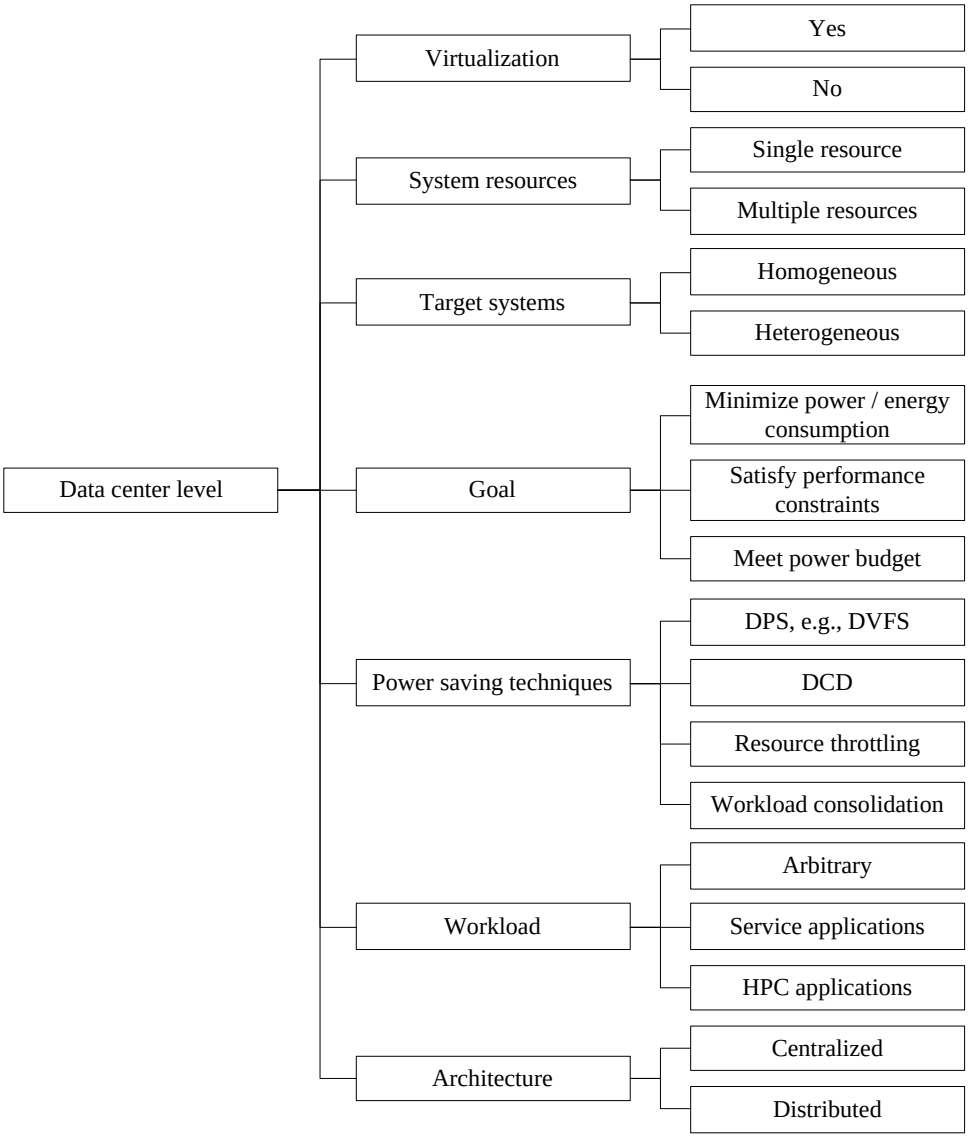


Figure 2.7: The data center level taxonomy

Although the notion of a Cloud has existed in one form or another for some time now (its roots can be traced back to the mainframe era [93]), recent advances in virtualization technologies and the business trend of reducing the TCO in particular have made it much more appealing compared to when it was first introduced. There are many benefits from the adoption and deployment of Clouds, such as scalability and reliability; however, Clouds in essence aim to deliver more economical solutions to both parties (consumers and providers). Economical means that consumers only need to pay per their use and providers can capitalize poorly utilized resources.

From the provider's perspective, the maximization of their profit is the highest priority. In this regard, the minimization of energy consumption plays a crucial role. Recursively, energy consumption can be much reduced by increasing the resource utilization. Large profit-driven Cloud service providers typically develop and implement better power management, since they are interested in taking all necessary means to reduce energy costs to maximize their profit. It has been shown that a reduction in energy consumption by more effectively dealing with resource provisioning (avoidance of resource under/over provisioning) can be obtained [6].

One of the important requirements for a Cloud computing environment is providing reliable QoS. It can be defined in terms of SLAs that describe such characteristics as the minimum allowed throughput, maximum response time, or latency delivered by the deployed system. Although modern virtualization technologies can ensure performance isolation between VMs sharing the same physical node, aggressive consolidation and variability of the workload may result in performance degradation of applications. Performance degradation may lead to increased response times, timeouts, or failures. Therefore, Cloud providers have to deal with the energy-performance trade-off – minimization of energy consumption, while meeting the QoS requirements.

Another problem is that Cloud applications require movements of large data sets between the infrastructure and consumers; thus it is essential to consider both compute and network aspects of the energy efficiency [8, 9]. Energy usage in large-scale computing systems like Clouds yields many other concerns, such as carbon emissions and system reliability. In the following sections it is shown how recent research addresses the mentioned problems.

Load Management for Power and Performance in Clusters

Pinheiro et al. [97] proposed a technique for managing a non-virtualized cluster of physical machines with the objective of minimizing energy consumption, while providing the required QoS. The authors presented a new direction of research as all previous works focused on power efficiency in mobile systems or load balancing in clusters. The main technique to minimize power consumption is load concentration, or unbalancing, while switching idle compute nodes off. The approach requires dealing with the power-performance trade-off, as application performance can be degraded due to consolidation.

The authors used the throughput and execution time of applications as constraints for ensuring the QoS. The nodes are assumed to be homogeneous. The algorithm periodically monitors the workload and decides which nodes should be turned on or off to minimize power consumption by the system, while providing the expected performance. To estimate the performance delivered by the system, the authors applied a notion of demand for resources, where resources include CPU, disk, and network interface. This notion is used to predict performance degradation and throughput due to workload migration based on historical data. To determine the time to add or remove a node, the authors introduced a total demand threshold that is set statically for each node. Additionally, this threshold is intended to solve the problem of the latency caused by a node addition, but may lead to performance degradation in the case of a fast demand growth.

The actual load distribution across active compute nodes is not handled by the system and has to be managed by the applications. The resource management algorithm is executed on a master node that creates a single point of failure and may become a performance bottleneck in a large system. In addition, it is claimed that reconfiguration operations are time-consuming and the implementation of the algorithm adds or removes only one node at a time that may result in slow reaction in large-scale environments.

The authors also investigated the cooperation between the applications and OS in terms of power management decisions. They found that such cooperation can help to achieve more efficient control at the cost of requiring modification of the applications. To evaluate the approach, the authors conducted several experimental studies with two workload types: web applications and compute-intensive applications. The evaluation showed that the approach can be efficiently applied to various workload types.

Managing Energy and Server Resources in Hosting Centers

Chase et al. [32] studied the problem of managing server resources in Internet hosting centers. Servers are shared among multiple service applications with SLAs defined in terms of throughput and latency constraints. The authors developed Muse, an OS for an Internet hosting center aimed at managing and coordinating interactions between a data center's components. The objective is not just to schedule resources efficiently but also to minimize the consumption of electrical power by the system. The proposed approach is applied to reduce: operating costs (power consumption by the computing resources and cooling system); CO_2 emissions, and thus the impact on the environment; thermal vulnerability of the system due to cooling failures or high service load; and over-provisioning in capacity planning. Muse addresses these problems by automatically scaling back the power demand (and therefore waste heat) when appropriate. Such a control over the resource usage optimizes the trade-off between the service quality and price, enabling the support for flexible SLAs negotiated between consumers and the resource provider.

The main challenge is to determine the resource demand of each application at its current request load level, and to allocate resources in the most efficient way. To deal with this problem, the authors applied an economic framework: the system allocates resources in a way that maximizes the "profit" by balancing the cost of each resource unit against the estimated utility, or the "revenue" that is gained from allocating that resource unit to a service. Services "bid" for resources in terms of the volume and quality. This enables negotiation of the SLAs according to the available budget and QoS requirements, i.e., balancing the cost of resource usage (energy cost) and benefit gained due to the usage of this resource. This enables the data center to increase energy efficiency under a fluctuating workload, dynamically match the load and power consumption requirements, and respond gracefully to resource shortages.

The system maintains a set of active servers selected to serve requests for each service. Network switches are dynamically reconfigured to change the active set when necessary. Energy consumption is reduced by switching idle servers to power-saving modes (e.g., sleep, hibernation). The system is targeted at the web workload, which leads to a "noise" in the load data. The authors addressed this problem by applying the statistical "flip-flop" filter, which reduces the number of unproductive reallocations and leads to a more

stable and efficient control.

This work has created a foundation for numerous studies in the area of power-efficient resource management at the data center level; however, the proposed approach has a few weaknesses. The system deals only with the CPU management, but does not take into account other system resources such as memory, disk storage, and network interface. It utilizes APM, which is an outdated standard for Intel-based systems, while currently adopted by industry standard is ACPI. The thermal factor, as well as the latency due to switching physical nodes on/off are not directly taken into account. The authors pointed out that the management algorithm is stable, but it turns out to be relatively expensive during significant changes in the workload. Moreover, heterogeneity of the software configuration requirements is not handled, which can be addressed by virtualization.

Energy-Efficient Server Clusters

Elnozahy et al. [43] explored the problem of power-efficient resource management in a homogeneous cluster serving a single web application with SLAs defined in terms of response time constraints. The motivation for the work is the reduction of operating costs and server overheating. The approach applies two power management mechanisms: switching servers on and off (Vary-On Vary-Off, VOVO) and DVFS.

The authors proposed five resource management policies: Independent Voltage Scaling (IVS), Coordinated Voltage Scaling (CVS), VOVO, combined policy (VOVO-IVS), and coordinated combined policy (VOVO-CVS). The last mentioned policy is claimed to be the most advanced and is provided with a detailed description and mathematical model for determining CPU frequency thresholds. The thresholds define when it is appropriate to turn on an additional physical node or turn off an idle node. The main idea of the policy is to estimate the total CPU frequency required to provide the expected response time, determine the optimal number of physical nodes, and proportionally set their frequency.

The experimental results showed that the proposed IVS policy can provide up to 29% energy savings and is competitive with more complex schemes for some workloads. VOVO policy can produce saving up to 42%, whereas CVS policy in conjunction with VOVO (VOVO-CVS) results in 18% higher savings that are obtained using VOVO independently. However, the proposed approach is limited in the following aspects. The time

required for starting up an additional node is not taken into account in the model. Only a single application is assumed to be running in the cluster, and load balancing is supposed to be done by an external system. Moreover, the algorithm is centralized, which creates a single point of failure and reduces the system scalability. The workload data are not approximated, which can lead to inefficient decisions due to fluctuations in the demand. No other system resources except for the CPU are managed.

Managing Server Energy and Operational Costs in Hosting Centers

Chen et al. [33] proposed an approach to managing multiple server applications in hosting centers for minimizing energy consumption, while meeting SLA requirements. The approach consists in two base phases executed periodically: (1) allocating a number of servers to each application to serve the current workload level; and (2) setting the DVFS parameters on servers suitable for serving the corresponding application's current workload. After each server allocation phase, the servers becoming idle get switched off to conserve energy. One of the distinguishing characteristics of this research is the consideration of SLA requirements in terms of a bound on the response time as an explicit constraint of the optimization problem. The objective of the optimization problem is to minimize the total cost comprising the electricity cost and the cost of the impact of switching servers on / off, which significantly affects the long term reliability of the system, thus reducing the Mean Time Between Failures (MTBF).

The authors addressed the defined problem using a hybrid approach consisting of a queueing theory-based approach and control theoretic approach. The queueing theory-based approach predicts the workload for the near future and tunes the server allocation appropriately. Since it is based on a steady-state analysis, it may not be accurate for fine-grained transient behavior. The feedback-based control theoretic approach is invoked at shorter time intervals and applied to adjust the DVFS settings of the servers at finer granularities. The proposed hybrid scheme is suitable for practical applications, where is it desirable to adjust the server provisioning less frequently due to significant overheads, and perform DVFS control more frequently. The experimental evaluation demonstrated that the proposed approach leads to significant energy savings, while meeting the defined SLA requirements.

Energy Conservation in Heterogeneous Server Clusters

Heath et al. [58] investigated the problem of energy-efficient request distribution in heterogeneous clusters hosting server applications, such as web servers. This is the first research work that considered energy-efficient workload distribution in heterogeneous clusters and leveraged the heterogeneity to achieve additional energy savings. The proposed model is based on the idea of quantifying the performance of heterogeneous servers in terms of the throughput provided by the server resources, e.g., CPU, disk. The server power consumption is estimated according to the utilization of each resource. Next, the application deployed on the cluster is profiled to map the performance requirements of each application request type on the throughput of the server resources. The application-specific tuning allows the system to provide higher energy savings and throughput.

The authors proposed analytical models that use the expected cluster load to predict the overall throughput and power consumption as a function of the request distribution. Simulated annealing is applied to find a request distribution from clients to servers and among servers that minimizes the power / throughput ratio for each workload level. Since the optimization algorithm is time-consuming, it is executed offline to obtain the best request distribution for each workload intensity level. This information is used online by the master node to look up the best request distribution and reconfigure the system. To validate the proposed approach, the authors implemented a web server running on a heterogeneous cluster of traditional and blade servers. The experiments showed that the proposed approach is able to reduce energy consumption by the system by 42% compared with an energy-oblivious system, while resulting in only 0.35% loss in throughput.

Energy-Aware Consolidation for Cloud Computing

Srikantaiah et al. [108] investigated the problem of dynamic consolidation of applications serving small stateless requests in data centers to minimize energy consumption. First of all, the authors explored the impact of workload consolidation on the energy-per-transaction metric depending on both the CPU and disk utilization. The obtained experimental results showed that the consolidation influences the relationship between energy consumption and utilization of resources in a non-trivial manner. The authors

found that energy consumption per transaction results in "U"-shaped curve. When the utilization is low, the resource is not efficiently used leading to a higher cost in terms of the energy-performance metric. However, high resource utilization results in an increased cache miss rate, context switches, and scheduling conflicts leading to high energy consumption due to performance degradation and consequently longer execution time. For the described experimental setup, the optimal points of utilization are at 70% and 50% for the CPU and disk utilization, respectively.

According to the obtained results, the authors stated that the goal of energy-aware workload consolidation is to keep servers well utilized, while avoiding performance degradation caused by high utilization. They modeled the problem as a multi-dimensional bin packing problem, in which servers are represented by bins, and each resource (i.e., CPU, memory, disk, and network) is considered as a dimension of the bin. The bin size along each dimension is defined by the determined optimal utilization level. The applications with known resource utilization are represented by objects with an appropriate size in each dimension. The minimization of the number of bins leads to the minimization of energy consumption by switching idle nodes off.

The authors proposed a heuristic for the defined bin packing problem. The heuristic is based on the minimization of the sum of the Euclidean distances of the current allocations to the optimal point at each server. As a request for execution of a new application is received, the application is allocated to a server using the proposed heuristic. If the capacity of the active servers is fully utilized, a new server is switched on, and all the applications are reallocated using the same heuristic in an arbitrary order.

According to the experimental results, energy used by the proposed heuristic is about 5.4% higher than optimal. The proposed approach is suitable for heterogeneous environments; however, it has several shortcomings. First of all, resource requirements of applications are assumed to be known *a priori* and constant. Moreover, migration of state-full applications between nodes incurs performance and energy overheads, which are not modeled. Switching servers on/off also leads to significant costs that must be considered for a real-world system. Another problem with the approach is the necessity in an experimental study to obtain the optimal points of the resource utilization for each server. Furthermore, the decision of keeping the upper threshold of the resource utilization at

the optimal point is not completely justified as the utilization above the threshold can symmetrically provide the same energy-per-transaction level as lower utilization.

Optimal Power Allocation in Server Farms

Gandhi et al. [50] studied the problem of allocating an available power budget to servers in a heterogeneous server farm to minimize the mean execution time of HPC applications. The authors investigated how CPU frequency scaling techniques affect power consumption. They conducted experiments applying DFS (T-states), DVFS (P-states), and DVFS+DFS (coarse-grained P-states combined with fine-grained T-states) for CPU-intensive workloads. The results showed a linear power-to-frequency relationship for the DFS and DVFS techniques and cubic square relationship for DVFS+DFS.

Given the power-to-frequency relationship, the authors investigated the problem of finding the optimal power allocation as a problem of determining the optimal frequencies of the CPUs of each server, while minimizing the mean execution time. To investigate the effect of different factors on the mean execution time, the authors introduced a queueing model, which allows prediction of the mean response time as a function of the power-to-frequency relationship, arrival rate, peak power budget, and so on. The model allows determining the optimal power allocation for every configuration of the above factors.

The approach was experimentally evaluated against different types of workloads. The results showed that an efficient power allocation can significantly vary for different workloads. To gain the best performance constrained by a power budget, running a small number of servers at their maximum speed is not always optimal. Oppositely, depending on the workload it can be more efficient to run more servers but at lower performance levels. The experimental results showed that efficient power allocation can improve server the farm performance up to a factor of 5 and by a factor of 1.4 on average.

Environment-Conscious Scheduling of HPC Applications

Garg et al. [51] investigated the problem of energy and CO_2 efficient scheduling of HPC applications in geographically distributed Cloud data centers. The aim is to provide HPC users with the ability to leverage high-end computing resources supplied by Cloud

computing environments on demand and on a pay-as-you-go basis. The authors addressed the problem in the context of a Cloud resource provider and presented heuristics for energy-efficient meta-scheduling of applications across heterogeneous resource sites. Apart from reducing the maintenance costs, which results in a higher profit for the resource provider, the proposed approach decreases CO_2 footprints. The proposed scheduling algorithms take into account energy cost, carbon emission rate, workload, and CPU power efficiency, which change across different data centers depending on their location, design, and resource management system.

The authors proposed five scheduling policies: two of which minimize CO_2 emissions, two maximize the profit of resource providers, and a multi-objective policy that minimizes CO_2 emissions and maximizes the profit. The multi-objective policy finds for each application a data center that provides the lowest CO_2 emissions across all the data centers able to complete the application by the deadline. Then from all the application-data center pairs, the policy chooses the one that results in the maximal profit. These steps are repeated until all the applications are scheduled. Energy consumption is also reduced by applying DVFS to all the CPUs in data centers.

The proposed heuristics were evaluated using simulations of different scenarios. The experimental results showed that the energy-centric policies allow the reduction of energy costs by 33% on average. The proposed multi-objective algorithm can be effectively applied when limitations of CO_2 emissions are desired by resource providers or forced by governments. This algorithm leads to a reduction of the carbon emission rate, while maintaining a high level of profit.

VirtualPower: Coordinated Power Management

Nathuji and Schwan [85, 86] investigated the problem of power-efficient resource management in large-scale virtualized data centers. This is the first time when power management techniques were explored in the context of virtualized systems. Besides the hardware scaling and VMs consolidation, the authors apply a new power management technique in the context of virtualized systems called "soft resource scaling". The idea is to emulate hardware scaling by providing a VM less time for utilizing a resource using the VMM's scheduling capability. "Soft" scaling is useful when hardware scaling is not

supported or provides a very small power benefit. The authors found that the combination of "hard" and "soft" scaling may provide higher power savings due to usually limited number of hardware scaling states.

The goals of the proposed approach are support for the isolated and independent operation of guest VMs, and control and coordination of diverse power management policies applied by the VMs to resources. The system intercepts guest VMs' ACPI calls to perform changes in power states, maps them on "soft" states, and uses them as hints for actual changes in the hardware power state. This way, the system supports a guest VM's system level or application level power management policies, while maintaining the isolation between multiple VMs sharing the same physical node.

The authors proposed splitting resource management into local and global policies. At the local level, the system coordinates and leverages power management policies of guest VMs at each physical machine. An example of such a policy is the on-demand governor integrated into the Linux kernel. At this level, the application-level QoS is maintained as decisions about changes in power states are issued the guest OS.

The authors described several local policies aimed at the minimization of power consumption under QoS constraints, and at power capping. The global policies are responsible for managing multiple physical machines using the knowledge of rack- or blade-level hardware characteristics and requirements. These policies consolidate VMs using migration in order to free lightly loaded server and place them into power saving states. The experiments conducted by the authors showed that the usage of the proposed approach leads to efficient coordination of VM and application-specific power management policies, and reduces power consumption up to 34% with little or no performance penalties.

Coordinated Multilevel Power Management

Raghavendra et al. [100] investigated the problem of power management in a data center by combining and coordinating five diverse power management policies. The authors argued that although a centralized solution can be implemented to handle all aspects of power management, it is more likely for a business environment that different solutions from multiple vendors are applied. In this case, it is necessary to solve the problem of coordination between individual controllers to provide correct, stable, and efficient con-

trol. The authors classified existing solutions by a number of characteristics including the objective function, performance constraints, hardware/software, and local/global types of policies. Instead of trying to address the whole space, the authors focused on five individual solutions and proposed five appropriate power management controllers. They applied a feedback control loop to coordinate the controller actions.

The efficiency controller optimizes the average power consumption by individual servers. The controller monitors the utilization of resources and based on these data predicts the future demand and appropriately adjusts the P-state of the CPU. The server manager implements power capping at the server level. It monitors power consumption by the server and reduces the P-state if the power budget is violated. The enclosure manager and the group manager implement power capping at the enclosure and data center level, respectively. They monitor individual power consumption across a collection of machines and dynamically re-provision power across them to maintain the group-level power budget. Power budgets can be defined by system designers based on thermal or power delivery constraints, or by high-level power managers.

The VM controller reduces power consumption across multiple physical nodes by dynamically consolidating VMs and switching idle servers off. The authors provided an integer programming model for the VM allocation optimization problem. However, the proposed model does not provide a protection from unproductive migrations due to workload fluctuations and does not show how SLA can be guaranteed in cases of fast changes in the workload. Furthermore, the transition time for reactivating servers and the ability to handle multiple system resources apart from the CPU are not considered. The authors provided experimental results, which showed the ability of the system to reduce power consumption under different workloads. The authors made an interesting observation: the actual power savings can vary depending on the workload, but "the benefits from coordination are qualitatively similar for all classes of workloads".

Power and Performance Management via Lookahead Control

Kusic et al. [72] explored the problem of power and performance efficient resource management in virtualized data centers. The problem is narrowed to dynamic provisioning of VMs for multi-tiered web applications according to the current workload (number of

incoming requests). The SLAs for each application are defined in terms of the request processing rate. The clients pay for the provided service and receive a refund in a case of SLA violation as a penalty to the resource provider. The objective is to maximize the resource provider's profit by minimizing both power consumption and SLA violation. The problem is defined as a sequential optimization and addressed using the Limited Lookahead Control (LLC). Decision variables are the number of VMs to be provisioned for each service; the CPU share allocated to each VM; the number of servers to switch on or off; and the fraction of the incoming workload to distribute across the servers hosting.

The workload is assumed to be quickly changing, which means that the resource allocation must be adapted over short time periods – "in order of 10 seconds to a few minutes". This requirement makes the high performance of the optimization controller essential. The authors also incorporated in the model the time delays and costs incurred for switching hosts and VMs on/off. Dynamic VM consolidation via offline migration combined with switching hosts on/off are applied as power-saving mechanisms. However, DVFS is not performed due to low-power reduction effect as argued by the authors. The authors applied Kalman filter to estimate the number of future application requests, which is used to predict the future system state and perform necessary reallocations.

The authors provided a mathematical model for the optimization problem. The utility function is risk-aware and includes risks of "excessive switching caused by workload variability" as well as the transient power consumption and opportunity costs. However, the proposed model requires application-specific adjustments though simulation-based learning: the processing rate of VMs with different CPU shares must be known *a priori* for each application. Moreover, due to the complexity of the model, the execution time of the optimization controller reaches 30 min even for a small experimental setup (15 hosts), which is not suitable for large-scale real-world systems. The experimental results show that a server cluster managed using LLC saves 26% in the power consumption costs over a 24 hour period with the SLAs being violated for 1.6% of requests.

Resource Allocation Using Virtual Clusters

Stillwell et al. [110] studied the problem of resource allocation for HPC applications in virtualized homogeneous clusters. The objective is to maximize the resource utilization,

while optimizing a user-centric metric that encompasses both performance and fairness, which is referred to as the yield. The yield of a job is "a fraction of its maximum achievable compute rate that is achieved". A yield of 1 means that the job consumes computational resources at its peak rate. To formally define the basic resource allocation problem as a Mixed Integer Programming (MIP) model, the authors assumed that an application requires only one VM instance; the application's computational power and memory requirements are static and known *a priori*.

However, the solution of the model requires exponential time, and thus can only be obtained for small instances of the problem. The authors proposed several heuristics to solve the problem and evaluated them experimentally across different workloads. The results showed that the multi-capacity bin packing algorithm that sorts tasks in the descending order of their largest resource requirement outperforms or equals to all the other evaluated algorithms in terms of the minimum and average yield, and the failure rate.

Subsequently, the authors relaxed the stated assumptions and considered parallel applications and dynamic workloads. The researchers defined a MIP model for parallel applications and adapted the previously designed heuristics to the new model. Dynamic workloads allow the application of VM migration to address the variability of the workload. To limit the VM migration overhead, the authors fixed the amount of bytes that can be transferred at one time. The authors provided a MIP model for the defined problem; however, no heuristic was proposed to solve large-scale problem instances. Limitations of the proposed approach are that no other system resources except for the CPU are considered in the optimization, and that the resource demands of the applications are assumed to be known *a priori*, which is not typical in practice.

Multi-Tiered On-Demand Resource Scheduling for VM-Based Data Center

Song et al. [106] studied the problem of efficient resource allocation in multi-application virtualized data centers. The objective is to improve the utilization of resources leading to the reduced energy consumption. To ensure the QoS, the resources are allocated to applications proportionally according to the application priorities. Each application can be deployed using several VMs instantiated on different physical nodes. Only the CPU and RAM utilization are taken into account in resource management decisions.

In cases of limited resources, the performance of a low-priority application is intentionally degraded and the resources are allocated to critical applications. The authors proposed scheduling at three levels: the application-level scheduler dispatches requests across the application's VMs; the local level scheduler allocates resources to VMs running on a physical node according to their priorities; and the global-level scheduler controls the resource "flow" between the applications. Rather than applying VM migration to implement the global resource flow, the system pre-instantiates VMs on a group of physical nodes and allocates fractions of the total amount of resources assigned to an application to different VMs.

The authors presented a linear programming model for the resource allocation problem and a heuristic for this model. They provided experimental results for three different applications running in a cluster: a web application, a database, and a virtualized office application showing that the approach satisfies the defined SLAs. One of the limitations of the proposed approach is that it requires machine learning to obtain utility functions for each application. Moreover, it does not utilize VM migration to adapt the VM placement at run-time. The approach is suitable for environments, where applications can have explicitly defined priorities.

Shares- and Utilities-based Power Consolidation

Cardosa et al. [31] investigated the problem of power-efficient VM allocation in virtualized enterprise computing environments. They leveraged the *min*, *max*, and *shares* parameters supported by many modern VM managers. The min and max parameters allow the user to specify the minimum and maximum of CPU time that can be allocated to a VM. The shares parameter determines proportions, in which the CPU time is allocated to VMs sharing the same resource. Such approach suits only environments where VMs have pre-defined priorities.

The authors provided a mathematical formulation of the optimization problem. The objective function includes power consumption and utility gained from the execution of a VM, which is assumed to be known *a priori*. The authors provided several heuristics for the defined model and experimental results. A basic strategy is to place all the VMs at their maximum resource requirements in a first-fit manner and leave 10% of the spare

capacity to handle the future growth of the resource usage. The algorithm leverages the heterogeneity of the infrastructure by sorting physical machines in the increasing order of the power cost per unit of capacity.

The limitations of the basic strategy are that it does not leverage the relative priorities of different VMs, but always allocates a VM at its maximum resource requirements, and uses only 90% of a server's capacity. This algorithm was used as the benchmark policy and was improved upon eventually culminating in the recommended PowerExpandMinMax algorithm. In comparison to the basic policy, this algorithm uses the value of profit that can be gained by allocating an amount of resource to a particular VM. It leverages the ability to shrink a VM to minimum resource requirements when necessary, and expand it when it is allowed by the spare capacity and can bring additional profit. The power consumption cost incurred by each physical server is deducted from the profit to limit the number of servers in use.

The authors evaluated the proposed algorithms by large-scale simulations and experiments on a small data center testbed. The experimental results showed that the PowerExpandMinMax algorithm consistently outperforms the other policies across a broad spectrum of inputs – varying VM sizes and utilities, varying server capacities, and varying power costs. One of the experiments on a real testbed showed that the overall utility of the data center can be improved by 47%. A limitation of this work is that VM migration is not applied to adapt the VM allocation at run-time – the allocation is static. Another problem is that no other system resources except for the CPU are taken into account by the model. Moreover, the approach requires static definition of the application priorities that limits its applicability.

pMapper: Power and Migration Cost Aware Application Placement

Verma et al. [119] investigated the problem of dynamic placement of applications in virtualized systems, while minimizing power consumption and meeting the SLAs. To address the problem, the authors proposed the pMapper application placement framework. It consists of three managers and an arbitrator, which coordinates their actions and makes allocation decisions. Performance Manager monitors the behavior of applications and resizes the VMs according to the current resource requirements and SLAs. Power Manager

is in charge of adjusting hardware power states and applying DVFS. Migration Manager issues instructions for VM live migration to consolidate the workload. Arbitrator has a global view of the system and makes decisions about new placements of VMs and determines the VM reallocations necessary to achieve a new placement. The authors claim that the proposed framework is general enough to be able to incorporate different power and performance management strategies under SLA constraints.

The authors formulated the problem as a continuous optimization: at each time frame, the VM placement is optimized to minimize power consumption and maximize the performance. They made several assumptions to solve the problem, which are justified by experimental studies. The first is performance isolation, which means that a VM can be seen by an application running on that VM as a dedicated physical server with the characteristics equal to the VM parameters. The second assumption is that the duration of a VM live migration does not depend on the background load, and the cost of migration can be estimated based on the VM size and profit decrease caused by an SLA violation. The solution does not focus on specific applications and can be applied to any kind of workload. Another assumption is that the algorithm can minimize power consumption without knowing the actual amount of power consumed by the applications.

The authors presented several algorithms to solve the problem defined as a bin packing problem with variable bin sizes and costs. The bins, items to pack, and bin costs represent servers, VMs, and power consumption of servers, respectively. To solve the bin packing problem, the First-Fit Decreasing (FFD) algorithm was adapted to work for differently sized bins with item-dependent cost functions. The problem was divided into two sub-problems: (1) new utilization values are determined for each server based on the cost functions and required performance; and (2) the applications are placed onto servers to reach the target utilization. This algorithm is called min Power Packing (mPP). The first phase of mPP solves the cost minimization problem, whereas the second phase solves the application placement problem.

mPP was adapted to reduce the migration cost by keeping track of the previous placement, while solving the second phase. This variant is referred to as mPPH. Finally, the placement algorithm was designed that optimizes the power and migration cost trade-off (pMaP). A VM is chosen to be migrated only if the revenue due to the new placement ex-

ceeds the migration cost. pMap searches the space between the old and new placements and finds a placement that minimizes the overall cost (sum of the power and migration costs). The authors implemented the pMapper architecture with the proposed algorithms and performed experiments to validate the efficiency of the approach. The experimental results showed that the approach allows saving about 25% of power relatively to the Static and Load Balanced Placement algorithms. The researchers suggested several directions for future work such as the consideration of memory bandwidth, a more advanced application of idle states, and an extension of the theoretical formulation of the problem.

Resource Pool Management: Reactive Versus Proactive

Gmach et al. [55] studied the problem of energy-efficient dynamic VM consolidation in enterprise environments. The authors proposed a combination of a trace-based workload placement controller and a reactive migration controller. The trace-based workload placement controller collects data on resource usage by VMs instantiated in the data center and uses this historical information to optimize the allocation, while meeting the specified QoS requirements. This controller performs multi-objective optimization by finding a new placement of VMs that minimizes the number of servers needed to serve the workload, while limiting the number of VM migrations required to achieve the new placement. The bound on the number of migrations is assumed to be set by the system administrator depending on the acceptable VM migration overhead. The controller places VMs according to their peak resource usage over the period since the previous reallocation, which is set to 4 hours in the experimental study.

The reactive migration controller continuously monitors the resource utilization of physical nodes and detects when the servers are overloaded or underloaded. In contrast to the trace-based workload placement controller, it acts based on the real-time data on the resource usage and adapts the allocation on a small scale (every minute). The objective of this controller is to rapidly respond to fluctuations in the workload. The controller is parameterized by two utilization thresholds that determine overload and underload conditions. An overload occurs when the utilization of the CPU or memory of the server exceeds the given threshold. An underload occurs when the CPU or memory usage averaged over all the physical nodes falls below the specified threshold. The threshold values

are statically set according to the workload analysis and QoS requirements.

The authors proposed several policies based on different combinations of the described optimization controllers with different utilization thresholds. The simulation-based evaluation using 3 month workload traces from 138 SAP applications showed that the best results can be achieved by applying both the optimization controllers simultaneously. The best policy invokes the workload placement controller every 4 hours, and also when the servers are detected to be lightly utilized. The migration controller is executed in parallel to tackle server underload and overload. The policy provides the minimal CPU violation and requires 10-20% higher CPU capacity than the optimal solution.

GreenCloud: Energy-Efficient and SLA-based Management Cloud Resources

Buyya et al. [27] proposed the GreenCloud project aimed at the development of energy-efficient provisioning of Cloud resources, while meeting QoS requirements defined by the SLAs established through a negotiation between Cloud providers and consumers. The project explored the problem of power-aware allocation of VMs in Cloud data centers for application services based on user QoS requirements such as deadline and budget constraints [66]. The authors introduced a real-time virtual machine model. Under this model, a Cloud provider provisions VMs for requested real-time applications and ensures meeting the specified deadline constraints.

The problem is addressed at several levels. At the first level, a user submits a request to a resource broker for provisioning resources for an application consisting of a set of subtasks with specified CPU and deadline requirements. The broker translates the specified resource requirements into a request for provisioning VMs and submits the request to a number of Cloud data centers. The data centers return the price of provisioning VMs for the request if the deadline requirement can be fulfilled. The broker chooses the data center that provides the lowest price of resource provisioning. The selected data center's VM provisioner instantiates the requested VMs on the physical resources, followed by launching the user applications.

The authors proposed three policies for scheduling real-time VMs in a data center using DVFS to reduce energy consumption, while meeting the deadline constraints and maximizing the request acceptance rate. The LowestDVS policy adjusts the P-state of the

CPU to the lowest level, ensuring that all the real-time VMs meet their deadlines. The δ-Advanced-DVS policy over-scales the CPU speed up to δ% to increase the acceptance rate. The Adaptive-DVS policy uses an M/M/1 queueing model to calculate the optimal CPU speed if the arrival rate and service time of VMs can be estimated in advance.

The proposed approach was evaluated via simulations using the CloudSim toolkit [29]. The simulation results showed that the δ-Advanced-DVS provides the best performance in terms of the profit per unit of the consumed power, as the CPU performance is automatically adjusted according to the system load. The performance of the Adaptive-DVS is limited by the simplified queueing model.

vManage: Loosely Coupled Platform and Virtualization Management in Data Centers

Kumar et al. [71] proposed an approach for dynamic VM consolidation based on an estimation of "stability" – the probability that a proposed VM reallocation will remain effective for some time in the future. The approach implements policies for integrated VM placement considering both VM requirements including CPU, memory, and network constraints, as well as platform requirements, such as power budget. Predictions of future resource demands of applications are computed using a time-varying probability density function. It is assumed that the parameters of the distribution, such as the mean and standard deviation, are known *a priori*.

The authors suggested that the distribution parameter values can be obtained using offline profiling of applications and online calibration. However, offline profiling is unrealistic for Infrastructure as a Service (IaaS) environments, where the provider is not aware of the application deployed in the VMs by the users. Moreover, the authors assume that the resource utilization follows a normal distribution, whereas numerous studies [10, 45, 75] showed that the resource usage by applications is more complex and cannot be modeled using simple probability distributions. The experimental evaluation on a Xen-based infrastructure hosting 28 VMs serving a mixed workload showed that the approach reduces power consumption by 10%, provides 71% less SLA violations, and migrates 54% fewer VMs compared with the benchmark system.

2.5 Thesis Scope and Positioning

This thesis investigates energy-efficient dynamic VM consolidation under QoS constraints applied in virtualized data centers containing heterogeneous physical resources. The goal is to minimize energy consumption by dynamically switching servers to the sleep mode, as energy consumption is the major component of operating costs. Moreover, energy consumption causes CO_2 emissions to the environment, thus reducing energy consumption consequently reduces CO_2 emissions. In addition to dynamically deactivating servers, the approach can be transparently combined with the existing OS level DVFS solutions, such as the on-demand governor of the Linux kernel.

This work focuses on IaaS Cloud environments, e.g., Amazon EC2, where multiple independent users dynamically provision VMs and deploy various types of applications. Moreover, the Cloud provider is not aware of workloads that are executed in the VMs; therefore, the resource management system has to be application agnostic, i.e., able to handle arbitrary workloads. Since multiple types of applications can coexist in the system and share physical resources, there is a challenge of defining QoS requirements in a workload independent manner. In other words, QoS metrics, such as response time and throughput, are unsuitable, as their definition is application-specific. A workload independent QoS metric is required for defining QoS requirements in the SLAs to constrain the degree of VM consolidation and acceptable performance degradation.

To gain the most benefits of dynamic VM consolidation it is necessary to oversubscribe system resources, such as the CPU. This allows the system to leverage fluctuations in the resource consumption by VMs and achieve higher levels of utilization. However, resource oversubscription is risky from the QoS perspective, as it may lead to performance degradation of applications when the resource demand increases.

The approach proposed in this thesis oversubscribes the server CPUs by taking advantage of information on the real-time CPU utilization; however, it does not overcommit RAM. In this work, the maximum amount of RAM that can be consumed by a VM is used as a constraint when placing VMs on servers. One of the reasons for that is that RAM is a more critical resource compared with the CPU, as an application may fail due to insufficient RAM, whereas insufficiency CPU may just slow down the execution of the application. Another reason is that in contrast to the CPU, RAM usually does not be-

Table 2.4: The thesis scope

Characteristic	Thesis scope
Virtualization	Virtualized data centers
System resources	Multiple resources: CPU, RAM
Target systems	Heterogeneous IaaS Clouds
Goal	Minimize energy consumption under performance constraints
Power saving techniques	DVFS, dynamic VM consolidation, server power switching
Workload	Arbitrary mixed workloads
Architecture	Distributed dynamic VM consolidation system

come a bottleneck resource, and therefore, does not limit the number of VMs that can be instantiated on a server, as shown in the literature [4, 107].

Another aspect distinguishing the work presented in this thesis compared with the related research is the distributed architecture of the VM management system. A distributed VM management system is essential for large-scale Cloud providers, as it enables the natural scaling of the system when new compute nodes are added. An illustration of the importance of scalability is the fact that Rackspace, a well-known IaaS provider, has increased the total server count in the second quarter of 2012 to 84,978 up from 82,438 servers at the end of the first quarter [99]. Another benefit of making the VM management system distributed is the improved fault tolerance by eliminating single points of failure: even if a compute or controller node fails, it would not render the whole system inoperable.

There are a few related works reviewed in this chapter that are close to the proposed research direction, which are, however, different in one or more aspects. Approaches to dynamic VM consolidation proposed Kusic et al. [72] and Stillwell et al. [110] are application-specific, whereas the approach proposed in this thesis is application-agnostic, which is suitable for the IaaS model. Verma et al. [119] focused on static and semi-static VM consolidation techniques, as these types of consolidation are easier to implement in an enterprise environment. In contrast, this thesis investigates the problem of dynamic consolidation to take advantage of ne-grained workload variations. Other solutions proposed in the literature are centralized and do not have a direct way of controlling the QoS [55, 71, 86], which are essential characteristics for the next generation data centers and Cloud computing systems. The scope of this thesis is summarized in Table 2.4.

2.6 Conclusions

In recent years, energy efficiency has emerged as one of the most important design requirements for modern computing systems, ranging from single servers to data centers and Clouds, as they continue to consume enormous amounts of electrical power. Apart from high operating costs incurred by computing resources, this leads to significant emissions of CO_2 into the environment. For example, currently, IT infrastructures contribute about 2% of the total CO_2 footprints. Unless energy-efficient techniques and algorithms to manage computing resources are developed, IT's contribution in the world's energy consumption and CO_2 emissions is expected to rapidly grow. This is obviously unacceptable in the age of climate change and global warming. To facilitate further developments in the area, it is essential to survey and review the existing body of knowledge. This chapter presented a taxonomy and survey of various ways to improve power and energy efficiency in computing systems. Recent research advancements have been discussed and classified across the hardware, OS, virtualization, and data center levels.

It has been shown that intelligent management of computing resources can lead to a significant reduction of energy consumption by a system, while still meeting performance requirements. One of the significant advancements that have facilitated the progress in managing compute servers is the implementation of the ability to dynamically adjust the voltage and frequency of the CPU (DVFS), followed by the subsequent introduction and implementation of ACPI. These technologies have enabled the run-time software control over power consumption by the CPU traded for the performance. This chapter presented various approaches to controlling power consumption by hardware from the OS level applying DVFS and other power saving techniques and algorithms.

Virtualization has further advanced the area by introducing the ability to encapsulate the workload in VMs and consolidate multiple VMs to a single physical server, while providing fault and performance isolation between individual VMs. Consolidation has become especially effective after the adoption of multi-core CPUs allowing multiple VMs to be independently executed on a server leading to the improved utilization of resources and reduced energy consumption. Besides consolidation, leading virtualization vendors (i.e., Xen, VMware) similarly to the Linux OS implement continuous DVFS.

The power management problem becomes more complicated when considered at the

data center level. At this level, the system is represented by a set of interconnected compute nodes that need to be managed as a single resource in order to optimize their energy consumption. Efficient resource management is extremely important for data centers and Cloud computing systems comprising multiple compute nodes: due to a low average utilization of resources, the cost of energy consumed by the compute nodes and supporting infrastructure (e.g., cooling systems, power supplies, PDU) leads to an inappropriately high TCO. This chapter classified and discussed a number of recent research works that deal with the problem of energy-efficient resource management in non-virtualized and virtualized data centers.

Due to a narrow dynamic power range of servers, the most efficient power saving technique is consolidation of the workload to fewer physical servers combined with switching the idle servers off. This technique improves the utilization of resources and eliminates the static power consumed by idle servers, which accounts for up to 70% of the power consumed by fully utilized servers. In virtualized environments and Clouds, live and offline VM migration offered by virtualization have enabled the technique of dynamic VM consolidation leveraging workload variability. However, VM migration leads to energy and performance overheads requiring a careful analysis and intelligent techniques to eliminate non-productive migrations.

This chapter has concluded with a discussion of the scope and positioning of the current thesis in the context of the presented taxonomy and reviewed research. The proposed research direction of this thesis is energy-efficient distributed dynamic VM consolidation under performance constraints in IaaS Clouds. Nevertheless, there are many other open research challenges in energy-efficient computing that are becoming even more prominent in the age of Cloud computing – some of them are discussed in Chapter 7.

Chapter 3

Competitive Analysis of Online Algorithms for Dynamic VM Consolidation

Prior to designing new algorithms for dynamic VM consolidation, it is important to attempt a theoretical analysis of potential optimal algorithms. One of the aspects of dynamic VM consolidation is that due to the variability of workloads experienced by modern applications, the VM placement needs to be optimized continuously in an online manner. This chapter formally defines the single VM migration and dynamic VM consolidation problems. To understand the implications of the online nature of the problem, competitive analysis of optimal online deterministic algorithms for the defined problems and proofs of their competitive ratios are conducted and presented.

3.1 Introduction

THIS chapter presents an analysis of the cost and performance characteristics of online algorithms for the problem of energy and performance efficient dynamic VM consolidation. First, the chapter discusses a simplified problem of determining the time to migrate a VM from an oversubscribed host to minimize the cost consisting of the cost of energy consumption and the cost incurred by the Cloud provider due to a violation of the QoS requirements defined in the SLAs. Next, the cost of an optimal offline algorithm for this problem, as well as the competitive ratio of an optimal online deterministic algorithm are determined and proved. Then, a more complex problem of dynamic consolidation of VMs considering multiple hosts and multiple VMs is investigated. The competitive ratio of an optimal online deterministic algorithm for this problem is proved and presented.

As discussed in Chapter 2, most of the related approaches to energy-efficient resource management in virtualized data centers constitute "systems" work focusing more on the

implementation aspect rather than theoretical analysis. However, theoretical analysis of algorithms is important since it provides provable guarantees on the algorithm performance, as well as insights into the future algorithm design.

Recent analytic work on reducing the cost of energy consumption in data centers includes online algorithms for load balancing across geographically distributed data centers [76, 78]. In contrast, the focus of this work is on energy and performance efficient VM management within a data center. Plaxton et al. [98] analyzed online algorithms for resource allocation for a sequence of requests arriving to a data center. Irani et al. [62] proposed and proved the competitiveness of an online algorithm for dynamic power management of a server with multiple power states. Lin et al. [77] proposed a 3-competitive algorithm for request distribution over the servers of a data center to provide power-proportionality, i.e., power consumption by the resources in proportion to the load.

This work differs from the prior analytic literature in the way the system and workload are modeled. Rather than modeling the workload as a sequence of arriving requests, this work is based on an IaaS-like model, where a set of independent long-running applications of different types share the computing resources. Each application generates time-varying CPU utilization and is deployed on a VM, which can be migrated across physical servers transparently for the application. This model is a representation of an IaaS Cloud, where multiple independent users instantiate VMs, and the provider is not aware of the types of applications deployed on the VMs. No results have been found to be published on competitive analysis of online algorithms for the problem of energy and performance efficient dynamic consolidation of VMs in such environments.

In the definition and analysis of the problems in this chapter, it is assumed that future events cannot be predicted based on the knowledge of past events. Although this assumption may not be satisfied for all types of real-world workloads, it enables the theoretical analysis of algorithms that do not rely on predictions of the future workload. Moreover, the higher the variability of the workloads, the closer they are to satisfying the unpredictability assumption. Since Cloud applications usually experience highly dynamic workloads, the unpredictability assumption is justifiable.

The **key contributions** of this chapter are the following.

1. Formal definitions of the single VM migration and dynamic VM consolidation

problems.

2. A proof of the cost incurred by an optimal offline algorithm for the single VM migration problem.
3. Competitive analysis and proofs of the competitive ratios of optimal online deterministic algorithms for the single VM migration and dynamic VM consolidation problems.

The remainder of this chapter is organized as follows. Section 3.2 provides background information on competitive analysis. Sections 3.3 and 3.4 present a thorough analysis of the single VM migration and dynamic VM consolidation problems respectively. The chapter is concluded with a summary and discussion of future research directions in Section 3.5.

3.2 Background on Competitive Analysis

In a real world setting, a control algorithm does not have the complete knowledge of future events, and therefore, has to deal with an *online problem*. According to Borodin and El-Yaniv [25], optimization problems in which the input is received in an online manner and in which the output must be produced online are called *online problems*. Algorithms that are designed for online problems are called *online algorithms*. One of the ways to characterize the performance and efficiency of online algorithms is to apply competitive analysis. In the framework of competitive analysis, the quality of online algorithms is measured relatively to the best possible performance of algorithms that have complete knowledge of the future. An online algorithm ALG is *c-competitive* if there is a constant a, such that for all finite sequences I:

$$ALG(I) \leq c \cdot OPT(I) + a, \tag{3.1}$$

where $ALG(I)$ is the cost incurred by ALG for the input I; $OPT(I)$ is the cost of an optimal offline algorithm for the input sequence I; and a is a constant. This means that for all possible inputs, ALG incurs a cost within the constant factor c of the optimal offline cost plus a constant a. c can be a function of the problem parameters, but it must be independent of the input I. If ALG is c-competitive, it is said that ALG attains a *competitive ratio* c.

In competitive analysis, an online deterministic algorithm is analyzed against the input generated by an omnipotent malicious adversary. Based on the knowledge of the online algorithm, the adversary generates the worst possible input for the online algorithm, i.e. the input that maximizes the competitive ratio. An algorithm's *configuration* is the algorithm's state with respect to the outside world, which should not be confused with the algorithm's internal state consisting of its control and internal memory.

3.3 The Single VM Migration Problem

This section applies competitive analysis [25] to analyze a sub-problem of the problem of energy and performance efficient dynamic consolidation of VMs. There is a single physical server, or host, and M VMs allocated to that host. In this problem the time is discrete and can be split into N time frames, where each time frame is 1 second. The resource provider pays the cost of energy consumed by the physical server. It is calculated as $C_p t_p$, where C_p is the cost of power (i.e. energy per unit of time), and t_p is a time period. The resource capacity of the host and resource usage by VMs are characterized by a single parameter, the CPU performance.

The VMs experience dynamic workloads, which means that the CPU usage by a VM arbitrarily varies over time. The host is oversubscribed, i.e. if all the VMs request their maximum allowed CPU performance, the total CPU demand will exceed the capacity of the CPU. It is defined that when the demand of the CPU performance exceeds the available capacity, a violation of the SLAs established between the resource provider and customers occurs. An SLA violation results in a penalty incurred by the provider, which is calculated as $C_v t_v$, where C_v is the cost of SLA violation per unit of time, and t_v is the time duration of the SLA violation. Since it is necessary to represent the relative difference between C_p and C_v, without loss of generality, the following relations can be defined: $C_p = 1$ and $C_v = s$, where $s \in \mathbb{R}^+$. This is equivalent to defining $C_p = 1/s$ and $C_v = 1$.

At some point in time v, an SLA violation occurs and continues until N. In other words, due to the over-subscription and variability of the workload experienced by VMs, at the time v the overall demand for the CPU performance exceeds the available CPU

capacity and does not decrease until N. It is assumed that according to the problem definition, a single VM can be migrated out from the host. This migration decreases the CPU performance demand and makes it lower than the available CPU capacity.

Let n be the stopping time, which is equal to the latest of either the end of the VM migration or the beginning of the SLA violation. A VM migration takes time T. During a migration an extra host is used to accommodate the VM being migrated, and therefore, the total energy consumed during a VM migration is $2C_pT$. The problem is to determine the time m when a VM migration should be initiated to minimize the total cost consisting of the energy cost and the cost caused by an SLA violation if it takes place. Let r be the remaining time since the beginning of the SLA violation, i.e. $r = n - v$.

3.3.1 The Cost Function

To analyze the problem, a cost function is defined as follows. The total cost includes the cost caused by the SLA violation and the cost of the *extra* energy consumption. The extra energy consumption is the energy consumed by the destination host, where a VM is migrated to, and the energy consumed by the source host after the beginning of the SLA violation. In other words, all the energy consumption is taken into account except for the energy consumed by the source host from t_0 (the starting time) to v. The reason is that this part of energy cannot be eliminated by any algorithm by the problem definition. Another restriction is that the SLA violation cannot occur until a migration starting at t_0 can be finished, i.e. $v > T$. According to the problem statement, the cost function $C(v, m)$ is defined as shown in (3.2).

$$C(v,m) = \begin{cases} (v-m)C_p & \text{if } m < v, v - m \geq T, \\ (v-m)C_p + 2(m-v+T)C_p + (m-v+T)C_v & \text{if } m \leq v, v - m < T, \\ rC_p + (r-m+v)C_p + rC_v & \text{if } m > v. \end{cases} \tag{3.2}$$

The cost function C defines three cases, which cover all possible relationships between v and m. The cases of (3.2) are denoted by C_1, C_2, and C_3 respectively.

1. C_1 describes the case when the migration occurs before the occurrence of the SLA violation ($m < v$), but the migration starts not later than T before the beginning of

the SLA violation ($v - m \geq T$). In this case the cost is just $(v - m)C_p$, i.e. the cost of energy consumed by the extra host from the beginning of the VM migration to the beginning of the potential SLA violation. There is no cost of SLA violation, as according to the problem statement the stopping time is exactly the beginning of the potential SLA violation, so the duration of the SLA violation is 0.

2. C_2 describes the case when the migration occurs before the occurrence of the SLA violation ($m \leq v$), but the migration starts later than T before the beginning of the SLA violation ($v - m < T$). C_2 contains three terms: (a) $(v - m)C_p$, the cost of energy consumed by the extra host from the beginning of the migration to the beginning of the SLA violation; (b) $2(m - v + T)C_p$, the cost of energy consumed by both the main host and the extra host from the beginning of the SLA violation to n; (c) $(m - v + T)C_v$, the cost of the SLA violation from the beginning of the SLA violation to the end of the VM migration.
3. C_3 describes the case when the migration starts after the beginning of the SLA violation. In this case the cost consists of three terms: (a) rC_p, the cost of energy consumed by the main host from the beginning of the SLA violation to n; (b) $(r - m + v)C_p$, the cost of energy consumed by the extra host from the beginning of the VM migration to n; (c) rC_v, the cost of SLA violation from the beginning of the SLA violation to n.

The next section presents analysis of the cost of an optimal offline algorithm for the single VM migration problem based on the defined cost function.

3.3.2 The Cost of an Optimal Offline Algorithm

Theorem 3.1. *An optimal offline algorithm for the single VM migration problem incurs the cost of* $\frac{T}{s}$*, and is achieved when* $\frac{v-m}{T} = 1$.

Proof. To find the cost incurred by an optimal offline algorithm, the range of the cost function for the domain of all possible algorithms is analyzed. The quality of an algorithm for this problem depends of the relation between v and m, i.e. on the difference between the time when the VM migration is initiated by the algorithm and the time when the SLA violation starts. It is possible to define $v - m = aT$, where $a \in \mathbb{R}$. Therefore, $m = v - aT$, and $a = \frac{v-m}{T}$. Further, the three cases defined by the cost function (3.2) are analyzed.

1. $m < v, v - m \geq T$. Thus, $aT \geq T$ and $a \geq 1$. By the substitution of $m = v - aT$ in the first case of (3.2), (3.3) is obtained.

$$C_1(v, a) = (v - v + aT)C_p = aTC_p \tag{3.3}$$

2. $m \leq v, v - m < T$. Thus, $a \geq 0$ and $aT < T$. Therefore, $0 \leq a < 1$. By the substitution of $m = v - aT$ in the second case of (3.2), (3.4) is obtained.

$$\begin{aligned} C_2(v, a) &= (v - v + aT)C_p + 2(v - aT - v + T)C_p + (v - aT - v + T)C_v \\ &= aTC_p + 2T(1 - a)C_p + T(1 - a)C_v \\ &= T(2 - a)C_p + T(1 - a)C_v \end{aligned} \tag{3.4}$$

3. $m > v$. Thus, $a < 0$. By simplifying the third case of (3.2), (3.5) is obtained.

$$\begin{aligned} C_3(v, m) &= rC_p + (r - m + v)C_p + rC_v \\ &= (2r - m + v)C_p + rC_v \end{aligned} \tag{3.5}$$

   For this case, r is the time from the beginning of the SLA violation to the end of the migration. Therefore, $r = m - v + T$. By the substitution of m, $r = T(1 - a)$ is obtained. By the substitution of m and r in (3.5), (3.6) is derived.

$$\begin{aligned} C_3(v, a) &= (2T - 2aT - v + aT + v)C_p + T(1 - a)C_v \\ &= T(2 - a)C_p + T(1 - a)C_v \\ &= C_2(v, a) \end{aligned} \tag{3.6}$$

Since $C_3(v, a) = C_2(v, a)$, the function can be simplified to just two case. Both cases are linear in a and do not depend on v (3.7).

$$C(a) = \begin{cases} T(2 - a)C_p + T(1 - a)C_v & \text{if } a < 1, \\ aTC_p & \text{if } a \geq 1. \end{cases} \tag{3.7}$$

According to the problem definition, the following substitutions can be made: $C_p = 1/s$ and $C_v = 1$ (3.8).

$$C(a) = \begin{cases} \frac{T(2-a)}{s} + T(1-a) & \text{if } a < 1, \\ \frac{aT}{s} & \text{if } a \geq 1. \end{cases} \tag{3.8}$$

It is clear that (3.8) reaches its minimum $\frac{T}{s}$ at $a = 1$, i.e. when $m = v - T$. This solution corresponds to an algorithm that always initiates the VM migration exactly at $m = v - T$. Such an algorithm must have perfect knowledge of the time when the SLA violation will occur before it actually occurs. An algorithm that satisfies this requirement is an optimal offline algorithm for the single VM migration problem.

□

3.3.3 An Optimal Online Deterministic Algorithm

The analysis of the single VM migration problem is continued with finding an optimal online deterministic algorithm and its competitive ratio.

Theorem 3.2. *The competitive ratio of an optimal online deterministic algorithm for the single VM migration problem is* $2 + s$*, and the algorithm is achieved when* $m = v$*.*

Proof. Using the cost function found in Theorem 3.1, the competitive ratio of any online algorithm is defined as in (3.9).

$$\frac{ALG(I)}{OPT(I)} = \begin{cases} \frac{T(2-a)+sT(1-a)}{s} \cdot \frac{s}{T} = 2 + s - a(1+s) & \text{if } a < 1, \\ \frac{aT}{s} \cdot \frac{s}{T} = a & \text{if } a \geq 1, \end{cases} \tag{3.9}$$

where $a = \frac{v-m}{T}$. The configuration of any online algorithm for the single VM migration problem is the current time i; the knowledge of whether an SLA violation is in place; and v if $i \geq v$. Therefore, there are two possible classes of online deterministic algorithms for this problem:

1. Algorithms ALG_1 that define m as a function of i, i.e. $m = f(i)$ and $a = \frac{v-f(i)}{T}$.
2. Algorithms ALG_2 that define m as a function of v, i.e. $m = g(v)$ and $a = \frac{v-g(v)}{T}$.

For algorithms from the first class, a can grow arbitrarily large, as m is not a function of v, and the adversary will select v such that it is infinitely greater than $f(i)$. As $a \to \infty$, $\frac{ALG_1(I)}{OPT(I)} \to \infty$; therefore, all algorithms from the first class are not competitive.

For the second class, $m \geq v$, as m is a function of v, and v becomes known for an online algorithm when $i = v$. Therefore $\frac{ALG_2(I)}{OPT(I)} = 2 + s - a(1 + s)$, where $a \leq 0$. The minimum competitive ratio of $2 + s$ is obtained at $a = 0$. Thus, an optimal online deterministic algorithm for the single VM migration problem is achieved when $a = 0$, or equivalently $m = v$, and its competitive ratio is $2 + s$.

□

An optimal online deterministic algorithm for the single VM migration problem can be implemented by monitoring the state of the host and migrating a VM as soon as an SLA violation is detected.

3.4 The Dynamic VM Consolidation Problem

This section analyzes a more complex problem of dynamic VM consolidation considering multiple hosts and multiple VMs. For this problem, it is defined that there are n homogeneous hosts, and the capacity of each host is A_h. Although VMs experience variable workloads, the maximum CPU capacity that can be allocated to a VM is A_v. Therefore, the maximum number of VMs allocated to a host when they demand their maximum CPU capacity is $m = \frac{A_h}{A_v}$. The total number of VMs is nm. VMs can be migrated between hosts using live migration with a migration time t_m. As for the single VM migration problem defined in Section 3.3, an SLA violation occurs when the total demand for the CPU performance exceeds the available CPU capacity A_h. The cost of power is C_p, and the cost of SLA violation per unit of time is C_v. Without loss of generality, the following relations can be defined: $C_p = 1$ and $C_v = s$, where $s \in \mathbb{R}^+$. This is equivalent to defining $C_p = 1/s$ and $C_v = 1$. It is assumed that when a host is idle, i.e. there are no allocated VMs, it is switched off and consumes no power, or switched to the sleep mode with negligible power consumption. Non-idle hosts are referred to as active. The total cost C is defined as follows:

$$C = \sum_{t=t_0}^{T} \left(C_p \sum_{i=0}^{n} a_{ti} + C_v \sum_{j=0}^{n} v_{tj} \right), \tag{3.10}$$

where t_0 is the initial time; T is the total time; $a_{ti} \in \{0, 1\}$ indicating whether the host i

is active at the time t; $v_{tj} \in \{0, 1\}$ indicating whether the host j is experiencing an SLA violation at the time t. The problem is to determine what time, which VMs and where should be migrated to minimize the total cost C.

3.4.1 An Optimal Online Deterministic Algorithm

Theorem 3.3. *The upper bound of the competitive ratio of an optimal online deterministic algorithm for the dynamic VM consolidation problem is* $\frac{ALG(I)}{OPT(I)} \leq 1 + \frac{ms}{2(m+1)}$.

Proof. Similarly to the single VM migration problem, an optimal online deterministic algorithm for the dynamic VM consolidation problem migrates a VM from a host when an SLA violation occurs at this host. The algorithm always consolidates VMs to the minimum number of hosts, ensuring that the allocation does not cause an SLA violation. The omnipotent malicious adversary generates the CPU demand by VMs in a way that cause as much SLA violation as possible , while keeping as many hosts active (consuming energy) as possible.

As $mA_v = A_h$, for any $k > m$, $k \in \mathbb{N}$, $kA_v > A_h$. In other words, an SLA violation occurs at a host when at least $m + 1$ VMs are allocated to this host, and these VMs demand their maximum CPU capacity A_v. Therefore, the maximum number of hosts that experience an SLA violation simultaneously n_v is defined as in (3.11).

$$n_v = \left\lfloor \frac{nm}{m+1} \right\rfloor. \tag{3.11}$$

In a case of a simultaneous SLA violation at n_v hosts, the number of hosts not experiencing an SLA violation is $n_r = n - n_v$. The strategy of the adversary is to make the online algorithm keep all the hosts active all the time and make n_v hosts experience an SLA violation half of the time. To show how this is achieved, the time is split into periods of length $2t_m$. Then $T - t_0 = 2t_m\tau$, where $\tau \in \mathbb{R}^+$. Each of these periods can be split into two equal parts of length t_m. For these two parts of each period, the adversary acts as follows:

1. During the first t_m, the adversary sets the CPU demand by the VMs in a way to allocate exactly $m + 1$ VMs to n_v hosts by migrating VMs from n_r hosts. As the VM migration time is t_m, the total cost during this period of time is $t_m n C_p$, as all

the hosts are active during migrations, and there is no SLA violation.

2. During the next t_m, the adversary sets the CPU demand by the VMs to the maximum causing an SLA violation at n_v hosts. The online algorithm reacts to the SLA violation, and migrates the necessary number of VMs back to n_r hosts. During this period of time, the total cost is $t_m(nC_p + n_vC_v)$, as all the hosts are again active, and n_v hosts are experiencing an SLA violation.

Therefore, the total cost during a time period $2t_m$ is defined as follows:

$$C = 2t_m nC_p + t_m n_v C_v. \tag{3.12}$$

This leads to the following total cost incurred by an optimal online deterministic algorithm (ALG) for the input I:

$$ALG(I) = \tau t_m(2nC_p + n_vC_v). \tag{3.13}$$

An optimal offline algorithm for this kind of workload will just keep m VMs at each host all the time without any migrations. Thus, the total cost incurred by an optimal offline algorithm is defined as shown in (3.14).

$$OPT(I) = 2\tau t_m nC_p. \tag{3.14}$$

Having determined both costs, the competitive ratio of an optimal online deterministic algorithm can be derived (3.15).

$$\frac{ALG(I)}{OPT(I)} = \frac{\tau t_m(2nC_p + n_vC_v)}{2\tau t_m nC_p} = \frac{2nC_p + n_vC_v}{2nC_p} = 1 + \frac{n_vC_v}{2nC_p}. \tag{3.15}$$

Via the substitution of $C_p = 1/s$ and $C_v = 1$, (3.16) is obtained.

$$\frac{ALG(I)}{OPT(I)} = 1 + \frac{n_v s}{2n}. \tag{3.16}$$

First, consider the case when $\quad \bmod \frac{nm}{m+1} = 0$, and thus $n_v = \frac{nm}{m+1}$. For this case ($ALG_1(I)$) the competitive ratio is shown in (3.17).

$$\frac{ALG_1(I)}{OPT(I)} = 1 + \frac{nms}{2n(m+1)} = 1 + \frac{ms}{2(m+1)}. \tag{3.17}$$

The second case ($ALG_2(I)$) is when $\mod \frac{nm}{m+1} \neq 0$. Then, due to the remainder, n_v is less than in the first case. Therefore, the competitive ratio is defined as in (3.18).

$$\frac{ALG_2(I)}{OPT(I)} < 1 + \frac{ms}{2(m+1)}. \tag{3.18}$$

If both cases are combined, the competitive ratio can be defined as in (3.19), which is an upper bound on the competitive ratio of an optimal online deterministic algorithm for the dynamic VM consolidation problem.

$$\frac{ALG(I)}{OPT(I)} \leq 1 + \frac{ms}{2(m+1)}. \tag{3.19}$$

□

3.5 Conclusions

This chapter presented proofs of competitive ratios of online deterministic algorithms for the single VM migration and dynamic VM consolidation problems. However, it is known that non-deterministic, or randomized, online algorithms typically improve upon the quality of their deterministic counterparts [39]. Therefore, it can be expected that the competitive ratio of online randomized algorithms for the single VM migration problem (Section 3.3), which falls back to an optimal online deterministic algorithm when $i \geq v$, lies between $\frac{T}{s}$ and $2 + s$. Similarly, it can be expected that the competitive ratio of online randomized algorithms for the dynamic VM consolidation problem should be improved relatively to the upper bound determined in Theorem 3.3. In competitive analysis, randomized algorithms are analyzed against different types of adversaries than the omnipotent malicious adversary used for deterministic algorithms. For example, one of these adversaries is the oblivious adversary that generates a complete input sequence prior to the beginning of the algorithm execution. It generates an input based on knowledge of probability distributions used by the algorithm.

Another approach to analyzing randomized algorithms is finding the average-case performance of an algorithm based on distributional models of the input. However, in a real world setting, the workload experienced by VMs is more complex and cannot

be modeled using simple statistical distributions [10]. For example, it has been shown that web workloads have such properties as correlation between workload attributes, non-stationarity, burstiness, and self-similarity [45]. Job arrival times in Grid and cluster workloads have been identified to exhibit such patterns as pseudo-periodicity, long range dependency, and multifractal scaling [75]. In other words, focusing on real-world workloads requires lifting the unpredictability assumption stated in the introduction, as in practice often there are interrelations between subsequent workload states.

The next chapter presents adaptive algorithms that rely on statistical analysis of historical data of the workload to leverage the property of workload predictability. One of the assumptions is that workloads are not completely random, and future events can be predicted based on the past data. However, such algorithms cannot be analyzed using simple distributional or adversary models, such as oblivious adversary, as realistic workloads require more complex modeling, e.g. using Markov chains [105]. A workload model based on Markov chains is investigated in Chapter 5.

Chapter 4

Heuristics for Distributed Dynamic VM Consolidation

This chapter presents a distributed approach to energy and performance efficient dynamic VM consolidation. In this approach, resource managers deployed on the compute hosts locally determine when and which VMs to migrate from the hosts in cases of underload and overload conditions, whereas the placement of VMs selected for migration is done by a global manager, which can potentially be replicated. The chapter continues with an introduction of heuristics for dynamic VM consolidation based on the proposed approach, which significantly reduce energy consumption, while ensuring a high level of adherence to Service Level Agreements (SLAs). The high efficiency of the proposed algorithms is shown by extensive simulations using workload traces from more than a thousand PlanetLab VMs.

4.1 Introduction

AS indicated in the previous chapter, real-world workloads exhibit the properties of correlation between workload attributes, self-similarity, and long range dependency [45, 75], which enable forecasting of future system states based on the knowledge of the observed past states. This chapter presents a set of heuristics for the problem of energy and performance efficient dynamic VM consolidation, which apply statistical analysis of the observed history of system behavior to infer potential future states. The proposed algorithms consolidate and deconsolidate VMs when needed to minimize energy consumption by computing resources under QoS constraints.

The target compute environment is an Infrastructure as a Service (IaaS), e.g., Amazon EC2, where the provider is unaware of applications and workloads served by the VMs, and can only observe them from outside. Due to this property, IaaS environments are referred as being application-agnostic. The proposed approach to dynamic VM consoli-

dation consists in splitting the problem into 4 sub-problems:

1. Deciding if a host is considered to be underloaded, so that all VMs should be migrated from it, and the host should be switched to a low-power mode.
2. Deciding if a host is considered to be overloaded, so that some VMs should be migrated from it to other active or reactivated hosts to avoid violating the QoS requirements.
3. Selecting VMs to migrate from an overloaded host.
4. Placing VMs selected for migration on other active or reactivated hosts.

This approach has two major advantages compared with traditional VM consolidation algorithms discussed in Chapter 2: (1) splitting the problem simplifies the analytic treatment of the sub-problems; and (2) the approach can be implemented in a distributed manner by executing the underload / overload detection and VM selection algorithms on compute hosts, and the VM placement algorithm on replicated controller hosts. Distributed VM consolidation algorithms enable the natural scaling of the system when new compute hosts are added, which is essential for large-scale Cloud providers.

An illustration of the importance of scalability is the fact that Rackspace, a well-known IaaS provider, has increased the total server count in the second quarter of 2012 to 84,978 up from 82,438 servers at the end of the first quarter [99]. Another benefit of making VM consolidation algorithms distributed is the improved fault tolerance by eliminating single points of failure: even if a compute or controller host fails, it would not render the whole system inoperable.

In contrast to the studies discussed in Chapter 2, the proposed heuristics efficiently implement dynamic VM consolidation in a distributed manner according to the current utilization of resources applying live migration, switching idle nodes to the sleep mode, and thus, minimizing energy consumption. The proposed approach can effectively adhere to strict QoS requirements, as well as handle multi-core CPU architectures, heterogeneous infrastructure and heterogeneous VMs.

The proposed algorithms are evaluated by extensive simulations using the CloudSim simulation toolkit [29] and data on the CPU utilization by more than a thousand PlanetLab VMs collected every 5 minutes during 10 randomly selected days in March and April 2011 [92]. According to the results of experiments, the proposed algorithms significantly

reduce energy consumption, while providing a high level of adherence to the SLAs.

The **key contributions** of this chapter are the following.

1. The introduction of a distributed approach to energy and performance efficient dynamic VM consolidation.
2. Novel heuristics for the problem of energy and performance efficient dynamic VM consolidation following the introduced distributed approach.
3. An extensive simulation-based evaluation and performance analysis of the proposed algorithms.

The remainder of the paper is organized as follows. The next section introduces the system model used in the design of heuristics for dynamic VM consolidation. The proposed heuristics are presented in Section 4.3, followed by an evaluation and analysis of the obtained experimental results in Section 4.4. The chapter is concluded with Section 4.5 providing a summary of results and contributions.

4.2 The System Model

The target system is an IaaS environment, represented by a large-scale data center consisting of N heterogeneous physical nodes. Each node i is characterized by the CPU performance defined in Millions Instructions Per Second (MIPS), amount of RAM and network bandwidth. The servers do not have direct-attached storage, while the storage is provided by a Network Attached Storage (NAS) or Storage Area Network (SAN) to enable VM live migration. The type of the environment implies no knowledge of application workloads and time for which VMs are provisioned. In other words, the resource management system must be application-agnostic.

Multiple independent users submit requests for provisioning of M heterogeneous VMs characterized by requirements to the processing power defined in MIPS, amount of RAM and network bandwidth. The fact that the VMs are owned and managed by independent users implies that the resulting workload created by consolidating multiple VMs on a single physical node is mixed. The mixed workload is formed by combining various types of applications, such as HPC and web-applications, which utilize the resources simultaneously. The users establish SLAs with the resource provider to formalize

the QoS requirements. The provider pays a penalty in cases of SLA violations.

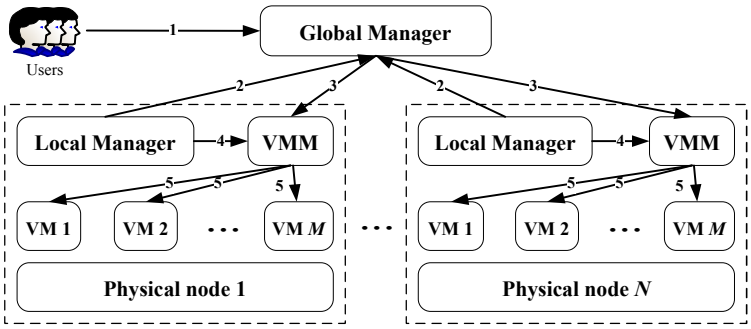


Figure 4.1: The system model

As mentioned earlier, the approach to dynamic VM consolidation proposed in this chapter follows a distributed model, where the problem is divided into 4 sub-problems:

1. Host underload detection.
2. Host overload detection.
3. VM selection.
4. VM placement.

Splitting the problems improves the scalability of the system, as the host underload / overload detection and VM placement algorithms are executed locally by each compute host. It follows that the software layer of the system is tiered comprising local and global managers (Figure 4.1). The local managers reside on each node as a module of the VMM. Their objective is the continuous monitoring of the node's CPU utilization, and detecting host underload and overload conditions (4).

In case of a host overload, the local manager running on the overloaded host initiates the configured VM selection algorithm to determine which VMs to offload from the host. The global manager resides on the master node and collects information from the local managers to maintain the overall view of the system's resource utilization (2). Based on the decisions made by the local managers, the global manager issues VM migration commands to optimize the VM placement (3). VMMs perform actual migration of VMs as well as changes in power modes of the nodes (5).

4.2.1 Multi-Core CPU Architectures

It is assumed that physical servers are equipped with multi-core CPUs. A multi-core CPU with n cores each having m MIPS is modeled as a single-core CPU with the total capacity of nm MIPS. This is justified since applications, as well as VMs, are not tied down to processing cores and can be executed on an arbitrary core using a time-shared scheduling algorithm. The only limitation is that the capacity of each virtual CPU core allocated to a VM must be less or equal to the capacity of a single physical CPU core. The reason is that if the CPU capacity required for a virtual CPU core is higher than the capacity of a single physical core, then a VM must be executed on more than one physical core in parallel. However, automatic parallelization of VMs with a single virtual CPU cannot be assumed.

4.2.2 The Power Model

Power consumption by computing nodes in data centers is mostly determined by the CPU, memory, disk storage, power supplies and cooling systems [83]. As discussed in Chapter 2, recent studies [44], [72] have shown that power consumption by servers can be accurately described by a linear relationship between the power consumption and CPU utilization, even when Dynamic Voltage and Frequency Scaling (DVFS) is applied. The reason lies in the limited number of states that can be set to the frequency and voltage of a CPU and the fact that voltage and performance scaling is not applied to other system components, such as memory and network interfaces.

However, due to the proliferation of multi-core CPUs and virtualization, modern servers are typically equipped with large amounts of memory, which begins to dominate the power consumption by a server [83]. This fact combined with the difficulty of modeling power consumption by modern multi-core CPUs makes building precise analytical models a complex research problem. Therefore, instead of using an analytical model of power consumption by a server, this work utilizes real data on power consumption provided by the results of the SPECpower benchmark[1].

Two server configurations with dual-core CPUs published in February 2011 have been

[1]The SPECpower benchmark. http://www.spec.org/power_ssj2008/

Table 4.1: Power consumption by the selected servers at different load levels in Watts

Server	**0%**	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
HP ProLiant G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP ProLiant G5	93.7	97	101	105	110	116	121	125	129	133	135

selected: HP ProLiant ML110 G4 (Intel Xeon 3040, 2 cores $\times$ 1860 MHz, 4 GB), and HP ProLiant ML110 G5 (Intel Xeon 3075, 2 cores $\times$ 2660 MHz, 4 GB). The configuration and power consumption characteristics of the selected servers are shown in Table 4.1. The reason why servers with more cores were not chosen is that it is important to simulate a large number of servers to evaluate the effect of VM consolidation. Thus, simulating less powerful CPUs is advantageous, as lighter workload is required to overload a server. Nevertheless, dual-core CPUs are sufficient to show how multi-core CPUs are handled by the proposed algorithms.

4.2.3 The Cost of VM Live Migration

Live migration of VMs allows transferring a VM between physical nodes without suspension and with a short downtime. However, live migration has a negative impact on the performance of applications running in a VM during a migration. Voorsluys et al. have performed an experimental study to investigate the value of this impact and find a way to model it [122]. They found that performance degradation and downtime depend on the application behavior, i.e., how many memory pages the application updates during its execution. However, for the class of applications with dynamic workloads, such as web-applications, the average performance degradation including the downtime can be estimated as approximately 10% of the CPU utilization.

In addition, it is required to model the resource consumption by the VM being migrated on the destination node. It follows that each VM migration may cause an SLA violation; therefore, it is crucial to minimize the number of VM migrations. The length of a live migration depends on the total amount of memory used by the VM and available network bandwidth. This model is justified since the images and data of VMs are stored on a shared storage accessible over the network, which is required to enable live migration; therefore, copying the VM's storage is not required. Thus, to simplify the model,

the migration time and performance degradation experienced by a VM j are estimated as shown in (4.1).

$$T_{m_j} = \frac{M_j}{B_j}, \qquad U_{d_j} = 0.1 \cdot \int_{t_0}^{t_0+T_{m_j}} u_j(t)\, dt, \tag{4.1}$$

where U_{d_j} is the total performance degradation by VM j, t_0 is the time when the migration starts, T_{m_j} is the time taken to complete the migration, $u_j(t)$ is the CPU utilization by VM j, M_j is the amount of memory used by VM j, and B_j is the available network bandwidth.

4.2.4 SLA Violation Metrics

Meeting QoS requirements is highly important for Cloud computing environments. QoS requirements are commonly formalized in the form of SLAs, which can be determined in terms of such characteristics as minimum throughput or maximum response time delivered by the deployed system. Since these characteristics can vary for different applications, it is necessary to define a workload independent metric that can be used to evaluate the QoS delivered for any VM deployed on the IaaS.

This work defines that the SLAs are satisfied when 100% of the performance requested by applications inside a VM is provided at any time bounded only by the parameters of the VM. Two metrics for measuring the level of SLA violations in an IaaS environment are proposed: (1) the fraction of time during which active hosts have experienced the CPU utilization of 100%, Overload Time Fraction (OTF); and (2) the overall performance degradation by VMs due to migrations, Performance Degradation due to Migrations (PDM) (4.2). The reasoning behind the OTF metric is the observation that if a host serving applications is experiencing the 100% utilization, the performance of the applications is bounded by the host's capacity; therefore, VMs are not being provided with the required performance level.

$$OTF = \frac{1}{N}\sum_{i=1}^{N}\frac{T_{s_i}}{T_{a_i}}, \qquad PDM = \frac{1}{M}\sum_{j=1}^{M}\frac{C_{d_j}}{C_{r_j}}, \tag{4.2}$$

where N is the number of hosts; T_{s_i} is the total time during which the host i has experienced the utilization of 100% leading to an SLA violation; T_{a_i} is the total of the host i being in the active state (serving VMs); M is the number of VMs; C_{d_j} is the estimate of the

performance degradation of the VM j caused by migrations; C_{r_j} is the total CPU capacity requested by the VM j during its lifetime. In this work, C_{d_j} is estimated to be 10% of the CPU utilization in MIPS during all migrations of the VM j.

Both the OTF and PDM metrics independently characterize the level of SLA violations in the system; therefore, a combined metric that encompasses both performance degradation due to host overloading and VM migrations is proposed, denoted SLA Violation (SLAV). The metric is calculated as shown in (4.3).

$$SLAV = OTF \cdot PDM. \tag{4.3}$$

4.3 Heuristics for Distributed Dynamic VM Consolidation

This section presents several heuristics for dynamic consolidation of VMs based on an analysis of historical data of the resource usage by VMs. The problem is divided into four parts: (1) determining when a host is considered to be underloaded leading to the need to migrate all the VMs from this host and switch the host into the sleep mode; (2) determining when a host is considered to be overloaded requiring a migration of one or more VMs from this host to reduce the load; (3) selecting VMs that should be migrated from an overloaded host; and (4) finding a new placement of the VMs selected for migration from the hosts. The sub-problems are discussed in the following sections.

4.3.1 Host Underload Detection

Although complex underload detection strategies can be applied, for the purpose of simulations in this chapter a simple approach is used. First, all the overloaded hosts are found using the selected overload detection algorithm, and the VMs selected for migration are allocated to the destination hosts. Then, the system finds a compute host with the minimal utilization compared with the other hosts, and attempts to place all the VMs from this host on other hosts, while keeping them not overloaded. If such a placement is feasible, the VMs are set for migration to the determined target hosts. Once the migrations are completed, the source host is switched to the sleep mode to save energy. If all the VMs from the source host cannot be placed on other hosts, the host is kept active.

This process is iteratively repeated for all non-overloaded hosts.

4.3.2 Host Overload Detection

Each compute host periodically executes an overload detection algorithm to de-consolidate VMs when needed in order to avoid performance degradation and SLA violation. This section describes several heuristics proposed for the host overload detection problem.

A Static CPU Utilization Threshold

One of the simplest overload detection algorithms is based on an idea of setting a CPU utilization threshold distinguishing the non-overload and overload states of the host. When the algorithm is invoked, it compares the current CPU utilization of the host with the defined threshold. If the threshold is exceeded, the algorithm detects a host overload.

An Adaptive Utilization Threshold: Median Absolute Deviation

The previous paragraph described a simple heuristic for detecting host overloads based on setting a static CPU utilization threshold. However, fixed values of the utilization threshold are unsuitable for an environment with dynamic and unpredictable workloads, in which different types of applications can share a physical node. The system should be able to automatically adjust its behavior depending on the workload patterns exhibited by the applications.

This section presents a heuristic algorithm for auto-adjustment of the utilization threshold based on statistical analysis of historical data collected during the lifetime of VMs. The algorithm applies a robust statistical method, which is more effective than classical methods for data containing outliers or coming from non-normal distributions. The proposed adaptive-threshold algorithm adjusts the value of the CPU utilization threshold depending on the strength of the deviation of the CPU utilization. The higher the deviation, the lower the value of the upper utilization threshold. This is explained by an observation that a higher deviation increases the likelihood of the CPU utilization reaching 100% and causing an SLA violation.

Robust statistics provides an alternative approach to classical statistical methods [60]. The motivation is to produce estimators that are not excessively affected by small departures from model assumptions. The Median Absolute Deviation (MAD) is a measure of statistical dispersion. It is a more robust estimator of scale than the sample variance or standard deviation, as it behaves better with distributions without a mean or variance, such as the Cauchy distribution.

The MAD is a robust statistic, being more resilient to outliers in a data set than the standard deviation. In standard deviation, the distances from the mean are squared leading to large deviations being on average weighted more heavily. This means that outliers may significantly influence the value of standard deviation. In the MAD, the magnitude of the distances of a small number of outliers is irrelevant.

For a univariate data set $X_1, X_2, ..., X_n$, the MAD is defined as the median of the absolute deviations from the median of the data set:

$$MAD = median_i(|X_i - median_j(X_j)|), \tag{4.4}$$

that is, the MAD is the median of the absolute values of deviations (residuals) from the data's median. In the proposed overload detection algorithm, the CPU utilization threshold (T_u) is defined as shown in (4.5).

$$T_u = 1 - s \cdot MAD, \tag{4.5}$$

where $s \in \mathbb{R}^+$ is a parameter of the method defining how strongly the system tolerates host overloads. In other words, the parameter s allows the adjustment of the safety of the method: a lower value of s results in a higher tolerance to variation in the CPU utilization, while possibly increasing the level of SLA violations caused by the consolidation. Once the threshold is calculated, the algorithm acts similarly to the static threshold algorithm by comparing the current CPU utilization with the calculated threshold.

An Adaptive Utilization Threshold: Interquartile Range

This section proposes a method for setting an adaptive CPU utilization threshold based on another robust statistic. In descriptive statistics, the Interquartile Range (IQR), also

called the midspread or middle fifty, is a measure of statistical dispersion. It is equal to the difference between the third and first quartiles: $IQR = Q_3 - Q_1$. Unlike the (total) range, the interquartile range is a robust statistic, having a breakdown point of 25%, and thus, is often preferred to the total range. For a symmetric distribution (i.e., such that the median equals the average of the first and third quartiles), half of the IQR equals the MAD. Using IQR, similarly to (4.5) the CPU utilization threshold is defined in (4.6).

$$T_u = 1 - s \cdot IQR, \tag{4.6}$$

where $s \in \mathbb{R}^+$ is a parameter of the method defining the safety of the method similarly to the parameter s of the method proposed in Section 4.3.2.

Local Regression

The next heuristic is based on the Loess method (from the German *löss* – short for *local regression*) proposed by Cleveland [36]. The main idea of the local regression method is fitting simple models to localized subsets of data to build up a curve that approximates the original data. The observations (x_i, y_i) are assigned neighborhood weights using the *tricube weight function* shown in (4.7).

$$T(u) = \begin{cases} (1 - |u|^3)^3 & \text{if } |u| < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

Let $\Delta_i(x) = |x_i - x|$ be the distance from x to x_i, and let $\Delta_{(i)}(x)$ be these distances ordered from smallest to largest. Then, the neighborhood weight for the observation (x_i, y_i) is defined by the function $w_i(x)$ (4.8).

$$w_i(x) = T\left(\frac{\Delta_i(x)}{\Delta_{(q)}(x)}\right), \tag{4.8}$$

for x_i such that $\Delta_i(x) < \Delta_{(q)}(x)$, where q is the number of observations in the subset of data localized around x. The size of the subset is defined by a parameter of the method called the *bandwidth*. For example, if the degree of the polynomial fitted by the method is 1, the parametric family of functions is $y = a + bx$. The line is fitted to the data using the

weighted least-squares method with weight $w_i(x)$ at (x_i, y_i). The values of a and b are found by minimizing the function shown in (4.9).

$$\sum_{i=1}^{n} w_i(x)(y_i - a - bx_i)^2. \tag{4.9}$$

In the proposed algorithm, this approach is applied to fit a trend polynomial to the last k observations of the CPU utilization, where $k = \lceil q/2 \rceil$. A polynomial is fit for a single point, the last observation of the CPU utilization (i.e., the right boundary x_k of the data set). The problem of the boundary region is well-known as leading to a high bias [65]. According to Cleveland [38], fitted polynomials of degree 1 typically distort peaks in the interior of the configuration of observations, whereas polynomials of degree 2 remove the distortion but result in higher biases at boundaries. Therefore, for host overload detection, polynomials of degree 1 are chosen to reduce the bias at the boundary.

Let x_k be the last observation, and x_1 be the k^{th} observation from the right boundary. For the problem under consideration, x_i satisfies $x_1 \leq x_i \leq x_k$; thus, $\Delta_i(x_k) = x_k - x_i$, and $0 \leq \frac{\Delta_i(x_k)}{\Delta_1(x_k)} \leq 1$. Therefore, the tricube weight function can be simplified as $T^*(u) = (1 - u^3)^3$ for $0 \leq u \leq 1$, and the weight function is the following:

$$w_i(x) = T^*\left(\frac{\Delta_i(x_k)}{\Delta_1(x_k)}\right) = \left(1 - \left(\frac{x_k - x_i}{x_k - x_1}\right)^3\right)^3. \tag{4.10}$$

In the proposed Local Regression (LR) algorithm, using the described method derived from Loess, a new trend line $\widehat{g}(x) = \widehat{a} + \widehat{b}x$ is found for each new observation. This trend line is used to estimate the next observation $\widehat{g}(x_{k+1})$. If the inequalities (4.11) are satisfied, the algorithm detects a host overload, requiring some VMs to be offloaded from the host.

$$s \cdot \widehat{g}(x_{k+1}) \geq 1, \qquad x_{k+1} - x_k \leq t_m, \tag{4.11}$$

where $s \in \mathbb{R}^+$ is the safety parameter; and t_m is the maximum time required for a migration of any of the VMs allocated to the host.

Robust Local Regression

The version of Loess described in Section 4.3.2 is vulnerable to outliers that can be caused by leptokurtic or heavy-tailed distributions. To make Loess robust, Cleveland proposed the addition of the robust estimation method *bisquare* to the least-squares method for fitting a parametric family [37]. This modification transforms Loess into an iterative method. The initial fit is carried out with weights defined using the tricube weight function. The fit is evaluated at the x_i to get the fitted values $\widehat{y}_i$, and the residuals $\widehat{\epsilon}_i = y_i - \widehat{y}_i$. At the next step, each observation (x_i, y_i) is assigned an additional *robustness weight* r_i, whose value depends on the magnitude of $\widehat{\epsilon}_i$. Each observation is assigned the weight $r_i w_i(x)$, where r_i is defined as in (4.12).

$$r_i = B\left(\frac{\widehat{\epsilon}_i}{6s}\right), \tag{4.12}$$

where $B(u)$ is the *bisquare weight function* (4.13), and s is the MAD for the least-squares fit or any subsequent weighted fit (4.14).

$$B(u) = \begin{cases} (1-u^2)^2 & \text{if } |u| < 1, \\ 0 & \text{otherwise,} \end{cases} \tag{4.13}$$

$$s = median|\widehat{\epsilon}_i|. \tag{4.14}$$

Using the estimated trend line, the method described in Section 4.3.2 is applied to estimate the next observation. If the inequalities (4.11) are satisfied, the host is detected to be overloaded. This host algorithm is denoted Local Regression Robust (LRR).

4.3.3 VM Selection

Once a host overload is detected, the next step is to select VMs to offload from the host to avoid performance degradation. This section presents three policies for VM selection.

The Minimum Migration Time Policy

The Minimum Migration Time (MMT) policy migrates a VM v that requires the minimum time to complete a migration relatively to the other VMs allocated to the host. The migration time is estimated as the amount of RAM utilized by the VM divided by the spare network bandwidth available for the host j. Let V_j be a set of VMs currently allocated to the host j. The MMT policy finds a VM v that satisfies the conditions formalized in (4.15).

$$v \in V_j | \forall a \in V_j, \frac{RAM_u(v)}{NET_j} \leq \frac{RAM_u(a)}{NET_j}, \tag{4.15}$$

where $RAM_u(a)$ is the amount of RAM currently utilized by the VM a; and NET_j is the network bandwidth available for migration from the host j.

The Random Selection Policy

The Selection Choice (RS) policy randomly selects a VM to be migrated from the host according to a uniformly distributed discrete random variable $X \stackrel{d}{=} U(0, |V_j|)$, whose values index a set of VMs V_j allocated to the host j.

The Maximum Correlation Policy

The Maximum Correlation (MC) policy is based on the idea proposed by Verma et al. [118]. The idea is that the higher the correlation between the resource usage by applications running on an oversubscribed server, the higher the probability of the server overloading. According to this idea, those VMs are selected to be migrated that have the highest correlation of the CPU utilization with the other VMs.

To estimate the correlation between the CPU utilization of VMs, the *multiple correlation coefficient* [1] is applied. It is used in multiple regression analysis to assess the quality of the prediction of the dependent variable. The multiple correlation coefficient corresponds to the squared correlation between the predicted and the actual values of the dependent variable. It can also be interpreted as the proportion of the variance of the dependent variable explained by the independent variables.

Let $X_1, X_2, ..., X_n$ be n random variables representing the CPU utilization of n VMs

allocated to a host. Let Y represent one of the VMs that is currently considered for being migrated. Then $n-1$ random variables are independent, and 1 variable Y is dependent. The objective is to evaluate the strength of the correlation between Y and $n-1$ remaining random variables. The $(n-1) \times n$ augmented matrix containing the observed values of the $n-1$ independent random variables is denoted by $\mathbf{X}$, and the $(n-1) \times 1$ vector of observations for the dependent variable Y is denoted by $\mathbf{y}$ (4.16). The matrix $\mathbf{X}$ is called augmented because the first column is composed only of 1.

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1,1} & \dots & x_{n-1,n-1} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{4.16}$$

A vector of predicted values of the dependent random variable $\widehat{Y}$ is denoted by $\widehat{\mathbf{y}}$ and is obtained as shown in (4.17).

$$\widehat{\mathbf{y}} = \mathbf{X}\mathbf{b} \qquad \mathbf{b} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{4.17}$$

Having found a vector of predicted values, it is now possible to compute the multiple correlation coefficient $R^2_{Y,1,\dots,n-1}$, which is equal to the squared coefficient of correlation between the observed values $\mathbf{y}$ of the dependent variable Y and the predicted values $\widehat{\mathbf{y}}$ (4.18).

$$R^2_{Y,X_1,\dots,X_{n-1}} = \frac{\sum_{i=1}^{n}(\mathbf{y}_i - m_Y)^2(\widehat{\mathbf{y}}_i - m_{\widehat{Y}})^2}{\sum_{i=1}^{n}(\mathbf{y}_i - m_Y)^2 \sum_{i=1}^{n}(\widehat{\mathbf{y}}_i - m_{\widehat{Y}})^2}, \tag{4.18}$$

where m_Y and $m_{\widehat{Y}}$ are the sample means of Y and $\widehat{Y}$ respectively. The multiple correlation coefficient is calculated for each X_i, which is denoted as $R^2_{X_i,X_1,\dots,X_{i-1},X_{i+1},\dots,X_n}$. The MC policy finds a VM v that satisfies the conditions defined in (4.19).

$$v \in V_j | \forall a \in V_j, R^2_{X_v,X_1,\dots,X_{v-1},X_{v+1},\dots,X_n} \geq R^2_{X_v,X_1,\dots,X_{a-1},X_{a+1},\dots,X_n}. \tag{4.19}$$

4.3.4 VM Placement

VM placement can be seen as a bin packing problem with variable bin sizes and prices, where bins represent the physical nodes; items are the VMs that have to be allocated; bin

sizes are the available CPU capacities of the nodes; and prices correspond to the power consumption by the nodes. As the bin packing problem is NP-hard, it is reasonable to apply a heuristic, such as the Best Fit Decreasing (BFD) algorithm, which has been shown to use no more than $11/9 \cdot OPT + 1$ bins (where OPT is the number of bins provided by the optimal solution) [130].

Algorithm 1 The Power Aware Best Fit Decreasing (PABFD) algorithm

Input: *hostList*, *vmList*
Output: *vmPlacement*
1: sort *vmList* in the order of decreasing utilization
2: **for** *vm* in *vmList* **do**
3: *minPower* ← MAX
4: *allocatedHost* ← NULL
5: **for** *host* in *hostList* **do**
6: **if** *host* has enough resources for *vm* **then**
7: *power* ← estimatePower(*host*, *vm*)
8: **if** *power* < *minPower* **then**
9: *allocatedHost* ← *host*
10: *minPower* ← *power*
11: **if** *allocatedHost* ≠ NULL **then**
12: add (*allocatedHost*, *vm*) to *vmPlacement*
13: **return** *vmPlacement*

This section presents a modification of the BFD algorithm denoted Power Aware Best Fit Decreasing (PABFD) shown in Algorithm 1. The algorithm sorts all the VMs in decreasing order of their current CPU utilization and allocates each VM to a host that provides the least increase of the power consumption caused by the allocation. This approach allows the algorithm to leverage the heterogeneity of hosts by choosing the most power-efficient ones first. The complexity of the algorithm is nm, where n is the number of hosts and m is the number of VMs that have to be allocated.

4.4 Performance Evaluation

4.4.1 Experiment Setup

As the targeted system is an IaaS, a Cloud computing environment that is supposed to create a view of infinite computing resources to users, it is essential to evaluate the proposed resource allocation algorithms on a large-scale virtualized data center infrastruc-

ture. However, it is extremely difficult to conduct repeatable large-scale experiments on a real infrastructure, which is required to evaluate and compare the proposed algorithms. Therefore, to ensure the repeatability of experiments, simulations were chosen as a way to evaluate the performance of the proposed heuristics.

The CloudSim toolkit [29] was chosen as a simulation platform, as it is a modern simulation framework aimed at Cloud computing environments. In contrast to alternative simulation toolkits (e.g. SimGrid, GangSim), it allows the modeling of virtualized environments, supporting on-demand resource provisioning, and their management. It has been extended to enable energy-aware simulations, as the core framework does not provide this capability. Apart from the energy consumption modeling and accounting, the ability to simulate service applications with dynamic workloads has been incorporated. The implemented extensions are included in the 2.0 version of the CloudSim toolkit.

The simulated data center comprised 800 heterogeneous physical nodes, half of which were HP ProLiant ML110 G4 servers, and the other half consisted of HP ProLiant ML110 G5 servers. The characteristics of the servers and data on their power consumption are given in Section 4.2.2. The frequencies of the servers' CPUs were mapped onto MIPS ratings: 1860 MIPS each core of the HP ProLiant ML110 G5 server, and 2660 MIPS each core of the HP ProLiant ML110 G5 server. Each server had 1 GB/s network bandwidth.

The characteristics of the VM types corresponded to Amazon EC2 instance types[2] with the only exception that all the VMs were single-core, which is explained by the fact that the workload data used for the simulations come from single-core VMs (Section 4.4.3). For the same reason the amount of RAM was divided by the number of cores for each VM type: High-CPU Medium Instance (2500 MIPS, 0.85 GB); Extra Large Instance (2000 MIPS, 3.75 GB); Small Instance (1000 MIPS, 1.7 GB); and Micro Instance (500 MIPS, 613 MB). Initially the VMs were allocated according to the resource requirements defined by the VM types. However, during the lifetime, VMs utilized less resources according to the input workload data, creating opportunities for dynamic consolidation.

[2]Amazon EC2 instance types. http://aws.amazon.com/ec2/instance-types/

Table 4.2: Characteristics of the workload data (CPU utilization)

Date	Number of VMs	Mean	St. dev.	Quartile 1	Median	Quartile 3
03/03/2011	1052	12.31%	17.09%	2%	6%	15%
06/03/2011	898	11.44%	16.83%	2%	5%	13%
09/03/2011	1061	10.70%	15.57%	2%	4%	13%
22/03/2011	1516	9.26%	12.78%	2%	5%	12%
25/03/2011	1078	10.56%	14.14%	2%	6%	14%
03/04/2011	1463	12.39%	16.55%	2%	6%	17%
09/04/2011	1358	11.12%	15.09%	2%	6%	15%
11/04/2011	1233	11.56%	15.07%	2%	6%	16%
12/04/2011	1054	11.54%	15.15%	2%	6%	16%
20/04/2011	1033	10.43%	15.21%	2%	4%	12%

4.4.2 Performance Metrics

In order to evaluate and compare the performance of the algorithms, several performance metrics were used. One of the metrics was the total energy consumption by the physical servers of the data center. Energy consumption was calculated according to the model defined in Section 4.2.2. Metrics used to evaluate the level of SLA violations caused by the system were SLAV, OTF and PDM defined in Section 4.2.4. Another metric was the number of VM migrations initiated during the VM placement adaptation.

The main metrics are energy consumption by physical nodes and SLAV, however, these metrics are typically negatively correlated as energy can usually be decreased by the cost of the increased level of SLA violations. The objective of the resource management system is to minimize both energy and SLA violations. Therefore, a combined metric is proposed that captures both energy consumption and the level of SLA violations, denoted Energy and SLA Violations (ESV) (4.20).

$$ESV = E \cdot SLAV. \tag{4.20}$$

4.4.3 Workload Data

To make a simulation-based evaluation applicable, it is important to conduct experiments using workload traces from a real system. For experiments in this chapter, the data provided as a part of the CoMon project, a monitoring infrastructure for PlanetLab [92], were used. The data contains the CPU utilization by more than a thousand VMs from servers

Table 4.3: Comparison of VM selection policies using paired T-tests

Policy 1 (ESV $\times 10^{-3}$)	**Policy 2 (ESV $\times 10^{-3}$)**	**Difference ($\times 10^{-3}$)**	**P-value**
RS (4.03)	MC (3.83)	0.196 (0.134, 0.258)	P-value < 0.001
RS (4.03)	MMT (3.23)	0.799 (0.733, 0.865)	P-value < 0.001
MC (3.83)	MMT (3.23)	0.603 (0.533, 0.673)	P-value < 0.001

located at more than 500 places around the world collected during 10 randomly selected days in March and April 2010. The interval of utilization measurements is 5 minutes.

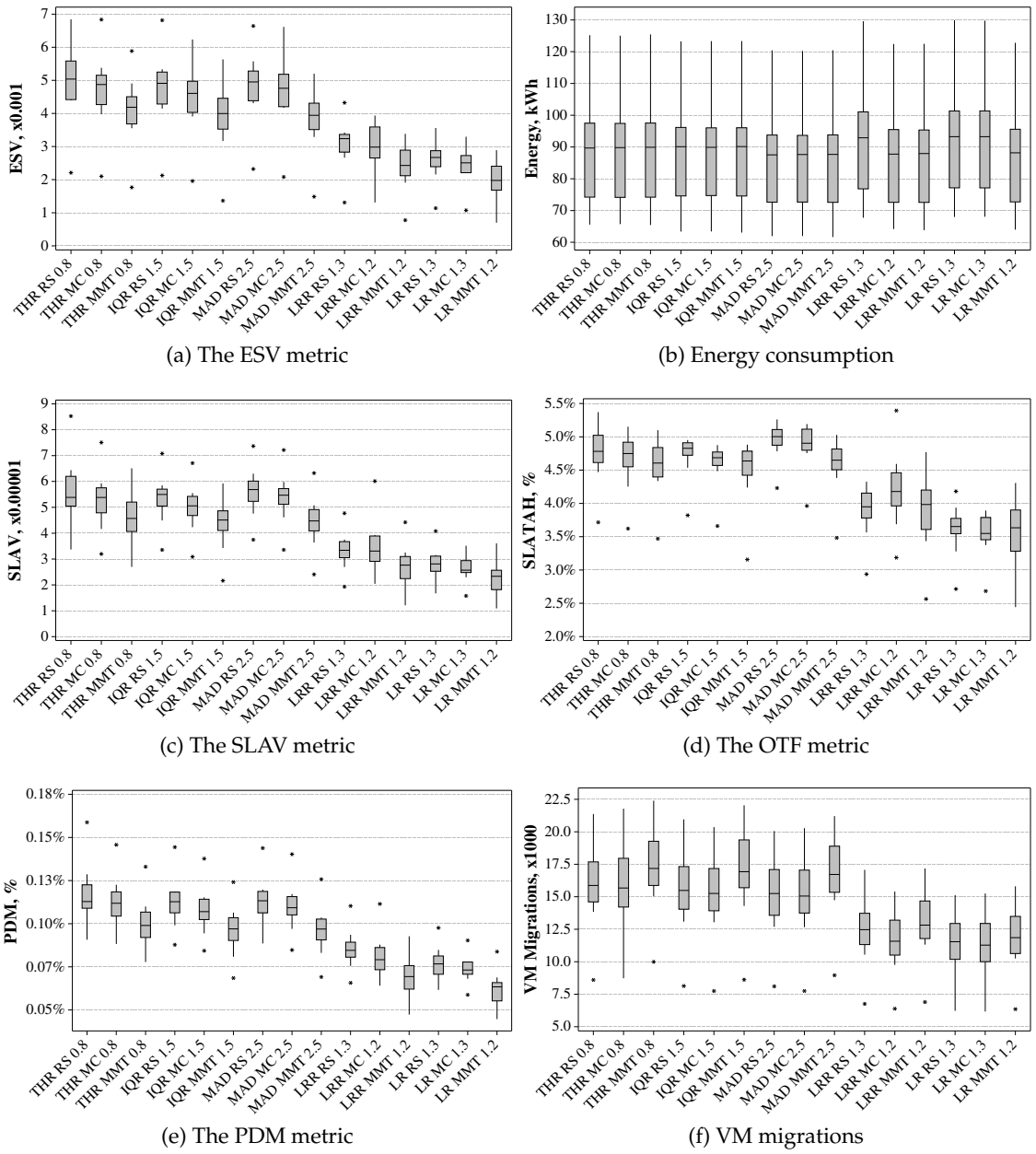
The characteristics of the data for each day are shown in Table 4.2. The data confirm the statement made in Chapter 2: the average CPU utilization is far below 50%. During the simulations, each VM was randomly assigned a workload trace from one of the VMs from the corresponding day. Since the objective of the experiments was to stress the consolidation algorithms, VM consolidation was not limited by the memory bounds.

4.4.4 Simulation Results and Analysis

Using the workload data described in Section 4.4.3, all combinations of the five proposed host overloading detection algorithms (THR, IQR, MAD, LR, and LRR) and three VM selection algorithms (MMT, RS, and MC) were simulated. Moreover, the parameters of each overload detection algorithm were varied as follows: for THR from 0.6 to 1.0 increasing by 0.1; for IQR and MAD from 0.5 to 3.0 increasing by 0.5; for LR and LRR from 1.0 to 1.4 increasing by 0.1. These variations resulted in 81 combinations of the algorithms and parameters.

According to Ryan-Joiner's normality test, the values of the ESV metric produced by the algorithm combinations did not follow a normal distribution with the P-value $<$ 0.01. Therefore, the median values of the ESV metric were used to compare the algorithm combinations and select the parameter of each algorithm combination that minimizes the median ESV metric calculated over 10 days of the workload traces. The results produced by the selected algorithms are shown in Figure 4.2.

According to Ryan-Joiner's normality test, the values of the ESV metric produced by the selected algorithm combinations follow a normal distribution with the P-value $>$ 0.1. Three paired T-tests were conducted to determine the VM selection policy that minimizes the ESV metric across all algorithm combinations (Table 4.3). The T-tests showed that the



(a) The ESV metric

(b) Energy consumption

(c) The SLAV metric

(d) The OTF metric

(e) The PDM metric

(f) VM migrations

Figure 4.2: Algorithm comparison in regard to the ESV, SLAV, OTF, and PDM metrics, as well as energy consumption, and the number of VM migrations

usage of the MMT policy leads to a statistically significantly lower value of the ESV metric with the P-value < 0.001. Next, the combinations of the overload detection algorithms with the MMT policy are analyzed.

To meet the assumptions of the ANOVA model, the values of the ESV metric for the algorithm combinations with the MMT policy were transformed using the square root function. The standardized residuals from the transformed data pass Ryan-Joiner's test

with the P-value > 0.1, justifying the assumption that the sample comes from a normal distribution. A plot of the standardized residuals against the fitted values showed that the assumption of equal variances is met.

Having the assumptions of the model met, the F-test was applied to check whether there is a statistically significant difference between the results produced by the combinations of the overload detection algorithms with the MMT policy with the selected parameters. The test showed that there is a statistically significant difference with the P-value < 0.001. Tukey's pairwise comparisons are summarized in Table 4.4.

According to the results of Tukey's pairwise comparisons, there is no statistically significant difference between the THR-MMT-0.8, IQR-MMT-1.5 and MAD-MMT-2.5 algorithms (group A), and between the LRR-MMT-1.2 and LR-MMT-1.2 algorithms (group B). However, there is a statistically significant difference between the local regression based algorithms and the other algorithms. Nevertheless, a paired T-test for a comparison of the means of the ESV metric produced by LRR-MMT-1.2 and LR-MMT-1.2 showed that there is a statistically significant difference with the P-value < 0.001. The mean difference is 4.21×10^{-4} with a 95% CI: $(3.23 \times 10^{-4}, 5.19 \times 10^{-4})$.

As a paired T-test provides more precise results than Tukey's pairwise comparisons, it follows that LR-MMT-1.2 provides the best results compared to all the other combinations of algorithms in regard to the ESV metric. Moreover, the trade-off between energy consumption and SLA violations can be adjusted by varying the safety parameter of the LR algorithm. The results of the combinations of each overload detection algorithm with the best parameters and the MMT policy, along with the benchmark algorithms are shown in Table 4.5. The benchmark policies include Non Power Aware (NPA), DVFS, and THR combined with the MMT policy. The NPA policy makes all the hosts consume

Table 4.4: Tukey's pairwise comparisons using the transformed ESV. Values that do not share a letter are significantly different.

Policy	**SQRT(ESV)** ($\times 10^{-2}$)	**95% CI**	**Grouping**
THR-MMT-0.8	6.34	(5.70, 6.98)	A
IQR-MMT-1.5	6.16	(5.44, 6.87)	A
MAD-MMT-2.5	6.13	(5.49, 6.77)	A
LRR-MMT-1.2	4.82	(4.22, 5.41)	B
LR-MMT-1.2	4.37	(3.83, 4.91)	B

the maximum power all the time.

Based on the observed simulation results, several conclusions can be made:

1. Dynamic VM consolidation algorithms significantly outperform static allocation policies, such as NPA and DVFS.
2. Statistics-based overload detection algorithms substantially outperform the static CPU utilization threshold algorithm due to a vastly reduced level of SLA violations.
3. The MMT policy produces better results compared to the MC and RS policies, meaning that the minimization of the VM migration time is more important than the minimization of the correlation between VMs allocated to a host.
4. Host overload detection algorithms based on local regression outperform the threshold based algorithms due to a decreased level of SLA violations and the number of VM migrations.
5. The LR algorithm produces better results than its robust modification, which can be explained by the fact that, for the simulated workload, it is more important to quickly react to load spikes instead of smoothing out such outlying observations.

In the simulations, the mean value of the sample means of the time before a host is switched to the sleep mode for the LR-MMT-1.2 algorithm combination was 1933 seconds with the 95% CI: (1740, 2127). This means that on average a host is switched to the sleep mode after approximately 32 minutes of activity. This value is effective for real-world systems, as modern servers allow low-latency transitions to the sleep mode consuming low power. Meisner et al. [81] showed that a typical blade server consuming 450 W in the fully utilized state consumes approximately 10.4 W in the sleep mode, while the transition delay is 300 ms.

The mean number of host transitions to the sleep mode (the total number of hosts is 800) per day was 1272 with 95% CI: (1211, 1333). The mean value of the sample means of the time before a VM is migrated from a host for the same algorithm combination was 15.3 seconds with the 95% CI: (15.2, 15.4). The mean value of the sample means of the execution time of the LR-MMT-1.2 algorithm on a server with an Intel Xeon 3060 (2.40 GHz) processor and 2 GB of RAM was 0.20 ms with the 95% CI: (0.15, 0.25).

Table 4.5: Simulation results of the best algorithm combinations and benchmark algorithms (median values)

Policy	**ESV** ($\times 10^{-3}$)	**Energy** (kWh)	**SLAV** ($\times 10^{-5}$)	**OTF**	**PDM**	**VM migr.** ($\times 10^{3}$)
NPA	0	2419.2	0	0%	0%	0
DVFS	0	613.6	0	0%	0%	0
THR-MMT-1.0	20.12	75.36	25.78	24.97%	0.10%	13.64
THR-MMT-0.8	4.19	89.92	4.57	4.61%	0.10%	17.18
IQR-MMT-1.5	4.00	90.13	4.51	4.64%	0.10%	16.93
MAD-MMT-2.5	3.94	87.67	4.48	4.65%	0.10%	16.72
LRR-MMT-1.2	2.43	87.93	2.77	3.98%	0.07%	12.82
LR-MMT-1.2	1.98	88.17	2.33	3.63%	0.06%	11.85

4.5 Conclusions

To maximize their Return On Investment (ROI), Cloud providers need to apply energy-efficient resource management strategies, such as dynamic VM consolidation with switching idle servers to power-saving modes. However, such consolidation is not trivial, as it may result in violations of the SLAs negotiated with customers. Based on the analysis in Chapter 3, this chapter proposed novel heuristics for distributed dynamic VM consolidation that rely on an analysis of historical data on the CPU utilization by VMs to leverage the predictability of the workload.

Simulations of a large-scale data center using workload traces from more than a thousand PlanetLab VMs have shown that the proposed local regression based host overload detection algorithm combined with the MMT VM selection policy significantly outperforms other dynamic VM consolidation algorithms in regard to the ESV metric due to a substantially reduced level of SLA violations and the number of VM migrations.

A disadvantage of the proposed algorithms is the inability to explicitly specify a QoS constraint: the performance of the algorithms in regard to QoS can only be adjusted indirectly by tuning the parameters of the algorithms. The next chapter investigates a host overload detection algorithm based on a Markov chain model, which allows the explicit specification of a QoS goal.

Chapter 5

The Markov Host Overload Detection Algorithm

This chapter investigates the problem of host overload detection as a part of dynamic VM consolidation. Determining when it is best to reallocate VMs from an overloaded host is an aspect of dynamic VM consolidation that directly influences the resource utilization and Quality of Service (QoS) delivered by the system. The influence on the QoS is explained by the fact that server overloads cause resource shortages and performance degradation of applications. Current solutions to the problem of host overload detection are generally heuristic-based, or rely on statistical analysis of historical data. The limitations of these approaches are that they lead to sub-optimal results and do not allow explicit specification of a QoS goal. This chapter presents a novel approach that for any known stationary workload and a given state configuration optimally solves the problem of host overload detection by maximizing the mean inter-migration time under the specified QoS goal based on a Markov chain model. The algorithm is heuristically adapted to handle unknown non-stationary workloads using the Multisize Sliding Window workload estimation technique, and evaluated by simulations.

5.1 Introduction

THE previous chapter introduced a distributed approach to dynamic VM consolidation that consists in splitting the problem into 4 sub-problems:

1. Deciding when a host is considered to be underloaded, so that its VMs should be migrated, and the host should be switched to a low-power mode.
2. Deciding when a host is considered to be overloaded, so that some VMs should be migrated from it to other hosts to meet the QoS requirements.
3. Selecting VMs to migrate from an overloaded host.
4. Allocating the VMs selected for migration to other active or re-activated hosts.

This chapter focuses on the second sub-problem – the problem of host overload detection. Detecting when a host becomes overloaded directly influences the QoS, since if the resource capacity is completely utilized, it is highly likely that the applications are experiencing resource shortage and performance degradation. What makes the host overload detection problem complex is the necessity to optimize the time-averaged behavior of the system, while handling a variety of heterogeneous workloads placed on a single host.

To address this problem, most of the current approaches to dynamic VM consolidation apply either heuristic-based techniques, such as utilization thresholds [54, 55, 121, 135]; decision-making based on statistical analysis of historical data [23, 57]; or simply periodic adaptation of the VM allocation [86, 119]. The limitations of these approaches are that they lead to sub-optimal results and do not allow the system administrator to explicitly set a *QoS goal*. In other words, the performance in regard to the QoS delivered by the system can only be adjusted indirectly by tuning parameters of the applied host overload detection algorithm. In contrast, the approach proposed in this chapter enables the system administrator to explicitly specify a QoS goal in terms of a workload independent QoS metric. The underlying analytical model allows a derivation of an optimal randomized control policy for any known stationary workload and a given state configuration. The **key contributions** of this chapter are the following:

1. An analytical model showing that to improve the quality of VM consolidation, it is necessary to maximize the mean time between VM migrations initiated by the host overload detection algorithm.
2. An *optimal offline algorithm* for host overload detection, and proof of its optimality.
3. A novel Markov chain model that allows a derivation of a randomized control policy that *optimally* solves the problem of maximizing the mean time between VM migrations under an explicitly specified QoS goal for any known stationary workload and a given state configuration in the *online* setting.
4. A heuristically adapted algorithm for handling unknown non-stationary workloads using the Multisize Sliding Window workload estimation approach [80], which leads to comparable to the best benchmark algorithm performance in terms of the inter-migration time, while providing the advantage of explicit specification of a QoS goal. The adapted algorithm leads to approximately 88% of the

mean inter-migration time produced by the optimal offline algorithm for the input workload traces used in the experiments.

The proposed algorithm is evaluated by simulations using real-world workload traces from more than a thousand PlanetLab[1] VMs hosted on servers located in more than 500 places around the world. The experiments show that the introduced algorithm outperforms the benchmark algorithms, while meeting the QoS goal in accordance with the theoretical model. The algorithm uses a workload independent QoS metric and transparently adapts its behavior to various workloads; therefore, it can be applied in an environment with unknown non-stationary workloads, such as IaaS.

It is important to note that the model proposed in this chapter is based on Markov chains requiring a few fundamental modeling assumptions. First of all, the workload must satisfy the Markov property, which implies memoryless state transitions and an exponential distribution of state transition delays. These assumptions must be taken into account in an assessment of the applicability of the proposed model to a particular system. A more detailed discussion of the modeling assumptions and validation of the assumptions is given in Section 5.6.5.

The remainder of the chapter is organized as follows. The next section discusses the related work followed by the objective of host overload detection and workload independent QoS metric in Sections 5.3 and 5.4 respectively. An optimal offline algorithm for the problem of host overload detection is introduced in Section 5.5. In Section 5.6, a Markov model for the problem of host overload detection is presented, and approximated for unknown non-stationary workloads in Section 5.7. In Section 5.8, a control algorithm is proposed followed by a multi-core CPU model in Section 5.9 and an experimental evaluation in Section 5.10. The chapter is concluded with Section 5.11 discussing the results and future research directions.

5.2 Related Work

Prior approaches to host overload detection for energy-efficient dynamic VM consolidation proposed in the literature can be broadly divided into 3 categories: periodic

[1]The PlanetLab project. http://www.planet-lab.org/

adaptation of the VM placement (no overload detection), threshold-based heuristics, and decision-making based on statistical analysis of historical data. One of the first works, in which dynamic VM consolidation has been applied to minimize energy consumption in a data center, has been performed by Nathuji and Schwan [86]. They explored the energy benefits obtained by consolidating VMs using migration and found that the overall energy consumption can be significantly reduced. Verma et al. [119] modeled the problem of power-aware dynamic VM consolidation as a bin-packing problem and proposed a heuristic that minimizes the data center's power consumption, taking into account the VM migration cost. However, the authors did not apply any algorithm for determining *when* it is necessary to optimize the VM placement – the proposed heuristic is simply periodically invoked to adapt the placement of VMs.

Zhu et al. [135] studied the dynamic VM consolidation problem and applied a heuristic of setting a static CPU utilization threshold of 85% to determine when a host is overloaded. The host is assumed to be overloaded when the threshold is exceeded. The 85% utilization threshold has been first introduced and justified by Gmach et al. [54] based on their analysis of workload traces. In their more recent work, Gmach et al. [55] investigated benefits of combining both periodic and reactive threshold-based invocations of the migration controller. VMware Distributed Power Management [121] operates based on the same idea with the utilization threshold set to 81%. However, static threshold heuristics are unsuitable for systems with unknown and dynamic workloads, as these heuristics do not adapt to workload changes and do not capture the time-averaged behavior of the system. This approach has been outperformed by the algorithms proposed in Chapter 4. In this chapter, static and dynamic threshold heuristics are used as benchmark algorithms in the experimental evaluation of the proposed approach.

Jung et al. [64] investigated the problem of dynamic consolidation of VMs running multi-tier web-applications to optimize a global utility function, while meeting SLA requirements. The approach is workload-specific, as the SLA requirements are defined in terms of the response time precomputed for each transaction type of the applications. When the request rate deviates out of an allowed interval, the system adapts the placement of VMs and the states of the hosts. Zheng et al. [134] proposed automated experimental testing of the efficiency of a reallocation decision prior to its application, once the

response time, specified in the SLAs, is violated. In the approach proposed by Kumar et al. [71], the resource allocation is adapted when the application's SLAs are violated. Wang et al. [123] applied control loops to manage resource allocation under response time QoS constraints at the cluster and server levels. If the resource capacity of a server is insufficient to meet the applications' SLAs, a VM is migrated from the server. All these works are similar to threshold-based heuristics in that they rely on instantaneous values of performance characteristics but do not leverage the observed history of system states to estimate the future behavior of the system and optimize the time-averaged performance.

Guenter et al. [57] implemented an energy-aware dynamic VM consolidation system focused on web-applications, whose SLAs are defined in terms of the response time. The authors applied weighted linear regression to predict the future workload and proactively optimize the resource allocation. This approach is in line with the Local Regression (LR) algorithm proposed in Chapter 4, which is used as one of the benchmark algorithms in this chapter. Bobroff et al. proposed a server overload forecasting technique based on time-series analysis of historical data [23]. Unfortunately, the algorithm description given in the paper is too high level, which does not allow us to implement it to compare with the approach proposed in this chapter. Weng et al. [127] proposed a load-balancing system for virtualized clusters. A cluster-wide cost of the VM allocation is periodically minimized to detect overloaded and underloaded hosts, and reallocate VMs. This is a related work but with the opposite objective – the VMs are deconsolidated to balance the load across the hosts.

As mentioned above, the common limitations of the prior works are that, due to their heuristic basis, they lead to sub-optimal results and do not allow the system administrator to explicitly set a QoS goal. In this work, a novel approach to the problem of host overload detection is proposed inspired by the work of Benini et al. [21] on power management of electronic systems using Markov decision processes. A Markov chain model is created for the case of a known stationary workload and a given state configuration. Using a workload independent QoS metric, a Non-Linear Programming (NLP) problem formulation is derived. The solution of the derived NLP problem is the optimal control policy that maximizes the time between VM migrations under the specified QoS constraint in the online setting. Since most real-world systems, including IaaS, experi-

ence highly variable non-stationary workloads, the Multisize Sliding Window workload estimation technique proposed by Luiz et al. [80] is applied to heuristically adapt the proposed model to non-stationary stochastic environments and practical applications. Although the final approach is a heuristic, in contrast to the related works it is based on an analytical model that allows the computation of an optimal control policy for any known stationary workload and a given state configuration.

5.3 The Objective of a Host Overload Detection Algorithm

This section shows that to improve the quality of VM consolidation, it is necessary to maximize the time intervals between VM migrations from overloaded hosts. Since VM consolidation is applied to reduce the number of active hosts, the VM consolidation quality is inversely proportional to H, the mean number of active hosts over n time steps:

$$H = \frac{1}{n} \sum_{i=1}^{n} a_i, \tag{5.1}$$

where a_i is the number of active hosts at the time step $i = 1, 2, \ldots, n$. A lower value of H represents a better quality of VM consolidation.

To investigate the impact of decisions made by host overload detection algorithms on the quality of VM consolidation, consider an experiment, where at any time step the host overload detection algorithm can initiate a migration from a host due to an overload. There are two possible consequences of a decision to migrate a VM relevant to host overload detection: *Case 1*, when a VM to be migrated from an overloaded host cannot be placed on another active host due to insufficient resources, and therefore, a new host has to be activated to accommodate the VM; and *Case 2*, when a VM to be migrated can be placed on another active host. To study host overload detection in isolation, it is assumed that no hosts are switched off during the experiment, i.e., once a host is activated, it remains active until n.

Let p be the probability of *Case 1*, i.e., an extra host has to be activated to migrate a VM from an overloaded host determined by the host overload detection algorithm. Then, the probability of *Case 2* is $(1 - p)$. Let T be a random variable denoting the time between two subsequent VM migrations initiated by the host overload detection algorithm. The

expected number of VM migrations initiated by the host overload detection algorithm over n time steps is $n/E[T]$, where $E[T]$ is the expected inter-migration time.

Based on the definitions given above, the number of extra hosts switched on due to VM migrations initiated by the host overload detection algorithm over n time steps can be defined as $X \sim B(n/E[T], p)$, which is a binomially distributed random variable. The expected number of extra hosts activated is $E[X] = np/E[T]$. Let A be a random variable denoting the time during which an extra host is active between the time steps 1 and n. The expected value of A can be defined as follows:

$$\begin{aligned} E[A] &= \sum_{i=1}^{\left\lfloor \frac{n}{E[T]} \right\rfloor} (n - (i-1)E[T])p \\ &= \left\lfloor \frac{n}{E[T]} \right\rfloor \frac{p}{2} \left(n + n - \left(\left\lfloor \frac{n}{E[T]} \right\rfloor - 1 \right) E[T] \right) \\ &\leq \frac{np}{2} \left(1 + \frac{n}{E[T]} \right). \end{aligned} \tag{5.2}$$

(5.1) can be rewritten as follows:

$$\begin{aligned} H &= \frac{1}{n} \sum_{i=1}^{n} a_i \\ &= \frac{1}{n} \sum_{i=1}^{n} a_1 + \frac{1}{n} \sum_{i=1}^{n} (a_i - a_1) \\ &= a_1 + \frac{1}{n} \sum_{i=1}^{n} (a_i - a_1). \end{aligned} \tag{5.3}$$

The first term a_1 is a constant denoting the number of hosts that have been initially active and remain active until the end of the experiment. The second term $H^* = \frac{1}{n} \sum_{i=1}^{n} (a_i - a_1)$ is the mean number of hosts switched on due to VM migrations being active per unit of time over n time steps. It is important to analyze the average behavior, and thus estimate the expected value of H^*. It is proportional to a product of the expected number of extra hosts switched on due to VM migrations and the expected activity time of an extra host normalized by the total time, as shown in (5.4).

$$\begin{aligned} E[H^*] \propto & \frac{1}{n} E[X] E[A] \\ \leq & \frac{1}{n} \frac{np}{E[T]} \frac{np}{2} \left(1 + \frac{n}{E[T]}\right) \\ = & \frac{np^2}{2E[T]} \left(1 + \frac{n}{E[T]}\right). \end{aligned} \tag{5.4}$$

Since the objective is to improve the quality of VM consolidation, it is necessary to minimize $E[H^*]$. From (5.4), the only variable that can be directly controlled by a host overload detection algorithm is $E[T]$; therefore, to minimize $E[H^*]$ the objective of a host overload detection algorithm is to maximize $E[T]$, i.e., to maximize the mean time between migrations from overloaded hosts.

5.4 A Workload Independent QoS Metric

To impose QoS requirements on the system, an extension of the *workload independent QoS metric* introduced in Chapter 4 is applied. For the purpose of this chapter, a host can be in one of two states in regard to its load level: (1) serving regular load; and (2) being overloaded. It is assumed that if a host is overloaded, the VMs allocated to the host are not being provided with the required performance level leading to performance degradation. To evaluate the overall performance degradation, a metric denoted Overload Time Fraction (OTF) is defined as follows:

$$OTF(u_t) = \frac{t_o(u_t)}{t_a}, \tag{5.5}$$

where u_t is the CPU utilization threshold distinguishing the non-overload and overload states of the host; t_o is the time, during which the host has been overloaded, which is a function of u_t; and t_a is the total time, during which the host has been active. Using this metric, SLAs can be defined as the maximum allowed value of OTF. For example, if in the SLAs it is stated that OTF must be less or equal to 10%, it means that on average a host is allowed to be overloaded for not more than 10% of its activity time. Since the provider is interested in maximizing the resource utilization while meeting the SLAs, from his perspective this requirement corresponds to the QoS goal of $OTF \rightarrow 10\%$, while

$OTF \leq 10\%$. The definition of the metric for a single host can be extended to a set of hosts by substituting the time values by the aggregated time values over the set of hosts.

The exact definition of the state of a host, when it is overloaded, depends on the specific system requirements. However, the value of the CPU utilization threshold u_t defining the states of a host does not affect the model proposed in this chapter, the model allows setting the threshold to any value. For example, in the experiments of this chapter, it is defined that a host is overloaded, when its CPU utilization is 100%, in which case the VMs allocated to this host do not get the required CPU capacity leading to performance degradation. The reasoning behind this is the observation that if a host serving applications is experiencing 100% utilization, the performance of the applications is constrained by the host's capacity; therefore, the VMs are not being provided with the required performance level.

It has been claimed in the literature that the performance of servers degrade, when their load approaches 100% [108, 133]. For example, the study of Srikantaiah et al. [108] has shown that the performance delivered by the CPU degrades when the utilization is higher than 70%. If due to system requirements, it is important to avoid performance degradation, the proposed OTF metric allows the specification of the CPU utilization threshold at the required level below 100%. The host is considered to be overloaded, when the CPU utilization is higher than the specified threshold.

In general, other system resources, such as memory, disk, and network bandwidth, should also be taken into account in the definition of QoS requirements. However, in this chapter, only the CPU is considered, as it is one of the main resources that are usually oversubscribed by Cloud providers. Therefore, in the analysis of this chapter, it is assumed that the other system resources are not significantly oversubscribed and do not become performance bottlenecks.

Verma et al. [118] proposed a similar metric for estimating the SLA violation level in a system, which they defined as the number of time instances, when the capacity of a server is less than the demand of all applications placed on it. However, their metric shows a non-normalized absolute value, which, for example, cannot be used to compare systems processing the same workload for different periods of time. In contrast, the OTF metric is normalized and does not depend on the length of the time period under consideration.

In the next section, based on the objective of a host overload detection algorithm derived in Section 5.3 and the OTF metric introduced in this section, an optimal offline algorithm for the host overload detection problem is proposed, and its optimality is proved.

5.5 An Optimal Offline Algorithm

As shown in Section 5.3, it is necessary to maximize the mean time between VM migrations initiated by the host overload detection algorithm, which can be achieved by maximizing each individual inter-migration time interval. Therefore, the problem formulation is limited to a single VM migration, i.e., the time span of a problem instance is from the end of a previous VM migration and to the end of the next. Given the results of Sections 5.3 and 5.4, the problem of host overload detection can be formulated as an optimization problem (5.6)-(5.7).

$$t_a(t_m, u_t) \rightarrow \max \tag{5.6}$$

$$\frac{t_o(t_m,u_t)}{t_a(t_m,u_t)} \leq M, \tag{5.7}$$

where t_m is the time when a VM migration has been initiated; u_t is the CPU utilization threshold defining the overload state of the host; $t_o(t_m, u_t)$ is the time, during which the host has been overloaded, which is a function of t_m and u_t; t_a is the total time, during which the host has been active, which is also a function of t_m and u_t; and M is the limit on the maximum allowed OTF value, which is a QoS goal expressed in terms of OTF. The aim of a host overload detection algorithm is to select the t_m that maximizes the total time until a migration, while satisfying the constraint (5.7). It is important to note that the optimization problem (5.6)-(5.7) is only relevant to host *overload* detection, and does not relate to host underload situations. In other words, maximizing the activity time of a host is only important for highly loaded hosts. Whereas for underloaded hosts, the problem is the opposite – the activity time needs to be minimized; however, this problem is not the focus of the current chapter and should be investigated separately.

In the offline setting, the state of the system is known at any point in time. Consider an offline algorithm that passes through the history of system states backwards starting

from the last known state. The algorithm decrements the time and re-calculates the OTF value $\frac{t_o(t_m,u_t)}{t_a(t_m,u_t)}$ at each iteration. The algorithm returns the time that corresponds to the current iteration if the constraint (5.7) is satisfied (Algorithm 2).

Algorithm 2 The Optimal Offline (OPT) algorithm

Input: A system state *history*
Input: M, the maximum allowed OTF
Output: A VM migration time
1: **while** *history* is not empty **do**
2: **if** OTF of *history* $\leq M$ **then**
3: **return** the time of the last *history* state
4: **else**
5: drop the last state from *history*

Theorem 5.1. *Algorithm 2 is an optimal offline algorithm (OPT) for the problem of host overload detection.*

Proof. Let the time interval covered by the system state history be $[t_0, t_n]$, and t_m be the time returned by Algorithm 2. Then, according to the algorithm the system states corresponding to the time interval $(t_m, t_n]$ do not satisfy the constraint (5.7). Since t_m is the right bound of the interval $[t_0, t_m]$, then t_m is the maximum possible time that satisfies the constraint (5.7). Therefore, t_m is the solution of the optimization problem (5.6)-(5.7), and Algorithm 2 is an optimal offline algorithm for the host overload detection problem. □

5.6 A Markov Chain Model for Host Overload Detection

In this section, the proposed model is based on the definitions of Markov chains, a mathematical framework for statistical modeling of real-world processes.

5.6.1 Background on Markov Chain

This section introduces the basic definitions of the Markov chains modeling framework. Bolch [24] provides a detailed introduction to Markov chains.

A stochastic process $\{X_0, X_1, \ldots, X_{n+1}, \ldots\}$ at the consecutive points of observation $0, 1, \ldots, n+1$ constitutes a *Discrete-Time Markov Chain* (DTMC) if the following relation on the conditional Probability Mass Function (PMF) holds $\forall n \in \mathbb{N}_0$, and $\forall s_i \in \mathcal{S} = \mathbb{N}_0$:

$$P(X_{n+1} = s_{n+1} | X_n = s_n, X_{n-1} = s_{n-1}, \ldots, X_0 = s_0) = P(X_{n+1} = s_{n+1} | X_n = s_n). \quad (5.8)$$

Given an initial state s_0, a DTMC evolves step by step according to the *one-step transition probabilities*:

$$p_{ij}^{(1)}(n) = P(X_{n+1} = s_{n+1} = j | X_n = s_n = i). \quad (5.9)$$

If the conditional PMF is independent of the *time parameter* n, the DTMC is referred to as *time-homogeneous* and (5.9) reduces to: $p_{ij} = P(X_{n+1} = j | X_n = i)\ \forall n \in T$. Starting from a state i, the DTMC transitions to a state j, so that $\sum_j p_{ij} = 1$, where $0 \leq p_{ij} \leq 1$. The one-step transition probabilities p_{ij} are usually summarized in a non-negative transition probability matrix $\mathbf{P} = [p_{ij}]$.

Let $t \in \mathcal{T}$ be the *time parameter*, where $\mathcal{T} \subseteq \mathbb{R}^+ = [0, \infty)$; let $\mathcal{S}$ be the *state space* of the stochastic process comprising all possible values of X_t (for each $t \in \mathcal{T}$). A stochastic process $\{X_t : t \in \mathcal{T}\}$ constitutes a *Markov process* if for all $0 = t_0 < t_1 < \ldots < t_n < t_{n+1}$, $\forall n \in \mathbb{N}$, and $\forall s_i \in \mathcal{S}$ the conditional Cumulative Distribution Function (CDF) of $X_{t_{n+1}}$ depends only on the previous value X_{t_n} and not on the earlier values $X_{t_0}, X_{t_1}, \ldots, X_{t_{n-1}}$:

$$P(X_{t_{n+1}} \leq s_{n+1} | X_{t_n} = s_n, X_{t_{n-1}} = s_{n-1}, \ldots, X_{t_0} = s_0) = P(X_{t_{n+1}} \leq s_{n+1} | X_{t_n} = s_n). \quad (5.10)$$

A stochastic process $\{X_t : t \in \mathcal{T}\}$ constitutes a *Continuous-Time Markov Chain* (CTMC) if for arbitrary $t_i \in \mathbb{R}_0^+$, with $0 = t_0 < t_1 < \ldots < t_n < t_{n+1}$, $\forall n \in \mathbb{N}$, and $\forall s_i \in \mathcal{S} = \mathbb{N}_0$ for the conditional PMF, the relation (5.10) holds. In other words, a CTMC is a Markov process restricted to a discrete, finite, or countably infinite state space $\mathcal{S}$, and a continuous-parameter space $\mathcal{T}$. The right-hand side of (5.10) is referred to as the *transition probability* $p_{ij}(u, v)$ of the CTMC to travel from state i to state j during the period of time $[u, v)$, with $u, v \in \mathcal{T}$ and $u \leq v$: $p_{ij}(u, v) = P(X_v = j | X_u = i)$. If the transition probabilities $p_{ij}(u, v)$ depend only on the time difference $t = v - u$ and not on the actual values of u and v, the CTMC is *time-homogeneous* with simplified transition probabilities: $p_{ij}(t) = P(X_{u+t} = j | X_u = i)$.

The analysis in this chapter focuses on time-homogeneous Markov chains, which

can also be described as Markov chains with stationary transition probabilities. Time-homogeneous Markov chains correspond to stationary workloads, i.e., workloads, whose statistical properties do not change over time. Section 5.7 shows how a time-homogeneous Markov model can be adapted to cases of non-stationary workloads.

Another characteristic that describes transitions of a CTMC between the states is the *instantaneous transition rate* $q_{ij}(t)$ of the CTMC traveling from state i to state j. The non-negative, finite, continuous functions $q_{ij}(t)$ satisfy the following conditions:

$$\begin{aligned} q_{ij}(t) &= \lim_{\Delta t \to 0} \frac{p_{ij}(t, t+\Delta t)}{\Delta t}, \quad i \neq j, \\ q_{ii}(t) &= \lim_{\Delta t \to 0} \frac{p_{ii}(t, t+\Delta t) - 1}{\Delta t}, \end{aligned} \tag{5.11}$$

where Δt is chosen such that $\sum_{j \in \mathcal{S}} q_{ij}(t)\Delta t + o(\Delta t) = 1; i, j \in \mathcal{S}$. A matrix $\mathbf{Q} = [q_{ij}]\ \forall i, j \in \mathcal{S}$ is called the *infinitesimal generator matrix* of the transition probability matrix $\mathbf{P}(t) = [p_{ij}(t)]$. The elements q_{ii} on the main diagonal of $\mathbf{Q}$ are given by $q_{ii} = -\sum_{j \in \mathcal{S}, j \neq i} q_{ij}$.

A vector $\boldsymbol{\pi}(t) = [\pi_i(t)]\ \forall i \in \mathcal{S}$ contains the probabilities that the CTMC will be in the state i at the time t. Using the Kolmogorov forward equation [24], the following equation for the *unconditional state probability vector* $\boldsymbol{\pi}(t)$ can be derived:

$$\frac{\mathrm{d}\boldsymbol{\pi}(t)}{\mathrm{d}t} = \boldsymbol{\pi}(t)\mathbf{Q}. \tag{5.12}$$

A transition probability matrix $\mathbf{P}$ of an ergodic DTMC (e.g., a DTMC with all the transition probabilities being non-zero) can be transformed into an infinitesimal generator matrix of the corresponding CTMC as follows:

$$\mathbf{Q} = \mathbf{P} - \mathbf{I}, \tag{5.13}$$

where $\mathbf{I}$ is the identity matrix. Next, using the definitions given in this section, a Markov chain model for the host overload detection problem is introduced.

5.6.2 The Host Model

Each VM allocated to a host at each point in time utilizes a part of the CPU capacity determined by the application workload. The CPU utilization created over a period of time by a set of VMs allocated to a host constitutes the host's workload. For the initial

analysis, it is assumed that the workload is known *a priori*, stationary, and satisfies the Markov property. In other words, the CPU utilization of a host measured at discrete time steps can be described by a single time-homogeneous DTMC.

There is a controller component, which monitors the CPU utilization of the host and according to a host overload detection algorithm decides when a VM should be migrated from the host to satisfy the QoS requirements, while maximizing the time between VM migrations. According to Section 5.5, the problem formulation is limited to a single VM migration, i.e., the time span of a problem instance is from the end of a previous VM migration to the end of the next.

To describe a host as a DTMC, states are assigned to N subsequent intervals of the CPU utilization. For example, if $N = 11$, the state 1 is assigned to all possible values of the CPU utilization within the interval $[0\%, 10\%)$, 2 to the CPU utilization within $[10\%, 20\%)$, $\ldots$, N to the value 100%. The state space $\mathcal{S}$ of the DTMC contains N states, which correspond to the defined CPU utilization intervals. Using this state definition and knowing the workload of a host in advance, by applying the Maximum Likelihood Estimation (MLE) method it is possible to derive a matrix of transition probabilities $\mathbf{P}$. The matrix is constructed by estimating the probabilities of transitions $\widehat{p}_{ij} = \frac{c_{ij}}{\sum_{k \in \mathcal{S}} c_{ik}}$ between the defined N states of the DTMC for $i, j \in \mathcal{S}$, where c_{ij} is the number of transitions between states i and j.

An additional state $(N + 1)$ is added to the Markov chain called an *absorbing state*. A state $k \in \mathcal{S}$ is said to be an absorbing state if and only if no other state of the Markov chain can be reached from it, i.e., $p_{kk} = 1$. In other words, once the Markov chain reaches the state k, it stays in that state indefinitely. The resulting extended state space is $\mathcal{S}^* = \mathcal{S} \cup \{(N + 1)\}$. For the problem discussed in this chapter, the absorbing state $(N + 1)$ represents the state where the DTMC transitions once a VM migration is initiated. According to this definition, the control policy can be described by a vector of the probabilities of transitions from any non-absorbing state to the absorbing state $(N + 1)$, i.e., the probabilities of VM migrations, which are denoted m_i, where $i \in \mathcal{S}$. To add the state $(N + 1)$ into the model, the initial transition probability matrix $\mathbf{P}$ is extended with a column of unknown transition probabilities $\mathbf{m} = [m_i] \; \forall i \in \mathcal{S}$ resulting in an extended matrix of transition probabilities $\mathbf{P}^*$:

$$\mathbf{P}^* = \begin{pmatrix} p^*_{11} & \cdots & p^*_{1N} & m_1 \\ \vdots & \ddots & \vdots & \vdots \\ p^*_{N1} & \cdots & p^*_{NN} & m_N \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{5.14}$$

where p^*_{ij} are defined as follows:

$$p^*_{ij} = p_{ij}(1 - m_i), \quad \forall i, j \in \mathcal{S}. \tag{5.15}$$

In general, the workload experienced by the host's VMs can lead to any CPU utilization from 0% to 100%; therefore, the original DTMC can be assumed to be ergodic. Later, the extended DTMC will be restricted to the states in $\mathcal{S}$; therefore, using $\mathbf{Q} = \mathbf{P} - \mathbf{I}$ [24], the extended matrix of transition probabilities $\mathbf{P}^*$ can be transformed into a corresponding extended matrix of transition rates $\mathbf{Q}^*$:

$$\mathbf{Q}^* = \begin{pmatrix} p^*_{11} - 1 & \cdots & p^*_{1N} & m_1 \\ \vdots & \ddots & \vdots & \vdots \\ p^*_{N1} & \cdots & p^*_{NN} - 1 & m_N \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{5.16}$$

In the next section, a QoS constraint is formulated in terms of the introduced model, derived extended matrix of transition rates $\mathbf{Q}^*$, and OTF metric.

5.6.3 The QoS Constraint

Let

$$\mathbf{L}(t) = \int_0^t \boldsymbol{\pi}(u)\, \mathrm{d}u, \tag{5.17}$$

then $\mathbf{L}_i(t)$ denotes the *total expected time* the CTMC spends in the state i during the interval $[0, t)$. By integrating an equation for the *unconditional state probability vector* $\boldsymbol{\pi}(t)$: $\mathrm{d}\boldsymbol{\pi}(t)/\mathrm{d}t = \boldsymbol{\pi}(t)\mathbf{Q}$ on both sides, a new differential equation for $\mathbf{L}(t)$ is derived [24]:

$$\frac{\mathrm{d}\,\mathbf{L}(t)}{\mathrm{d}t} = \mathbf{L}(t)\mathbf{Q} + \boldsymbol{\pi}(0), \quad \mathbf{L}(0) = 0. \tag{5.18}$$

The expected time spent by the CTMC before absorption can be calculated by finding the limit $\mathbf{L}_{\mathcal{S}}(\infty) = \lim_{t\to\infty} \mathbf{L}_{\mathcal{S}}(t)$ restricting the state space to the states in $\mathcal{S}$. The limit exists due to a non-zero probability of a transition to the absorbing state $(N+1)$. However, the limit does not exist for the state $(N+1)$. Therefore, to calculate $\mathbf{L}_{\mathcal{S}}(\infty)$, the extended infinitesimal generator matrix $\mathbf{Q}^*$ is restricted to the states in $\mathcal{S}$, resulting in a matrix $\mathbf{Q}^*_{\mathcal{S}}$ of the size $N \times N$. The initial probability vector $\boldsymbol{\pi}(0)$ is also restricted to the states in $\mathcal{S}$ resulting in $\boldsymbol{\pi}_{\mathcal{S}}(0)$. Restricting the state space to non-absorbing states allows the computation of $\lim_{t\to\infty}$ on both sides of (5.18) resulting in the following *linear* equation [24]:

$$\mathbf{L}_{\mathcal{S}}(\infty)\mathbf{Q}^*_{\mathcal{S}} = -\boldsymbol{\pi}_{\mathcal{S}}(0). \tag{5.19}$$

Let N denote the state of a host when it is overloaded, e.g., when the CPU utilization is equal to 100%, then the expected time spent in the state N before absorption can be calculated by finding $L_N(\infty)$ from a solution of the system of linear equations (5.19). Similarly, the total expected time of the host being active can be found as $\sum_{i\in\mathcal{S}} L_i(\infty)$. Letting the VM migration time be T_m, the expected OTF can be calculated as follows:

$$OTF = \frac{T_m + L_N(\infty)}{T_m + \sum_{i\in\mathcal{S}} L_i(\infty)}. \tag{5.20}$$

5.6.4 The Optimization Problem

By the solution of (5.19), *closed-form* equations for $L_1(\infty), L_2(\infty), \ldots, L_N(\infty)$ are obtained. The unknowns in these equations are $m_1, m_2, \ldots, m_N$, which completely describe the policy of the controller. For the problem investigated in this chapter, the utility function is the total expected time until absorption, as the objective is to maximize the inter-migration time. To introduce the QoS goal in the problem formulation, a limit M on the maximum allowed value of the OTF metric is specified as a constraint resulting in the following optimization problem:

$$\begin{aligned} &\sum_{i\in\mathcal{S}} L_i(\infty) \to \max \\ &\frac{T_m + L_N(\infty)}{T_m + \sum_{i\in\mathcal{S}} L_i(\infty)} \leq M. \end{aligned} \tag{5.21}$$

The equations (5.21) form an NLP problem. The solution of this NLP problem is the vector $\mathbf{m}$ of the probabilities of transitions to the absorbing state, which forms the optimal control policy defined as a PMF $\mathbf{m} = [m_i]\ \forall i \in \mathcal{S}$. At every time step, the optimal control policy migrates a VM with the probability m_i, where $i \in \mathcal{S}$ is the current state. The control policy is *deterministic* if $\exists k \in \mathcal{S} : m_k = 1$ and $\forall i \in \mathcal{S}, i \neq k : m_i = 0$, otherwise the policy is *randomized*.

Since the total time until absorption and T_m are non-negative, the problem formulation (5.21) can be simplified to (5.22).

$$\begin{aligned} &\sum_{i \in \mathcal{S}} L_i(\infty) \rightarrow \max \\ &(1 - M)(T_m + L_N(\infty)) - M \sum_{i \in \mathcal{S}} L_i(\infty) \leq 0. \end{aligned} \tag{5.22}$$

5.6.5 Modeling Assumptions

The introduced model allows the computation of the optimal control policy of a host overload detection controller for a given stationary workload and a given state configuration. It is important to take into account that this result is based on a few fundamental modeling assumptions. First of all, it is assumed that the system satisfies the Markov property, or in other words, the sojourn times (i.e., the time a CTMC remains in a state) are exponentially distributed. Assuming an exponential distribution of sojourn times may not be accurate in many systems. For instance, state transition delays can be deterministic due to a particular task scheduling, or follow other than exponential statistical distribution, such as a bell-shaped distribution. Another implication of the Markov property is the assumption of memoryless state transitions, which means that the future state can be predicted solely based on the knowledge of the current state. It is possible to envision systems, in which future states depend on more than one past state.

Another assumption is that the workload is stationary and known *a priori*, which does not hold in typical computing environments. In the next section, it is shown how the introduced model can be heuristically adapted to handle unknown non-stationary workloads. The proposed heuristically adapted model removes the assumption of stationary and known workloads; however, the assumptions implied by the Markov property must

still hold. In Section 5.10, the proposed heuristically adapted model is evaluated, and the assumptions are tested through a simulation study using real workload traces from more than a thousand PlanetLab VMs. The simulation results show that the model is efficient for this type of mixed computing workloads.

With a correct understanding of the basic model assumptions and careful assessment of the applicability of the proposed model to a particular system, an application of the model can bring substantial performance benefits to the resource management algorithms. As demonstrated by the simulation study in Section 5.10, the proposed approach outperforms the benchmark algorithms in terms of both the mean inter-migration time and the precision of meeting the specified QoS goal.

5.7 Non-Stationary Workloads

The model introduced in Section 5.6 works with the assumption that the workload is stationary and known. However, this is not the case in systems with unknown non-stationary workloads, such as IaaS. One of the ways to adapt the model defined for known stationary workloads to the conditions of initially unknown non-stationary workloads is to apply the Sliding Window workload estimation approach proposed by Chung et al. [34]. The base idea is to approximate a non-stationary workload as a sequence of stationary workloads $\mathcal{U} = (u_1, u_2, \ldots, u_{N_u})$ that are enabled one after another. In this model, the transition probability matrix $\mathbf{P}$ becomes a function of the current stationary workload $\mathbf{P}(u)$.

Chung et al. [34] called a policy that makes ideal decisions for a current stationary workload u_i the *best adaptive policy*. However, the best adaptive policy requires the perfect knowledge of the whole sequence of workloads $\mathcal{U}$ and the times, at which the workloads change. In reality, a model of a workload u_i can only be built based on the observed history of the system behavior. Moreover, the time at which the current workload changes is unknown. Therefore, it is necessary to apply a heuristic that achieves results comparable to the best adaptive policy. According to the Sliding Window approach, a time window of length l_w *slides* over time always capturing last l_w events. Let c_{ij} be the observed number of transitions between states i and j, $i, j \in \mathcal{S}$, during the last window l_w. Then, apply-

ing the MLE method, the transition probability p_{ij} is estimated as $\widehat{p}_{ij} = \frac{c_{ij}}{\sum_{k \in \mathcal{S}} c_{ik}}$. As the window length $l_w \to \infty$, the estimator $\widehat{p}_{ij}$ converges to the real value of the transition probability p_{ij} if the length of the current stationary workload u_i is equal to l_w [34].

However, the Sliding Window approach introduces 3 sources of errors in the estimated workload:

1. The biased estimation error, which appears when the window length l_w is shorter than the length of a sequence of outliers.
2. The resolution error (referred to as the sampling error by Luiz et al. [80]), which is introduced due to the maximum precision of the estimates being limited to $1/l_w$.
3. The adaptation time (referred to as the identification delay by Luiz et al. [80]), which is a delay required to completely fill the window with new data after a switch from a stationary workload u_{i-1} to a new stationary workload u_i.

Luiz et al. [80] extended the Sliding Window approach by employing multiple windows with different sizes, where a window to use is selected dynamically using the information about the previous system state and variances of the estimates obtained from different windows. They referred to the extended approach as the Multisize Sliding Window approach. The proposed algorithm dynamically selects the best window size to eliminate the bias estimate error and benefit from both the small sampling error of large window sizes and small identification error of small window sizes. In this chapter, the Multisize Sliding Window approach is applied to the model introduced in Section 5.6 to adapt it to initially unknown non-stationary workloads.

The calculation of the expected OTF (5.20) is adapted by transforming it to a function of $t \in \mathbb{R}^+$ to incorporate the information that is known by the algorithm at the time of decision making:

$$OTF(t) = \frac{T_m + y(t) + L_N(\infty)}{T_m + t + \sum_{i \in \mathcal{S}} L_i(\infty)}, \tag{5.23}$$

where $y(t)$ is a function returning the total time spent in the state N during the time interval $[0, t]$.

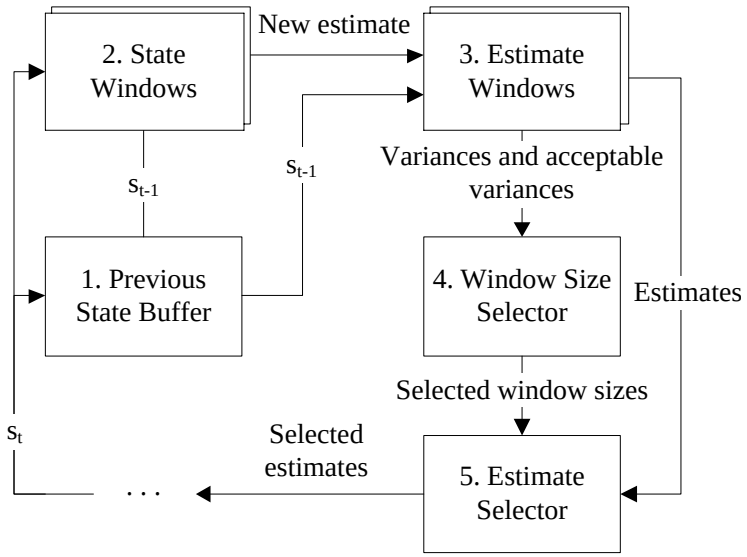


Figure 5.1: The Multisize Sliding Window workload estimation

5.7.1 Multisize Sliding Window Workload Estimation

This section briefly introduces the Multisize Sliding Window approach; for more details, reasoning and analysis please refer to Luiz et al. [80]. A high level view of the estimation algorithm is shown in Figure 5.1. First of all, to eliminate the biased estimation error, the previous history is stored separately for each state in $\mathcal{S}$ resulting in S *state windows* W_i, $i = 1, 2, \ldots, S$. Let J, D, and N_J be positive numbers; $\mathcal{L} = (J, J+D, J+2D, \ldots, J+(N_J-1)D)$ a sequence of window sizes; and $l_{w_{\max}} = J + (N_J - 1)D$ the maximum window size. At each time t, the Previous State Buffer stores the system state s_{t-1} at the time $t-1$ and controls the window selector, which selects a window W_i such that $s_{t-1} = i$. The notation $W_i^k(t)$ denotes the content of the window W_i in a position k at the time t. The selected window shifts its content one position to the right to store the current system state: $W_i^{k+1}(t) = W_i^k(t)$, $\forall k = 1, \ldots, l_{w_{\max}}$; discards the rightmost element $W_i^{l_{w\max}}(t)$; and stores s_t in the position $W_i^1(t)$. Once the selected state window W_i is updated, new probability estimates are computed based on this state window for all window sizes as follows:

$$\widehat{p}_{ij}(t,m) = \frac{\sum_{k=1}^{\mathcal{L}_m}(W_i^k(t) == j)}{\mathcal{L}_m}, \tag{5.24}$$

where "==" is the equivalence operation, i.e., $(1 == 1) = 1, (1 == 0) = 0$. A computed probability estimate is stored in N_J out of the SSN_J *estimate windows* $E_{ijm}(t)$, where $i, j \in$

$\mathcal{S}$, and m is the estimate window size index, $1 \leq m \leq N_J$. N_J estimate windows $E_{ijm}(t)$ are selected such that $s_{t-1} = i$ and $s_t = j$, $\forall m = 1, \ldots, N_J$. Similarly to the update process of the state windows, the selected estimate windows shift their contents one position to the right, discard the rightmost element $E_{ijm}^{\mathcal{L}_m}(t)$, and store $\widehat{p}_{ij}(t, \mathcal{L}_m)$ in the position $E_{ijm}^{1}(t)$. To evaluate the precision of the probability estimates, the variance $S(i, j, t, m)$ of the probability estimates obtained from every updated estimate window is estimated:

$$\begin{aligned} \bar{p}_{ij}(t, m)) &= \frac{1}{\mathcal{L}_m} \sum_{k=1}^{\mathcal{L}_m} E_{ijm}^{k}(t), \\ S(i, j, t, m) &= \frac{1}{\mathcal{L}_m - 1} \sum_{k=1}^{\mathcal{L}_m} (E_{ijm}^{k}(t) - \bar{p}_{ij}(t, \mathcal{L}_m))^2, \end{aligned} \tag{5.25}$$

where $\bar{p}_{ij}(t, m)$ is the mean value of the probability estimates calculated from the state window W_i of length $\mathcal{L}_m$. To determine what values of the variance can be considered to be low enough, a function of acceptable variance $V_{ac}(\widehat{p}_{ij}(t, m), m)$ is defined [80]:

$$V_{ac}(\widehat{p}_{ij}(t, m), m) = \frac{\widehat{p}_{ij}(t, \mathcal{L}_m)(1 - \widehat{p}_{ij}(t, \mathcal{L}_m))}{\mathcal{L}_m}. \tag{5.26}$$

Using the function of acceptable variance, probability estimates are considered to be adequate if $S(i, j, t, m) \leq V_{ac}(\widehat{p}_{ij}(t, m), m)$. Based on the definitions given above, a window size selection algorithm can be defined (Algorithm 3). According to the selected window sizes, transition probability estimates are selected from the estimate windows.

Algorithm 3 The window size selection algorithm

Input: J, D, N_J, t, i, j
Output: The selected window size

```
l_w ← J
for k = 0 to N_J − 1 do
    if S(i, j, t, k) ≤ V_ac(p̂_ij(t, k), k) then
        l_w ← J + kD
    else
        break loop
return l_w
```

The presented approach addresses the errors mentioned in Section 5.7 as follows:

1. The biased estimation error is eliminated by introducing dedicated history windows for each state: even if a burst of transitions to a particular state is longer

than the length of the window, the history of transitions from the other states is preserved.

2. The sampling error is minimized by selecting the largest window size constrained by the acceptable variance function.
3. The identification error is minimized by selecting a smaller window size when the variance is high, which can be caused by a change to the next stationary workload.

5.8 The Control Algorithm

A control algorithm based on the model introduced in Section 5.6 is referred to as the Optimal Markov Host Overload Detection (MHOD-OPT) algorithm. The MHOD-OPT algorithm adapted to unknown non-stationary workloads using the Multisize Sliding Window workload estimation technique introduced in Section 5.7 is referred to as the Markov Host Overload Detection (MHOD) algorithm. A high-level view of the MHOD-OPT algorithm is shown in Algorithm 4. In the online setting, the algorithm is invoked periodically at each time step to make a VM migration decision.

Algorithm 4 The MHOD-OPT algorithm

Input: Transition probabilities
Output: A decision on whether to migrate a VM
1: Build the objective and constraint functions
2: Invoke the brute-force search to find the **m** vector
3: **if** a feasible solution exists **then**
4: Extract the VM migration probability
5: **if** the probability is < 1 **then**
6: **return false**
7: **return true**

Closed-form equations for $L_1(\infty), L_2(\infty), \ldots, L_N(\infty)$ are precomputed offline from (5.19); therefore, the run-time computation is not required. The values of transition probabilities are substituted into the equations for $L_1(\infty), L_2(\infty), \ldots, L_N(\infty)$, and the objective and constraint functions of the NLP problem are generated by the algorithm. To solve the NLP problem, a brute-force search algorithm with a step of 0.1 is applied, as its performance was sufficient for the purposes of simulations. In MHOD-OPT, a decision to migrate a VM is made only if either no feasible solution can be found, or the migration

probability corresponding to the current state is 1. The justification for this is the fact that if a feasible solution exists and the migration probability is less than 1, then for the current conditions there is no hard requirement for an immediate migration of a VM.

Algorithm 5 The MHOD algorithm

Input: A CPU utilization history
Output: A decision on whether to migrate a VM
1: **if** the CPU utilization history size $> T_l$ **then**
2: Convert the last CPU utilization value to a state
3: Invoke the Multisize Sliding Window estimation to obtain the estimates of transition probabilities
4: Invoke the MHOD-OPT algorithm
5: **return** the decision returned by MHOD-OPT
6: **return false**

The MHOD algorithm shown in Algorithm 5 can be viewed as a wrapper over the MHOD-OPT algorithm, which adds the Multisize Sliding Window workload estimation. During the initial learning phase T_l, which in this experiments of this chapter was set to 30 time steps, the algorithm does not migrate a VM. Once the learning phase is over, the algorithm applies the Multisize Sliding Window technique to estimate the probabilities of transitions between the states and invokes the MHOD-OPT algorithm passing the transition probability estimates as the argument. The result of the MHOD-OPT algorithm invocation is returned to the user.

5.9 The CPU model

The models and algorithms proposed in this chapter are suitable for both single core and multi-core CPU architectures. The capacity of a single core CPU is modeled in terms of its clock frequency F. A VM's CPU utilization u_i is relative to the VM's CPU frequency f_i and is transformed into a fraction of the host's CPU utilization U. These fractions are summed up over the N VMs allocated to the host to obtain the host's CPU utilization, as shown in (5.27).

$$U = F \sum_{i}^{N} f_i u_i. \tag{5.27}$$

For the purpose of the host overload detection problem, multi-core CPUs are modeled

Table 5.1: An artificial non-stationary workload

	0-60 s	**60-86 s**	**86-160 s**
p_{00}	1.0	0.0	1.0
p_{01}	0.0	1.0	0.0
p_{10}	1.0	0.0	1.0
p_{11}	0.0	1.0	0.0

as proposed in Chapter 4. A multi-core CPU with n cores each having a frequency f is modeled as a single core CPU with the nf frequency. In other words, F in (5.27) is replaced by nf. This simplification is justified, as applications and VMs are not tied down to a specific core, but can by dynamically assigned to an arbitrary core by a time-shared scheduling algorithm. The only physical constraint is that the CPU capacity allocated to a VM cannot exceed the capacity of a single core. Removing this constraint would require the VM to be executed on more than one core in parallel. However, automatic parallelization of VMs and their applications cannot be assumed.

5.10 Performance Evaluation

5.10.1 Importance of Precise Workload Estimation

The purpose of this section is to show that the precision of the workload estimation technique is important to achieve high performance of the MHOD algorithm. To show this, an artificial workload was constructed that illustrates a case when the MHOD algorithm with the Multisize Sliding Window workload estimation leads to lower performance compared to MHOD-OPT due to its inability to adapt quickly enough to a highly non-stationary workload.

it is defined that the host can be in one of two possible states $\{0, 1\}$, where the state 1 means that the host is being overloaded. Let the non-stationary workload be composed of a sequence of three stationary workloads, whose probabilities of transitions between the states are shown in Table 5.1. Simulations were used to evaluate the algorithms. For this experiment, the OTF constraint was set to 30%, and the sequence of window sizes for the Multisize Sliding Window workload estimation was $(30, 40, 50, 60, 70, 80, 90, 100)$. The

Table 5.2: Comparison of MHOD, MHOD-OPT and OPT

	MHOD-30	MHOD-OPT-30	OPT-30
OTF	29.97%	16.30%	16.30%
Time	87	160	160

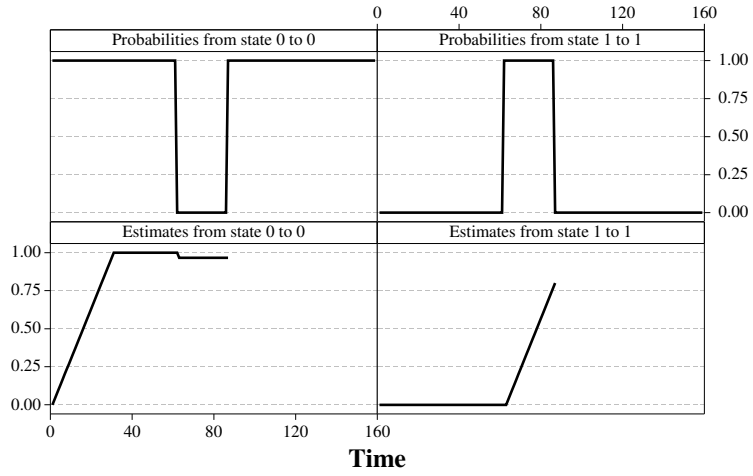


Figure 5.2: The estimated $\widehat{p}_{00}$ compared to p_{00}

code of the simulations is written in Clojure[2]. To foster and encourage reproducibility of experiments, the source code of the simulations has been made publicly available online[3].

The simulation results are shown in Table 5.2. According to the results, for the workload defined in Table 5.1 the MHOD-OPT algorithm provides exactly the same performance as the optimal offline algorithm (OPT). However, the MHOD algorithm migrates a VM at the beginning of the third stationary workload because it is not able to immediately recognize the change of the workload, as shown for p_{00} and $\widehat{p}_{00}$ in Figure 5.2.

In summary, even though the Multisize Sliding Window workload estimation provides high quality of estimation [80], in some cases it may result in an inferior performance of the MHOD algorithm compared to MHOD-OPT. This result was expected, as MHOD-OPT skips the estimation phase and utilizes the knowledge of real transition probabilities. The artificial workload used in this section was specifically constructed to show that imprecise workload estimation may lead to unsatisfactory performance of the MHOD algorithm. However, as shown in the next section, the MHOD algorithm performs closely to OPT for real-world workloads.

[2]The Clojure programming language. http://clojure.org/
[3]The simulation source code. http://github.com/beloglazov/tpds-2013-simulation/

5.10.2 Evaluation Using PlanetLab Workload Traces

In an environment with multiple hosts, the MHOD algorithm operates in a decentralized manner, where independent instances of the algorithm are executed on every host. Therefore, to evaluate the MHOD algorithm under a real-world workload, a single host with a quad-core CPU serving a set of heterogeneous VMs was simulated. The clock frequency of a single core of the host was set to 3 GHz, which according to the model introduced in Section 5.9 transforms into 12 GHz. These CPU characteristics correspond to a mid-range Amazon EC2 physical server type [82]. The amount of the host's memory was assumed to be enough for the VMs. The CPU frequency of a VM was randomly set to one of the values approximately corresponding to the Amazon EC2 instance types[4]: 1.7 GHz, 2 GHz, 2.4 GHz, and 3 GHz. The CPU utilization of the VMs was simulated based on the data provided as a part of the CoMon project, a monitoring infrastructure for PlanetLab [92]. The project provides the data measured every 5 minutes from more than a thousand VMs running in more than 500 locations around the world. For the experiments of this chapter, 10 days were randomly selected from the workload traces collected during March and April 2011.

For a simulation run, a randomly generated set of VMs with the CPU utilization traces assigned is allocated to the host. At each time step, the host overload detection algorithm makes a decision of whether a VM should be migrated from the host. The simulation runs until either the CPU utilization traces are over, or until a decision to migrate a VM is made by the algorithm. At the end of a simulation run, the resulting value of the OTF metric is calculated according to (5.5). The algorithm of assigning the workload traces to a set of VMs is presented in Algorithm 6. To avoid trivial cases and stress the algorithms with more dynamic workloads, the original workload traces were filtered. The maximum allowed OTF after the first 30 time steps was constrained to 10% and the minimum overall OTF was constrained to 20%. Using the workload assignment algorithm, 100 different sets of VMs that meet the defined OTF constraints were pregenerated. Every algorithm was run for each set of VMs. The workload data used in the experiments are publicly available online[5].

[4] Amazon EC2 instance types. http://aws.amazon.com/ec2/instance-types/
[5] The workload data. http://github.com/beloglazov/tpds-2013-workload/

Algorithm 6 The workload trace assignment algorithm

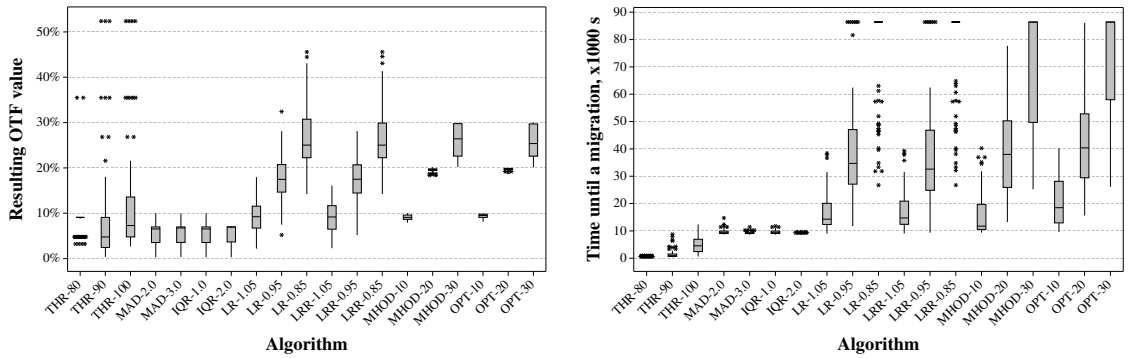
Input: A set of CPU utilization traces
Output: A set of VMs
1: Randomly select the host's minimum CPU utilization at the time 0 from 80%, 85%, 90%, 95%, and 100%
2: **while** the host's utilization $<$ the threshold **do**
3: Randomly select the new VM's CPU frequency
4: Randomly assign a CPU utilization trace
5: Add the new VM to the set of created VMs
6: **return** the set of created VMs

Benchmark Algorithms

In addition to the optimal offline algorithm introduced in Section 5.5, a number of benchmark algorithms were implemented. The benchmark algorithms were run with different parameters to compare with the proposed MHOD algorithm. This section gives a brief overview of the benchmark algorithms; a detailed description of each of them is given in Chapter 4. The first algorithm is a simple heuristic based on setting a CPU utilization threshold (THR), which monitors the host's CPU utilization and migrates a VM if the defined threshold is exceeded. This threshold-based heuristic was applied in a number of related works [54, 55, 121, 135]. The next two algorithms apply statistical analysis to dynamically adapt the CPU utilization threshold: based on the median absolute deviation (MAD), and on the interquartile range (IQR).

Two other algorithms are based on estimation of the future CPU utilization using *local regression* and a modification of the method robust to outliers, referred to as *robust local regression*. These algorithms are denoted Local Regression (LR) and Local Regression Robust (LRR) respectively. The LR algorithm is in line with the regression-based approach proposed by Guenter et al. [57]. Another algorithm continuously monitors the host's OTF and decides to migrate a VM if the current value exceeds the defined parameter. This algorithm is referred to as the OTF Threshold (OTFT) algorithm. The last benchmark algorithm, the OTF Threshold Migration Time (OTFTM) algorithm, is similar to OTFT; however, it uses an extended metric that includes the VM migration time:

$$OTF(t_o, t_a) = \frac{T_m + t_o}{T_m + t_a}, \tag{5.28}$$



(a) The resulting OTF value

(b) The time until a migration

Figure 5.3: The resulting OTF value and time until a migration produced by the MHOD and benchmark algorithms

where t_o is the time, during which the host has been overloaded; t_a is the total time, during which the host has been active; and T_m is the VM migration time.

MHOD Compared with Benchmark Algorithms

To shorten state configuration names of the MHOD algorithm, they are referred to by denoting the thresholds between the utilization intervals. For example, a 3-state configuration $([0\%, 80\%), [80\%, 100\%), 100\%)$ is referred to as 80-100. The following 2- and 3-state configurations of the MHOD algorithm were simulated: 80-100, 90-100, and 100 (a 2-state configuration). Each state configuration with the OTF parameter set to 10%, 20% and 30% was simulated. For experiments, the VM migration time was set to 30 seconds.

In order to find out whether different numbers of states and different state configurations of the MHOD algorithm significantly influence the algorithm's performance in regard to the time until a migration and the resulting OTF value, paired t-tests were conducted. The tests on the produced time until a migration data for comparing MHOD 80-100 with MHOD 100 and MHOD 90-100 with MHOD 100 showed non-statistically significant differences with the p-values 0.20 and 0.34 respectively. This means that the simulated 2- and 3-state configurations of the MHOD algorithm on average lead to approximately the same time until a migration. However, there are statistically significant differences in the resulting OTF value produced by these algorithms: 0.023% with 95% Confidence Interval (CI) (0.001%, 0.004%) and p-value $= 0.033$ for MHOD 100 compared

Table 5.3: Paired T-tests with 95% CIs for comparing the time until a migration produced by MHOD, LR and LRR

Alg. 1 ($\times 10^3$)	**Alg. 2** ($\times 10^3$)	**Diff.** ($\times 10^3$)	p**-value**
MHOD (39.64)	LR (44.29)	4.65 (2.73, 6.57)	< 0.001
MHOD (39.23)	LRR (44.23)	5.00 (3.09, 6.91)	< 0.001

with MHOD 80-100; and 0.022% with 95% CI (0.000%, 0.004%) and p-value $= 0.048$ for MHOD 100 compared with MHOD 90-100. However, differences in the resulting OTF value in the order of less than 0.1% are not practically significant; therefore, the conclusion is that the simulated 2- and 3-state configurations produce approximately the same results. Further in this section, only the $([0\%, 100\%), 100\%)$ 2-state configuration of MHOD is compared with the benchmark algorithms, as it requires simpler computations compared with the 3-state configurations.

The experimental results comparing the 2-state configuration of the MHOD algorithm (for the MHOD algorithm, the OTF parameter is denoted in the suffix of the algorithm's name, e.g., for 10%, 20% and 30%: MHOD-10, MHOD-20 and MHOD-30) with the benchmark algorithms are depicted in Figures 5.3a and 5.3b. It is remarkable how closely the resulting OTF value of the MHOD algorithm resembles the value set as the parameter of the algorithm for 10% and 20%. The wider spread for 30% is explained by the characteristics of the workload: in many cases the overall OTF is lower than 30%, which is also reflected in the resulting OTF of the optimal offline algorithm (OPT-30). The experimental results show that the algorithm is capable of meeting the specified OTF goal, which is consistent with the theoretical model introduced in Section 5.6.

Figures 5.3a and 5.3b show that the THR, MAD and IQR algorithms are not competitive compared with the LR, LRR and MHOD algorithms, as the produced time until a migration is low and does not significantly improve by adjustments of the algorithm parameters. To compare the LR and LRR algorithms with the MHOD algorithms, additional simulations of the MHOD algorithm with the OTF parameter matching the mean value of the resulting OTF produced by LR and LRR were conducted. The following OTF parameter values of the MHOD algorithm were set to match the mean resulting OTF values of LR and LRR: to match LR-1.05, LR-0.95 and LR-0.85 – 9.9%, 18.2% and 31% respectively; to match LRR-1.05, LRR-0.95 and LRR-0.85 – 9.9%, 17.9% and 30.4% respectively.

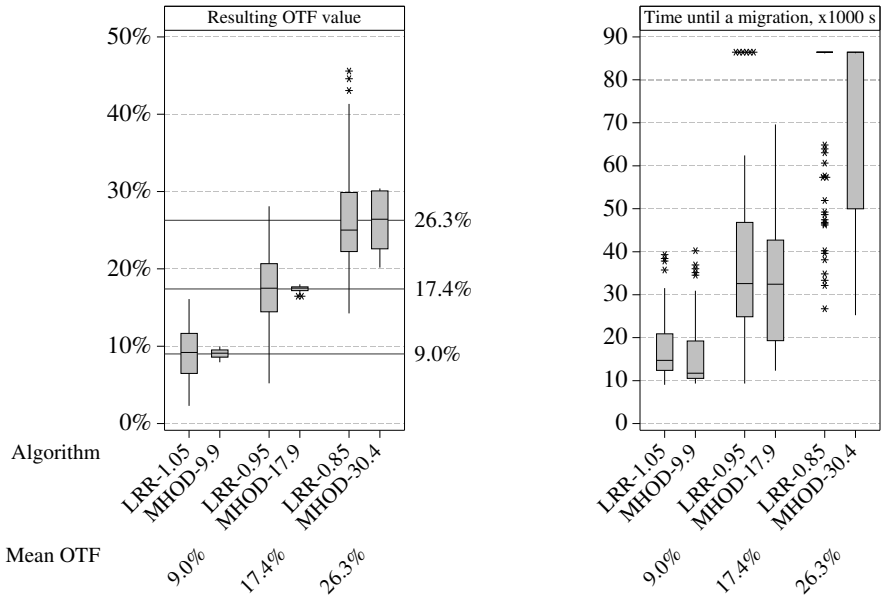


Figure 5.4: Comparison of MHOD with LRR

As intended, paired t-tests for the comparison of MHOD with LR and MHOD with LRR showed non-statistically significant differences in the resulting OTF values with both p-values > 0.9. Results of paired t-tests for comparing the time until a migration produced by the algorithms with matching resulting OTF values are shown in Table 5.3. The MHOD and LRR algorithms are graphically compared in Figure 5.4.

According to the results, there is a statistically significant difference in the time until a migration produced by the algorithms: the MHOD algorithm on average leads to approximately 10.5% and 11.3% shorter time until a migration than LR and LRR respectively with the same mean resulting OTF values. This means that the MHOD algorithm leads to a slightly lower quality of VM consolidation compared with the LR and LRR algorithms, while providing the advantage of explicit specification of a QoS goal in terms of the OTF metric. In contrast, the performance of the LR and LRR algorithms in regard to the QoS can only be adjusted indirectly by tuning the safety parameter. As seen in Figure 5.4, the lower time until a migration produced of the MHOD algorithm can be partially explained by the fact that the spread of the resulting OTF produced by the LRR algorithm is much wider than that of MHOD, while MHOD precisely meets the specified QoS goal. This means that in many cases LRR provides worse QoS than MHOD, which leads to a higher time until a migration.

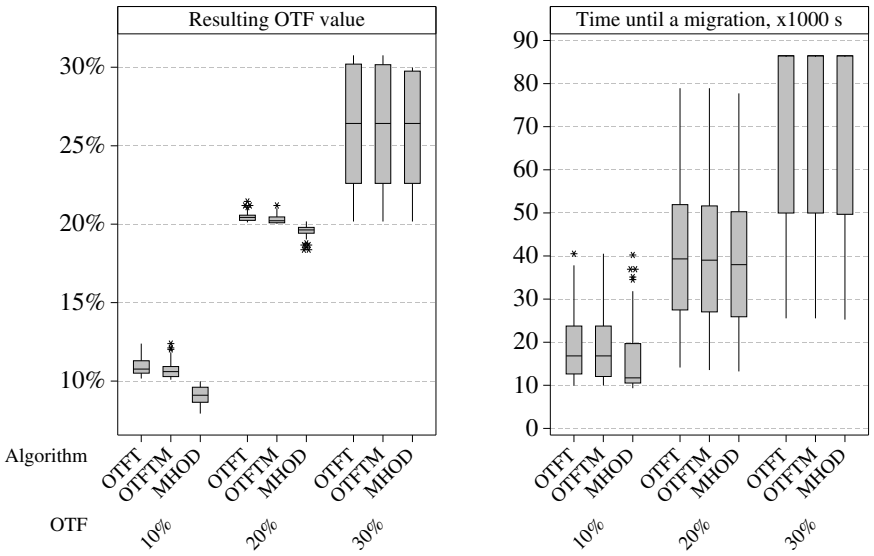


Figure 5.5: Comparison of OTFT, OTFTM and MHOD

Comparison of MHOD with OTFT and OTFTM

OTFT and OTFTM are two other algorithms that apart from the MHOD algorithm allow explicit specification of the QoS goal in terms of the OTF parameter. To compare the performance of the OTFT, OTFTM and MHOD algorithms, another performance metrics introduced. This metric is the percentage of SLA violations relatively to the total number of VM migrations, where SLA requirements are defined as $OTF \leq M$, M is the limit on the maximum allowed resulting OTF value. The SLA violation counter is incremented if after a VM migration the resulting OTF is higher than the value M specified in the SLAs.

The OTFT, OTFTM and MHOD algorithms were simulated using the PlanetLab workload described earlier. The algorithms were simulated with the following values of the OTF parameter set as the SLA requirement: 10%, 20% and 30%. The simulation results are shown in Figure 5.5. The graphs show that MHOD leads to slightly lower resulting OTF values and time until a migration. The SLA violation levels caused by the algorithms are shown in Table 5.4. It is clear that the MHOD algorithm substantially outperforms the OTFT and OTFTM algorithms in the level of SLA violations leading to only 0.33% SLA violations, whereas both OTFT and OTFTM cause SLA violations of 81.33%.

The obtained results can be explained by the fact that both OTFT and OTFTM are unable to capture the overall behavior of the system over time and fail to meet the SLA requirements. In contrast, the MHOD algorithm leverages the knowledge of the past

system states and by estimating future states avoids SLA violations. For instance, in a case of a steep rise in the load, OTFT and OTFTM react too late resulting in an SLA violation. In contrast, MHOD acts more intelligently and by predicting the potential rise migrates a VM before an SLA violation occurs. As a result, for the simulated PlanetLab workload the MHOD algorithm keeps the level of SLA violations at less than 0.5%.

Comparison of MHOD with OPT

Figures 5.3a and 5.3b include the results produced by the optimal offline algorithm (OPT) for the same values of the OTF parameter set for the MHOD algorithm: 10%, 20% and 30%. The results of paired t-tests comparing the performance of OPT with MHOD are shown in Table 5.5. The results show that there is no statistically significant difference in the resulting OTF value, which means that for the simulated PlanetLab workload the MHOD algorithm on average leads to approximately the same level of adherence to the QoS goal as the optimal offline algorithm.

There is a statistically significant difference in the time until a migration with the mean difference of 4,639 with 95% CI: (3617, 5661). Relatively to OPT, the time until a migration produced by the MHOD algorithm converts to 88.02% with 95% CI: (86.07%, 89.97%). This means that for the simulated PlanetLab workload, the MHOD algorithm on average delivers approximately 88% of the performance of the optimal offline algorithm, which is highly efficient for an online algorithm.

5.11 Conclusions

In this chapter, a Markov chain model and control algorithm have been proposed for the problem of host overload detection as a part of dynamic VM consolidation. The model

Table 5.4: SLA violations by OTFT, OTFTM and MHOD

OTF Parameter	OTFT	OTFTM	MHOD
10%	100/100	100/100	0/100
20%	100/100	100/100	1/100
30%	44/100	44/100	0/100
Overall	81.33%	81.33%	0.33%

Table 5.5: Paired T-tests for comparing MHOD with OPT

	OPT	**MHOD**	**Difference**	***p*-value**
OTF	18.31%	18.25%	0.06% (-0.03, 0.15)	$= 0.226$
Time	45,767	41,128	4,639 (3617, 5661)	< 0.001

allows a system administrator to explicitly set a QoS goal in terms of the OTF parameter, which is a workload independent QoS metric. For a known stationary workload and a given state configuration, the control policy obtained from the Markov model optimally solves the host overload detection problem in the online setting by maximizing the mean inter-migration time, while meeting the QoS goal.

Using the Multisize Sliding Window workload estimation approach, the model has been heuristically adapted to handle unknown non-stationary workloads. In addition, an optimal offline algorithm for the problem of host overload detection has been proposed to evaluate the efficiency of the MHOD algorithm. The conducted experimental study has led to the following conclusions:

1. For the simulated PlanetLab workload, 3-state configurations of the MHOD algorithm on average produce approximately the same results as the $([0, 100), 100)$ 2-state configuration of the MHOD algorithm; therefore, the 2-state configuration is preferred, as it requires simpler computations.
2. The 2-state configuration of the MHOD algorithm leads to approximately 11% shorter time until a migration than the LRR algorithm, the best benchmark algorithm. However, the MHOD algorithm provides the advantage of explicit specification of a QoS goal in terms of the OTF metric. In contrast, the performance of the LR and LRR algorithms in regard to the QoS can only be adjusted indirectly by tuning the safety parameter. Moreover, the spread of the resulting OTF value produced by the MHOD algorithm is substantially narrower compared with the LR and LRR algorithms, which means the MHOD algorithm more precisely meets the QoS goal.
3. The MHOD algorithm substantially outperforms the OTFT and OTFTM algorithms in the level of SLA violations resulting in less than 0.5% SLA violations compared to 81.33% of OTFT and OTFTM.
4. The MHOD algorithm on average provides approximately the same resulting OTF

value and approximately 88% of the time until a VM migration produced by the optimal offline algorithm (OPT).

5. The MHOD algorithm enables explicit specification of a desired QoS goal to be delivered by the system through the OTF parameter, which is successfully met by the resulting value of the OTF metric.

The introduced model is based on Markov chains requiring a few fundamental assumptions. It is assumed that the workload satisfies the Markov property, which may not be true for all types of workloads. Careful assessment of the assumptions discussed in Section 5.6.5 is important in an investigation of the applicability of the proposed model to a particular system. However, the experimental study involving multiple mixed heterogeneous real-world workloads has shown that the algorithm is efficient in handling them. For the simulated PlanetLab workload the MHOD algorithm performed within a 12% difference from the performance of the optimal offline algorithm, which is highly efficient for an online algorithm. In the next chapter, the MHOD algorithm is implemented and evaluated as part of a framework for dynamic VM consolidation tailored to OpenStack Clouds[6].

[6]The OpenStack Cloud platform. http://openstack.org/

Chapter 6

OpenStack Neat: A Framework for Distributed Dynamic VM Consolidation

The previous chapters have introduced a distributed approach to energy-efficient dynamic VM consolidation and several algorithms for each of the 4 sub-problems, namely, host underload / overload detection, VM selection, and VM placement. This chapter presents OpenStack Neat, a framework for distributed dynamic VM consolidation in OpenStack Clouds. The implementation of the framework includes the algorithms proposed in the previous chapters. The architecture and implementation of the framework are described in detail, followed by an experimental evaluation on a 5-node OpenStack installation using workload traces from PlanetLab VMs.

6.1 Introduction

THIS chapter introduces an architecture and implementation of *OpenStack Neat*[1], an open source software framework for distributed dynamic VM consolidation in Cloud data centers based on the OpenStack platform[2]. The framework is designed and implemented as a transparent add-on to OpenStack, which means that the OpenStack installation need not be modified or specifically configured to benefit from OpenStack Neat. Figure 6.1 depicts a typical deployment of the key components of OpenStack and OpenStack Neat, which may include multiple instances of compute and controller hosts. The framework acts independently of the base OpenStack platform and applies VM consolidation processes by invoking public Application Programming Interfaces (APIs) of OpenStack. The purpose of the OpenStack Neat framework is twofold: (1) providing a

[1]The OpenStack Neat framework. http://openstack-neat.org/
[2]The OpenStack Cloud platform. http://openstack.org/

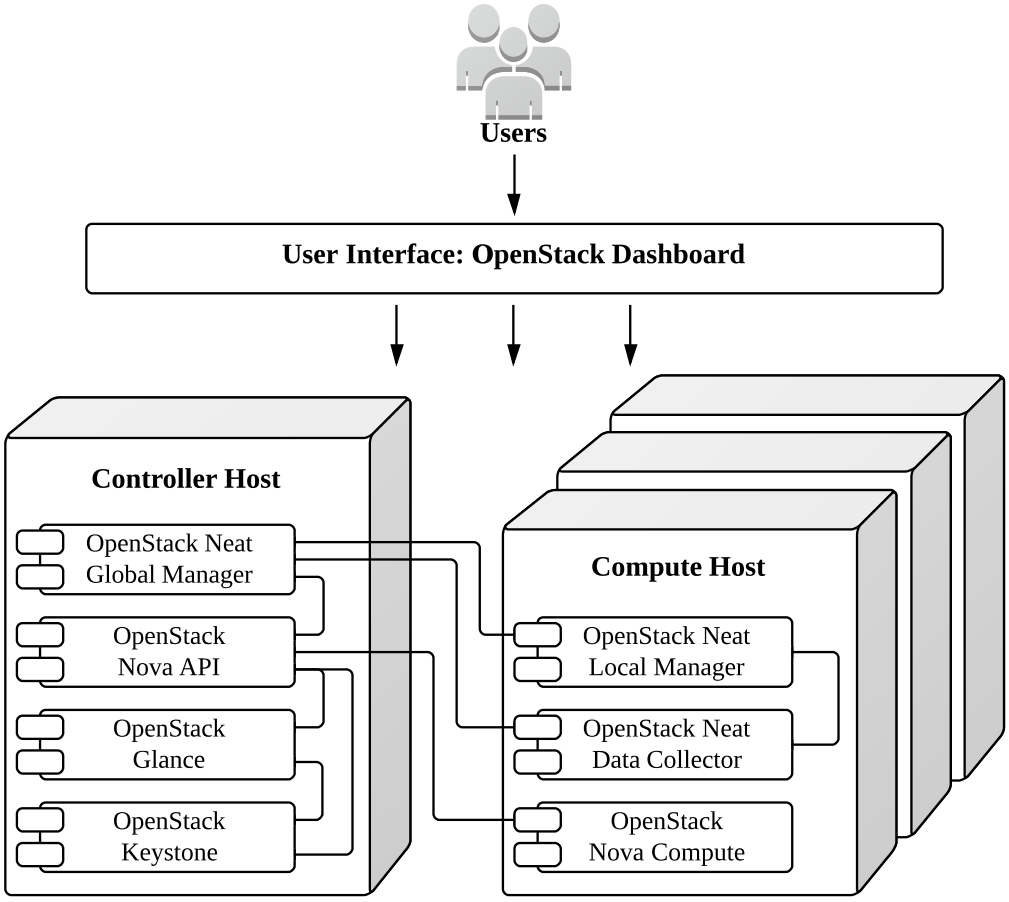


Figure 6.1: The combined deployment of OpenStack and OpenStack Neat

fully operational open source software for dynamic VM consolidation that can be applied to existing OpenStack Clouds; and (2) providing an extensible software framework for conducting research on dynamic VM consolidation.

OpenStack Neat is designed and implemented following the distributed approach to dynamic VM consolidation introduced and evaluated Chapters 4 and 5. The target environment is an Infrastructure as a Service (IaaS), e.g., Amazon EC2, where the provider is unaware of applications and workloads served by the VMs, and can only observe them from outside. The proposed approach to distributed dynamic VM consolidation consists in splitting the problem into 4 sub-problems: underload / overload detection, VM selection, and VM placement, as discussed in Chapter 4.

In addition, to facilitate research efforts and future advancements in the area of dynamic VM consolidation, this chapter outlines a benchmark suite for evaluating and com-

paring dynamic VM consolidation algorithms comprising OpenStack Neat as the base software framework, real-world workload traces from PlanetLab, performance metrics, and evaluation methodology, as discussed in Section 6.6.

The **key contributions** of this chapter are the following:

- An architecture of an extensible software framework for dynamic VM consolidation designed to transparently integrate with OpenStack installations and allowing configuration-based substitution of multiple implementations of algorithms for each of the 4 defined sub-problems of dynamic VM consolidation.
- An open source software implementation of the framework in Python released under the Apache 2.0 license and publicly available online[3].
- An implementation of several algorithms for dynamic VM consolidation proposed and evaluated by simulations in the previous chapters.
- An initial version of a benchmark suite comprising the software framework, workload traces, performance metrics, and methodology for evaluating and comparing dynamic VM consolidation solutions following the distributed model.
- Experimental evaluation of the framework on a 5-node OpenStack deployment using real-world application workload traces collected from more than a thousand PlanetLab VMs hosted on servers located in more than 500 places around the world [92]. According to the estimates of potential energy savings, the algorithms reduce energy consumption by up to 33% with a limited performance impact.

The current implementation of OpenStack Neat assumes a single instance of the controller responsible for placing VMs selected for migrations on hosts. However, due to distributed underload / overload detection and VM selection algorithms, the overall scalability is significantly improved compared with existing centralized solutions. Furthermore, it is potentially possible to implement replication of OpenStack Neat's global manager, which would provide a completely distributed system, as discussed in Section 6.7.4.

The remainder of the chapter is organized as follows. The next section discusses the related work, followed by the overall design and details of each component of the OpenStack Neat framework in Section 6.3. Section 6.4 describes the implemented VM consol-

[3]The OpenStack Neat framework. http://openstack-neat.org/

idation algorithms. Section 6.5 discusses software development techniques and libraries applied in the implementation of the framework. Section 6.6 proposes a benchmark suite for evaluating distributed dynamic VM consolidation algorithms. The experimental evaluation of the framework and analysis of the results are presented in Section 6.7. The chapter is concluded with a summary and discussion of future directions.

6.2 Related Work

Research work related to this paper can be divided into two categories: (1) theoretical work on various approaches to dynamic VM consolidation; and (2) practically implemented and publicly available open source software systems. The first category of work has been discussed in detail in Chapter 2. The framework presented in this chapter follows the distributed approach to dynamic VM consolidation proposed in the previous chapters, where every compute host locally solves the problems of underload / overload detection and VM selection. Then, it sends a request to a global manager to place only the selected for migration VMs on other hosts.

A similar approach was followed by Wood et al. [129] in their system called Sandpiper aimed at load balancing in virtualized data centers using VM live migration. The main objective of the system is to avoid host overloads referred to as hot spots by detecting them and migrating overloaded VMs to less loaded hosts. The authors applied an application-agnostic approach, referred to as a black-box approach, in which VMs are observed from outside, without any knowledge of applications resident in the VMs. A hot spot is detected when the aggregate usage of a host's resources exceeds the specified threshold for k out of n last measurements, as well as for the next predicted value. Another proposed approach is gray-box, when a certain application-specific data are allowed to be collected. The VM placement is computed heuristically by placing the most loaded VM to the least loaded host. The difference from the approach proposed in this chapter is that VMs are not consolidated; therefore, the number of active hosts is not reduced to save energy.

Despite the large volume of research published on the topic of dynamic VM consolidation, there are very few software implementations publicly available online. One of

the earliest open source implementation of a VM consolidation manager is the Entropy project[4]. Entropy is an open source VM consolidation manager for homogeneous clusters developed by Hermenier et al. [59] and released under the LGPL license. Entropy is built on top of Xen and focused on two objectives: (1) maintaining a configuration of the cluster, where all VMs are allocated sufficient resources; and (2) minimizing the number of active hosts.

To optimize the VM placement, Entropy applies a two-phase approach. First, a constraint programming problem is solved to find an optimal VM placement, which minimizes the number of active hosts. Then, another optimization problem is solved to find a target cluster configuration with the minimal number of active hosts that also minimizes the total cost of reconfiguration, which is proportional to the cost of VM migrations. In comparison to OpenStack Neat, Entropy may find a more optimal VM placement by computing a globally optimal solution for VM placement. However, the required optimization problems must be solved by a central controller with limited opportunities for replication, thus limiting the scalability of the system and introducing a single point of failure. This approach is applicable to relatively small-scale private Clouds; however, it cannot be applied to large-scale data centers with tens of thousands of nodes, such as Rackspace [99], where decentralization and fault-tolerance are essential.

Feller et al. [46, 47] proposed and implemented a framework for distributed management of VMs for private Clouds called Snooze[5], which is open source and released under the GPL v2 license. In addition to the functionality provided by the existing Cloud management platforms, such as OpenStack, Eucalyptus, and OpenNebula, Snooze implements dynamic VM consolidation as one of its base features. Another difference is that Snooze implements hierarchical distributed resource management. The management hierarchy is composed of three layers: local controllers on each physical node; group managers managing a set of local controllers; and a group leader dynamically selected from the set of group managers and performing global management tasks. The distributed structure enables fault-tolerance and self-healing by avoiding single points of failure and automatically selecting a new group leader if the current one fails.

Snooze also integrates monitoring of the resource usage by VMs and hosts, which

[4]The Entropy VM manager. http://entropy.gforge.inria.fr/
[5]The Snooze Cloud manager. http://snooze.inria.fr/

can be leveraged by VM consolidation policies. These policies are intended to be implemented at the level of group managers, and therefore can only be applied to subsets of hosts. This approach partially solves the problem of scalability of VM consolidation by the cost of losing the ability of optimizing the VM placement across all the nodes of the data center. OpenStack Neat enables scalability by distributed underload / overload detection and VM selection, and potentially replicating the VM placement controllers. In contrast to Snooze, it is able to apply global VM placement algorithms for the selected for migration VMs by taking into account the full set of hosts. Another difference is that OpenStack Neat transparently integrates with OpenStack, an established open source Cloud platform widely adopted and supported by the industry, thus ensuring long-term development of the platform.

6.3 System Design

The aim of the OpenStack Neat project is to provide an extensible framework for dynamic consolidation of VMs based on the OpenStack platform. Extensibility in this context means the ability to implement new VM consolidation algorithms and apply them in OpenStack Neat without the necessity to modify the source code of the framework itself. Different implementations of the algorithms can be plugged into the framework by modifying the appropriate options in the configuration file. More information on configuring and extending the framework is given in Sections 6.3.8 and 6.3.9 respectively.

OpenStack Neat provides an infrastructure required for monitoring VMs and hypervisors, collecting resource usage data, transmitting messages and commands between the system components, and invoking VM live migrations. The infrastructure is agnostic to VM consolidation algorithms in use and allows implementing custom decision-making algorithms for each of the 4 sub-problems of dynamic VM consolidation: host underload / overload detection, VM selection, and VM placement. The implementation of the framework includes the algorithms proposed in the previous chapters. The following sections discuss the requirements and assumptions, integration of the proposed framework with OpenStack, each of the framework's components, as well as configuration and extensibility of the framework.

6.3.1 Requirements and Assumptions

The components of the framework are implemented in the form of OS services running on the compute and controller hosts of the data center in addition to the core OpenStack services. The framework components interact through a Representational State Transfer (REST) interface; therefore, network communication via the corresponding port specified in the framework's configuration must be enabled.

OpenStack Neat relies on live migration to dynamically relocate VMs across physical machines. To enable live migration, it is required to set up a *shared storage* and correspondingly configure OpenStack Nova (i.e. the OpenStack Compute service) to use this storage for storing VM instance data. For instance, a shared storage can be provided using the Network File System (NFS), or the GlusterFS distributed file system [19].

OpenStack Neat uses a database for storing information about VMs and hosts, as well as resource usage data. It is possible to use the same database server used by the core OpenStack services. In this case, it is only required to create a new database and user for OpenStack Neat. The required database tables are automatically created by OpenStack Neat on the first launch of its services.

Another requirement is that all the compute hosts must have a user, which is enabled to switch the host into a low-power mode, such as *Suspend to RAM*. This user account is used by the global manager to connect to the compute hosts via the Secure Shell (SSH) protocol and switch them into the sleep mode when necessary. More information on deactivating and reactivating physical nodes is given in Section 6.3.4.

Since OpenStack Neat is implemented in Python, VM consolidation algorithms to be plugged in should also be implemented in Python. It may be required to implement VM consolidation algorithms in another programming language for various reasons, such as performance requirements. Integration of such algorithms can be achieved by providing Python wrappers that redirect calls to the corresponding external programs.

6.3.2 Integration with OpenStack

OpenStack Neat services are installed independently of the core OpenStack services. Moreover, the activity of the OpenStack Neat services is transparent to the core Open-

Stack services. This means that OpenStack does not need to be configured in a special way to be able to take advantage of dynamic VM consolidation implemented by OpenStack Neat. It also means, that OpenStack Neat can be added to an existing OpenStack installation without the need to modify its configuration.

The transparency is achieved by the independent resource monitoring implemented by OpenStack Neat, and the interaction with the core OpenStack services using their public APIs. The OpenStack APIs are used for obtaining information about the current state of the system and performing VM migrations. In particular, the APIs are used to get the current mapping of VMs to hosts, hardware characteristics of hosts, parameters of VM flavors (i.e., instance types), VM states, and invoke VM live migrations. Although OpenStack Neat performs actions affecting the current state of the system by relocating VMs across hosts, it is transparently handled by the core OpenStack services since VM migrations are invoked via the public OpenStack APIs, which is equivalent to invoking VM migrations manually by the system administrator.

In the following sections, hosts running the Nova Compute service, i.e., hosting VM instances, are referred to as compute hosts; and a host running the other OpenStack management services but not hosting VM instances is referred to as the controller host.

6.3.3 System Components

OpenStack Neat is composed of a number of components and data stores, some of which are deployed on the compute hosts, and some on the controller host, which can potentially have multiple replicas. As shown in Figure 6.2, the system is composed of three main components:

- *Global manager* – a component that is deployed on the controller host and makes global management decisions, such as mapping VM instances to hosts, and initiating VM live migrations.
- *Local manager* – a component that is deployed on every compute host and makes local decisions, such as deciding that the host is underloaded or overloaded, and selecting VMs to migrate to other hosts.
- *Data collector* – a component that is deployed on every compute host and is responsible for collecting data on the resource usage by VM instances and hypervisors,

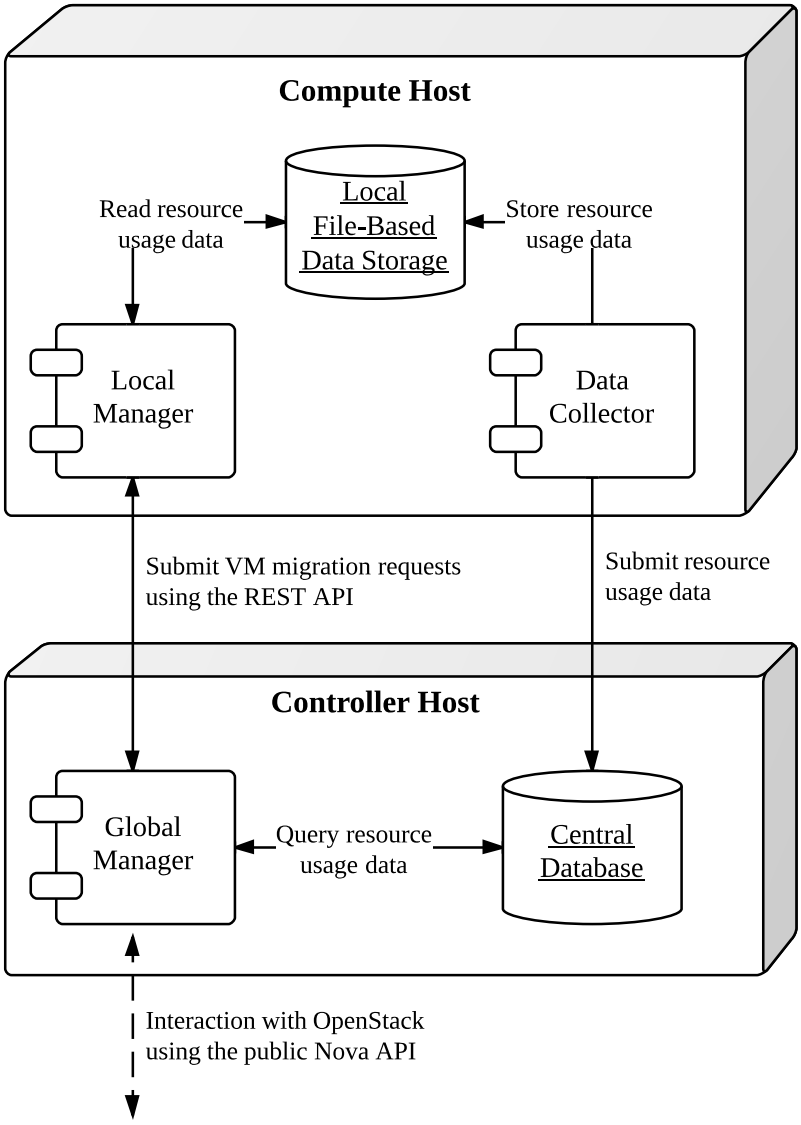


Figure 6.2: The deployment diagram

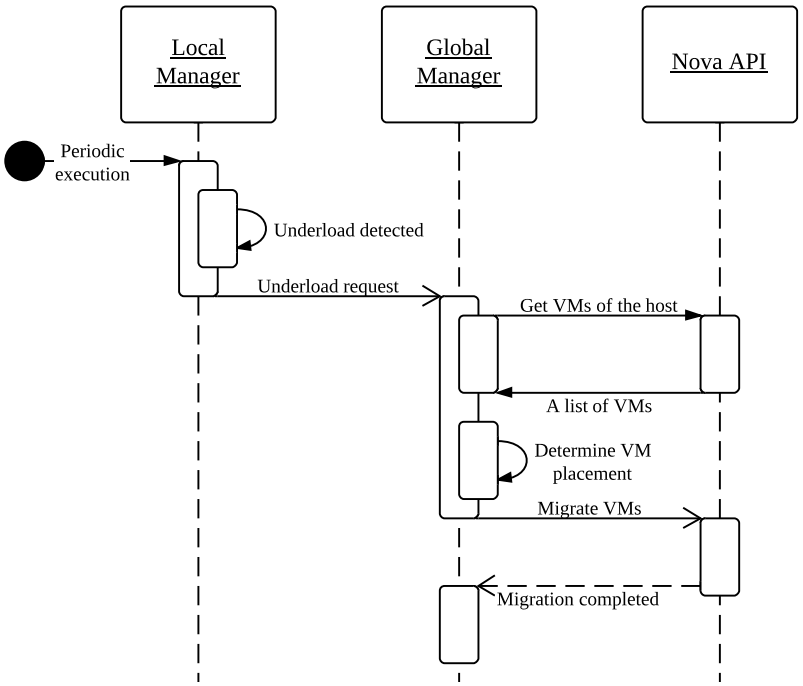


Figure 6.3: The global manager: a sequence diagram of handling an underload request

and then storing the data locally and submitting it to the central database.

The deployment model may vary for each particular system depending on its requirements. For instance, the central database can be deployed on a separate physical node, or be distributed across multiple physical nodes. The location and deployment of the database server is transparent to OpenStack Neat, which only requires a configuration parameter to be set to the network address of the database front-end server. For simplicity, in the experimental testbed used in this chapter, the database server is deployed on the same physical node hosting the global manager, as shown in Figure 6.2.

6.3.4 The Global Manager

The global manager is deployed on the controller host and is responsible for making VM placement decisions and initiating VM migrations. It exposes a REST web service, which accepts requests from local managers. The global manager processes two types of requests: (1) relocating VMs from an underloaded host; and (2) offloading a number of VMs from an overloaded host.

Figure 6.3 shows a sequence diagram of handling a host underload request by the

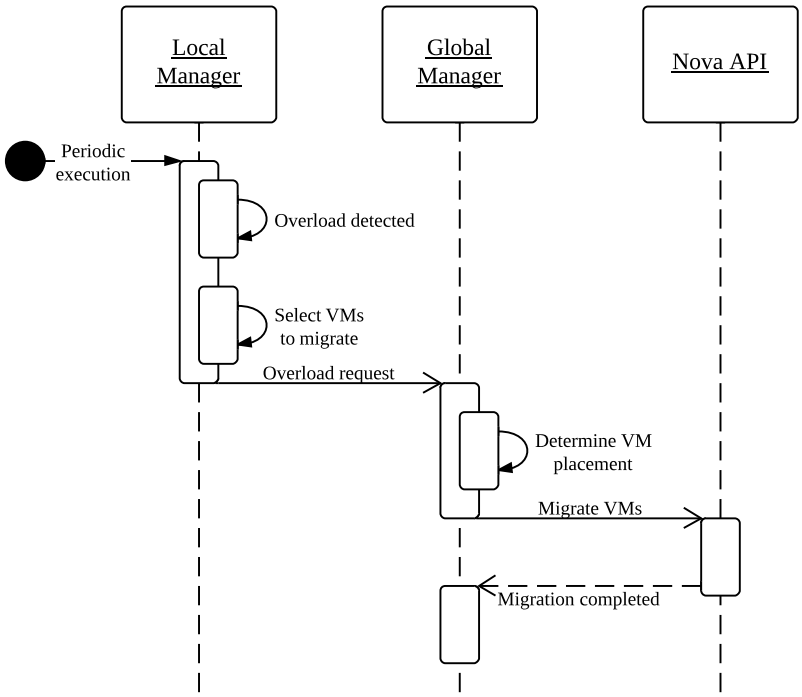


Figure 6.4: The global manager: a sequence diagram of handling an overload request

global manager. First, a local manager detects an underload of the host using the specified in the configuration underload detection algorithm. Then, it sends an underload request to the global manager including the name of the underloaded host. The global manager calls the OpenStack Nova API to obtain the list of VM currently allocated to the underloaded host. Once the list of VMs is received, the global manager invokes the VM placement algorithm with the received list of VMs along with their resource usage and states of hosts fetched from the database as arguments. Then, according to the VM placement generated by the algorithm, the global manager submits the appropriate VM live migration requests to the OpenStack Nova API, and monitors the VM migration process to determine when the migrations are completed. Upon the completion of the VM migrations, the global manager switches the now idle source host into the sleep mode using the procedure described in Section 6.3.4.

As shown in Figure 6.4, handling overload requests is similar to underload requests. The difference is that instead of sending just the host name, the local manager also sends a list of UUIDs of the VMs selected by the configured VM selection algorithm to be offloaded from the overloaded host. Once the request is received, the global manager in-

vokes the specified in the configuration VM placement algorithm and passes as arguments the list of VMs received from the local manager to be placed on other hosts along with other system information. If some of the VMs are placed on hosts that are currently in the sleep mode, the global manager reactivates them using the Wake-on-LAN technology, as described in Section 6.3.4. Then, similarly to handling underload requests, the global manager submits VM live migration requests to the OpenStack Nova API.

REST API

The global manager exposes a REST web service (REST API) for processing VM migration requests sent by local managers. The service Uniform Resource Locator (URL) is defined according to configuration options specified in */etc/neat/neat.conf*, which is discussed in detail in Section 6.3.8. The two relevant options are:

- *global_manager_host* – the name of the host running the global manager;
- *global_manager_port* – the port that should be used by the web service to receive requests.

Using these configuration options, the service URL is composed according to the following template: *http://global_manager_host:global_manager_port/*. The global manager processes two types of requests from local managers: host underloads, and host overloads discussed in the previous section. Both types of requests are served at a single resource '/' accessed using the PUT method of the Hypertext Transfer Protocol (HTTP). The type of a received request is determined by the global manager by analyzing the parameters included in the request. The following parameters are common to both types of requests:

- *username* – the admin user name specified in the configuration file, which is used to authenticate the client making the request as being allowed to access the web service. This parameter is sent SHA-1-encrypted to avoid sending the user name in the open form over the network.
- *password* – the admin password specified in the configuration file, which is used to authenticate the client making the request as being allowed to access the web service. Similarly to *username*, this parameter is also sent encrypted with the SHA-1 algorithm.
- *time* – the time when the request has been sent. This parameter is used by the

global manager to identify and enforce time-outs, which may happen if a request has been sent a long time ago rendering it non-representative of the current state of the system.

- *host* – the host name of the overloaded or underloaded host, where the local manager sending the request is deployed on.
- *reason* – an integer specifying the type of the request, where 0 represents a host underload request, and 1 represents a host overload request.

If the request type specified by the *reason* parameter is 1 (i.e., denoting an overload request), there is an extra mandatory parameter *vm_uuids*. This is a string parameter, which must contain a coma-separated list of Universally Unique Identifiers (UUIDs) of VMs selected for migration from the overloaded host.

If a request contains all the required parameters and the provided credentials are correct, the service responds with the HTTP status code *200 OK*. The service uses standard HTTP error codes to respond in cases of errors. The following error codes are used:

- 400 – bad input parameter: incorrect or missing parameters;
- 401 – unauthorized: user credentials are missing;
- 403 – forbidden: user credentials do not much the ones specified in the configuration file;
- 405 – method not allowed: the request has been made with a method other than the only supported PUT method;
- 422 – precondition failed: the request has been sent more than 5 seconds ago, which means that the states of the hosts or VMs may have changed – a retry is required.

Switching Power States of Hosts

One of the main features required to be supported by the hardware and OS in order to take advantage of dynamic VM consolidation to save energy is the Advanced Configuration and Power Interface (ACPI). The ACPI standard defines platform-independent interfaces for power management by the OS. The standard is supported by Linux, the target OS for the OpenStack platform. ACPI defines several sets of power states, the most relevant of which is the sleep state S3, referred to as *Suspend to RAM*. Meisner et al. [81] showed that power consumption of a typical blade server can be reduced from 450 W in

the active state to just 10.4 W in the S3 state. The transition latency is currently mostly constrained by the Power Supply Unit (PSU) of the server, which leads to the total latency of approximately 300 ms. This latency is acceptable for the purposes of dynamic VM consolidation, as VM live migrations usually take tens of seconds.

The Linux OS provides an API to programmatically switch the physical machine into the sleep mode. In particular, CentOS supports a *pm-utils* package, which includes command line programs for changing the power state of the machine. First of all, to check whether the Suspend to RAM state is supported, the following command can be used: `pm-is-supported --suspend`. If the command returns 0, the Suspend to RAM state is supported, otherwise it is not supported. If the state is supported, the following command can be used to enable it: `pm-suspend`.

It is possible to reactivate a physical machine over the network using the Wake-on-LAN technology. This technology has been introduced in 1997 by the Advanced Manageability Alliance (AMA) formed by Intel and IBM, and is currently supported by most modern servers. To reactivate a server using Wake-on-LAN, it is necessary to send over the network a special packet, called the *magic packet*. This can be done using the *etherwake* Linux program as follows: `ether-wake -i interface mac_address`, where `interface` is replaced with the name of the network interface to send the packet from, and `mac_address` is replaced with the actual Media Access Control (MAC) address of the host to be reactivated.

6.3.5 The Local Manager

The local manager component is deployed on every compute host as an OS service running in the background. The service periodically executes a function that determines whether it is necessary to reallocate VMs from the host. A high-level view of the workflow performed by the local manager is shown in Figure 6.5. At the beginning of each iteration it reads from the local storage the historical data on the resource usage by the VMs and hypervisor stored by the data collector. Then, the local manager invokes the specified in the configuration underload detection algorithm to determine whether the host is underloaded. If the host is underloaded, the local manager sends an underload request to the global manager's REST API to migrate all the VMs from the host and switch

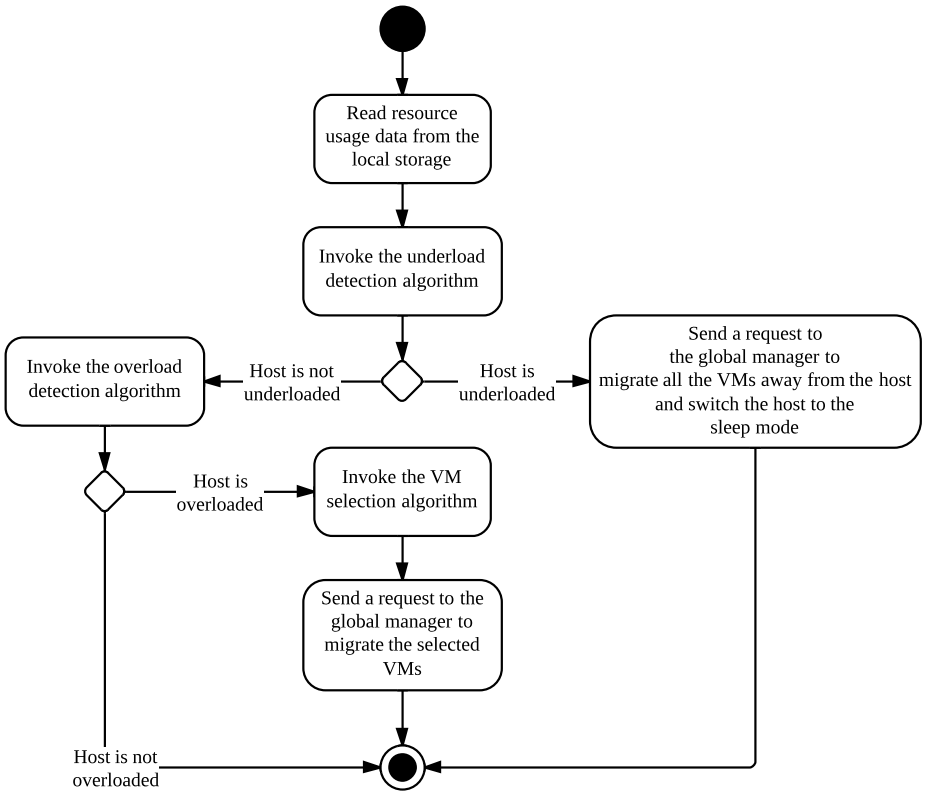


Figure 6.5: The local manager: an activity diagram

the host to a low-power mode.

If the host is not underloaded, the local manager proceeds to invoking the specified in the configuration overload detection algorithm. If the host is overloaded, the local manager invokes the configured VM selection algorithm to select VMs to offload from the host. Once the VMs to migrate from the host are selected, the local manager sends an overload request to the global manager's REST API to migrate the selected VMs. Similarly to the global manager, the local manager can be configured to use custom underload detection, overload detection, and VM selection algorithms using the configuration file discussed in Section 6.3.8.

6.3.6 The Data Collector

The data collector is deployed on every compute host as an OS service running in the background. The service periodically collects the CPU utilization data for each VM running on the host, as well as data on the CPU utilization by the hypervisor. The col-

lected data are stored in the local file-based data store, and also submitted to the central database. The data are stored as the average number of MHz consumed by a VM during the last measurement interval of length τ. In particular, the CPU usage $C_i^v(t_0, t_1)$ of a VM i, which is a function of the bounds of a measurement interval $[t_0, t_1]$, is calculated as shown in (6.1).

$$C_i^v(t_0, t_1) = \frac{n_i^v F(\tau_i^v(t_1) - \tau_i^v(t_0))}{t_1 - t_0}, \tag{6.1}$$

where n_i^v is the number of virtual CPU cores allocated to the VM i; F is the frequency of a single CPU core in MHz; and $\tau_i^v(t)$ is the CPU time consumed by the VM i up to the time t. The CPU usage of the hypervisor $C_j^h(t_0, t_1)$ is calculated as a difference between the overall CPU usage and the CPU usage by the set of VMs allocated to the host, as shown in (6.2).

$$C_j^h(t_0, t_1) = \frac{n_j^h F(\tau_j^h(t_1) - \tau_j^h(t_0))}{t_1 - t_0} - \sum_{i \in \mathcal{V}_j} C_i^v(t_0, t_1), \tag{6.2}$$

where n_j^h is the number of physical cores of the host j; $\tau_j^h(t)$ is the CPU time consumed by the host overall up to the time t; and $\mathcal{V}_j$ is the set of VM allocated to the host j. The CPU usage data are stored as integers. This data format is portable: the stored values can be approximately converted to the CPU utilization percentages for any host or VM type, supporting heterogeneous hosts and VMs.

The actual data are obtained using libvirt's API[6] in the form of the CPU time consumed by VMs and hosts overall to date. Using the CPU time collected at the previous time step, the CPU time for the last time interval is calculated. According to the CPU frequency of the host and the length of the time interval, the CPU time is converted into the required average MHz consumed by the VM over the last time interval. Then, using the VMs' CPU utilization data, the CPU utilization by the hypervisor is calculated. The collected data are stored both locally and submitted to the central database. The number of the latest data values to be stored locally and passed to the underload / overload detection and VM selection algorithms is defined by the *data_collector_data_length* option in the configuration file.

[6]The libvirt virtualization API. http://libvirt.org/

At the beginning of every iteration, the data collector obtains the set of VMs currently running on the host using the Nova API and compares them to the VMs running on the host at the previous time step. If new VMs have been found, the data collector fetches the historical data about them from the central database and stores the data in the local file-based data store. If some VMs have been removed, the data collector removes the data about these VMs from the local data store.

While OpenStack Neat oversubscribes the CPU of hosts by taking advantage of information on the real-time CPU utilization, it does not overcommit RAM. In other words, RAM is still a constraint in placing VMs on hosts; however, the constraint is the maximum amount of RAM that can be used by a VM statically defined by its instance type, rather than the real-time RAM consumption. One of the reasons for that is that RAM is a more critical resource compared with the CPU, as an application may fail due to insufficient RAM, whereas insufficient CPU may just slow down the execution of the application. Another reason is that in contrast to the CPU, RAM usually does not become a bottleneck resource, as shown by an analysis of workload traces and information from the industry [4, 107].

6.3.7 Data Stores

As shown in Figure 6.2, the system contains two types of data stores:

- *Central database* – a database server, which can be deployed either on the controller host, or on one or more dedicated hosts.
- *Local file-based data storage* – a data store deployed on every compute host and used for temporary caching the resource usage data to use by the local managers in order to avoid excessive database queries.

The details about the data stores are given in the following subsections.

Central Database

The central database is used for storing historical data on the resource usage by VMs and hypervisors, as well as hardware characteristics of hosts. The database is populated by the data collectors deployed on compute hosts. There are two main use cases when the

data are retrieved from the central database instead of the local storage of compute hosts. First, it is used by local managers to fetch the resource usage data after VM migrations. Once a VM migration is completed, the data collector deployed on the destination host fetches the required historical data from the database and stores them locally to use by the local manager.

The second use case of the central database is when the global manager computes a new placement of VMs on hosts. VM placement algorithms require information on the resource consumption of all the hosts in order to make global allocation decisions. Therefore, every time there is a need to place VMs on hosts, the global manager queries the database to obtain the up-to-date data on the resource usage by hypervisors and VMs.

Field	Type
id	Integer
hostname	String(255)
cpu_mhz	Integer
cpu_cores	Integer
ram	Integer

(a) The *hosts* table

Field	Type
id	Integer
host_id	Integer
timestamp	DateTime
cpu_mhz	Integer

(b) The *host_resource_usage* table

Field	Type
id	Integer
uuid	String(36)

(c) The *vms* table

Field	Type
id	Integer
vm_id	Integer
timestamp	DateTime
cpu_mhz	Integer

(d) The *vm_resource_usage* table

Table 6.1: The database schema

As shown in Table 6.1, the database schema contains 4 main tables: *hosts*, *host_resource_usage*, *vms*, and *vm_resource_usage*. The *hosts* table stores information about hosts, such as the host names, CPU frequency of a physical core in MHz, number of CPU cores, and amount of RAM in MB. The *vms* table stores the UUIDs of VMs assigned by OpenStack. The *host_resource_usage* and *vm_resource_usage* tables store data on the resource consumption over time by hosts and VMs respectively.

Local File-Based Data Store

A local manager at each iteration requires data on the resource usage by the VMs and hypervisor of the corresponding host in order to pass them to the underload / overload

detection and VM placement algorithms. To reduce the number of queries to the database over the network, apart from submitting the data into the database, the data collector temporarily stores the data locally. This way, the local manager can just read the data from the local file storage and avoid having to retrieve data from the central database.

The data collector stores the resource usage data locally in *local_data_directory/vms/* as plain text files, where *local_data_directory* is defined in the configuration file discussed in Section 6.3.8. The data for each VM are stored in a separate file named after the UUID of the VM. The data on the resource usage by the hypervisor are stored in the *local_data_directory/host* file. The format of the files is a new line separated list of integers representing the average CPU consumption in MHz during measurement intervals.

6.3.8 Configuration

The configuration of OpenStack Neat is stored in the */etc/neat/neat.conf* file in the standard INI format using the '#' character for denoting comments. It is assumed that this file exists on all the compute and controller hosts and contains the same configuration. The available configuration options, default values, and descriptions of the options are given in Table 6.2.

One of the ideas implemented in OpenStack Neat is providing the user with the ability to change the implementation and parameters of any of the 4 VM consolidation algorithms simply by modifying the configuration file. This provides the means of adding to the system and enabling custom VM consolidation algorithms without modifying the source code of the framework. The algorithms are configured using the options with the *algorithm_* prefix. More information on adding and enabling VM consolidation algorithms is given in Section 6.3.9.

Table 6.2: OpenStack Neat's configuration options

Configuration option = default value	**Description**
log_directory=/var/log/neat	The directory to store log files.
log_level=3	The level of emitted log messages (0-3).
sql_connection= mysql://neat:neatpassword@controller/neat	The host name and credentials for connecting to the database server specified in the format supported by SQLAlchemy.

Table 6.2: OpenStack Neat's configuration options (continued)

Configuration option = default value	Description
os_admin_tenant_name=admin	The admin tenant name authenticating in Nova.
os_admin_user=admin	The admin user name for authenticating in Nova.
os_admin_password=adminpassword	The admin password for authenticating in Nova.
os_auth_url=http://controller:5000/v2.0/	The OpenStack authentication URL.
vm_instance_directory=/var/lib/nova/instances	The directory, where OpenStack Nova stores the data of VM instances.
compute_hosts=compute1, compute2, ...	A coma-separated list of compute host names.
global_manager_host=controller	The global manager's host name.
global_manager_port=60080	The port of the REST web service exposed by the global manager.
db_cleaner_interval=7200	The time interval between subsequent invocations of the database cleaner in seconds.
local_data_directory=/var/lib/neat	The directory used by the data collector to store data on the resource usage by the VMs and hypervisor.
local_manager_interval=300	The time interval between subsequent invocations of the local manager in seconds.
data_collector_interval=300	The time interval between subsequent invocations of the data collector in seconds.
data_collector_data_length=100	The number of the latest data values stored locally by the data collector and passed to the underload / overload detection, and VM placement algorithms.
host_cpu_overload_threshold=0.8	The threshold on the overall (all cores) utilization of the physical CPU of a host, above which the host is considered to be overloaded. This is used for logging host overloads.
host_cpu_usable_by_vms=1.0	The threshold on the overall (all cores) utilization of the physical CPU of a host available for allocation to VMs.
compute_user=neat	The user name for connecting to the compute hosts for switching them into the sleep mode.
compute_password=neatpassword	The password of the user account used for connecting to the compute hosts for switching them into the sleep mode.
sleep_command=pm-suspend	A shell command used to switch a host into the sleep mode, the *compute_user* must have permissions to execute this command.
ether_wake_interface=eth0	The network interface to send a magic packet from the controller host using the ether-wake program.
network_migration_bandwidth=10	The network bandwidth in MB/s available for VM live migrations.
algorithm_underload_detection_factory= neat.locals.underload.trivial. last_n_average_threshold_factory	The fully qualified name of a Python factory function that returns a function implementing an underload detection algorithm.
algorithm_underload_detection_parameters= {"threshold": 0.5, "n": 2}	JSON encoded parameters to be parsed and passed to the underload detection algorithm factory.
algorithm_overload_detection_factory= neat.locals.overload.mhod.core.mhod_factory	The fully qualified name of a Python factory function that returns a function implementing an overload detection algorithm.

Table 6.2: OpenStack Neat's configuration options (continued)

Configuration option = default value	Description
algorithm_overload_detection_parameters= {"state_config": [0.8], "otf": 0.2, "history_size": 500, "window_sizes": [30, 40, 50, 60, 70, 80, 90, 100], "bruteforce_step": 0.2, "learning_steps": 10}	JSON encoded parameters to be parsed and passed to the specified overload detection algorithm factory.
algorithm_vm_selection_factory= neat.locals.vm_selection.algorithms. minimum_migration_time_max_cpu_factory	The fully qualified name of a Python factory function that returns a function implementing a VM selection algorithm.
algorithm_vm_selection_parameters= {"n": 2}	JSON encoded parameters to be parsed and passed to the specified VM selection algorithm factory.
algorithm_vm_placement_factory= neat.globals.vm_placement. bin_packing.best_fit_decreasing_factory	The fully qualified name of a Python factory function that returns a function implementing a VM placement algorithm.
algorithm_vm_placement_parameters= {"cpu_threshold": 0.8, "ram_threshold": 0.95, "last_n_vm_cpu": 2}	JSON encoded parameters to be parsed and passed to the specified VM placement algorithm factory.

6.3.9 Extensibility of the Framework

One of the main points of the framework's extensibility is the ability to add new VM consolidation algorithm to the system and enable them by updating the configuration file without the necessity in modifying the source code of the framework itself. There are 4 algorithms that can be changed through a modification of the configuration file: underload / overload detection, VM selection, and VM placement algorithms. The values of the corresponding configuration options should be fully qualified names of functions available as a part of one of the installed Python libraries. The fact that the functions are specified by their fully qualified names also means that they can be installed as a part of a Python library independent from OpenStack Neat. The 4 corresponding configuration options are the following:

1. *algorithm_underload_detection_factory*
2. *algorithm_overload_detection_factory*
3. *algorithm_vm_selection_factory*
4. *algorithm_vm_placement_factory*

Since an algorithm may need to be initialized prior to its usage, the factory function pattern is applied. The functions specified as values of any of the *algorithm_*_factory* configuration options are not functions that actually implement VM consolidation algo-

Table 6.3: Interfaces of VM consolidation algorithms and their factory functions

Algorithm	Factory arguments	Algorithm arguments	Algorithm return
Underload detection	1. time_step: int, ≥ 0 2. migration_time: float, ≥ 0 3. params: dict(str: *)	1. cpu_utilization: list(float) 2. state: dict(str: *)	1. decision: bool 2. state: dict(str: *)
Overload detection	1. time_step: int, ≥ 0 2. migration_time: float, ≥ 0 3. params: dict(str: *)	1. cpu_utilization: list(float) 2. state: dict(str: *)	1. decision: bool 2. state: dict(str: *)
VM selection	1. time_step: int, ≥ 0 2. migration_time: float, ≥ 0 3. params: dict(str: *)	1. vms_cpu: dict(str : list(int)) 2. vms_ram: dict(str : list(int)) 3. state: dict(str: *)	1. vms: list(str) 2. state: dict(str: *)
VM placement	1. time_step: int, ≥ 0 2. migration_time: float, ≥ 0 3. params: dict(str: *)	1. hosts_cpu_usage: dict(str : int) 2. hosts_cpu_total: dict(str : int) 3. hosts_ram_usage: dict(str : int) 4. hosts_ram_total: dict(str : int) 5. inactive_hosts_cpu: dict(str : int) 6. inactive_hosts_ram: dict(str : int) 7. vms_cpu: dict(str : list(int)) 8. vms_ram: dict(str : list(int)) 9. state: dict(str: *)	1. alloc.: dict(str: str) 2. state: dict(str: *)

rithms, rather they are functions that return initialized instances of functions implementing the corresponding VM consolidation algorithms. All functions implementing VM consolidation algorithms and their factories should adhere to the corresponding predefined interfaces. For example, all factory functions of overload detection algorithms must accept a time step, migration time, and algorithm parameters as arguments. The function must return another function that implements the required consolidation algorithm, which in turn must follow the interface predefined for overload detection algorithms.

Every function implementing an overload detection algorithm must: (1) accept as arguments a list of CPU utilization percentages and dictionary representing the state of the algorithm; and (2) return a tuple containing the decision of the algorithm as a boolean and updated state dictionary. If the algorithm is stateless, it should return an empty dictionary as the state. Definitions of the interfaces of functions implementing VM consolidation algorithms and their factories are given in Table 6.3. The types and descriptions of the arguments are given in Table 6.4.

Using the *algorithm_*_parameters* configuration options, it is possible to pass arbitrary dictionaries of parameters to VM consolidation algorithm factory functions. The parameters must be specified as an object in the JSON format on a single line. The specified JSON strings are automatically parsed by the system and passed to factory functions as Python

dictionaries. Apart from being parameterized, a consolidation algorithm may also preserve state across invocations. This can be useful for implementing stateful algorithms, or as a performance optimization measure, e.g., to avoid repeating costly computations. Preserving state is done by accepting a state dictionary as an argument, and returning the updated dictionary as the second element of the return tuple.

Table 6.4: Arguments of VM consolidation algorithms and their factory functions

Argument name	Type	Description
time_step	int, ≥ 0	The length of the time step in seconds.
migration_time	float, ≥ 0	The VM migration time in time seconds.
params	dict(str: *)	A dictionary containing the algorithm's parameters parsed from the JSON representation specified in the configuration file.
cpu_utilization	list(float)	A list of the latest CPU utilization percentages in the $[0, 1]$ range calculated from the combined CPU usage by all the VMs allocated to the host and the hypervisor.
state	dict(str: *)	A dictionary containing the state of the algorithm passed over from the previous iteration.
vms_cpu	dict(str : list(int))	A dictionary of VM UUIDs mapped on the lists of the latest CPU usage values by the VMs in MHz.
vms_ram	dict(str : int)	A dictionary of VM UUIDs mapped on the maximum allowed amounts of RAM for the VMs in MB.
host_cpu_usage	dict(str : int)	A dictionary of host names mapped on the current combined CPU usage in MHz.
host_cpu_total	dict(str : int)	A dictionary of host names mapped on the total CPU capacities in MHz calculated as a multiplication of the frequency of a single physical core by the number of cores.
host_ram_usage	dict(str : int)	A dictionary of host names mapped on the current amounts of RAM in MB allocated to the VMs of the hosts.
host_ram_total	dict(str : int)	A dictionary of host names mapped on the total amounts of RAM in MB available to VMs on the hosts.
inactive_hosts_cpu	dict(str : int)	A dictionary of the names of the currently inactive hosts mapped on the total CPU capacities in MHz.
inactive_hosts_ram	dict(str : int)	A dictionary of the names of the currently inactive hosts mapped on the total amounts of RAM in MB available to VMs on the hosts.

Currently, the data collector only collects data on the CPU utilization. It is possible to extend the system to collect other types of data that may be passed to the VM consolidation algorithms. To add another type of data, it is necessary to extend the *host_resource_usage* and *vm_resource_usage* database tables by adding new fields for storing the new types of data. Then, the *execute* function of the data collector should be extended to include the code required to obtain the new data and submit them to the database.

Finally, the local and global managers need to be extended to fetch the new type of data from the database to be passed to the appropriate VM consolidation algorithms.

6.3.10 Deployment

OpenStack Neat needs to be deployed on all the compute and controller hosts. The deployment consists in installing dependencies, cloning the project's Git repository, installing the project, and starting up the services. The process is cumbersome since multiple steps should be performed on each host. The OpenStack Neat distribution includes a number of Shell scripts that simplify the deployment process. The following steps are required to perform a complete deployment of OpenStack Neat:

1. Clone the project's repository on the controller host by executing:
   `git clone git://github.com/beloglazov/openstack-neat.git`
2. Install the required dependencies by executing the following command from the cloned repository if the OS of the controller is CentOS: `./setup/deps-centos.sh`
3. In the cloned repository, modify *neat.conf* to meet the requirements. In particular, it is necessary to enter the names of the available compute hosts. It is also necessary to create a database on the database server accessible with the details specified in the configuration file.
4. Install OpenStack Neat on the controller host by executing the following command from the project's directory: `sudo python setup.py install`. This command will also copy the modified configuration file to */etc/neat/neat.conf*.
5. Using the scripts provided in the package, it is possible to install OpenStack Neat on all the compute hosts specified in the configuration file remotely from the controller. First, the following command can be used to clone the repository on all the compute hosts: `./compute-clone-neat.py`.
6. Once the repository is cloned, OpenStack Neat and its dependencies can be installed on all the compute hosts by executing the two following commands on the controller: `./compute-install-deps.py; ./compute-install-neat.py`
7. Next, it is necessary to copy the modified configuration file to the compute hosts, which can be done by the following command: `./compute-copy-conf.py`
8. All OpenStack Neat services can be started on the controller and compute hosts

with the following single command `./all-start.sh`

Once all the steps listed above are completed, OpenStack Neat's services should be deployed and started up. If any service fails, the log files can be found in */var/log/neat/* on the corresponding host.

6.4 VM Consolidation Algorithms

As mentioned earlier, OpenStack Neat is based on the approach to the problem of dynamic VM consolidation, proposed in the previous chapters, which consists in dividing the problem into 4 sub-problems: (1) host underload detection; (2) host overload detection; (3) VM selection; and (4) VM placement. This section discusses some of the implemented algorithms. It is important to note that the presented algorithms are not the main focus of the current chapter. The focus of the chapter is the design of the framework for dynamic VM consolidation, which is capable of handling multiple implementations of consolidation algorithms, and can be switched between the implementations through configuration as discussed in Section 6.3.8.

6.4.1 Host Underload Detection

In the experiments of this chapter, a simple heuristic is used for the problem of underload detection shown in Algorithm 7. The algorithm calculates the mean of the n latest CPU utilization measurements and compares it to the specified threshold. If the mean CPU utilization is lower than the threshold, the algorithm detects a host underload situation. The algorithm accepts 3 arguments: the CPU utilization threshold, the number of last CPU utilization values to average, and a list of CPU utilization measurements.

Algorithm 7 The averaging threshold-based underload detection algorithm

Input: *threshold*, *n*, *utilization*
Output: Whether the host is underloaded
1: **if** *utilization* is not empty **then**
2: $utilization \leftarrow$ last n values of $utilization$
3: $meanUtilization \leftarrow$ sum($utilization$) / len($utilization$)
4: **return** $meanUtilization \leq threshold$
5: **return false**

6.4.2 Host Overload Detection

OpenStack Neat includes several overload detection algorithms, which can be enabled by modifying the configuration file. One of the simple included algorithms is the averaging Threshold-based (THR) overload detection algorithm. The algorithm is similar to Algorithm 7, while the only difference is that it detects overload situations if the mean of the n last CPU utilization measurements is *higher* than the specified threshold.

Another overload detection algorithm included in the default implementation of OpenStack Neat is based on estimating the future CPU utilization using *local regression* (i.e., the Loess method), referred to as the Local Regression Robust (LRR) algorithm shown in Algorithm 8, which has been introduced in Chapter 4. The algorithm calculates the Loess parameter estimates, and uses them to predict the future CPU utilization at the next time step taking into account the VM migration time. In addition, the LR algorithm accepts a safety parameter, which is used to scale the predicted CPU utilization to increase or decrease the sensitivity of the algorithm to potential overloads.

Algorithm 8 The Local Regression Robust (LRR) overload detection algorithm

Input: $threshold$, $param$, n, $migrationTime$, $utilization$
Output: Whether the host is overloaded
1: **if** len($utilization$) $< n$ **then**
2: **return false**
3: $estimates \leftarrow$ loessRobustParameterEstimates(last n values of $utilization$)
4: $prediction \leftarrow estimates[0] + estimates[1] \times (n + migrationTime)$
5: **return** $param \times prediction \geq threshold$

A more complex overload detection algorithm included in OpenStack Neat is the Markov Overload Detection (MHOD) algorithm introduced in Chapter 5. This algorithm allows the system administrator to specify a constraint on the OTF metric. Let a host can be in one of two states in regard to its CPU utilization: (1) serving regular load; and (2) being overloaded. It is assumed that if a host is overloaded, the VMs allocated to that host are not being provided with the required performance level, and therefore, experience performance degradation. The OTF metric allows quantifying the performance degradation over a period of time according to the definition of the overload state. The OTF metric is defined as shown in (6.3).

$$OTF(u_t) = \frac{t_o(u_t)}{t_a}, \tag{6.3}$$

where u_t is the CPU utilization threshold distinguishing the non-overload and overload states of a compute host; t_o is the time, during which the host has been overloaded, which is a function of u_t; and t_a is the total time, during which the host has been active. It has been claimed in the literature that the performance of servers degrade when their utilization approaches 100% [108, 133]. This problem is addressed in the OTF metric by adjusting the value of u_t, which in the experiments was set to 80%. Using this definition, the QoS requirements can be defined as the maximum allowed value of OTF. For instance, assume that OTF must be less or equal to 10%, and a host becomes overloaded when its CPU utilization is higher than 80%. This would mean that on average every host is allowed to have the CPU utilization higher than 80% for no longer than 10% of its total activity time. The date center-level OTF can be calculated by replacing the time values for a single host by the aggregated time values over the full set of hosts, as discussed in Section 6.6.2.

The MHOD algorithm enables the system administrator to explicitly specify a constraint on the OTF value as a parameter of the algorithm, while maximizing the time between VM migrations, thus, improving the quality of VM consolidation. The algorithm builds a Markov chain model based on the observed history of the CPU utilization applying the Multisize Sliding Window workload estimation method [80]. The model is used to generate the objective function and constraint of the optimization problem to find the VM migration probability. The optimization problem is attempted to be solved using the brute-force search method with a large step to find any feasible solution. If no feasible solution exists or the VM migration probability is less than 1, the algorithm detects a host overload. The algorithm is described in detail in Chapter 5.

6.4.3 VM Selection

Once a host overload has been detected, it is necessary to determine what VMs are the best to be migrated from the host. This problem is solved by VM selection algorithms. An example of such an algorithm is simply randomly selecting a VM from the set of

VMs allocated to the host. Another algorithm shown in Algorithm 9 is called Minimum Migration Time Maximum CPU utilization (MMTMC). This algorithm first selects VMs with the minimum amount of RAM to minimize the live migration time. Then, out of the selected subset of VMs, the algorithm selects the VM with the maximum CPU utilization averaged over the last n measurements to maximally reduce the overall CPU utilization of the host.

Algorithm 9 The MMTMC algorithm

Input: n, $vmsCpuMap$, $vmsRamMap$
Output: A VM to migrate
1: $minRam \leftarrow$ min(values of $vmsRamMap$)
2: $maxCpu \leftarrow 0$
3: $selectedVm \leftarrow$ **None**
4: **for** vm, cpu in $vmsCpuMap$ **do**
5: **if** $vmsRamMap[vm] > minRam$ **then**
6: **continue**
7: $vals \leftarrow$ last n values of cpu
8: $mean \leftarrow$ sum($vals$) / len($vals$)
9: **if** $maxCpu < mean$ **then**
10: $maxCpu \leftarrow mean$
11: $selectedVm \leftarrow vm$
12: **return** $selectedVm$

6.4.4 VM Placement

The VM placement problem can be seen as a bin packing problem with variable bin sizes, where bins represent hosts; bin sizes are the available CPU capacities of hosts; and items are VMs to be allocated with an extra constraint on the amount of RAM. As the bin packing problem is NP-hard, it is appropriate to apply a heuristic to solve it. OpenStack Neat implements a modification of the Best Fit Decreasing (BFD) algorithm, which has been shown to use no more than $11/9 \cdot OPT + 1$ bins, where OPT is the number of bins of the optimal solution [130].

The implemented modification of the BFD algorithm shown in Algorithm 10 includes several extensions: the ability to handle extra constraints, namely, consideration of currently inactive hosts, and a constraint on the amount of RAM required by the VMs. An inactive host is only activated when a VM cannot be placed on one of the already active hosts. The constraint on the amount of RAM is taken into account in the first fit manner,

Algorithm 10 The Best Fit Decreasing (BFD) VM placement algorithm

Input: n, $hostsCpu$, $hostsRam$, $inactiveHostsCpu$, $inactiveHostsRam$, $vmsCpu$, $vmsRam$
Output: A map of VM UUIDs to host names

$vmTuples \leftarrow$ empty list
for vm, cpu in $vmsCpu$ **do**
  $vals \leftarrow$ last n values of cpu
  append a tuple of the mean of $vals$, $vmsRam[vm]$, and vm to $vmTuples$
$vms \leftarrow$ sortDecreasing($vmTuples$)
$hostTuples \leftarrow$ empty list
for $host$, cpu in $hostsCpu$ **do**
  append a tuple of cpu, $hostsRam[host]$, $host$ to $hostTuples$
$hosts \leftarrow$ sortIncreasing($hostTuples$)
$inactiveHostTuples \leftarrow$ empty list
for $host$, cpu in $inactiveHostsCpu$ **do**
  append a tuple of cpu, $inactiveHostsRam[host]$, $host$ to $inactiveHostTuples$
$inactiveHosts \leftarrow$ sortIncreasing($inactiveHostTuples$)
$mapping \leftarrow$ empty map
for $vmCpu$, $vmRam$, $vmUuid$ in vms **do**
  $mapped \leftarrow$ **false**
  while not $mapped$ **do**
    $allocated \leftarrow$ **false**
    for _, _, $host$ in $hosts$ **do**
      if $hostsCpu[host] \geq vmCpu$ and $hostsRam[host] \geq vmRam$ **then**
        $mapping[vmUuid] \leftarrow host$
        $hostsCpu[host] \leftarrow hostsCpu[host]$ - $vmCpu$
        $hostsRam[host] \leftarrow hostsRam[host]$ - $vmRam$
        $mapped \leftarrow$ **true**
        $allocated \leftarrow$ **true**
        break
    if not $allocated$ **then**
      if $inactiveHosts$ is not empty **then**
        $activatedHost \leftarrow$ pop the first from $inactiveHosts$
        append $activatedHost$ to $hosts$
        $hosts \leftarrow$ sortIncreasing($hosts$)
        $hostsCpu[activatedHost[2]] \leftarrow activatedHost[0]$
        $hostsRam[activatedHost[2]] \leftarrow activatedHost[1]$
      else
        break
if len(vms) == len($mapping$) **then**
  return $mapping$
return empty map

i.e., if a host is selected for a VM as a best fit according to its CPU requirements, the host is confirmed if it just satisfies the RAM requirements. In addition, similarly to the averaging underload and overload detection algorithms, the algorithm uses the mean values of the last n CPU utilization measurements as the CPU constraints. The worst-case complexity of the algorithm is $(n + m/2)m$, where n is the number of physical nodes, and m is the number of VMs to be placed. The worst case occurs when every VM to be placed requires a new inactive host to be activated.

6.5 Implementation

OpenStack Neat is implemented in Python. The choice of the programming language has been mostly determined by the fact that OpenStack itself is implemented in Python; therefore, using the same programming language could potentially simplify the integration of the two projects. Since Python is a dynamic language, it has a number of advantages, such as concise code, no type constraints, and *monkey patching*, which refers to the ability to replace methods, attributes, and functions at run-time. Due to its flexibility and expressiveness, Python typically helps to improve productivity and reduce the development time compared with statically typed languages, such as Java and C++. The downsides of dynamic typing are the lower run-time performance and lack of compile time guarantees provided by statically typed languages.

To compensate for the reduced safety due to the lack of compile time checks, several programming techniques are applied in the implementation of OpenStack Neat to minimize bugs and simplify maintenance. First of all, the functional programming style is followed by leveraging the functional features of Python, such as higher-order functions and closures, and minimizing the use of the object-oriented programming features, such as class hierarchies and encapsulation. One important technique that is applied in the implementation of OpenStack Neat is the minimization of mutable state. Mutable state is one of the causes of side effects, which prevent functions from being referentially transparent. This means that if a function relies on some global mutable state, multiple calls to that function with the same arguments do not guarantee the same result returned by the function for each call.

The implementation of OpenStack Neat tries to minimize side effects by avoiding mutable state where possible, and isolating calls to external APIs in separate functions covered by unit tests. In addition, the implementation splits the code into small easy to understand functions with explicit arguments that the function acts upon without mutating their values. To impose constraints on function arguments, the Design by Contract (DbC) approach is applied using the PyContracts library. The approach prescribes the definition of formal, precise, and verifiable interface specifications for software components. PyContracts lets the programmer to specify contracts on function arguments via a special format of Python docstrings. The contracts are checked at run-time, and if any of the constraints is not satisfied, an exception is raised. This approach helps to localize errors and fail fast, instead of hiding potential errors. Another advantage of DbC is comprehensive and up-to-date code documentation, which can be generated from the source code by automated tools.

To provide stronger guarantees of the correctness of the program, it is important to apply unit testing. According to this method, each individual unit of source code, which in this context is a function, should be tested by an automated procedure. The goal of unit testing is to isolate parts of the program and show that they perform correctly. One of the most efficient unit testing techniques is implemented by the Haskell QuickCheck library. This library allows the definition of tests in the form of properties that must be satisfied, which do not require the manual specification of the test case input data. QuickCheck takes advantage of Haskell's rich type system to infer the required input data and generates multiple test cases automatically.

The implementation of OpenStack neat uses Pyqcy, a QuickCheck-like unit testing framework for Python. This library allows the specification of *generators*, which can be seen as templates for input data. Similarly to QuickCheck, Pyqcy uses the defined templates to automatically generate input data for hundreds of test cases for each unit test. Another Python library used for testing of OpenStack Neat is Mocktest. This library leverages the flexibility of Python's monkey patching to dynamically replace, or *mock*, existing methods, attributes, and functions at run-time. Mocking is essential for unit testing the code that relies on calls to external APIs. In addition to the ability to set artificial return values of methods and functions, Mocktest allows setting expectations on

Table 6.5: The OpenStack Neat codebase summary

Package	Files	Lines of code	Lines of comments
Core	21	2,144	1,946
Tests	20	3,419	260

the number of the required function calls. If the expectations are not met, the test fails. Currently, OpenStack Neat includes more than 150 unit tests.

OpenStack Neat applies Continuous Integration (CI) using the Travis CI service[7]. The aim of the CI practice is to detect integration problems early by periodically building and deploying the software system. Travis CI is attached to OpenStack Neat's source code repository through Git hooks. Every time modifications are pushed to the repository, Travis CI fetches the source code and runs a clean installation in a sandbox followed by the unit tests. If any step of the integration process fails, Travis CI reports the problem.

Despite all the precautions, run-time errors may occur in a deployed system. OpenStack Neat implements multi-level logging functionality to simplify the post-mortem analysis and debugging process. The verbosity of logging can be adjusted by modifying the configuration file. Table 6.5 provides information on the size of the current codebase of OpenStack Neat. Table 6.6 summarizes the set of open source libraries used in the implementation of OpenStack Neat.

6.6 A Benchmark Suite for Evaluating Distributed Dynamic VM Consolidation Algorithms

Currently, research in the area of dynamic VM consolidation is limited by the lack of a standardized suite of benchmark software, workload traces, performance metrics, and evaluation methodology. Most of the time, researchers develop their own solutions for evaluating the proposed algorithms, which are not publicly available later on. This complicates further research efforts in the area due to the limited opportunities for comparing new results with prior solutions. Moreover, the necessity of implementing custom evaluation software leads to duplication of efforts. This chapter outlines an initial version of a benchmark suite for evaluating dynamic VM consolidation algorithms following the

[7]OpenStack Neat on Travis CI. http://travis-ci.org/beloglazov/openstack-neat

Table 6.6: Open source libraries used by OpenStack Neat

Library	License	Description
Distribute	Python 2.0	A library for managing Python projects and distributions. http://bitbucket.org/tarek/distribute
Pyqcy	FreeBSD	A QuickCheck-like unit testing framework for Python. http://github.com/Xion/pyqcy
Mocktest	LGPL	A Python library for mocking objects and functions. http://github.com/gfxmonk/mocktest
PyContracts	LGPL	A Python library for Design by Contract (DbC). http://github.com/AndreaCensi/contracts
SQLAlchemy	MIT	A Python SQL toolkit, also used by the core OpenStack services. http://www.sqlalchemy.org/
Bottle	MIT	A micro web-framework for Python. http://bottlepy.org/
Requests	ISC	A Python HTTP client library. http://python-requests.org/
libvirt	LGPL	A virtualization toolkit with Python bindings. http://libvirt.org/
Python-novaclient	Apache 2.0	A Python Nova API client implementation. http://github.com/openstack/python-novaclient
NumPy	BSD	A library for scientific computing. http://numpy.scipy.org/
SciPy	BSD	A library of extra tools for scientific computing. http://scipy.org/

distributed approach of splitting the problem into 4 sub-problems discussed earlier. The proposed benchmark suite consists of 4 major components:

1. OpenStack Neat, a framework for distributed dynamic VM consolidation in OpenStack Clouds providing a base system implementation and allowing configuration-based switching of different implementations of VM consolidation algorithms.
2. A set of workload traces containing data on the CPU utilization collected every 5 minutes from more than a thousand PlanetLab VMs deployed on servers located in more than 500 places around the world [92].
3. A set of performance metrics capturing the following aspects: quality of VM consolidation; quality of service delivered by the system; overhead of VM consolidation in terms of the number of VM migration; and execution time of the consolidation algorithms.
4. Evaluation methodology prescribing the approach of preparing experiments, deploying the system, generating workload using the PlanetLab traces, as well as processing and analyzing the results.

The availability of such a benchmark suite will foster and facilitate research efforts and future advancements in the area of dynamic VM consolidation. In addition, researchers are encouraged to publicize and share the implemented consolidation algorithms to simplify performance comparisons with future solutions. One approach to sharing algorithm implementations is the addition of an extra package to the main branch of OpenStack Neat that will contain contributed algorithms. This would provide a central location, where anyone can find the up-to-date set of consolidation algorithms to use in their research. Therefore, processing and managing the inclusion of such submissions into the main public repository of OpenStack Neat will be done as a part of the project. The following sections provide more information on the workload traces, performance metrics, and evaluation methodology of the proposed benchmark suite. The performance evaluation discussed in Section 6.7 is an example of application of the benchmark suite.

6.6.1 Workload Traces

To make experiments reproducible, it is important to rely on a set of input traces to reliably generate the workload, which would allow the experiments to be repeated as many times as necessary. It is also important to use workload traces collected from a real system rather than artificially generated, as this would help to reproduce a realistic scenario. This chapter uses workload trace data provided as a part of the CoMon project, a monitoring infrastructure of PlanetLab [92]. The traces include data on the CPU utilization collected every 5 minutes from more than a thousand VMs deployed on servers located in more 500 places around the world. 10 days of workload traces collected during March and April 2011 have been randomly chosen, which resulted in the total of 11,746 24-hour long traces. The full set of workload traces is publicly available online[8].

The workload from PlanetLab VMs is representative of an IaaS Cloud environment, such as Amazon EC2, in the sense that the VMs are created and managed by multiple independent users, and the infrastructure provider is not aware of what particular applications are executing in the VMs. Furthermore, this implies that the overall system workload is composed of multiple independent heterogeneous applications, which also corresponds to an IaaS environment. However, there is difference from a public Cloud

[8]The PlanetLab traces. http://github.com/beloglazov/planetlab-workload-traces

provider, such as Amazon EC2. The difference is that PlanetLab is an infrastructure mainly used for research purposes; therefore, the applications are potentially closer to the HPC type, rather than web services, which are common in public Clouds.

HPC applications are typically CPU-intensive with lower dynamics in the resource utilization compared with web services, whose resource consumption depends on the number of user requests and may vary over time. HPC workload is easier to handle for a VM consolidation system due to infrequent variation in the resource utilization. Therefore, to stress the system in the experiments, the original workload traces have been filtered to leave only the ones that exhibit high variability. In particular, only the traces that satisfy the following two conditions have been selected: (1) at least 10% of time the CPU utilization is lower than 20%; and (2) at least 10% of time the CPU utilization is higher than 80%. This significantly reduced the number of workload traces resulting in only 33 out of 11,746 24-hour traces left. The set of selected traces and filtering script are available online [14].

The resulting number of traces was sufficient for the experiments, whose scale was limited by the size of the testbed described in Section 6.7.1. If a larger number of traces is required to satisfy larger scale experiments, one approach is to relax the conditions of filtering the original set of traces. Another approach is to randomly sample with replacement from the limited set of traces. If another set of suitable workload traces becomes publicly available, it can be included in the benchmark suite as an alternative.

6.6.2 Performance Metrics

For effective performance evaluation and comparison of algorithms it is essential to define performance metrics that capture the relevant characteristics of the algorithms. One of the objectives of dynamic VM consolidation is the minimization of energy consumption by the physical nodes, which can be a metric for performance evaluation and comparison. However, energy consumption is highly dependent on the particular model and configuration of the underlying hardware, efficiency of power supplies, implementation of the sleep mode, etc. A metric that abstracts from the mentioned factors, but is directly proportional and can be used to estimate energy consumption, is the time of a host being idle, aggregated over the full set of hosts. Using this metric, the quality of VM consoli-

dation can be represented by the increase in the aggregated idle time of hosts. However, this metric depends on the length of the overall evaluation period and the number of hosts. To eliminate this dependency, a normalized metric is proposed that is referred to as the Aggregated Idle Time Fraction (AITF) defined as shown in (6.4).

$$AITF = \frac{\sum_{h \in \mathcal{H}} t_i(h)}{\sum_{h \in \mathcal{H}} t_a(h)}, \tag{6.4}$$

where $\mathcal{H}$ is a set of hosts; $t_i(h)$ is the idle time of the host h; and $t_a(h)$ is the total activity time of the host h. To quantify the overall QoS delivered by the system, the Aggregated Overload Time Fraction (AOTF) metric is applied, which is based on (6.3) and defined as in (6.5).

$$AOTF(u_t) = \frac{\sum_{h \in \mathcal{H}} t_o(h, u_t)}{\sum_{h \in \mathcal{H}} t_b(h)}, \tag{6.5}$$

where $t_o(h, u_t)$ is the overload time of the host h calculated according to the overload threshold u_t; and $t_b(h)$ is the total busy (non-idle) time of the host h. The overhead of dynamic VM consolidation in the system is proposed to be evaluated in terms of the number of VM migrations initiated as a part of dynamic consolidation. This metric is referred to as the VM Migration Count (VMMC). Apart from that, the execution time of various components of the system including the execution time of the VM consolidation algorithms is evaluated.

6.6.3 Performance Evaluation Methodology

One of the key points of the proposed performance evaluation methodology is the minimization of manual steps required to run an experiment through automation. Automation begins from scripted installation of the OS, OpenStack services and their dependencies on the testbed's nodes, as described in the OpenStack installation guide [19]. The next step is writing scripts for preparing the system for an experiment, which includes starting up the required services, booting VM instances, and preparing them for starting the workload generation.

While most of the mentioned steps are trivial, workload generation is complicated by the requirement of synchronizing the time of starting the workload generation on all the

VMs. Another important aspect of workload generation is the way workload traces are assigned to VMs. Typically, the desired behavior is assigning a unique workload trace out of the full set of traces to each VM. Finally, it is necessary to create and maintain a specific level of CPU utilization for the whole interval between changes of the CPU utilization level defined by the workload trace for each VM.

This problem is addressed using a combination of a CPU load generation program[9], and a workload distribution web service and clients deployed on VMs [14]. When a VM boots from a pre-configured image, it automatically starts a script that polls the central workload distribution web service to be assigned a workload trace. Initially, the workload distribution web service drops requests from clients deployed on VMs to wait for the moment when all the required VM instances are booted up and ready for generating workload. When all clients are ready, the web service receives a command to start the workload trace distribution. The web service starts replying to clients by sending each of them a unique workload trace. Upon receiving a workload trace, every client initiates the CPU load generator and passes the received workload trace as an argument. The CPU load generator reads the provided workload trace file, and starts generating CPU utilization levels corresponding to the values specified in the workload trace file for each time frame.

During an experiment, OpenStack Neat continuously logs various events into both the database and log files on each host. After the experiment, the logged data are used by special result processing scripts to extract the required information and compute performance metrics discussed in Section 6.6.2, as well as the execution time of various system components. This process should be repeated for each combination of VM consolidation algorithms under consideration. After the required set of experiments is completed, other scripts are executed to perform automated statistical tests and plotting graphs for comparing the algorithms.

The next section presents an example of application of the proposed benchmark suite, and in particular applies: (1) OpenStack Neat as the dynamic VM consolidation framework; (2) the filtered PlanetLab workload traces discussed in Section 6.6.1; (3) the performance metrics defined in Section 6.6.2; and (4) the proposed evaluation methodology.

[9]The CPU load generator. http://github.com/beloglazov/cpu-load-generator

The full set of scripts used in the experiments is available online [14].

6.7 Performance Evaluation

In this section, the benchmark suite proposed in Section 6.6 is applied to evaluate OpenStack Neat and several dynamic VM consolidation algorithm discussed in Section 6.4.

6.7.1 Experimental Testbed

The testbed used for performance evaluation of the system consisted of the following hardware:

- 1 x Dell Optiplex 745
  - Intel(R) Core(TM) 2 CPU (2 cores, 2 threads) 6600 @ 2.40GHz
  - 2GB DDR2-667
  - Seagate Barracuda 80GB, 7200 RPM SATA II (ST3808110AS)
  - Broadcom 5751 NetXtreme Gigabit Controller
- 4 x IBM System x3200 M3
  - Intel(R) Xeon(R) CPU (4 cores, 8 threads), X3460 @ 2.80GHz
  - 4GB DDR3-1333
  - Western Digital 250 GB, 7200 RPM SATA II (WD2502ABYS-23B7A)
  - Dual Gigabit Ethernet (2 x Intel 82574L Ethernet Controller)
- 1 x Netgear ProSafe 16-Port 10/100 Desktop Switch FS116

The Dell Optiplex 745 machine was chosen to serve as the controller host running all the major OpenStack services and the global manager of OpenStack Neat. The 4 IBM System x3200 M3 servers were used as compute hosts, i.e. running OpenStack Nova, and local managers and data collectors of OpenStack Neat. All the machines formed a local network connected via the Netgear FS116 network switch.

Unfortunately, there was a hardware problem preventing the system from taking advantage of dynamic VM consolidation to save energy. The problem was that the compute nodes of the testbed did not support the Suspend to RAM power state, which is the most suitable for the purpose of dynamic VM consolidation. This state potentially

provides very low switching latency, on the order of 300 ms, while reducing the energy consumption to a negligible level [81]. Therefore, rather than measuring the actual energy consumption by the servers, the AITF metric introduced in Section 6.6.2 was applied to evaluate the system, which can be seen as a representation of potential energy savings.

6.7.2 Experimental Setup and Algorithm Parameters

From the point of view of experimenting with close to real world conditions, it is interesting to allocate as many VMs on a compute host as possible. This would create a more dynamic workload and stress the system. At the same time, it is important to use full-fledged VM images representing realistic user requirements. Therefore, the Ubuntu 12.04 Cloud Image [30] was used in the experiments, which is one of the Ubuntu VM images available in Amazon EC2.

Since the compute hosts of the testbed contained limited amount of RAM, to maximize the number of VMs served by a single host, it was necessary to use a VM instance type with the minimum amount of RAM sufficient for Ubuntu 12.04. The minimum required amount of RAM was empirically determined to be 128 MB. This resulted in the maximum of 28 VMs being possible to instantiate on a single compute host. Therefore, to maximize potential benefits of dynamic VM consolidation on the testbed containing 4 compute nodes, the total number of VM instances was set to 28, so that in an ideal case all of them can be placed on a single compute host, while the other 3 hosts are kept idle. Out of the 33 filtered PlanetLab workload traces discussed in Section 6.6.1, 28 traces were randomly selected, i.e., one unique 24-hour trace for each VM instance. The full set of selected traces is available online [14].

During the experiments, all the configuration parameters of OpenStack Neat were set to their default values listed in Table 6.2 except for the configuration of the overload detection algorithm. The overload detection algorithm was changed for each experiment by going through the following list of algorithms and their parameters:

1. MAX-ITF algorithm – a base line algorithm, which never detects host overloads leading to the maximum ITF for the host, where the algorithm is used.
2. The THR algorithm with the n parameter set to 2, and the CPU utilization threshold set to 0.8, 0.9, and 1.0.

3. The LRR algorithm with the safety parameter set to 0.9, 1.0, and 1.1.
4. The MHOD algorithm with the OTF parameter set to 0.2, 0.3, and 0.4.

Each experiment was run 3 times to handle the variability caused by random factors, such as the initial VM placement, workload trace assignment, and component communication latency. All the system initialization and result processing scripts, along with the experiment result packages are available online [14].

6.7.3 Experimental Results and Analysis

The results of experiments are graphically depicted in Figure 6.6. The mean values of the obtained AITF and AOTF metrics, and the number of VM migrations along with their 95% Confidence Intervals (CIs) are displayed in Table 6.7. The results of MAX-ITF show that for the current experiment setup it is possible to obtain high values of AITF of up to 50%, while incurring a high AOTF of more than 40%. All the THR, LRR, and MHOD allow tuning of the AITF values by adjusting the algorithm parameters. For the THR algorithm, the mean AITF increases from 36.9% to 49.2% with the corresponding decrease in the QoS level from 15.4% to 42.2% by varying the CPU utilization threshold from 0.8 to 1.0. The mean number of VM migrations decreases from 167.7 for the 80% threshold to 11.3 for the 100% threshold. The THR algorithm with the CPU utilization threshold set to 100% reaches the mean AITF shown by the MAX-ITF algorithm, which is expected as setting the threshold to 100% effectively disables host overload detection. Similarly, adjusting the safety parameter of the LRR algorithm from 1.1 to 0.9 leads to an increase of the mean AITF from 37.9% to 47.3% with a growth of the mean AOTF from

Table 6.7: The experimental results (mean values with 95% CIs)

Algorithm	AITF	AOTF	VM migrations
THR-0.8	36.9% (35.6, 38.2)	15.4% (12.5, 18.3)	167.7 (152.7, 182.6)
THR-0.9	43.0% (42.6, 43.5)	27.0% (25.7, 28.1)	75.3 (70.2, 80.5)
THR-1.0	49.2% (49.2, 49.4)	42.2% (33.0, 51.3)	11.3 (9.9, 12.8)
LRR-1.1	37.9% (37.9, 38.0)	17.8% (12.8, 22.7)	195.7 (158.3, 233.0)
LRR-1.0	40.3% (38.1, 42.4)	23.8% (21.4, 26.1)	93.7 (64.6, 122.8)
LRR-0.9	47.3% (45.2, 49.4)	34.4% (28.8, 40.0)	28.3 (23.2, 33.5)
MHOD-0.2	37.7% (36.8, 38.5)	16.0% (13.5, 18.5)	158.3 (153.2, 163.5)
MHOD-0.3	38.1% (37.7, 38.5)	17.9% (16.8, 18.9)	138.0 (81.6, 194.4)
MHOD-0.4	40.7% (37.0, 44.4)	21.4% (16.7, 26.0)	116.3 (26.6, 206.0)
MAX-ITF	49.2% (49.1, 49.3)	40.4% (35.8, 44.9)	14.0 (7.4, 20.6)

17.8% to 34.4% and decrease of the mean number of VM migrations from 195.7 to 28.3. THR-1.0 reaches the mean AITF of 49.2% with the mean AOTF of 42.2%, while LRR-0.9 reaches a close mean AITF of 47.3% with the mean AOTF of only 34.4%, which is a significant decrease compared with the AOTF of THR-1.0.

Varying the OTF parameter of the MHOD algorithm from 0.2 to 0.4 leads to an increase of the mean AITF from 37.7% to 40.7% with an increase of the mean AOTF from 16.0% to 21.4%. First of all, it is important to note that the algorithm meets the specified QoS constraint by keeping the value of the AOTF metric below the specified OTF parameters. However, the resulting mean AOTF is significantly lower than the specified OTF parameters: 17.9% for the 30% OTF, and 21.4% for the 40% OTF. This can be explained by a combination of two factors: (1) the MHOD algorithm is parameterized by the per-host OTF, rather than AOTF, which means that it meets the OTF constraint for each host independently; (2) due to the small scale of the experimental testbed, a single underloaded host used for offloading VMs from overloaded hosts is able to significantly skew the AITF metric. The AITF metric is expected to be closer to the specified OTF parameter for large-scale OpenStack Neat deployments. A comparison of the results produced by LRR-1.1 and LRR-1.0 with MHOD-0.2 and MHOD-0.4 reveals that the MHOD algorithm leads to lower values of the AOTF metric (higher level of QoS) for approximately equal values of the AITF metric.

Using the obtained AITF and AOTF metrics for each algorithm and data on power consumption by servers, it is possible to compute estimates of potential energy savings relatively to a non-power-aware system assuming that hosts are switched to the sleep mode during every idle period. To obtain a lower bound on the estimated energy savings, it is assumed that when dynamic VM consolidation is applied, the CPU utilization of each host is 80% when it is active and non-overloaded, and 100% when it is overloaded. According to the data provided by Meisner et al. [81], power consumption of a typical blade server is 450 W in the fully utilized state, 270 W in the idle state, and 10.4 W in the sleep mode. Using the linear server power model proposed by Fan et al. [44] and the power consumption data provided by Meisner et al. [81], it is possible to calculate power consumption of a server at any utilization level.

To calculate the base energy consumption by a non-power-aware system, it is as-

Table 6.8: Energy consumption estimates

Algorithm	Energy, kWh	Base energy, kWh	Energy savings
THR-0.8	25.99	34.65	24.99%
THR-0.9	24.01	33.80	28.96%
THR-1.0	22.09	32.93	32.91%
LRR-1.1	25.66	34.50	25.63%
LRR-1.0	24.96	34.18	26.97%
LRR-0.9	22.60	33.20	31.93%
MHOD-0.2	25.70	34.53	25.59%
MHOD-0.3	25.59	34.48	25.76%
MHOD-0.4	24.72	34.12	27.54%
MAX-ITF	22.07	32.94	33.01%

sumed that in such a system all the compute hosts are always active with the load being distributed across them. Since, the power model applied in this study is linear, it is does not matter how exactly the load is distributed across the servers. The estimated energy consumption levels for each overload detection algorithm, along with the corresponding base energy consumption by a non-power-aware system, and percentages of the estimated energy savings are presented in Table 6.8.

According to the estimates, MAX-ITF leads to the highest energy savings over the base energy consumption of approximately 33% by the cost of substantial performance degradation (AOTF = 40.4%). The THR, LRR, and MHOD algorithms lead to energy savings from approximately 25% to 32% depending on the specified parameters. Similarly to the above comparison of algorithms using the AITF metric, LRR-0.9 produces energy savings close to those of THR-1.0 (31.93% compared with 32.91%), while significantly reducing the mean AOTF from 42.2% to 34.4%. The MHOD algorithm produces approximately equal or higher energy savings than the LRR algorithm with lower mean AITF values, i.e., higher levels of QoS, while also providing the advantage of specifying a QoS constraint as a parameter of the algorithm. The obtained experimental results confirm the hypothesis that dynamic VM consolidation is able to significantly reduce energy consumption in an IaaS Cloud with a limited performance impact.

Table 6.9 lists mean values of the execution time along with 95% CIs measured for each overload detection algorithm during the experiments for some of the system components: processing underload and overload requests by the Global Manager (GM), overload detection algorithms executed by the Local Manager (LM), and iterations of the Data Collector (DC). Request processing by the global manager takes on average between 30

Table 6.9: The execution time of components in seconds (mean values with 95% CIs)

Algorithm	GM underload	GM overload	LM overload	DC
THR	33.5 (26.4, 40.5)	60.3 (54.0, 66.7)	0.003 (0.000, 0.006)	0.88 (0.84, 0.92)
LRR	34.4 (27.6, 41.1)	50.3 (47.8, 52.8)	0.006 (0.003, 0.008)	0.76 (0.73, 0.80)
MHOD	41.6 (27.1, 56.1)	53.7 (50.9, 56.6)	0.440 (0.429, 0.452)	0.92 (0.88, 0.96)
MAX-ITF	41.7 (9.6, 73.7)	–	0.001 (0.000, 0.001)	1.03 (0.96, 1.10)

and 60 seconds, which is mostly determined by the time required to migrate VMs. The mean execution time of the MHOD algorithm is higher than those of THR and LRR, while still being under half a second resulting in a negligible overhead considering that it is executed at most once in 5 minute. The mean execution time of an iteration of the data collector is similarly under a second, which is also negligible considering that it is executed only once in 5 minutes.

6.7.4 Scalability Remarks

Scalability and eliminating single points of failure are important benefits of designing a dynamic VM consolidation system in a distributed way. According to the approach adopted in the design of OpenStack Neat, the underload / overload detection and VM selection algorithms are able to inherently scale with the increased number of compute hosts. This is due to the fact that they are executed independently on each compute host and do not rely on information about the global state of the system. In regard to the database setup, there exist distributed database solutions, e.g., the MySQL Cluster [90].

On the other hand, in the current implementation of OpenStack Neat, there assumed to be only one instance of the global manager deployed on a single controller host. This limits the scalability of VM placement decisions and creates a single point of failure. However, even with this limitation the overall scalability of the system is significantly improved compared with existing completely centralized VM consolidation solutions. Compared with centralized solutions, the only functionality implemented in OpenStack Neat by the central controller is the placement of VMs selected for migration, which constitute only a fraction of the total number of VMs in the system. To address the problem of a single point of failure, it is possible to run a second instance of the global manager, which initially does not receive requests from the local managers and gets automatically

activated when the primary instance of the global manager fails. However, the problem of scalability is more complex since it is necessary to have multiple independent global managers concurrently serving requests from local managers.

Potentially it is possible to implement replication of the global manager in line with OpenStack's approach to scalability by replication of its services. From the point of view of communication between the local and global managers, replication can be simply implemented by a load balancer that distributes requests from the local managers across the set of replicated global managers. A more complex problem is synchronizing the activities of the replicated global managers. It is necessary to avoid situations when two global managers place VMs on a single compute host simultaneously, since that would imply that they use an out-of-date view of the system state. One potential solution to this problem could be a continuous exchange of information between global managers during the process of execution of the VM placement algorithm, i.e., if a host is selected by a global manager for a VM, it should notify the other global managers to exclude that host from their sets of available destination hosts.

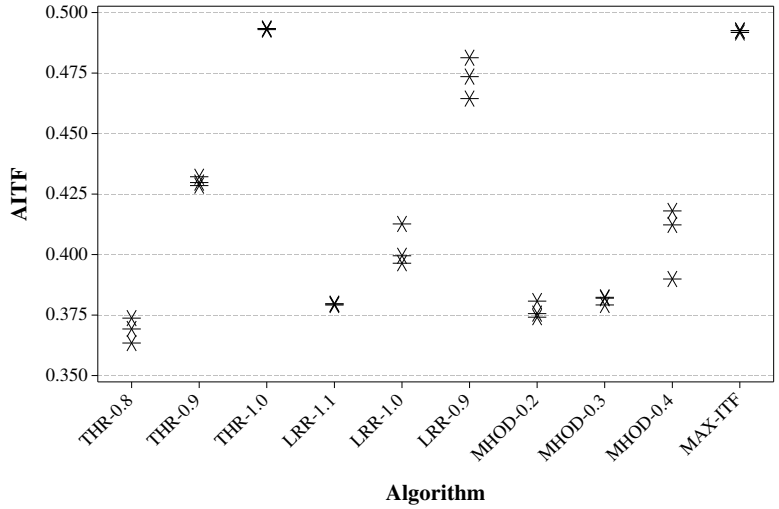
6.8 Conclusions

This chapter has presented a design and implementation of an open source framework for dynamic VM consolidation in OpenStack Clouds, called OpenStack Neat. The framework follows a distributed model of dynamic VM consolidation, where the problem is divided into 4 sub-problems: host underload detection, host overload detection, VM selection, and VM placement. Through its configuration, OpenStack Neat can be customized to use various implementations of algorithms for each for the 4 sub-problems of dynamic VM consolidation. OpenStack Neat is transparent to the base OpenStack installation by interacting with it using the public APIs, and not requiring any modifications of OpenStack's configuration. In addition, a benchmark suite has been proposed that comprises OpenStack Neat as the base software framework, a set of PlanetLab workload traces, performance metrics, and methodology for evaluating and comparing dynamic VM consolidation algorithms following the distributed model.

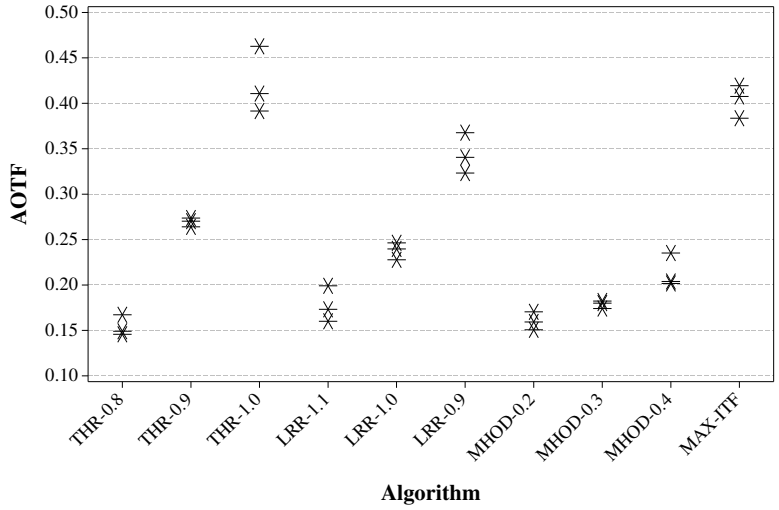
The experimental results and estimates of energy consumption have shown that Open-

Stack Neat is able to reduce energy consumption by the compute nodes of a 4-node testbed by 25% to 33%, while resulting in a limited application performance impact from approximately 15% to 40% AOTF. The MHOD algorithm has led to approximately equal or higher energy savings with lower mean AOTF values compared with the other evaluated algorithms, while also allowing the system administrator to explicitly specify a QoS constraint in terms of the OTF metric.

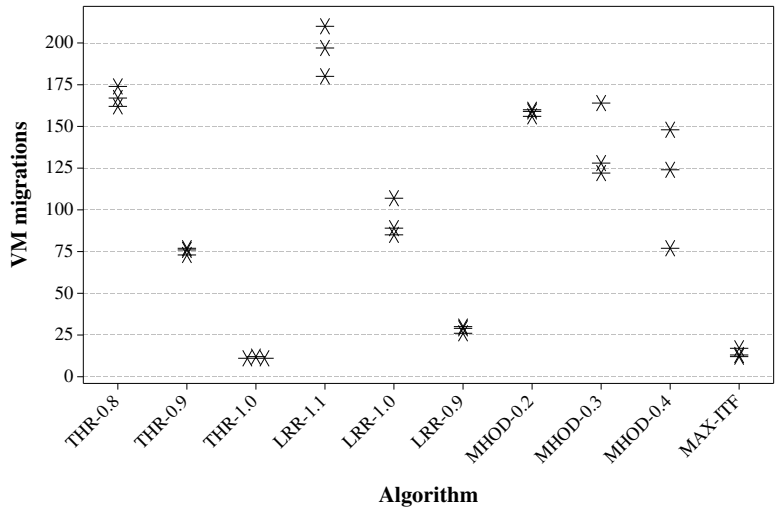
The performance overhead of the framework is nearly negligible taking on average only a fraction of a second to execute iterations of the components. The request processing of the global manager takes on average between 30 and 60 seconds and is mostly determined by the time required to migrate VMs. The results have shown that dynamic VM consolidation brings significant energy savings with a limited impact on the application performance. The proposed framework can be applied in both further research on dynamic VM consolidation, and real OpenStack Cloud deployments to improve the utilization of resources and reduce energy consumption.



(a) The AITF metric



(b) The AOTF metric



(c) The number of VM migrations

Figure 6.6: The experimental results

Chapter 7
Conclusions and Future Directions

This chapter summarizes the research work on energy-efficient resource management in Cloud data centers presented in this thesis and highlights the main findings. It also discusses open research problems in the area and outlines a number of future research directions.

7.1 Conclusions and Discussion

CLOUD computing has made the vision of computing resources as a utility another step closer to the reality. As the technology advances and network access becomes faster and with lower latency, the model of delivering computing power remotely over the Internet will proliferate. Therefore, Cloud data centers are expected to grow and accumulate a larger fraction of the world's computing resources. In this context, energy-efficient management of data center resources is a crucial problem in regard to both the operating costs and CO_2 emissions to the environment.

This thesis has proposed and investigated a suite of novel techniques for implementing distributed dynamic VM consolidation in IaaS Clouds under workload-independent QoS constraints. The proposed approach improves the utilization of data center resources and reduces energy consumption, while satisfying the defined QoS requirements. Pike Research [96] forecasts that research, such as presented in this thesis, will facilitate the reduction of the data center energy expenditures from $23.3 billion in 2010 to $16.0 billion in 2020, as well as cause a 28% reduction in greenhouse gas emissions from the 2010 levels as a result of adoption of the Cloud computing model in delivering IT services.

In addition to the complexity of implementing dynamic VM consolidation, IaaS Clouds being the target environments imply extra requirements, such as satisfying workload-independent QoS constraints and distributed architecture of the VM management system

to provide scalability and eliminate single points of failure. To address the formulated research issues, this thesis has achieved each of the objectives delineated in Chapter 1. In particular, Chapter 2 presented an in-depth review, analysis, and taxonomy of the area of energy-efficient resource management in computing systems at the hardware and firmware, OS, virtualization, and data centers levels. The research literature analysis has helped to identify gaps, open challenges, and clearly determine the research direction undertaken in the thesis.

In Chapter 3, dynamic VM consolidation algorithms have been analyzed to obtain theoretical performance estimates and insights into designing online algorithms for dynamic VM consolidation. The chapter has concluded with a discussion of potential benefits of randomized online algorithms. Another conclusion has been that real workloads often exhibit interrelations between subsequent workload states; therefore, dynamic VM consolidation algorithm can be improved by assuming that future workload states can be predicted to some degree based on the history of the observed system behavior. However, such algorithms cannot be analyzed using simple distributional or adversary models, such as oblivious adversary, as realistic workloads require more complex modeling, e.g. using Markov chains [105].

To address the next objective, Chapter 4 has proposed a workload-independent QoS metric that can be used in defining system-wide QoS constraints. The proposed OTF metric is based on the observation that the application performance degrades when the utilization of the server's CPU reaches 100%. According to this idea, the metric is defined as a fraction of time when the server has been overloaded relatively to the total activity time. The metric can be extended to span multiple servers forming the AOTF metric by substituting the time values with the aggregate values over the full set of servers. Moreover, the metric is adjustable to system requirements by tuning the threshold of the CPU utilization defining the overload state. For instance, a server can be considered to be overloaded when its utilization exceeds 80%. Such tuning is useful in cases when the application performance is known to degrade after a certain level of CPU utilization.

In addition, Chapter 4 has proposed an approach to distributed dynamic VM consolidation consisting in splitting the problem into 4 sub-problems: (1) host underload detection; (2) host overload detection; (3) VM selection; and (4) VM placement. Splitting the

problem allows executing algorithms for the first 3 sub-problems in a distributed manner independently on each compute node. VM placement decisions still need to be made by a global manager; however, the load on the global manager is significantly reduced since it only makes placement decisions for the VMs selected for migration. Moreover, the global manager can be replicated, thus eliminating a single point of failure and making the system completely distributed and decentralized.

Host overload detection is the sub-problem of distributed dynamic VM consolidation that directly influences the QoS delivered by the system. Therefore, to provide reliable QoS, Chapter 5 has investigated a host overload detection algorithm based on a Markov chain model, which allows an explicit specification of a QoS goal in terms of the OTF metric. Instead of tuning parameters of the algorithms to obtain the required QoS level, the proposed MHOD algorithm accepts a parameter that serves as an OTF constraint.

Apart from the theoretical exploration and simulation-based evaluation of the proposed algorithms, the presented work resulted in an open source implementation of OpenStack Neat[1], a software framework for distributed dynamic VM consolidation in IaaS Clouds. The framework presented in Chapter 6 is implemented as an add-on to the OpenStack Cloud platform[2], which can be transparently attached to existing OpenStack installations. The framework is extensible, as it enables configuration-based substitution of algorithms for each sub-problem of distributed dynamic VM consolidation. Experiments conducted on a 5-node testbed have shown that dynamic VM consolidation is able to reduce energy consumption by the compute nodes by up to 30% with a limited application performance impact.

The framework has been designed and implemented to both be used as a tool in further academic research on distributed dynamic VM consolidation algorithms, and be applied in real-world OpenStack Cloud deployments to improve the utilization of resources and save energy. In addition, to facilitate further research efforts in the area, Chapter 6 has outlined a benchmark suite comprising OpenStack Neat as the base software framework, a set of PlanetLab workload traces, performance metrics, and methodology for evaluating and comparing distributed dynamic VM consolidation algorithms.

[1]The OpenStack Neat framework. http://openstack-neat.org/
[2]The OpenStack platform. http://openstack.org/

7.2 Future Research Directions

Despite substantial contributions of the current thesis in energy-efficient distributed dynamic VM consolidation, there are a number of open research challenges that need to be addressed in order to further advance the area.

7.2.1 Advanced Distributed VM Placement Algorithms

Replication of the global managers would lead to multiple instances of the VM placement algorithm being executed concurrently on multiple controller nodes. It is important to develop advanced VM placement algorithms that would efficiently exchange information between the algorithm instances to reduce the impact of the distribution and provide the quality of VM placement close to centralized algorithms.

The currently proposed version of the VM placement algorithm implemented in OpenStack Neat processes each VM placement request received from local managers individually. There is scope for improvement in this regard, e.g., by buffering several subsequent requests constrained by the time period and buffer size, and making a VM placement decision taking into account the buffered requests simultaneously. It is important to carefully design and analyze such algorithms to provide close to optimal solutions, as even slight variation of the parameter values may lead to significant performance deviation. One approach is to apply the competitive analysis framework to analyze the proposed algorithms and derive performance bounds.

Another potential improvement of the VM placement algorithm is implementing extra constraints on the VM placement. Such constraints can be useful when it is required to co-allocate VMs on the physical server or a set of servers. For example, these requirements can be imposed by the users due to performance or privacy concerns.

7.2.2 VM Network Topologies

In virtualized data centers VMs often communicate with each other, establishing virtual network topologies. However, due to VM migrations or non-optimized allocation, the communicating VMs may end up hosted on logically distant physical nodes leading to costly data transfers between them. If the communicating VMs are allocated to hosts in

different racks or enclosures, the network communication may involve network switches that consume significant amounts of energy.

To eliminate this data transfer overhead and minimize energy consumption, it is necessary to observe the communication between VMs and place the communicating VMs on the same or closely located nodes. In particular, to provide efficient reallocations, it is required to develop power consumption models of the network devices and estimate the cost of data transfer depending on the traffic volume. Moreover, VM migrations consume additional energy and have a negative impact on the performance. The VM placement algorithm has to ensure that the cost of migration does not exceed the benefits.

7.2.3 Exploiting VM Resource Usage Patterns

IaaS environments allow the users to provision VMs on-demand and deploy any kind of applications. This leads to the fact that different applications (e.g., HPC applications, web services) may share a physical node. However, it is not obvious how these applications influence each other in terms of performance, as they can be data, network or compute intensive creating variable or static load on the resources.

The problem is to determine what kinds of applications can be co-allocated to provide the most efficient overall resources usage. For example, it can be achieved by choosing the applications that do not intensively use the same resource. A compute intensive (scientific) application can be effectively combined with a web application (file server), as the former mostly relies on the CPU performance, whereas the latter utilizes disk storage and network bandwidth.

Therefore, it is important to investigate Cloud application workloads in order to identify common behaviors, patterns, and explore load forecasting approaches that can potentially lead to more efficient resource provisioning and consequently higher energy efficiency. Furthermore, it is necessary to develop VM consolidation algorithms that will use the information about the historical workload patterns and application behavior to select which applications will share physical resources. These will minimize the overlapping of the resource usage by applications, and thus their influence on each other, as well as reducing the amount of migration needed when their demand for resources changes.

7.2.4 Thermal-Aware Dynamic VM Consolidation

A significant part of electrical energy consumed by computing resources is transformed into heat. High temperature leads to a number of problems, such as the reduced system reliability and availability, as well as the decreased life time of the hardware. In order to keep the system components within their safe operating temperature and prevent failures and crashes, the emitted heat must be dissipated. The cooling problem becomes extremely important for modern blade and 1U rack servers that pack computing resources with high density and complicate heat dissipation. For example, in a 30,000 ft^2 data center with 1,000 standard computing racks, each consuming 10 kW, the initial cost of purchasing and installing the infrastructure is \$2-\$5 million; with an average cost of \$10/MWh, the annual costs of cooling alone are \$4-\$8 million [94]. Therefore, apart from the hardware improvements, it is essential to address the problem of optimizing the cooling system operation from the resource management side.

One of the ways to minimize the cooling operating costs is to continuously monitor thermal state of physical nodes and reallocate VMs from a node when it becomes overheated. In this case, the cooling system of the offloaded node can be slowed down allowing natural heat dissipation. Moreover, there has been research work on modeling the thermal topology of a data center that can lead to more efficient workload placement [63]. Therefore, it is necessary to investigate how and when to reallocate VMs to minimize the power drawn by the cooling system, while preserving safe temperature of the resources and minimizing the migration overhead and performance degradation.

7.2.5 Dynamic and Heterogeneous SLAs

Cloud data centers need to provide strict QoS guarantees, which are documented in the form of SLAs. Resource provisioning within a data center directly influences whether the SLAs are met. Current Cloud data centers host applications from clients distributed globally. These clients have very different requirements, which may also vary over time. For example, an organization using Cloud services may require tighter response time guarantees in day time than in night time. To address the problem, it is necessary to develop algorithms that exploit time variation in SLAs of the users to minimize the number of

physical servers required.

Currently, to simplify management, resources in Cloud data centers are allocated to clients depending only on that client's SLAs, regardless of the SLAs of other users. The intrinsic differences among various SLAs can make huge differences in the amount of resources allocated to each user. Although heterogeneous requirements of users make scheduling and VM consolidation algorithms complex, they can be exploited to improve energy-efficiency. It is important to devise and analyze algorithms that make use of such heterogeneity. Moreover, to meet the requirements of large-scale Cloud data centers, it is necessary to design a solution scalable to handling thousands of users.

7.2.6 Power Capping

As discussed in Chapter 2, power capping is a power management technique aimed at meeting a power budget available for a computing system. This technique is useful in environments where the overall power consumption is constrained by either the capacity of the power delivery infrastructure, or by financial aspects. For instance, if the power consumption by data center resources is tending to exceed the available power budget, the resource management system may intentionally degrade the performance of the applications by down-scaling the hardware speed in order to keep the overall power consumption within the budget. A potential future research direction could be an extension of the distributed dynamic VM consolidation approach proposed in this thesis to take into account the power budget aspect and implement power capping.

7.2.7 Competitive Analysis of Dynamic VM Consolidation Algorithms

As indicated in the literature review, the current approaches to dynamic VM consolidation are weak in terms of providing performance guarantees. As proposed in this thesis, one of the ways to prove performance bounds is to divide the problem of energy-efficient dynamic VM consolidation into a few sub-problems that can be analyzed individually. It is important to analytically model the problem and derive optimal and near optimal approximation algorithms that provide provable efficiency. This has been done in Chapter 3 by proving competitive ratios of online deterministic algorithms for dynamic VM

consolidation. The MHOD algorithm proposed in Chapter 5 is also derived based on an analytical Markov chain model that allows a derivation of an optimal control policy for known stationary workloads.

However, the performance guarantees of the proposed heuristic algorithms for the problems of underload detection and VM selection have not yet been obtained. It is important to conduct competitive analysis of these algorithms to derive theoretical performance bounds and obtain insights into improving their performance. Furthermore, meeting QoS constraints should be formally incorporated into the algorithms.

7.2.8 Replicated Global Managers

Scalability and eliminating single points of failure are important benefits of designing a dynamic VM consolidation system in a distributed way. The approach proposed in this thesis makes a big step towards completely distributed and decentralized dynamic VM consolidation, as the underload / overload detection and VM selection algorithms are able to inherently scale with the increased number of compute hosts – these algorithms are executed independently on each compute node and do not rely on information about the global state of the system.

However, it is assumed that there is only one instance of the global manager deployed on a single controller host, which makes decisions regarding placement of VMs selected for migration. This limits the scalability of VM placement decisions and creates a single point of failure. To address the problem of a single point of failure, it is possible to run a second instance of the global manager, which initially does not receive requests from the local managers and gets automatically activated when the primary instance of the global manager fails. However, the scalability problem is more complex since it requires multiple independent global managers concurrently serving requests from local managers.

A potential solution is to investigate replication of the global manager, i.e., executing multiple instances of the VM placement algorithm concurrently on multiple nodes. From the point of view of communication between the local and global managers, replication can be implemented by a load balancer that distributes requests from the local managers across the set of replicated global managers. A more complex research problem is synchronizing activities of the replicated global managers. It is necessary to avoid situations

when two global managers place VMs on a single compute host simultaneously, since that would imply that they use an out-of-date view of the system state. One potential solution to this problem could be the continuous exchange of information between global managers during the process of execution of the VM placement algorithm, i.e., if a host is selected by a global manager for a VM, the other global managers should be notified to exclude that host from their sets of available destination hosts. This problem is an interesting direction for future research, and its solution would lead to a completely distributed and decentralized dynamic VM consolidation system.

7.3 Final Remarks

Clouds have evolved as the next-generation IT platform for hosting applications in science, business, social networking, and media content delivery. Energy-efficient management of Cloud infrastructure resources, and in particular dynamic VM consolidation explored in this thesis, will enable resource providers to successfully offer scalable service provisioning with lower energy requirements, costs, and CO_2 emissions. Research, such as presented in this thesis, combined with open source technologies, such as the OpenStack platform and the OpenStack Neat framework[3], will undoubtedly drive further innovation in Cloud computing and development of next generation computing systems.

[3]The OpenStack Neat framework. http://openstack-neat.org/

Bibliography

[1] H. Abdi, "Multiple correlation coefficient," in *Encyclopedia of Measurement and Statistics*, N. J. Salkind, Ed. Sage, Thousand Oaks, CA, USA, 2007, pp. 648–651.

[2] S. Albers, "Energy-efficient algorithms," *Communications of the ACM*, vol. 53, no. 5, pp. 86–96, 2010.

[3] V. Anagnostopoulou, S. Biswas, H. Saadeldeen, A. Savage, R. Bianchini, T. Yang, D. Franklin, and F. T. Chong, "Barely alive memory servers: Keeping data active in a low-power state," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 8, no. 4, pp. 31:1–31:20, 2012.

[4] M. Andreolini, S. Casolari, and M. Colajanni, "Models and framework for supporting runtime decisions in web-based systems," *ACM Transactions on the Web (TWEB)*, vol. 2, no. 3, pp. 17:1–17:43, 2008.

[5] L. L. Andrew, M. Lin, and A. Wierman, "Optimality, fairness, and robustness in speed scaling designs," in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2010, pp. 37–48.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of Cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2009.

[7] ASHRAE, TC, "Thermal guidelines for data processing environments," American Society of Heating and Refrigerating and Air-Conditioning Engineers, Tech. Rep. 9.9, 2004.

[8] J. Baliga, R. Ayre, K. Hinton, and R. S. Tucker, "Green Cloud computing: Balancing energy in processing, storage and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[9] J. Baliga, K. Hinton, and R. S. Tucker, "Energy consumption of the Internet," in *Proceedings of the International Conference on the Optical Internet (COIN) with the 32nd Australian Conference on Optical Fibre Technology (ACOFT)*, 2007, pp. 1–3.

[10] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 1, pp. 151–160, 1998.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003, pp. 16–177.

[12] L. A. Barroso, "The price of performance," *Queue*, vol. 3, no. 7, pp. 48–53, 2005.

[13] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[14] A. Beloglazov, "Scripts for setting up and analyzing results of experiments using OpenStack Neat," (accessed on 26/11/2012). [Online]. Available: http://github.com/beloglazov/spe-2012-experiments

[15] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Future Generation Computer Systems (FGCS)*, vol. 28, no. 5, pp. 755–768, 2011.

[16] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers," *Concurrency and Computation: Practice and Experience (CCPE)*, vol. 24, no. 13, pp. 1397–1420, 2012.

[17] ——, "Managing overloaded hosts for dynamic consolidation of virtual machines in Cloud data centers under quality of service constraints," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 24, no. 7, pp. 1366–1379, 2013.

[18] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A taxonomy and survey of energy-efficient data centers and Cloud computing systems," *Advances in Computers, M. Zelkowitz (ed.)*, vol. 82, pp. 47–111, 2011.

[19] A. Beloglazov, S. F. Piraghaj, M. Alrokayan, and R. Buyya, "Deploying OpenStack on CentOS using the KVM hypervisor and GlusterFS distributed file system," CLOUDS-TR-2012-3, CLOUDS Laboratory, The University of Melbourne, Australia, Tech. Rep., 2012.

[20] L. Benini, A. Bogliolo, and G. D. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299–316, 2000.

[21] L. Benini, A. Bogliolo, G. A. Paleologo, and G. D. Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813–833, 1999.

[22] M. Blackburn, "Five ways to reduce data center server power consumption," The Green Grid, Tech. Rep., 2008.

[23] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2007, pp. 119–128.

[24] G. Bolch, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Blackwell, 2006.

[25] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, New York, 1998, vol. 53.

[26] G. Buttazzo, "Scalable applications for energy-aware processors," in *Embedded Software*, ser. Lecture Notes in Computer Science, A. Sangiovanni-Vincentelli and J. Sifakis, Eds. Springer Berlin Heidelberg, 2002, vol. 2491, pp. 153–165.

[27] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for Cloud computing: A vision, architectural elements, and open

challenges," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2010, pp. 1–12.

[28] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[29] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of Cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[30] Canonical Ltd., "Ubuntu 12.04 (Precise Pangolin) Cloud images," (accessed on 22/11/2012). [Online]. Available: http://uec-images.ubuntu.com/precise/current/

[31] M. Cardosa, M. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2009, pp. 327–334.

[32] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 103–116, 2001.

[33] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, pp. 303–314, 2005.

[34] E. Y. Chung, L. Benini, A. Bogliolo, Y. H. Lu, and G. D. Micheli, "Dynamic power management for nonstationary service requests," *IEEE Transactions on Computers*, vol. 51, no. 11, pp. 1345–1361, 2002.

[35] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005, pp. 273–286.

[36] W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots," *Journal of the American Statistical Association*, vol. 74, no. 368, pp. 829–836, 1979.

[37] ——, *Visualizing data*. Hobart Press, Summit, New Jersey, 1993.

[38] W. S. Cleveland and C. Loader, "Smoothing by local regression: Principles and methods," *Statistical Theory and Computational Aspects of Smoothing*, vol. 1049, pp. 10–49, 1996.

[39] S. B. David, A. Borodin, R. Karp, G. Tardos, and A. Widgerson, "On the power of randomization in online algorithms," in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, 1990, pp. 379–386.

[40] S. Devadas and S. Malik, "A survey of optimization techniques targeting low power VLSI circuits," in *Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference (DAC)*, 1995, pp. 242–247.

[41] G. Dhiman, K. Mihic, and T. Rosing, "A system for online power prediction in virtualized environments using gaussian mixture models," in *Proceedings of the 47th Annual ACM/IEEE Design Automation Conference (DAC)*, 2010, pp. 807–812.

[42] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive disk spin-down policies for mobile computers," *Computing Systems*, vol. 8, no. 4, pp. 381–413, 1995.

[43] E. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Power-Aware Computer Systems*, ser. Lecture Notes in Computer Science, B. Falsafi and T. Vijaykumar, Eds. Springer Berlin Heidelberg, 2003, vol. 2325, pp. 179–197.

[44] X. Fan, W. D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*, 2007, pp. 13–23.

[45] D. G. Feitelson, "Workload modeling for performance evaluation," in *Performance Evaluation of Complex Systems: Techniques and Tools*, ser. Lecture Notes in Computer Science, M. C. Calzarossa and S. Tucci, Eds. Springer Berlin Heidelberg, 2002, vol. 2459, pp. 114–141.

[46] E. Feller, L. Rilling, and C. Morin, "Snooze: A scalable and autonomic virtual machine management framework for private Clouds," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012, pp. 482–489.

[47] E. Feller, C. Rohr, D. Margery, and C. Morin, "Energy management in IaaS Clouds: A holistic approach," in *Proceedings of the 5th IEEE International Conference on Cloud Computing (IEEE CLOUD)*, 2012, pp. 204–212.

[48] K. Flautner, S. Reinhardt, and T. Mudge, "Automatic performance setting for dynamic voltage scaling," *Wireless networks*, vol. 8, no. 5, pp. 507–520, 2002.

[49] J. Flinn and M. Satyanarayanan, "Managing battery lifetime with energy-aware adaptation," *ACM Transactions on Computer Systems (TOCS)*, vol. 22, no. 2, pp. 137–179, 2004.

[50] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *Proceedings of the 11th Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance)*, 2009, pp. 157–168.

[51] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of HPC applications on distributed Cloud-oriented data centers," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 71, no. 6, pp. 732–749, 2011.

[52] Gartner, Inc., "Gartner estimates ICT industry accounts for 2 percent of global CO2 emissions," 2007, (accessed on 17/01/2013). [Online]. Available: http://www.gartner.com/it/page.jsp?id=503867

[53] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for pareto-optimal congurations in parameterized systems-on-a-chip," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2001, pp. 25–30.

[54] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *Proceedings of the 38th IEEE International Conference on Dependable Systems and Networks (DSN)*, 2008, pp. 326–335.

[55] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," *Computer Networks*, vol. 53, no. 17, pp. 2905–2922, 2009.

[56] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithm for dynamic speed-setting of a low-power CPU," in *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking (MobiCom)*, 1995, pp. 13–25.

[57] B. Guenter, N. Jain, and C. Williams, "Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning," in *Proceedings of the 30st Annual IEEE International Conference on Computer Communications (INFOCOM)*, 2011, pp. 1332–1340.

[58] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr, and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2005, pp. 186–195.

[59] F. Hermenier, X. Lorca, J. Menaud, G. Muller, and J. Lawall, "Entropy: A consolidation manager for clusters," in *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, 2009, pp. 41–50.

[60] P. J. Huber, E. Ronchetti, and E. Corporation, *Robust statistics*. Wiley Online Library, 1981, vol. 1.

[61] C. H. Hwang and A. C. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 5, no. 2, pp. 226–241, 2000.

[62] S. Irani, R. Gupta, and S. Shukla, "Competitive analysis of dynamic power management strategies for systems with multiple power savings states," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2002, pp. 117–123.

[63] P. Johnson and T. Marker, "Data centre energy efficiency product profile," Equipment Energy Efficiency Committee (E3) of the Australian Government Department of the Environment, Water, Heritage and the Arts (DEWHA), Tech. Rep., 2009.

[64] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in Cloud infrastructures," in *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS)*, 2010, pp. 62–73.

[65] M. G. Kendall and J. K. Ord, *Time-series*. Oxford University Press, Oxford, 1973.

[66] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of Cloud resources for real-time services," in *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science (MGC)*, 2009, pp. 1–6.

[67] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: The Linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[68] G. Koch, "Discovering multi-core: Extending the benefits of moore's law," *Technology*, vol. 1, 2005.

[69] J. G. Koomey, "Estimating total power consumption by servers in the US and the world," Lawrence Berkeley National Laboratory, Tech. Rep., 2007.

[70] ——, "Growth in data center electricity use 2005 to 2010," Analytics Press, Tech. Rep., 2011.

[71] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, "vManage: Loosely coupled platform and virtualization management in data centers," in *Proceedings of the 6th International Conference on Autonomic Computing (ICAC)*, 2009, pp. 127–136.

[72] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[73] S. Lee and T. Sakurai, "Run-time voltage hopping for low-power real-time systems," in *Proceedings of the 37nd Annual ACM/IEEE Design Automation Conference (DAC)*, 2000, pp. 806–809.

[74] L. Lefèvre and A.-C. Orgerie, "Designing and evaluating an energy efficient Cloud," *The Journal of Supercomputing*, vol. 51, no. 3, pp. 352–373, 2010.

[75] H. Li, "Workload dynamics on clusters and grids," *The Journal of Supercomputing*, vol. 47, no. 1, pp. 1–20, 2009.

[76] M. Lin, Z. Liu, A. Wierman, and L. L. H. Andrew, "Online algorithms for geographical load balancing," in *Proceedings of the 3rd International Green Computing Conference (IGCC)*, 2012, pp. 1–10.

[77] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM)*, 2011, pp. 1098–1106.

[78] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. H. Andrew, "Greening geographic load balancing," in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2011, pp. 233–244.

[79] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with PACE," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1, pp. 50–61, 2001.

[80] S. O. Luiz, A. Perkusich, and A. M. N. Lima, "Multisize sliding window in workload estimation for dynamic power management," *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1625–1639, 2010.

[81] D. Meisner, B. Gold, and T. Wenisch, "PowerNap: eliminating server idle power," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 205–216, 2009.

[82] K. Mills, J. Filliben, and C. Dabrowski, "Comparing VM-placement algorithms for on-demand Clouds," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2011, pp. 91–98.

[83] L. Minas and B. Ellison, *Energy Efficiency for Information Technology: How to Reduce Power Consumption in Servers and Data Centers*. Intel Press, 2009.

[84] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[85] R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting platform heterogeneity for power efficient data centers," in *Proceedings of the 4th International Conference on Autonomic Computing (ICAC)*, 2007, pp. 5–5.

[86] R. Nathuji and K. Schwan, "VirtualPower: Coordinated power management in virtualized enterprise systems," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 265–278, 2007.

[87] R. Neugebauer and D. McAuley, "Energy is just another resource: Energy accounting and energy pricing in the nemesis OS," in *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS)*, 2001, pp. 59–64.

[88] P. W. Ong and R. H. Yan, "Power-conscious software design – a framework for modeling software on hardware," in *Proceedings of the IEEE Symposium on Low Power Electronics*, 1994, pp. 36–37.

[89] Open Compute Project, "Energy efficiency," (accessed on 21/11/2012). [Online]. Available: http://opencompute.org/about/energy-efficiency/

[90] Oracle Corporation, "MySQL cluster CGE," (accessed on 23/11/2012). [Online]. Available: http://www.mysql.com/products/cluster/

[91] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, vol. 2, 2006, pp. 215–230.

[92] K. S. Park and V. S. Pai, "CoMon: a mostly-scalable monitoring system for PlanetLab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

[93] D. F. Parkhill, *The challenge of the computer utility*. Addison-Wesley Reading, MA, 1966.

[94] C. D. Patel, C. E. Bash, R. Sharma, M. Beitelmal, and R. Friedrich, "Smart cooling of data centers," in *Proceedings of the Pacific RIM/ASME International Electronics Packaging Technical Conference and Exhibition (InterPACK)*, 2003, pp. 129–137.

[95] F. Petrini, J. Moreira, J. Nieplocha, M. Seager, C. Stunkel, G. Thorson, P. Terry, and S. Varadarajan, "What are the future trends in highperformance interconnects for

parallel computers," in *Proceedings of the 12th Annual IEEE Symposium on High Performance Interconnects (HOTI)*, 2004, pp. 3–3.

[96] Pike Research, "Cloud computing to reduce global data centre energy expenditures by 38% in 2020," 2010, (accessed on 21/01/2013). [Online]. Available: http://www.pikeresearch.com/newsroom/cloud-computing-to-reduce-global-data-center-energy-expenditures-by-38-in-2020

[97] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP)*, 2001, pp. 182–195.

[98] C. G. Plaxton, Y. Sun, M. Tiwari, and H. Vin, "Reconfigurable resource scheduling," in *Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2006, pp. 93–102.

[99] Rackspace, US Inc., "Rackspace hosting reports second quarter 2012 results," 2012, (accessed on 06/11/2012). [Online]. Available: http://ir.rackspace.com/phoenix.zhtml?c=221673&p=irol-newsArticle&ID=1723357

[100] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "power" struggles: Coordinated multi-level power management for the data center," *SIGARCH Computer Architecture News*, vol. 36, no. 1, pp. 48–59, 2008.

[101] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, "Resource kernels: A resource-centric approach to real-time and multimedia systems," *Readings in multimedia computing and networking, Morgan Kaufmann*, pp. 476–490, 2001.

[102] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level power management for dense blade servers," in *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, 2006, pp. 66–77.

[103] S. Rowe, "Usenet archives," 1992, (accessed on 16/01/2013). [Online]. Available: https://groups.google.com/d/msg/comp.misc/XE25RmO1gIo/-ScxdONYkfkJ

[104] D. G. Sachs, W. Yuan, C. J. Hughes, A. Harris, S. V. Adve, D. L. Jones, R. H. Kravets, and K. Nahrstedt, "GRACE: A hierarchical adaptation framework for saving energy," Computer Science, University of Illinois, Tech. Rep. UIUCDCS-R-2004-2409, 2004.

[105] B. Song, C. Ernemann, and R. Yahyapour, "Parallel computer workload modeling with Markov chains," in *Proceedings of the 11th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2005)*, 2005, pp. 47–62.

[106] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun, "Multi-tiered on-demand resource scheduling for VM-based data center," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009), Shanghai, China*, 2009, pp. 148–155.

[107] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE Transactions on Services Computing (TSC)*, vol. 3, no. 4, pp. 266–278, 2010.

[108] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for Cloud computing," in *Proceedings of the 2008 USENIX Workshop on Power Aware Computing and Systems (HotPower)*, 2008, pp. 1–5.

[109] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 1, pp. 42–55, 1996.

[110] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation using virtual clusters," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 260–267.

[111] J. Stoess, C. Klee, S. Domthera, and F. Bellosa, "Transparent, power-aware migration in virtualized systems," in *Proceedings of GI/ITG Fachgruppentreffen Betriebssysteme*, 2007, pp. 1–6.

[112] J. Stoess, C. Lang, and F. Bellosa, "Energy management for hypervisor-based virtual machines," in *Proceedings of the USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, 2007, pp. 1–14.

[113] C. L. Su, C. Y. Tsui, and A. M. Despain, "Saving power in the control path of embedded processors," *IEEE Design & Test of Computers*, vol. 11, no. 4, pp. 24–31, 1994.

[114] V. Tiwari, P. Ashar, and S. Malik, "Technology mapping for low power," in *Proceedings of the 30nd Annual ACM/IEEE Design Automation Conference (DAC)*, 1993, pp. 74–79.

[115] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: An overview," in *Proceedings of the IEEE Symposium on Low Power Electronics*, 1994, pp. 38–39.

[116] V. Vardhan, D. G. Sachs, W. Yuan, A. F. Harris, S. V. Adve, D. L. Jones, R. H. Kravets, and K. Nahrstedt, "Integrating fine-grained application adaptation with global adaptation for saving energy," in *Proceedings of the International Workshop on Power-Aware Real-Time Computing, Jersey City, NJ*, 2005, pp. 1–14.

[117] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys (CSUR)*, vol. 37, no. 3, pp. 195–237, 2005.

[118] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proceedings of the 2009 USENIX Annual Technical Conference*, 2009, pp. 28–28.

[119] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, 2008, pp. 243–264.

[120] VMware Inc., "vSphere resource management guide," Tech. Rep., 2009.

[121] ——, "VMware distributed power management concepts and use," Tech. Rep., 2010.

[122] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in Clouds: A performance evaluation," in *Proceedings of the 1st International Conference on Cloud Computing (CloudCom)*, 2009, pp. 1–12.

[123] X. Wang and Y. Wang, "Coordinating power control and performance management for virtualized server clusters," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 22, no. 2, pp. 245–259, 2011.

[124] G. Wei, J. Liu, J. Xu, G. Lu, K. Yu, and K. Tian, "The on-going evolutions of power management in Xen," Intel Corporation, Tech. Rep., 2009.

[125] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Mobile Computing*, pp. 449–471, 1996.

[126] A. Weissel and F. Bellosa, "Process cruise control: event-driven clock scaling for dynamic power management," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2002, pp. 238–246.

[127] C. Weng, M. Li, Z. Wang, and X. Lu, "Automatic performance tuning for the virtualized cluster system," in *Proceedings of the 29th International Conference on Distributed Computing Systems (ICDCS)*, 2009, pp. 183–190.

[128] A. Wierman, L. L. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems," in *Proceedings of the 28th Conference on Computer Communications (INFOCOM)*, 2009, pp. 2007–2015.

[129] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation*, 2007, pp. 229–242.

[130] M. Yue, "A simple proof of the inequality FFD (L)< 11/9 OPT (L)+ 1,for all l for the FFD bin-packing algorithm," *Acta Mathematicae Applicatae Sinica (English Series)*, vol. 7, no. 4, pp. 321–331, 1991.

[131] H. Zeng, C. S. Ellis, and A. R. Lebeck, "Experiences in managing energy with ecosystem," *IEEE Pervasive Computing*, vol. 4, no. 1, pp. 62–68, 2005.

[132] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "ECOSystem: Managing energy as a first class operating system resource," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 123–132, 2002.

[133] Q. Zheng and B. Veeravalli, "Utilization-based pricing for power management and profit optimization in data centers," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 72, no. 1, pp. 27–34, 2011.

[134] W. Zheng, R. Bianchini, G. Janakiraman, J. Santos, and Y. Turner, "JustRunIt: Experiment-based management of virtualized data centers," in *Proceedings of the 2009 USENIX Annual Technical Conference*, 2009, pp. 18–33.

[135] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser *et al.*, "1000 Islands: Integrated capacity and workload management for the next generation data center," in *Proceedings of the 5th International Conference on Autonomic Computing (ICAC)*, 2008, pp. 172–181.