



universität
wien

DISSERTATION

Titel der Dissertation

Energy Efficient Heterogeneous
Virtualized Data Centers

Verfasser

Mag.rer.nat. Thomas Treutner

angestrebter akademischer Grad

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2012

Studienkennzahl lt. Studienblatt:	A 786 881
Dissertationsgebiet lt. Studienblatt:	Informatik
Betreuer:	Univ. Prof. Dipl.-Ing. Dr. Helmut Hlavacs

Kurzfassung

Meine Dissertation befasst sich mit software-gesteuerter Steigerung der Energie-Effizienz von Rechenzentren. Deren Anteil am weltweiten Gesamtstrombedarf wurde auf 1-2% geschätzt, mit stark steigender Tendenz. Server verursachen oft innerhalb von 3 Jahren Stromkosten, die die Anschaffungskosten übersteigen. Die Steigerung der Effizienz aller Komponenten eines Rechenzentrums ist daher von hoher ökonomischer und ökologischer Bedeutung. Meine Dissertation befasst sich speziell mit dem effizienten Betrieb der Server. Ein Großteil wird sehr ineffizient genutzt, Auslastungsbereiche von 10-20% sind der Normalfall, bei gleichzeitig hohem Strombedarf. In den letzten Jahren wurde im Bereich der *Green Data Centers* bereits Erhebliches an Forschung geleistet, etwa bei Kühltechniken. Viele Fragestellungen sind jedoch derzeit nur unzureichend oder gar nicht gelöst. Dazu zählt, inwiefern eine virtualisierte und heterogene Server-Infrastruktur möglichst stromsparend betrieben werden kann, ohne dass Dienstqualität und damit Umsatzziele Schaden nehmen. Ein Großteil der bestehenden Arbeiten beschäftigt sich mit homogenen Cluster-Infrastrukturen, deren Rahmenbedingungen nicht annähernd mit Business-Infrastrukturen vergleichbar sind. Hier dürfen verringerte Stromkosten im Allgemeinen nicht durch Umsatzeinbußen zunichte gemacht werden. Insbesondere ist ein automatischer Trade-Off zwischen mehreren Kostenfaktoren, von denen *einer* der Energiebedarf ist, nur unzureichend erforscht.

In meiner Arbeit werden mathematische Modelle und Algorithmen zur Steigerung der Energie-Effizienz von Rechenzentren erforscht und bewertet. Es soll immer nur so viel an stromverbrauchender Hardware online sein, wie zur Bewältigung der momentan anfallenden Arbeitslast notwendig ist. Bei sinkender Arbeitslast wird die Infrastruktur konsolidiert und nicht benötigte Server abgedreht. Bei steigender Arbeitslast werden zusätzliche Server aufgedreht, und die Infrastruktur skaliert. Idealerweise geschieht dies vorausschauend anhand von Prognosen zur Arbeitslastentwicklung. Die Arbeitslast, gekapselt in VMs, wird in beiden Fällen per Live Migration auf andere Server verschoben. Die Frage, welche VM auf welchem Server laufen soll, sodass in Summe möglichst wenig Strom verbraucht wird *und* gewisse Nebenbedingungen nicht verletzt werden (etwa SLAs), ist ein kombinatorisches Optimierungsproblem in mehreren Variablen. Dieses muss regelmäßig neu gelöst werden, da sich etwa der Ressourcenbedarf der VMs ändert. Weiters sind Server hinsichtlich ihrer Ausstattung und ihres Strombedarfs nicht homogen. Aufgrund der Komplexität ist eine exakte Lösung praktisch unmöglich. Eine Heuristik aus verwandten Problemklassen (*vector packing*) wird angepasst, ein meta-heuristischer Ansatz aus der Natur (Genetische Algorithmen) umformuliert. Ein einfach konfigurierbares Kostenmodell wird formuliert, um Energieeinsparungen gegenüber der Dienstqualität abzuwägen. Die Lösungsansätze werden mit Load-Balancing verglichen. Zusätzlich werden die Forecasting-Methoden SARIMA und Holt-Winters evaluiert. Weiters werden Modelle entwickelt, die den negativen Einfluss einer Live Migration auf die Dienstqualität voraussagen können, und Ansätze evaluiert, die diesen Einfluss verringern. Abschließend wird untersucht, inwiefern das Protokollieren des Energieverbrauchs Auswirkungen auf Aspekte der Security und Privacy haben kann.

Abstract

My thesis is about increasing the energy efficiency of data centers by using a management software. It was estimated that world-wide data centers already consume 1-2% of the globally provided electrical energy. Furthermore, a typical server causes higher electricity costs over a 3 year lifespan than the purchase cost. Hence, increasing the energy efficiency of all components found in a data center is of high ecological as well as economic importance. The focus of my thesis is to increase the efficiency of servers in a data center. The vast majority of servers in data centers are underutilized for a significant amount of time, operating regions of 10-20% utilization are common. Still, these servers consume huge amounts of energy. A lot of efforts have been made in the area of *Green Data Centers* during the last years, e.g., regarding cooling efficiency. Nevertheless, there are still many open issues, e.g., operating a virtualized, heterogeneous business infrastructure with the minimum possible power consumption, under the constraint that Quality of Service, and in consequence, revenue are not severely decreased. The majority of existing work is dealing with homogeneous cluster infrastructures, where large assumptions can be made. Especially, an automatic trade-off between competing cost categories, with energy costs being just *one* of them, is insufficiently studied.

In my thesis, I investigate and evaluate mathematical models and algorithms in the context of increasing the energy efficiency of servers in a data center. The amount of online, power consuming resources should at all times be close to the amount of actually required resources. If the workload intensity is decreasing, the infrastructure is consolidated by shutting down servers. If the intensity is rising, the infrastructure is scaled by waking up servers. Ideally, this happens pro-actively by making forecasts about the workload development. Workload is encapsulated in VMs and is live migrated to other servers. The problem of mapping VMs to physical servers in a way that minimizes power consumption, but does not lead to severe Quality of Service violations, is a multi-objective combinatorial optimization problem. It has to be solved frequently as the VMs' resource demands are usually dynamic. Further, servers are not homogeneous regarding their performance and power consumption. Due to the computational complexity, exact solutions are practically intractable. A greedy heuristic stemming from the problem of vector packing and a meta-heuristic genetic algorithm are investigated and evaluated. A configurable cost model is created in order to trade-off energy cost savings with QoS violations. The base for comparison is load balancing. Additionally, the forecasting methods SARIMA and Holt-Winters are evaluated. Further, models able to predict the negative impact of live migration on QoS are developed, and approaches to decrease this impact are investigated. Finally, an examination is carried out regarding the possible consequences of collecting and storing energy consumption data of servers on security and privacy.

Contents

1	Introduction	1
1.1	Motivation and Problem Statement	1
1.2	Synopsis	3
1.3	Summary of Publications	4
1.4	Acknowledgments	5
I	Background	7
2	Energy Consumption of Data Centers	9
2.1	Energy Consumption Structure	10
2.1.1	Servers	11
2.1.2	Storage	13
2.1.3	Communication	14
2.2	Methodology and Benchmarks	14
2.2.1	Energy Proportionality	14
2.2.2	Power Usage Effectiveness	15
2.2.3	SPECpower_ssj2008	18
2.3	Green Data Centers	20
3	System Virtualization	23
3.1	Introduction	23
3.2	History	24
3.3	Technologies and Implementations	27
3.3.1	Full Virtualization and Binary Translation	27
3.3.2	Paravirtualization	28
3.3.3	Hardware-assisted Virtualization	28
3.3.4	Operating System Level Virtualization	29
3.3.5	Hypervisor Based Virtualization	30
3.4	Key Benefits	30
3.4.1	Dynamic Resource Allocation	30
3.4.2	Live Migration	32
3.5	Implementations of Virtual Machine Live Migration	32
3.5.1	Non-Live Stop-and-Copy Migration	33

3.5.2	Iterative Pre-Copy Live Migration	34
3.5.3	Post-Copy Live Migration	35
3.5.4	Live Migration over Wide Area Networks	36
3.5.5	Full System Trace and Replay	36
3.5.6	Increasing Live Migration Efficiency	37
4	Energy Efficient Dynamic Consolidation	39
4.1	Static Consolidation	39
4.2	Workload and Utilization Patterns	40
4.3	Dynamic Consolidation	43
4.4	Workload and Resource Requirement Prediction	44
4.5	Energy Efficient Mapping of Virtual Machines to Physical Servers	45
4.5.1	Knapsack Problem	46
4.5.2	Bin Packing Problem	47
4.5.3	Virtual Machine Mapping Problem	47
4.6	Quality of Service	51
4.6.1	Energy Costs, Quality of Service and Revenue	51
4.6.2	Quality of Service During Live Migration	52
II	New Approaches in Energy Efficient Data Centers	55
5	MetaVisor - Cost Model Driven Dynamic Consolidation	57
5.1	Overview of Chapter	57
5.2	Introduction and Related Work	57
5.3	Open Issues and Own Contribution	61
5.4	Scenario, Assumptions and Constraints	63
5.5	Prototype Architecture	65
5.6	Cost Model	69
5.6.1	Energy Consumption	69
5.6.2	Live Migration	70
5.6.3	Server Boots and Shutdowns	70
5.6.4	Server Overload and Utilization Thresholds	70
5.6.5	Utilization Variance	72
5.7	Virtual Machine Mapping Algorithms	73
5.7.1	Balanced First Fit Heuristic	73
5.7.2	Genetic Algorithm	76
5.8	Utilization Trace Data	78
5.9	VM workload forecasting	79
5.10	Evaluation	83
5.11	Results	86
5.11.1	Interval Length and Forecasting	86

5.11.2	Overload Cost Parameter Influence	90
5.11.3	Migration Cost Parameter Influence	92
5.11.4	Energy Cost Parameter Influence	93
5.11.5	VM Consolidation Inertia	93
5.11.6	RM Idle Timeout	94
5.11.7	Performance Evaluation	96
5.12	Discussion	99
5.13	Future Work	100
6	Predicting Quality of Service During Iterative Pre-Copy Live Migration	101
6.1	Overview of Chapter	101
6.2	Introduction and Related Work	101
6.3	Open Issues and Own Contribution	102
6.4	Experimental Setup	103
6.4.1	Infrastructure	103
6.4.2	Methodology	104
6.5	Data Overview	105
6.6	Data Analysis	113
6.6.1	Model Quality	113
6.7	Model Selection	115
6.7.1	Akaike Information Criterion (AIC)	115
6.7.2	LEAPS	115
6.8	Qualitative Results	115
6.8.1	Low Load	116
6.8.2	Medium Load	117
6.8.3	High Load	117
6.9	Regression Analysis	118
6.9.1	Results for AIC	118
6.9.2	Results for LEAPS	121
6.10	Discussion	124
6.11	Future Work	125
7	Service Level Management for Iterative Pre-Copy Live Migration	127
7.1	Overview of Chapter	127
7.2	Introduction and Related Work	127
7.3	Open Issues and Own Contribution	128
7.4	Experimental Setup	129
7.4.1	Infrastructure	129
7.4.2	Methodology	130
7.5	Results	131
7.5.1	Unpatched KVM	131
7.5.2	Bandwidth Increased KVM	133

7.5.3	Bandwidth Increased KVM, Strict Downtime	133
7.5.4	Xenalike KVM	134
7.5.5	Service Level Managed KVM	135
7.5.6	Comparison	138
7.6	Discussion	140
7.7	Future Work	141
8	Security and Privacy of Energy Efficient Data Centers	143
8.1	Overview of Chapter	143
8.2	Introduction and Related Work	144
8.3	Open Issues and Own Contribution	145
8.4	Experimental Details	146
8.4.1	Experimental Infrastructure in Vienna	147
8.4.2	Experimental Infrastructure in Lyon	147
8.5	Evaluation	148
8.6	Results	150
8.6.1	Results of Experiments in Vienna	150
8.6.2	Results of Experiments in Lyon	154
8.7	Discussion	159
8.8	Future Work	161
9	Conclusions and Future Work	163
	Bibliography	167
A	MetaVisor and MetaSim GUI	181
B	Basic Configuration for the MetaSim Simulation.	183
C	Patch to Introduce Stop Conditions to the KVM Live Migration.	189
	List of Figures	191
	List of Tables	195

Chapter 1

Introduction

1.1 Motivation and Problem Statement

A vast majority of servers in classical data centers of all scales are underutilized for a significant amount of time. These servers operate at a very low rate of efficiency [BH07] and consume huge amounts of energy. Since the EPA¹ report to the congress on server and data center energy efficiency [ENE07] in 2007, these topics have gained more and more public and even official interest, visible as the *Green IT* trend. It is estimated by the Uptime Institute [Bri07] that beginning with 2009, a typical server causes higher electricity costs over a three year lifespan than the purchase cost.

According to a recent report by Koomey [Koo11] about the growth in both worldwide and US data center electricity use, in 2010, data centers have consumed approximately 1.1% to 1.5% of the globally produced electricity. In the US, the estimation of data center power consumption is 1.7% to 2.2% of the country's total electricity production. Interestingly, the worldwide electricity consumption of data centers has grown about 56% from 2005 to 2010, compared to a doubling from 2000 to 2005. The most probable reasons for this slower growth are a generally increased interest in data center energy efficiency caused by the first reports about data center energy consumption, the slower growth and even net reduction in the total number of installed servers due to virtualization, and the 2008 financial crisis with its effects on worldwide economic development.

Nevertheless, it is estimated that worldwide data centers have drained an average electrical power of 23.2 to 31.0 GW in 2010. To visualize this abstract number, a reduction of a single percent could make it possible to completely shut down a typical reactor block of the Fukushima-Daiichi nuclear power plant as it is no longer required. Even if a net reduction may not be realistically possible in the short run, every percent of growth that can be avoided by efforts in energy efficiency is a reactor block that may not be needed to be built.

As mentioned, concerns regarding energy consumption of data centers have already led to significant efforts in the last decade. Briefly, a data center's energy consumption can be divided into direct equipment (servers, storage, networking, etc.) and indirect equipment (cooling, batteries, etc.) consumption. For a more detailed discussion, see Chapter 2. The scope of this thesis is the reduction of server energy consumption, hence reducing a

¹Environmental Protection Agency, USA

major part of direct equipment consumption. Furthermore, decreasing direct equipment consumption generally leads to a reduction of indirect equipment consumption, as less heat loss requires less cooling. During the last decade, one of the most important approaches to decrease direct equipment consumption was *static server consolidation* using system virtualization. The general idea is to consolidate a set of idle or badly utilized servers to a smaller set of better utilized servers. There are many different implementations of virtualization, but in general, computing resources are virtualized to gain higher efficiency and flexibility.

When using system virtualization, virtual machines (VMs) are executed on physical servers (physical or real machines, RMs). The VMs are independent from each other and each VM runs its own operating system instance. A VM typically consists of one or more virtual CPUs (vCPUs), an amount of virtual main memory, one or more virtual disk images, virtual NICs, etc. VMs are controlled by a thin layer of management software, usually called hypervisor or Virtual Machine Monitor (VMM). The hypervisor's purpose is to arbitrate physical resources (e.g., CPU time slices and main memory) between the VMs. Xen [CFH⁺05], KVM [KKL⁺07b] and VMware vSphere² are typical examples for system virtualization products found in data centers. For a more detailed discussion, see Chapter 3.

However, business infrastructures typically have to bear workload with high variability and strong periodic components. For a detailed discussion, refer to Chapter 4. In such a scenario, static reservation of resources, although in a virtualized approach, is a dead end regarding efficiency. Virtualization can not change the fact that provisioning resources for day hours is a waste of resources during the night hours. On the contrary, provisioning resources for night hours will typically result in catastrophic service levels and annoyed users during the day.

The effort of my thesis is to investigate the potential energy savings of *dynamic consolidation*. Here, a management software system automatically, adaptively and dynamically manages resource provisioning with regard to the expected workload situation and places the workload (encapsulated in VMs) in a way that energy consumption is minimized, subject to avoiding both under- and over-provisioning. Energy efficiency is generally not worth the effort if quality metrics (e.g., QoS, QoE) are severely decreased. Hence, the amount of online, power consuming resources should at all times be close to the amount of actually required resources. Ideally, the resource provisioning happens pro-actively by making forecasts about the workload development and by scaling the infrastructure to the near future demand. This way, currently unrequired servers can be shut down to save energy, and the average utilization of the remaining servers is increased to a level of efficient operation. Here, it is important to have a target utilization in mind. Werner Vogels, CTO and Vice President of Amazon, has mentioned [Vog08] that achieving 40% utilization is a “major success for workloads that are not strictly CPU-bound” [Vog08], and achieving 50% is the “holy grail” [Vog08] of server consolidation. Additionally, it is crucial to make high quality forecasts, and to consider the management overhead, e.g., due to VM live migration.

²<http://www.vmware.com/products/datacenter-virtualization/vsphere/overview.html>, last visited on 31.8.2012

1.2 Synopsis

This thesis is structured into two Parts, each consisting of several Chapters.

In Part I, the background and status quo of approaches to energy efficient data centers are summarized and the key technologies necessary for the work at hand are discussed. In Chapter 2, an overview on energy consumption within data centers is given. Further, the most important advances and open issues in energy efficiency, as well as metrics available to measure energy efficiency are discussed. In Chapter 3, the concept of system virtualization and its relevance regarding server consolidation are briefly introduced. Especially, Section 3.5 gives an overview of different implementations of virtual machine live migration. Chapter 4 describes the approach of dynamically consolidating an infrastructure to the current or predicted workload situation. A short discussion of workload patterns commonly found in data centers sheltering business infrastructures is given, based on real utilization traces thankfully provided by the University of Vienna Central IT department. The underlying multi-objective combinatorial optimization problem of mapping VMs to physical servers, aiming at reducing power consumption, is described. Especially, the key aspects and open issues of dynamic server consolidation for heterogeneous infrastructures are summarized and possible effects on Quality of Service are discussed.

Part II contains presentations and results of the research efforts I have carried out during the work on this thesis. Chapter 5 presents results for a cost model driven dynamic consolidation approach using a meta-heuristic genetic algorithm, compared with a greedy heuristic and load-balancing. SARIMA and Holt-Winters forecasting methods are evaluated and compared with more frequent control loop iterations. Further, several investigations, e.g., regarding the influence of parameters, the runtime on different hardware platforms and the degree of parallelization are carried out. Chapter 6 presents models able to predict the negative impact of live migration on QoS and experiments for several workload scenarios. Chapter 7 presents both a low and a high level approach to decrease this impact. Chapter 8 presents experiments and evaluation results regarding security and privacy implications of energy efficient infrastructures that collect and store energy consumption data of servers. Finally, Chapter 9 draws conclusions and discusses interesting options regarding future work.

1.3 Summary of Publications

My interest in researching energy efficient computing infrastructures was initiated by my diploma thesis *Virtual Home Environments* in 2008. Research results were disseminated as follows:

- *H. Hlavacs, R. Weidlich, T. Treutner, Energy Saving in Future Home Environments, 2nd Home Networking Conference at IFIP Wireless Days, Dubai, Nov. 24-27, 2008.*
- *A. Berl, H. de Meer, H. Hlavacs and T. Treutner, Virtualization in Future Home Environments, IEEE Communications Magazine 47-12 (2009), pp.62-67.*
- *H. Hlavacs, R. Weidlich, T. Treutner, Load-Dependent Energy Saving in Future Home Environments, to appear in Telecommunication Systems 51-4 (Dec. 2012), Springer, 2012.*
- *H. Hlavacs, R. Weidlich, T. Treutner, Energy Efficient Peer-to-Peer File Sharing, accepted at Journal of Supercomputing (Springer), special issue on Energy-efficient High-Performance Parallel and Distributed Computing.*

During the work on this thesis, research results have been continuously disseminated to the scientific community. The cost model driven dynamic consolidation using meta-heuristic genetic algorithms described in Chapter 5 was disseminated as

- *H. Hlavacs, T. Treutner, Genetic Algorithms for Energy Efficient Virtualized Data Centers, accepted at 6th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud (SVM12), Las Vegas, Oct. 26, 2012.*

Preliminary results regarding models allowing QoS prediction during live migration were disseminated as

- *H. Hlavacs, T. Treutner, Migration of Virtual Machines: Performance and QoS, to appear in COST Action IC0804 proceedings 2011, ISBN : 978-2-917490-18-1 - EAN : 9782917490181.*

Final and detailed results are presented in Chapter 6 and were disseminated as

- *H. Hlavacs, T. Treutner, Predicting Web Service Levels During VM Live Migrations, 5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud (SVM11), 24th October, 2011, Paris, France.*

Research results regarding service level management for iterative pre-copy live migration presented in Chapter 7 were disseminated as

- *T. Treutner, H. Hlavacs, Service Level Management for Iterative Pre-Copy Live Migration, accepted at the 8th International Conference on Network and Service Management (CNSM 2012), 22-26 October, Las Vegas, USA.*

Security and privacy implications of energy efficient infrastructures presented in Chapter 8 were disseminated as

- *H. Hlavacs, T. Treutner, J.-P. Gelas, L. Lefèvre and A.-C. Orgerie, Energy consumption side-channel attack at Virtual Machines in a Cloud, accepted at International Conference on Cloud and Green Computing (CGC 2011), Sydney, Australia, Dec. 12-14 2011.*

Preliminary results were disseminated as

- *H. Hlavacs, T. Treutner, Privacy Risks for Virtual Machines in a Cloud, to appear in COST Action IC0804 proceedings 2011, ISBN : 978-2-917490-18-1 - EAN : 9782917490181 and*
- *M. Niedermayer, N. Ali, H. de Meer, H. Hlavacs, T. Treutner, L. Lefèvre, J.-P. Gelas and I. Koutsopoulos, Energy monitoring and its impact on individual user privacy, 4th Future Internet Cluster Workshop, 2011, Budapest.*

1.4 Acknowledgments

First and foremost, I want to thank my thesis advisor Prof. Helmut Hlavacs for his supervision and guidance while introducing me to research. I am especially thankful for his willingness and flexibility to let me develop and pursue my own ideas, yet offering advice and support.

Then I would like to express special gratitude to Ralph Staudigl and Ulf Busch from the University of Vienna Central IT Department for their efforts to make this research possible by allowing me to examine real VM utilization traces.

I want to thank Laurent Lefèvre, Jean-Patrick Gelas, Anne-Cécile Orgerie and Olivier Mornard from the INRIA RESO/LIP Laboratory at the Ecole Normale Supérieure de Lyon most sincerely for inviting me to Lyon. They have given me a warm and uncomplicated welcome and an environment to be highly productive while having me as their guest in July 2011.

I am also grateful for the many friends, acquaintances and research colleagues that had an open ear for discussions on the topic or just about difficult situations and moments I believe everyone experiences during the work on and creation of a thesis.

I want to thank my parents for their interest in my academic education and also for helping me to finance such expensive recreational hobbies as owning a motorcycle sometimes can be.

Part I

Background

Chapter 2

Energy Consumption of Data Centers

In general, power consumption of data centers can only be estimated. Data center operators are often very secretive about their power demand, energy costs and energy efficiency achievements. Detailed information about a data center's power demand could be used to estimate the infrastructure size. Knowing by how much a company is expanding its infrastructure is an important part of estimating the growth of a company, its customer base, revenue, etc. To gain competitive advantage, major companies invest significant efforts into increasing energy efficiency. Thus, such knowledge was disseminated seldom in the past and only rough estimates can be made.

According to a recent report by Koomey [Koo11] about the growth in both worldwide and US data center electricity use, in 2010, data centers have consumed approximately 1.1% to 1.5% of the globally provided electricity. These percentages represent an energy consumption of approximately 203.4 TWh to 271.8 TWh. In the year 2010, US data centers have consumed 1.7% to 2.2% of the country's total provided electricity.

As a consequence, it is estimated that worldwide data centers have drained an average electrical power of 27 GW in 2010. To visualize this abstract number, even a small reduction in the order of percents could make it possible to completely shut down a typical reactor block of the Fukushima-Daiichi¹ nuclear power plant as it is no longer required. Even if a net reduction may not be realistically possible in the short run due to higher computing demand, growth that can be avoided by efforts in energy efficiency is a reactor block that may not be needed to be built.

Of course, data center energy consumption is not only an ecological issue, but also an economical. With a total energy consumption of approximately 203.4 TWh to 271.8 TWh, and a very moderate price assumption of 10 Euro Cents per kWh for bulk purchasers, the total energy bill sums up to approximately 20 to 27 Billion Euro.

However, the price of energy is generally not fixed, but varies from location and time. Qureshi et al. [QWB⁺09] have analyzed historical energy prices from [Pla09] for six different US regions over a time span of more than three years. As a result, the day-ahead price

¹<http://pris.iaea.org/PRIS/CountryStatistics/ReactorDetails.aspx?current=377>, last visited on 13.6.2012

per MWh varies from 30 \$ to 150 \$, with typical prices from 50 \$ to 100 \$. The total energy consumption of data centers in the US was estimated by Koomey [Koo11] to 67.1 TWh to 85.6 TWh for 2010. This results in an estimated total yearly energy bill of 3.4 to 8.6 Billion \$ for the US alone.

Furthermore, the majority of small and medium enterprises do not build and operate their own data centers, but use co-location and rack housing services. Here, energy is mostly sold in packages at fixed, but higher prices, e.g., 30 Euro Cents per kWh².

2.1 Energy Consumption Structure

In order to identify potential savings, it is important to have information about how energy consumption is structured within a data center. The first comprehensive study available was carried out by Koomey [Koo08], and is an extension of [Koo07], where only server consumption was examined. It is based on shipment numbers from IDC and typical power demands of current servers. In 2011, an update was published by Koomey [Koo11] containing estimates for 2010. A compilation of these estimates is shown in Table 2.1.

Category	Component	2000	2005	2010
Direct Consumption	Servers	41%	40%	38%
	Storage	4%	5%	9%
	Communications	5%	5%	6%
Indirect Consumption	Infrastructure	50%	50%	47%
Totals	Total [TWh]	70.8	152.5	237.65
	% of World Production	0.5%	1%	1.3%

Table 2.1: Estimated worldwide energy consumption of data centers , compiled from [Koo08] and [Koo11].

In Table 2.1, estimates for the worldwide energy consumption of data centers and the inner structure of data center energy consumption are shown. Within ten years, the total energy consumed by data centers has more than tripled. Interestingly, the worldwide electricity consumption of data centers has grown about 56% from 2005 to 2010, compared to a doubling from 2000 to 2005 [Koo11]. The most probable reasons for this slower growth are a generally increased interest in data center energy efficiency caused by the first reports about data center energy consumption, the slower growth and even net reduction in the total number of installed servers due to virtualization, and the 2008 financial crisis with its effects on worldwide economic development.

²<http://www.ipax.at/rackhousinginterxion.html>,
[rackhousingipone.html](http://www.ipax.at/rackhousingipone.html), last visited on 14.6.2012

<http://www.ipax.at/>

Typically, the energy consumption within a data center is structured into *direct* and *indirect* energy consumption. Only devices directly related to the IT workload are counted as IT equipment and are contributing to the *direct* energy consumption. The most important components within this category are servers, storage and network devices. Everything else required for running an IT infrastructure contributes to the *indirect* energy consumption, e.g., cooling and ventilation, power distribution and supply units, batteries, uninterruptible power supplies, etc. In the following, the inner structure of data center energy consumption is briefly discussed based on Table 2.1 in the respective Sections. The aim of this thesis is to reduce the amount of energy consumed by servers, hence other components are discussed as far as relevant for the work at hand.

2.1.1 Servers

As visible in Table 2.1, servers have been responsible for 38% of the total worldwide data center energy consumption in 2010. Compared to 2000, we can see a small decrease in the relative amount. Nevertheless, the absolute amount has approximately tripled. Of course, servers typically consist of several distinctive components, each having a different power consumption profile. A comprehensive study regarding potential energy savings is done in [Car11]. The general problem relevant for this thesis is that the current generation of computers is consuming a significant amount of power even when being idle. Ideally, computers should be “energy proportional”[BH07], a concept discussed in Section 2.2.1.

Name	Idle Consumption	Peak Consumption
Dell PowerEdge 1850	230 W	250 W
Dell PowerEdge 1950	250 W	340 W
Sun Fire V60x	190 W	205 W
Sun Fire X2100	190 W	205 W
Sun Fire X2200	200 W	220 W
HP Compaq DL360	160 W	180 W
HP Integrity rx2600	190 W	340 W

Table 2.2: Idle and Peak Power Consumption of typical legacy servers, compiled from [DHKC09].

In Table 2.2, idle and peak power consumption of several typical legacy servers as plotted in [DHKC09] is shown. These servers are mostly powered by single or dual core processors, and are therefore nowadays to be considered as legacy systems. We can clearly see that the gap between a servers idle and peak power consumption is generally very small. The only exception, the HP Integrity rx2600, is an Itanium driven server.

Since 2008, more data about server power consumption profiles is available due to the introduction of the SPECpower_ssj2008³ benchmark, which is briefly discussed in Section 2.2.3.

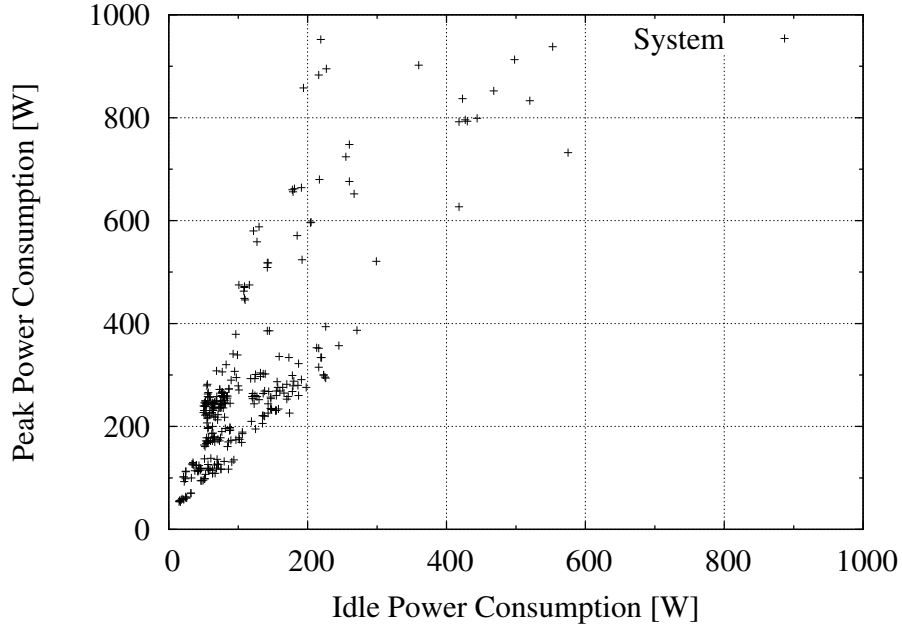


Figure 2.1: Idle vs. peak power consumption of single server systems participating in the SPECpower_ssj2008 benchmark.

Figure 2.1 shows the correlation between idle and peak power consumption of single server systems participating⁴ in the SPECpower_ssj2008 benchmark. We can clearly see technical progress compared to Table 2.2. E.g., there are servers that have an idle consumption of 200 W, and a peak consumption of 600 W. However, the majority of servers still consumes 50 W to 200 W when being idle. The reason for this is that most components of a server consume lots of power although they are not utilized.

The most important exception here is the CPU. Fan et al. [FWB07] have studied the dynamic power range of CPUs, with the result that there is a strong linear correlation between CPU utilization and its power consumption. This allows to use Equation 2.1 as a good approximation for calculating the power consumption of a system depending on the CPU utilization. The term $P_{peak} - P_{idle}$ is commonly denoted as *dynamic range*, as its impact is depending on the varying utilization.

$$P_{CPU} = P(U_{CPU}) = P_{idle} + (P_{peak} - P_{idle}) \cdot U_{CPU} \quad (2.1)$$

³http://www.spec.org/power_ssj2008/, last visited on 14.6.2012

⁴http://www.spec.org/cgi-bin/osgresults?conf=power_ssj2008;op=dump;format=csvdump, last visited on 14.6.2012

Technically, the power consumption of a CPU is the sum of dynamic and static power [KAB⁺03], and typically denoted by the approximation

$$P_{CPU} = ACV^2f + VI_{leak} \quad (2.2)$$

where A is the fraction of gates actively switching, C the capacitance of all gates, V the operating voltage, f the CPU frequency and I_{leak} leakage current. V^2 is contributing heavily to the consumption, which explains DVFS (Dynamic Voltage and Frequency Scaling) efforts to save power. There are technical constraints regarding the achievable frequencies given a specific voltage. Hence, the frequency must be scaled down in coordination with the voltage.

Modern CPUs offer a multitude of low power and sleep states, as discussed in [Car11]. This way, P_{idle} can be minimized to a point where an idle CPU is consuming only very little power [CAB⁺07]. Technologies like Intel SpeedStep and AMD PowerNow were originally developed for mobile devices like laptops. Here, power consumption was always critical as it reduces battery runtime and increases heat emission. Over the years, these mechanisms have been adopted for desktop and server computers too, with regular extensions, improvements and refinements of available power saving states. However, Pierson and Casanova have shown [PC11] that, regarding an optimal job placement in a Grid scenario, a large number of power states is exploitable only for smaller infrastructures. For bigger infrastructures, only one intermediate level between on and off is required.

2.1.2 Storage

Storage devices have consumed 9% of the total worldwide electricity used in data centers in 2010. This portion is more than a doubling compared to 2000, as visible in Table 2.1. In absolute numbers, there is a factor of 6 between the year 2000 and 2010. The most probable reason is the boom of data intensive services, e.g., internet search, online bidding platforms, social networking sites or multimedia streaming. Additionally, Cloud services storing private customer data are becoming very popular since the last years.

In virtualized data center infrastructures, storage devices are typically used to shelter virtual machine disk images. When using virtual machine live migration, it is inevitable to host the disk images on a SAN or NAS, reachable from both servers involved in the live migration. Although disk drives indeed offer power saving mechanisms, their power consumption depends on the workload [MPL09], and there are efforts regarding energy-proportional storage systems [VKUR10] the situation in data centers is generally a different one. Here, the disk drives are just one of several components comprising a storage device. Further, there are higher demands regarding availability, so several disk will be used in RAID mode, and have to be synchronized frequently. Thus, disks in a data center will only seldom be able to go into a low power mode. In the probable case of multiple storage devices synchronized over a storage area network, such opportunities to save power will be even rarer. Hence, data center operators currently have to rely on disk drives that consume

less energy even when highly utilized.

Solid state drives (SSDs) are the outcome of recent advances in flash memory technology, and offer significantly lower power consumption. However, their price per storage capacity is still many times higher than for common disk drives.

2.1.3 Communication

Communication devices, most importantly switches, routers and hardware firewalls, have consumed 6% of the total worldwide electricity used in data centers in 2010. Compared to 2000, this is only a small increase in the relative amount as visible in Table 2.1. In absolute numbers, the consumption of communication devices has tripled from 2000 to 2010. It was measured in [HDP09] that these devices have a highly static power consumption profile, with only marginal influence of the workload intensity. This allows the approximation of a static power consumption. Ananthanarayanan and Katz [AK08] have proposed to decrease the power consumption of switches by dynamically shutting down ports, with a guaranteed and bounded latency.

2.2 Methodology and Benchmarks

2.2.1 Energy Proportionality

Barroso and Hölzle [BH07] have presented real utilization data from over 5000 Google servers during half a year. The astonishing result shown in Figure 2.2 is that, despite Google being an industry leader in energy efficient data center design and operation as mentioned in Section 2.2.2, the typical operating region is between 15% and 55% utilization. Compared to infrastructure operators that do not care too much about energy efficiency, with resulting utilizations of 10% or less, this may seem as a high level of efficiency. However, there is still a high amount of energy wasted.

The reason is that computers are currently not "energy-proportional" [BH07], meaning their energy consumption is not strongly correlated with their utilization for all utilization levels. Instead, typical computers consume half their peak consumption when they are idle and do no work, so the dynamic range $P_{peak} - P_{idle}$ of the overall system is limited to approximately 50%. The high idle consumption is a consequence of power saving technologies that primarily aim at the CPU, e.g., DVFS and clock gating. Other components like motherboard, memory modules, extension cards etc. have been mostly ignored in the past and therefore have a very narrow dynamic consumption range, or even no significant at all.

Barroso and Hölzle [BH07] take a simple but still useful approach in defining energy efficiency. They divide the utilization by its corresponding power consumption as in Equation 2.3.

$$Eff = \frac{U}{P(U)} \quad (2.3)$$

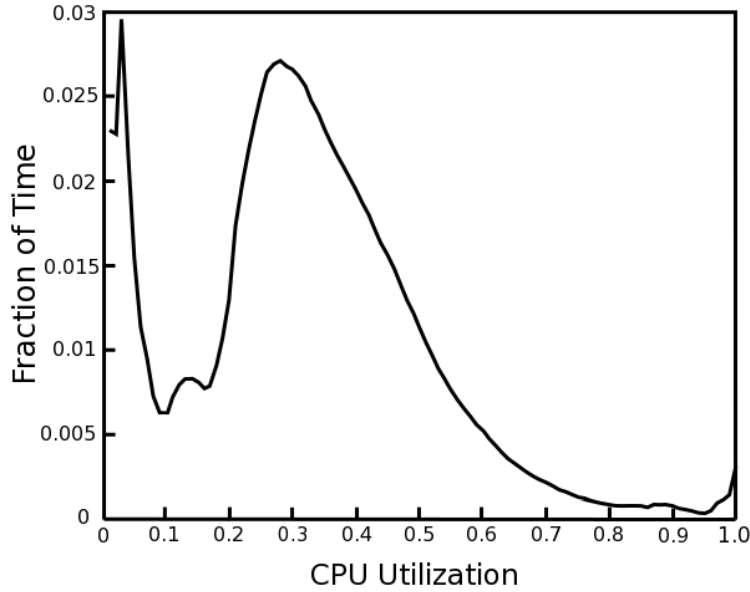


Figure 2.2: Google server utilization as in [BH07]. The typical operating region is between 15% and 55% utilization.

It is clearly visible in Figure 2.3 that current computers with their high and static idle consumption achieve only low efficiency when considering their typical levels of utilization.

The long-term approach to achieve higher efficiency is easy to describe, but much harder and tedious to realize. *All* components of computers must be designed *energy-proportional*, with wide dynamic power ranges and a strong correlation between utilization and power consumption. This would provide high rates of energy efficiency in typical operating regions, as visible in Figure 2.4. To achieve this, a shift of focus is necessary from the CPU, where highly sophisticated power saving mechanisms have been developed in the last years, to all other computer components. Until such a new generation of hardware is available, other approaches must be investigated. An obvious way to cope with imperfect hardware is to create software systems that offer remedy.

This thesis aims at achieving higher efficiency on current hardware by shifting the typical operating region to the maximum possible. Of course, this must be done under consideration of constraints, most importantly, quality of service. Shifting the operating region to 40% to 60% as shown in Figure 2.5 would result in significantly higher energy efficiency.

2.2.2 Power Usage Effectiveness

As mentioned above, data center energy consumption is not only caused by direct IT equipment such as servers, but also by infrastructure components like cooling. An early metric to compare data centers with regard to their energy efficiency was introduced by the Green Grid consortium [Gri07], denoted as Power Usage Effectiveness (PUE) and defined as

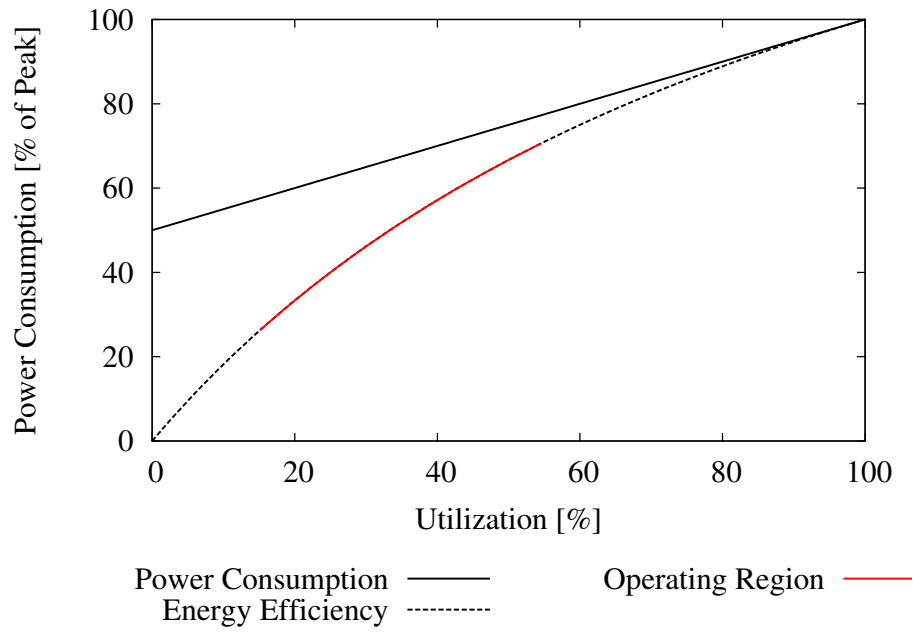


Figure 2.3: Efficiency profile of a current server, lacking energy proportionality as in [BH07].

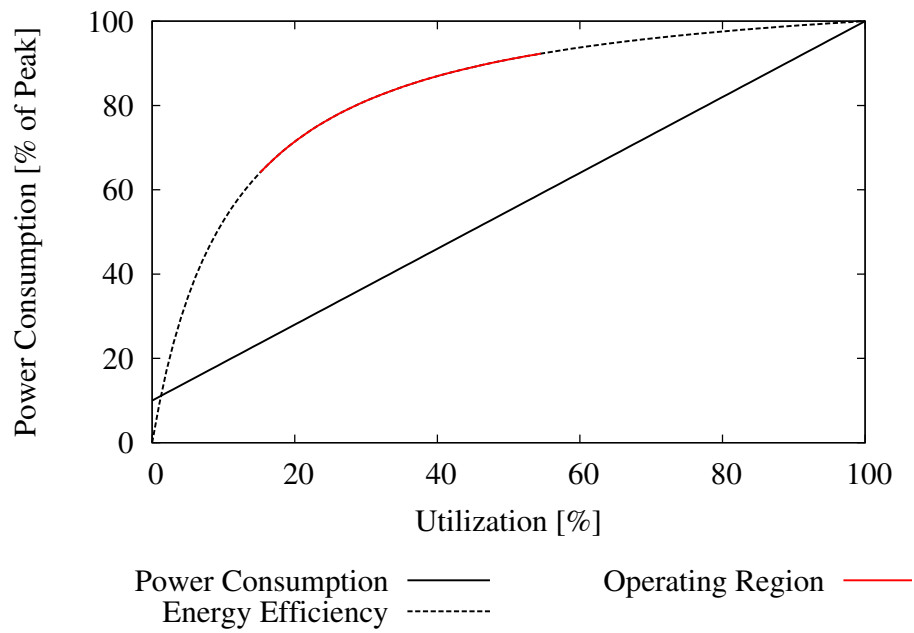


Figure 2.4: A hypothetical, more energy-proportional server as in [BH07].

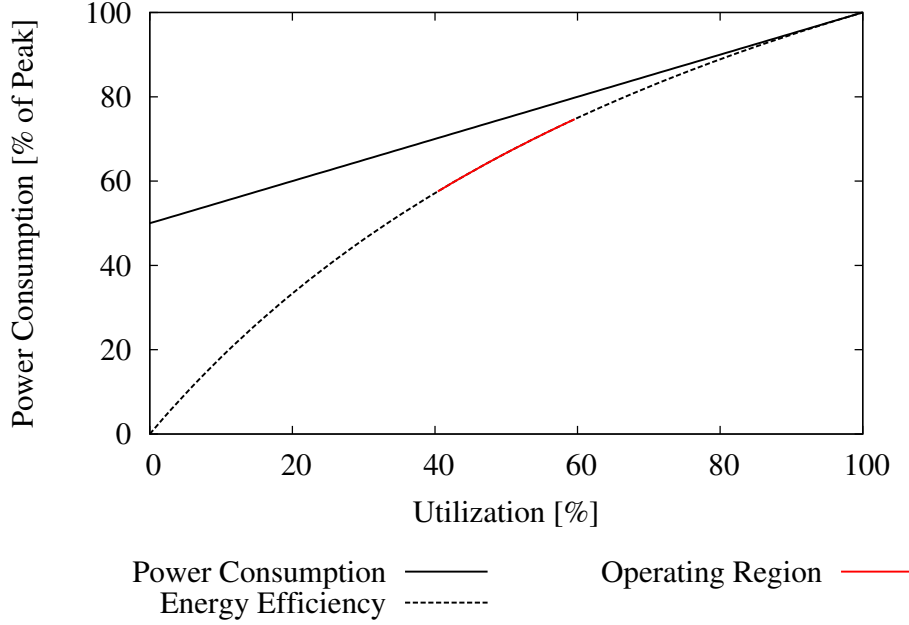


Figure 2.5: Shifting the operating region to 40-60% would provide software-managed energy efficiency.

$$PUE = \frac{TotalFacilityPower}{ITEquipmentPower} \quad (2.4)$$

Only devices directly related to the IT workload are counted as IT equipment, e.g., servers, storage and switches. Everything else required for running an IT infrastructure contributes to the total facility power, e.g., cooling, power distribution, battery, UPS, etc. Hence, the PUE value gives the ratio between direct computing device consumption and the raw data center consumption including all overheads. E.g., if the PUE of a particular data center is 2.5, and a new server demands 400 W, the data center power consumption will be increased by 1000 W including overheads, here 600 W.

Of course, realistic values for the PUE are of high importance here. According to a report by Greenberg et al. [GMT⁺06] which investigates 22 data centers and even predates the formalization of the PUE, the ratio between total data center consumption and server power consumption was empirically found to range from 1.33 to 3. In 2007, the EPA report [ENE07] estimated an average PUE of 2.0 for the period 2000 to 2006. This estimation is consistent with Table 2.1, where direct and indirect consumption amounts are approximately equal, yielding a PUE of 2. The EPA has also described different future scenarios, ranging from a PUE of 1.9 for a "current trends" scenario to 1.2 for a "state-of-the-art" scenario.

Since then, Google has officially announced PUE values for their data centers that are even lower than the most optimistic scenario examined by the EPA. On average, Google achieves a PUE of 1.16, with a particular data center achieving a PUE of 1.09, extremely

close⁵ to the ideal PUE value of 1. Of course, considerable amounts of both knowledge and resources are required to gain such high level of power usage effectiveness. However, as there is a big interest regarding knowledge and best-practices by data center operators, energy costs are expected to rise further, and some of the best-practices like hot/cold-isles can be implemented quite easily, we can fairly assume the average PUE to be less than 2.0 in the future.

Nevertheless, methods for decreasing *direct* IT equipment power consumption are not generally suitable for the masses yet. The vast mass of data center operators have to purchase off-the-shelf servers and need to rely on intelligent and energy saving hardware design to be provided by the manufacturer. It is not realistic to generalize the possibilities companies like Google indeed have, e.g., deploying servers with specifically designed mainboards and circuitry usable without the typical power supply unit and sheltering an onboard battery instead of a rack-UPS to minimize transformation losses. On the other hand, e.g., rearranging servers to implement hot/cold-isles reducing cooling power consumption and therefore *overhead indirect* consumption is a realistic short-term possibility.

It is of high importance to realize the PUE metric only addresses the relation between *direct equipment* power consumption and *indirect overhead* power consumption of data centers, aiming at minimizing the *indirect overhead* power consumption. It does not address several aspects important in the context of this thesis. Most importantly, the PUE metric gives no information about the amount of energy required per workload unit handled by the data center and the level of proportionality between workload intensity and computing equipment power consumption. The aim of this work is to increase this proportionality, hence decreasing *direct equipment* power consumption when it is possible due to the current level of utilization.

As a consequence, the PUE metric is relevant in the context of this work to estimate additional, indirect energy savings. For every kWh of *direct equipment* energy consumption saved due to the effort of the work at hand, an additional kWh of *indirect overhead* consumption can be saved, assuming a PUE of 2.0 for an average data center.

2.2.3 SPECpower_ssj2008

As described in Section 2.2.1, current computing hardware lacks energy proportionality, meaning that a completely idle computer still consumes approximately 50% of its peak consumption. In combination with the observation that most servers are badly utilized most of the time, this explains in a nutshell why there is a big potential of decreasing *direct equipment* power consumption of data centers.

An important benchmark addressing the level of proportionality is SPECpower_ssj2008⁶, benchmarking a server with a Java workload at load levels from 100% to 0% in steps of

⁵<http://www.google.com/about/datacenters/inside/efficiency/power-usage.html>, last visited on 3.9.2012

⁶http://www.spec.org/power_ssj2008/, last visited on 31.8.2012

10%, relative to the server's maximum transaction throughput found in a calibration phase [GKL08]. For each load graduation i , the corresponding throughput ssj_ops_i and average power consumption $\bar{W}_i$ are measured. The primary metric is then

$$specpoints = \sum_i \frac{ssj_ops_i}{\bar{W}_i}, \quad (2.5)$$

giving the overall ratio of throughput per power.

Ranganathan et al. [RRM09] discuss several further energy-efficiency metrics, but SPECpower_ssj2008 currently addresses energy proportionality best and a lot of results are available. However, it is focusing strictly on throughput, with other metrics highly important to users like average response times being out of consideration. Additionally, it primarily aims at benchmarking a single system under test (SUT), although benchmarking a set of identical systems is within the test's specification. Clearly, evaluating a set of heterogeneous servers with regard to their energy efficiency in consideration of QoS-metrics is currently not in the range of benchmark.

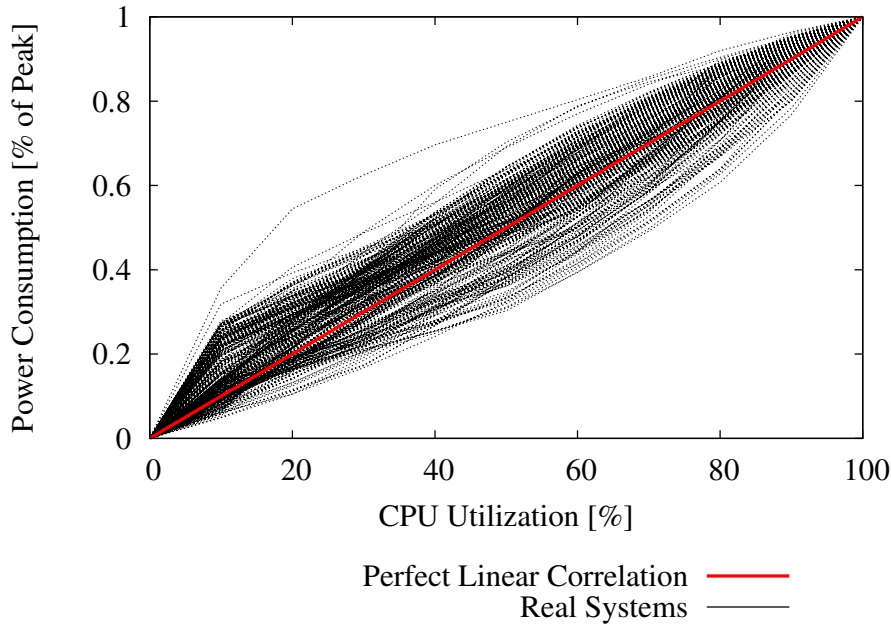


Figure 2.6: System power consumption, relative to peak, depending on the CPU utilization for almost 300 systems participating in SPECpower_ssj2008.

However, SPECpower_ssj2008 is an interesting data source to determine the power signature of real and common servers. Figure 2.6 shows the power consumption of the whole server, relative to its peak power consumption, depending on the CPU utilization, for all available single SUT, currently almost 300. We can see both convex and concave deviation from a perfectly linear rise. Furthermore, at lot of systems show a drastically bigger slope

from 0% to 10% utilization than for the rest. A plausible explanation for this is that most newer CPUs offer enhanced deep sleep states to aggressively save power when the CPU is completely idle. We can also see that residuals of 10% to 20% regarding the power consumption can easily occur if a server's power consumption is just measured at 0% and 100% utilization and interpolated. However, as a lot of common off-the-shelf servers are already available in the SPECpower_ssj2008 database, new measurements for software systems requiring power consumption models may be unnecessary.

2.3 Green Data Centers

During the last years, building and operating data centers in a *green* way, meaning energy efficient as well as ecologically sustainable [CMR⁺10], attracted a lot of attention. These requirements are in addition to the obvious necessity that a data center is economically efficient. In the sections above, a description of energy consumption characteristics of computing equipment found in a data center, resulting inefficiencies and potential solutions within the focus of this thesis were given. To give a broader view, problems, challenges and the current state of research regarding green data centers are briefly discussed in the following.

Ensuring a data center's economic efficiency is a challenging task. Fan et al. [FWB07] study power consumption at the data center level and describe some of the challenges involved regarding power provisioning. It is hard to neither under-provision nor over-provision a data center's power infrastructure, as the actual power consumption of a whole data center depends on a multitude of factors, and is additionally varying. The task is to fully use the data center's power budget in order to amortize the non-recurring facility costs. Fan et al. [FWB07] have found that the difference between the theoretical peak power usage and the actual usage can be up to 40%, giving much room for adding more computing equipment within the already existing power budget.

However, designing power provisioning is only one of several tasks when building a data center. A cost model for planning, developing and operating a data center was presented by Chandrakant et al. [PS05], addressing costs for space, power, cooling and operation. Here, the increasing thermal power density is an important challenge, as it makes cooling more expensive and the occurrence of hot-spots more probable. Chen et al. [Kek06a, Kek06b, Kek06c] have provided a comprehensive study about the feasibility of locally controlling temperature in a data center. Moore et al. [MCRS05, MCR06] have proposed temperature-aware workload scheduling. Raghavendra et al. [RRT⁺08] have presented a coordinated approach of managing power delivery, electricity consumption and heat dissipation.

As mentioned in Section 2.1, a significant amount of energy is spent in data centers for cooling purposes. Typically, air conditioning and ventilation are used to provide cool air and to transport the heat dissipated by computing equipment to the data center's environment. In general, the bigger the difference between the environment's air temperature and the data center's target temperature is, the bigger is the amount of energy required. Due to physical reasons, air is a rather inefficient medium for such purposes. Due to its high ther-

mal capacity, water is a much more efficient and suitable substance. However, the technical challenges involved in using water for cooling purposes are drastically higher. A complex piping system has to be built, reaching from all high priority heat dissipating components of all kinds of computing equipment to recuperators. Further, ordinary water conducts electricity and would cause a catastrophe in case of a pipe leak. Finally, providing cool water requires lots of energy, and finding a use case for water that was warmed about only a few degrees is difficult. In this context, Sharma et al. [SSB⁺08] have studied the water demand of data center cooled with water, and have proposed a trade-off between water efficiency and energy efficiency.

SuperMUC⁷ is Europe's fastest super-computer, ranking number four world wide. It recently went online and presents a novel solution to the problem. Counterintuitively, warm inlet water of approximately 40 degrees Celsius is used instead common 16 degrees Celsius. To provide water with 40 degrees, simple cooling techniques can be used as temperatures above 35 degrees are very rare in Germany. As a consequence, the outlet water is quite hot with up to 70 degrees Celsius, and can be used, e.g., for heating purposes. CPUs are safely operable at up to 85 degrees Celsius, thus overheating is not an issue. This cooling system, named Aquasar and built by ETH Zurich and IBM, is said⁸ to reduce the energy consumption by 40%, and CO₂ emissions by up to 85%.

Concurrently, an approach becoming more popular is to build data centers where its environment can be exploited for cooling purposes. Ideally, the data center's energy demand is provided by renewable sources. E.g., the Green Mountain Data Centre⁹ near Stavanger, Norway, is placed in a fjord, where vast amounts of cold water are available throughout the year. Facebook is currently building¹⁰ a data center in Lulea, Sweden. Lulea is about 100 km south of the polar circle, offers a cool and dry climate for easy cooling by air, plentiful renewable hydroelectric energy, and excellent network connectivity with five carriers. Google has built¹¹ a data center in Hamina, Finland, where it also uses cool seawater for cooling. Currently, Google is building¹² a second data center in Hamina. A location often discussed as perfect for green data centers is Iceland. It offers cold air and water, renewable energy from hydroelectric and geothermal sources, and high quality infrastructure. However, latency issues¹³ make it unlikely that Iceland will become a top location for green data centers. Being in the midst of Europe and North America, there is no customer base nearby. For services requiring low latency to a large customer base, this is a significant drawback.

⁷<http://www.lrz.de/services/compute/super muc/systemdescription/>, last visited on 10.8.2012

⁸http://www.ethlife.ethz.ch/archive_articles/090623_liqiud_cooled_computer/index, last visited on 10.8.2012

⁹<http://www.greenmountain.no>, last visited on 10.8.2012

¹⁰<http://goo.gl/KHi jd>, last visited on 10.8.2012

¹¹<http://www.google.com/about/datacenters/locations/hamina/index.html>, last visited on 10.8.2012

¹²<http://www.wired.com/wiredenterprise/2012/08/google-finland-data-center/>, last visited on 10.8.2012

¹³<http://goo.gl/g3LF8>, last visited on 10.8.2012

There are also increasing efforts to disseminate knowledge about how to build a greener data center. Greenberg et al. [GMT⁺06] have compiled best practices for cooling and power provisioning learned from studying 22 data centers. Best practices regarding, but not limited to, energy efficient data centers, are also discussed by Barroso and Holzle [BH09]. Da Costa et al. [DdAG⁺10] have presented the GREEN-NET framework to decrease the energy consumption of grid and cloud computing infrastructures. It informs users about a job's energy consumption, and gives them possibilities to reduce it. Concurrently, it manages the used infrastructure to save energy by switching off nodes. Further, there are several industry platforms and cooperations to reduce data center energy consumption. Facebook has founded the Open Compute Project¹⁴, offering detailed know-how about low-cost, efficient server and storage systems, and how to design an energy efficient data center. AMD has founded the Green Grid¹⁵ project, which was recently joined also by Google and Intel, who were previously active in the Climate Savers Computing project¹⁶.

¹⁴<http://opencompute.org>, last visited on 10.8.2012

¹⁵<http://www.thegreengrid.org>, last visited on 10.8.2012

¹⁶<http://www.climatesaverscomputing.org/>, last visited on 10.8.2012

Chapter 3

System Virtualization

Nowadays, modern data centers frequently use system virtualization. In the context of this thesis, dynamic resource allocation and live migration are the most important advantages offered by system virtualization. Dynamic resource allocation refers to the fact that a virtual machine's resource allocation can be changed on the fly and without service interruption. The most interesting tuning options are less/more virtual CPUs and main memory. The scheduling of CPU cycles is done automatically by the hypervisor or virtual machine monitor, although it can often be influenced. Live migration provides the possibility to establish a more energy efficient target distribution of virtual machines, as it can be used for dynamic consolidation (turning off servers) and scaling (using more or faster servers). For building more energy efficient computing infrastructures in the future, the ability to live migrate VMs between physical servers without any perceptible service interruption is pivotal. It allows to dynamically shift workload encapsulated in VMs between servers with regard to a cost model, where energy costs of a computing infrastructure play an important role, next to, e.g., response times. These two key benefits are further discussed in Section 3.4 and Section 3.5, where several implementations of live migration and the current state of research are presented.

3.1 Introduction

In general, virtualization is a concept for abstracting computing resources from a physical hardware layer to a logical, virtualized layer. This abstraction layer can be realized in various ways for different purposes. Adams and Agesen [AA06] provide a detailed discussion of x86 virtualization. Common purposes in data center are system, network and storage virtualization. An entity, be it a human user or a computer software, is then able to use a resource without having knowledge about the inner workings of the resource, as visualized in Figure 3.1. A modern operating system offers a multitude of abstract interfaces and mechanisms to allow a user to concentrate on the specific task. Simple but typical examples are virtual memory, scheduling, privilege management, network protocols and file systems.

Briefly, virtualization can be used in two different directions. On the one hand, a computing resource (e.g., a server) can be separated into n isolated and independent logical resources (e.g., database, web and mail server). This process can be described by *resource*

partitioning and is depicted in Figure 3.2. In the ideal case, applications and users are completely unaware of using a virtualized resource. Further, complete isolation inhibits noticing the existence of other virtualized resources.

On the other hand, n computing resources (e.g., servers) can be aggregated to a single, big, logical resource (e.g., to a cluster, grid or more recently a cloud). In the ideal case, applications and users are completely unaware of its inner details. In a cluster scenario, users would just submit computing jobs and have no knowledge about workflows like job placement or load balancing. This process can be denoted as *resource aggregation* and is depicted in Figure 3.3.

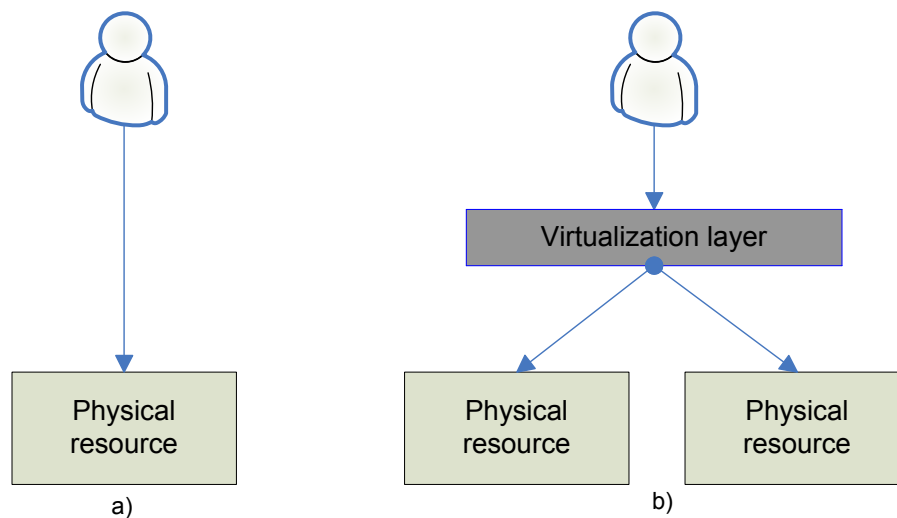


Figure 3.1: a) Without virtualization; b) With virtualization.

3.2 History

The history of virtualization is tightly coupled to the history of computing in general. A short overview of the development is given in the following.

Beginning of the Computing Era. When the first machines similar to what is now understood as a computer came up, their computing power was very little compared to today's standards, but their price was astronomically high. Further, they were used inefficiently, as programs had to be run serially, and long time gaps between these runs were common. Scheduling was done by human operators. In a first step to increase efficiency, batch job control systems were developed, allowing to schedule and compute tasks in a certain, desired order. However, it was not yet possible to split a machines computing power into time

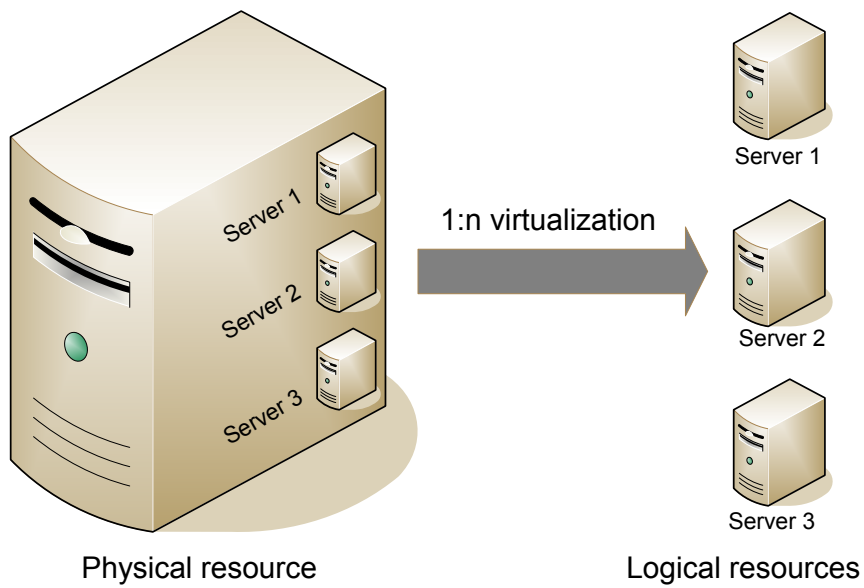


Figure 3.2: Resource partitioning using virtualization.

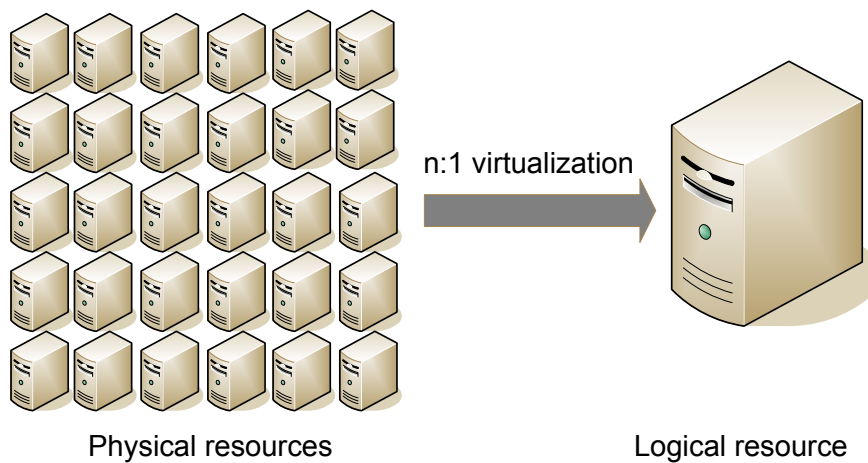


Figure 3.3: Resource aggregation using virtualization.

slices and to assign them to a number of existing process. Further, programmers still had to explicitly manage different memory hierarchies. To solve these problems, significant effort was necessary in the following decades.

Virtual memory. One of the first implementations using virtualization was memory paging. In the late 1950s, a group at the University of Manchester introduced automatic page replacement in the Atlas system [CJ06]. It enabled programmers to allocate more memory

than physically available by having a logical memory address space. A virtual memory system is also required for multitasking, where processes need to be isolated from each other. Further, the different memory hierarchies (main memory and disk space) were accessible uniformly. Nowadays, virtual memory is state of the art and part of almost all modern operating systems.

Virtual Machines. In the late 1960s, IBM presented the System/360 model 67 [CJ06]. It implemented a self-virtualized processor instruction set and a virtual machine operating system, often denoted by *virtual machine monitor*. It was possible to run several different operating systems on a single processor machine, with all virtual machines running isolated and protected from each other. Further, IBM developed the timesharing technology, yielding the wide adoption of VM/timesharing systems such as IBM VM/370.

Peak. The 1970s are clearly representing the peak in usage and popularity of virtualization. Decreased costs and increased flexibility compared to other approaches are the most important reasons for this. Software developed for older, less powerful legacy computers could be run on newer, more powerful systems without significant adaption effort. Beneath a family of compatible microcode CISC processors, virtualization was a key technology to support legacy software. [TvS08]

Decline. In the 1980s and 1990s, low-cost mini- and personal computers with increasing computing power and available around the corner took over. The IBM mainframe almost disappeared, and with it, virtualization. The concept of virtualization survived in niches, e.g. Sun's Java Virtual Machine.

Recurrence. In the late 1990s, servers using personal computer technology rapidly became both more powerful and cheaper. It soon became standard procedure to buy more servers and add them to the infrastructure instead of hosting additional tasks or customers within the existing server pool. The resulting inefficiency regarding management and costs lead to the conclusion that consolidation is necessary to decrease complexity and total costs of ownership. In the last few years, static consolidation as described in Section 4.1 was used to solve the problem. Each virtual machines runs in its own, isolated environment. Ideally, all kinds of problems occurring inside a virtual machine are strictly contained. Isolation was one of the most important demands that lead to the tremendous growth of the installed server base in the 1990s. Here, isolation was done by dedicating hardware to a specific task, project or customer. Interestingly, modern operating systems should be able to offer a sufficient level of isolation by process isolation and privilege management. However, even hypervisors are currently unable to guarantee complete isolation, whether because of exploitable bugs or because of indirect measures such as timing attacks. E.g., if VM network connectivity is achieved by virtual Ethernet bridging, cracking a single VM often leads to the chance of sniffing traffic of other VMs on the same host.

Status Quo. The availability of VMware [RG05] products, offering x86 virtualization using binary translation, helped in creating a second boom of virtualization. Since then, a multitude of both free and commercial virtualization solutions is available¹. Both AMD [AMD07] and Intel [Int07] have included support for hardware supported virtualization into their instruction sets several years ago. Binary translation is thus not necessary anymore, and virtualization can be used with small performance overhead [VMw07]. Today, typical servers offer multi-socket, multi-core CPUs and dozens of GiB main memory and are thus able to host a number of tasks that previously had to be distributed over several nodes. Without virtualization, it would be sometimes even hard to adequately utilize the available computing power of a typical server.

3.3 Technologies and Implementations

There are several common techniques and implementations of system virtualization [AA06, RG05, WCC⁺08]. Historically, the perhaps first classification of system virtualization was done by Goldberg [Gol73] in 1973. In 1974, Popek and Goldberg [PG74] have proposed formal requirements for virtualizable architectures. The most important technologies and implementations for the work at hand are discussed in the following Sections.

3.3.1 Full Virtualization and Binary Translation

The perhaps most brute-force approach is to fully virtualize a computer's hardware in order to allow unmodified operating systems to run in a virtual machine. The guest operating system is completely unaware of being virtualized. In the past, this approach was primarily used for hardware emulation purposes, e.g., emulating a hardware platform to do cross-platform software development. The unmodified operating system will invoke privileged instructions as if it had full and exclusive hardware access. Thus, these instructions have to be trapped and translated in a proper, integrity conserving way before execution. This process, typically called binary translation [AKS00], is very resource intensive, even if the native ISA and the virtualized ISA are the same. The x86 ISA was originally not designed for virtualizing multiple operating systems. In the x86 privilege ring model, there is only a single privileged ring. Running multiple kernels is thus impossible without further efforts. Hence, full virtualization was the only feasible x86 virtualization approach for a long time. This has changed with the instruction set extensions Intel VT and AMD SVM mentioned in Section 3.3.3.

To decrease the performance penalty, several technologies were developed. In the x86 ecosystem, binary translation was extended to adaptive binary translation [AA06] by VMware and made their early products very popular. It automatically detects those in-

¹http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines, last visited on 15.8.2012

structions which have to be translated, and ignores the other for the sake of performance. Once translated instructions are cached to further decrease the performance penalty.

3.3.2 Paravirtualization

Paravirtualization requires the guest operating system to be modified, in order to cooperate with the hypervisor to ensure integrity. This way, the amount of instructions that have to be translated can be decreased further. However, modification of guest operating systems is impossible for binary-only distributions like Microsoft Windows.

A typical example in the x86 ecosystem is Xen [BDF⁺03]. However, since the inclusion of the generic, hypervisor agnostic *paravirt_ops*² infrastructure into the mainline Linux kernel, paravirtualization was also partly adopted by VMware, KVM, etc. With the increasing availability of hardware-assisted virtualization, paravirtualizing whole guest operating system became mostly unnecessary again and several vendors dropped their support. Nevertheless, the concept of paravirtualization was found to be still useful for virtualizing I/O devices such as network and disk controllers, e.g., the *virtio* platform³

3.3.3 Hardware-assisted Virtualization

Basically, binary translation and paravirtualization are workarounds for the x86 ISA, which until recently offered no hardware support for virtualization. Today, almost every processor model manufactured by AMD and Intel has an extended instruction set allowing to do native virtualization. Here, unmodified guest operating systems can run in a virtual machine. A comparison of the instruction set extensions AMD SVM and Intel VT is available in [Hav05].

AMD SVM is a concept of intercepting critical instruction in hardware, as described in [EP07, Cha05b]. It virtualized the parts of the Memory Management Unit (MMU), especially the Shadow Page Tables (SPT), which are then called Nested Page Tables (NPT). As a result, each virtual machine has its own virtualized page tables. It also offers a virtualized TLB, denoted as Tagged Translation Lookaside Buffer (TTLB). The TTLB has the advantage that only entries belonging to a specific VM have to be flushed at a context switch, but not the whole buffer. Further, it speeds up lookups, similar to set-associative caches.

Intel VT [Cha05a] is basically a subset of AMD SVM, and adds a new privilege ring, the *VMX root mode*. It is even more privileged than the traditional x86 Ring 0, where a kernel runs. The hypervisor runs in the *VMX root mode*, and virtualized guests run in *VMX mode*. To switch between virtual machines, the instructions *VM entry* and *VM exit* are necessary.

²<http://wiki.xensource.com/xenwiki/XenParavirtOps>, last visited on 16.8.2012

³<http://www.linux-kvm.org/page/Virtio>, last visited on 16.8.2012

3.3.4 Operating System Level Virtualization

OS level virtualization refers to the approach of splitting an existing operating system into several isolated virtual environments. Common terms for these environments are containers, partitions, zones or jails. Typical examples for x86 are OpenVZ and Linux VServer. Each container has its isolated userspace, but all containers share a common kernel, as visible in Figure 3.4. This is both the biggest advantage and disadvantage of OS level virtualization. Sharing a kernel is very memory efficient, and OS level virtualization is a popular choice for scenarios where a big number of VMs using the same operating system is required, e.g., virtual private web server hosting. Each virtualized kernel-space can be specifically configured to allow a certain number of open file handles, network connections, active processes or amount of memory usage. Further, starting a kernel for each VM is unnecessary, so a vast number of VMs can be started or stopped within short time [QNC06].

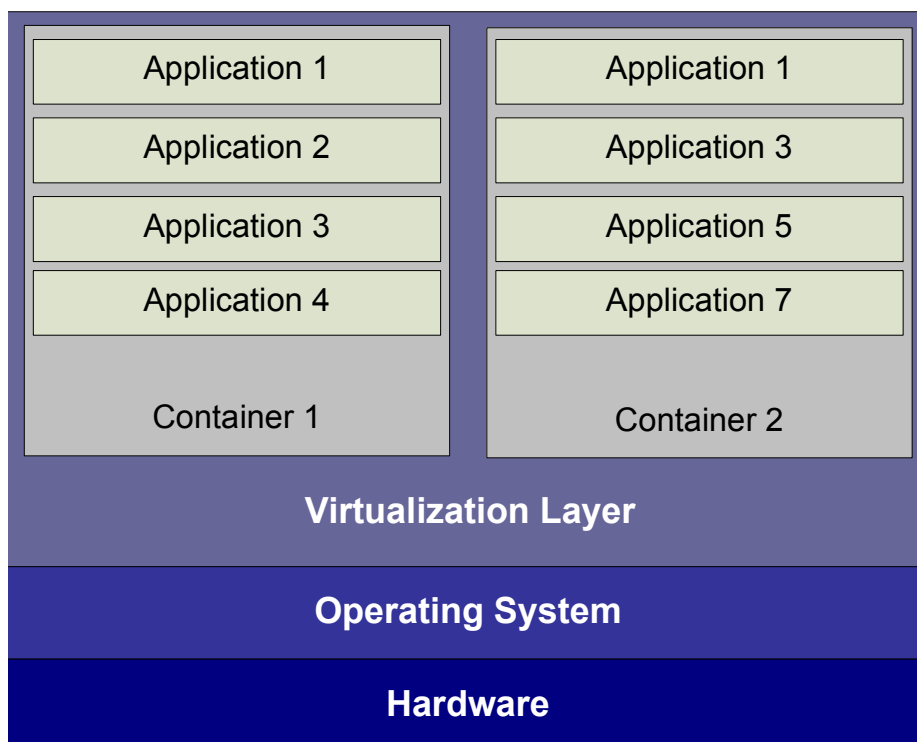


Figure 3.4: Operating System Level Virtualization.

However, sharing a common kernel can also be a big drawback, depending on the scenario. With OS level virtualization, it is impossible to run different operating systems in different virtual machines.

3.3.5 Hypervisor Based Virtualization

Using a hypervisor or virtual machine monitor is perhaps the most sophisticated approach to virtualization and typically found in data centers. It is also similar to the classical approach introduced by IBM. Goldberg [Gol73] distinguishes between type 1 and type 2 hypervisors. Type 1 hypervisors run exclusively on the bare metal hardware, as shown in Figure 3.5. Type 2 hypervisors run on an operating system, which runs natively on the hardware. Type 2 hypervisors is often referred to as hosted virtualization.

Examples for the former are Xen, VMware ESX, Hyper-V, etc. Examples for the latter are VMware Workstation and VirtualBox. Regarding the classification of KVM, there are different opinions⁴. It is argued by KVM developers that the KVM kernel module turns the Linux kernel into a type 1 hypervisor. A brief visualization of its architecture is shown in Figure 3.6.

Type 1 hypervisors such as Xen often have a very small footprint. E.g., Xen is implemented within less than 50000 lines of code [Xen06]. On top of this, several guest operating systems can be run, each in its own virtual machine. Nowadays, guest operating systems can be run unmodified with most hypervisors, if CPU hardware support is provided.

3.4 Key Benefits

The common advantages and disadvantages of virtualization have been discussed sufficiently in the past, e.g., [SML10, Tre08]. Generally, isolation, encapsulation, flexibility, efficiency, availability and scalability are quoted as advantages. There are many examples where virtualization indeed makes it easier to fulfill such demands, as it adds a layer of abstraction to the physical layer and thus creates a logical view onto resources. On second thought, such an abstraction layer also adds complexity, making certain aspects of management harder in a virtualized scenario. A simple example is that when a virtualized infrastructure uses dynamic consolidation (migrating virtual machines and powering down unrequired physical servers) traditional monitoring software, e.g. Nagios, must be adapted and informed to not give an alarm in such cases. Hence, the involved overhead and the added layer of complexity are usually pointed out as disadvantages. In the following, the two key benefits offered by virtualization regarding this thesis are briefly discussed.

3.4.1 Dynamic Resource Allocation

A virtual machine's resource allocation can be changed on the fly and without service interruption. The most interesting tuning options are less/more virtual CPUs and main memory. The only constraint is that allocations can not go beyond the configured maximum allocation

⁴<http://virtualizationreview.com/blogs/mental-ward/2009/02/kvm-baremetal-hypervisor.aspx>, last visited on 16.9.2012

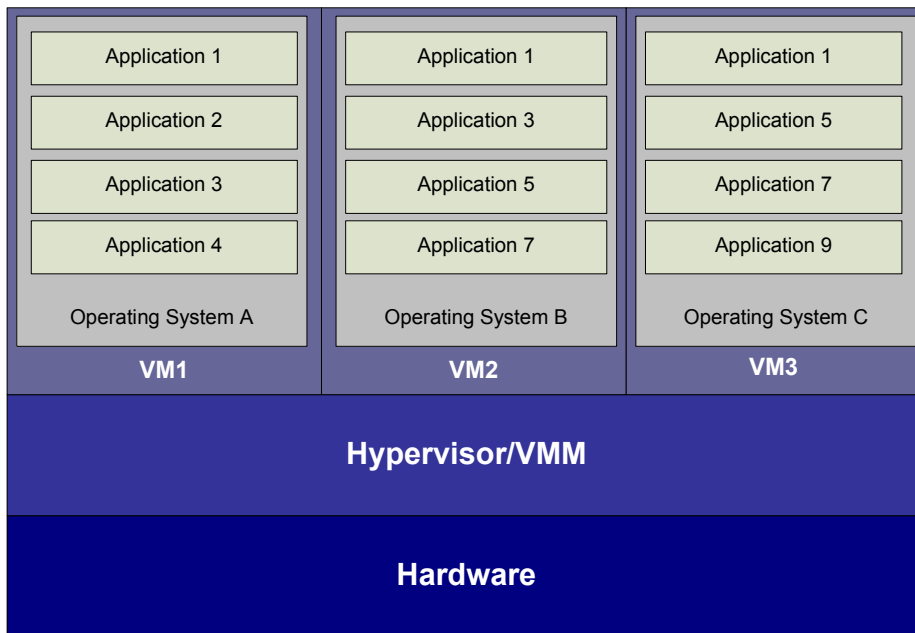


Figure 3.5: Type 1 Hypervisor based virtualization.

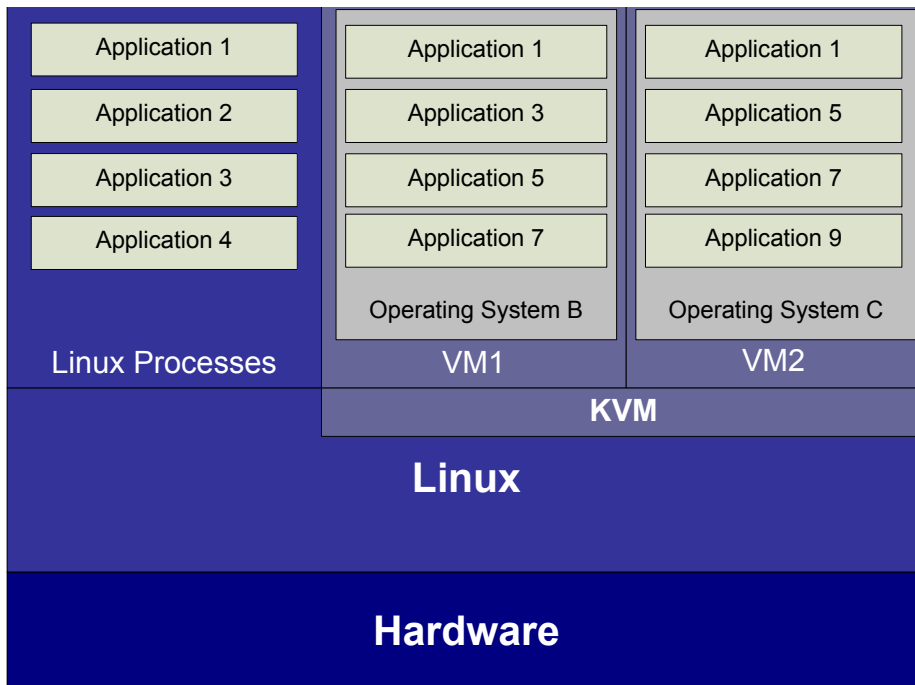


Figure 3.6: The KVM hypervisor architecture.

the VM was booted with, as the guest operating system needs to initialize the corresponding internal data structures at startup.

E.g., a VM can get more main memory when required and physically available. When the additional memory is no longer required, the VM can be shrunk to its original size again. It is important to note that in general such a process has to be triggered manually. In contrast, CPU time slice assignment is automatically done by the hypervisor. When a vCPU requests its execution, the hypervisor follows a scheduling policy and decides how to react. A vCPU can be executed only by a single real CPU at a given time, so it can not consume more time slices than a single real CPU can provide. This is important, as a VM needs to have at least n vCPUs to be able to consume the computing power of n real CPUs. Although the assignment of CPU time slices is done automatically, the number of vCPUs of a given VM is not adjusted automatically but is subject to manual interaction.

Statically over-provisioning vCPUs is generally not beneficial, as both the VM's and the hypervisor's CPU scheduler would act in an uncoordinated way. The guest operating system could try to balance its workload amongst its vCPUs even in low workload scenarios, resulting in an unnecessarily high amount of context switches.

3.4.2 Live Migration

A virtual machine's place of execution (host) can be changed dynamically and, allegedly, without significant service interruption. It provides the possibility to establish an energy efficient target distribution of virtual machines, as it can be used for dynamic consolidation (turning off servers) and scaling (using more or faster servers). There are several approaches [LJL⁺09, HDG09, CFH⁺05] in realizing live migration, although the iterative pre-copy approach [CFH⁺05] is the only one publicly implemented and available for experimental studies. Interestingly, even modern data centers which already use system virtualization often do not fully profit from the key benefits described above. Common reasons for data centers not utilizing dynamic resource allocation and live migration are fears concerning the quality of feasibility and possible side effects, the yet unclear potential of saving energy and money and a manifold lack of high quality automation in typical virtualization solutions.

3.5 Implementations of Virtual Machine Live Migration

Live migration of virtual machines can be seen as a technological successor to process migration [MDP⁺00, Dou89, DO91, Zay87, Smi88] in non-virtualized environments, e.g., OpenMOSIX⁵. Interestingly, process migration never became widely adopted due to several reasons, economical as well as technical. Most of them are caused by the fact that programs

⁵<http://sourceforge.net/projects/openmosix/>, last visited at 5.6.2012

needed to be changed to be migratable. With modern CPUs having hardware support for virtualizing operating systems, these do not need to be changed, let alone user space programs running inside a VM. Furthermore, virtualization is already widely adopted in current data centers as mentioned above.

Live migration of virtual machines is an important mechanism offered by typical virtualization layers. It allows to change the place of execution (host, server, hypervisor, etc.) of a running virtual machine. Ideally, this process is transparent to the guest operating system and its applications. In the past, it was primarily used for availability purposes. If a physical server requires maintenance, its virtual machines can be migrated to another physical machine without service interruption.

Within the scope of this thesis, live migration is a key mechanism to rearrange workload encapsulated in virtual machines. It provides the possibility to establish a more energy efficient target distribution of virtual machines, as it can be used for dynamic consolidation (turning off servers) and scaling (using more or faster servers). Of course, it can also be used for load balancing purposes. Therefore, the ability to live migrate virtual machines between physical servers without any perceptible service interruption is pivotal for building more energy efficient computing infrastructures in the future. It enables us to dynamically shift workload between servers with regard to a cost model, where energy costs of a computing infrastructure play an important role (next to, e.g., response times). This way, the currently lacking energy proportionality of servers mentioned in Section 2.2.1 can be solved by dynamically consolidating to fewer servers in times of lower workload intensity. Based on the actual resource demand, it is easily possible to dynamically change the workload distribution among servers.

3.5.1 Non-Live Stop-and-Copy Migration

Before discussing the iterative pre-copy live migration implementation in the following Section 3.5.2, it is instructive to briefly describe non-live migration. Often, this approach is denoted as stop-and-copy migration, corresponding to the two most important process phases. Here, an administrator or management system triggers the migration of a virtual machine. Then, the VM is paused, meaning its virtual CPUs are deactivated. Hence, the VM can not change its state. Following this, the virtual machine's state is copied to the destination server. The virtual machine's state is comprised of several parts, e.g., its main memory, virtual CPU state (registers, etc.), the states of virtual devices as NICs, etc. When the transfer was successfully carried out, the VM is unpaused at the target. Typically, ARP announce packets will be sent to inform the virtual network bridge of its new member.

Clearly, the VM is completely unresponsive during the migration, and all the services it is offering are unavailable. Moreover, any network connections established will be most likely lost due to time-outs. To draw an example, transferring the comparably small amount of 1 GiB of main memory over Gbit-Ethernet will approximately take 10 s. If no central storage system sheltering the virtual machine's disk images is used, then these have to be transferred too, drastically increasing the downtime.

3.5.2 Iterative Pre-Copy Live Migration

The iterative pre-copy live migration implementation was the first one available and is nowadays the most common. In contrast to the non-live stop-and-copy implementation, the virtual machine's state is copied to the destination *before* it is stopped, hence *pre-copy*. The idea is to significantly decrease the downtime by transferring as much as possible to the destination before actually switching the place of execution. The use of a central storage system sheltering the virtual disk images and reachable by both servers involved is indispensable for any implementation of live migration.

Iterative pre-copy live migration was first described by Clark et al. [CFH⁺05] for the *Xen* hypervisor. *Xen*'s live migration algorithm works iteratively and is an important building block for many other hypervisor's live migration mechanisms. Roughly, it has three phases: First, a full copy of the VM's memory pages is transferred from the source to the destination server. Then, only the pages which have changed during the last transfer (denoted as *dirty pages*) are copied again. This is done iteratively until the amount of dirty pages is reasonably small to finish the live migration: The VM is paused at the source host, the remaining dirty pages and the vCPU state are transferred, and finally the VM is unpaused at the destination host. As the CPU cache contents are not transferred, a new cache warmup is necessary.

Clearly, the VM is running during all except the final phase of the live migration, thus the VM is unresponsive only during the final iteration. Ideally, the final iteration is feasible quickly enough for network connections most probably staying alive. As memory pages can get dirty again, the crucial assumption here is that the amount of dirty pages quickly converges to a minimum, and service interruptions are imperceptible. If this assumption of convergence does not hold, the live migration may run indefinitely. A clear case of lacking convergence is when the page dirtying rate is exceeding the page transfer rate. Another problematic situation occurs if the page dirtying rate is getting close to the transfer rate, although not necessarily exceeding it. Then, the migration process will converge only very slowly, causing severe service degradation due to the lengthy and resource intensive migration process. As a consequence, stop conditions are necessary to ensure that the process does finish eventually. In *Xen*, three stop conditions [ASR⁺10] are defined:

1. Less than fifty pages were dirtied during the last iteration, meaning that the final iteration, during which the VM has to be offline, can be carried out within short time. Here, the process has converged, and the threshold of fifty dirty pages is just representing the fact that achieving zero dirty pages while a VM runs is unrealistic even for idle VMs.
2. 29 pre-copy iterations have been carried out, meaning that the complete address space was scanned 29 times for dirty pages, but condition 1 is still not achieved. The process does obviously not converge, and the 30th iteration is forced to be the final one.
3. More than three times the amount of memory the VM has allocated was transferred over the network, but condition 1 is still not achieved. E.g., if a VM has 1 GiB of

RAM allocated, then Xen will force the final iteration if more than 3 GiB were already transferred.

KVM's implementation [KKL⁺07a] of live migration is similar, but there is an important difference: Dirty memory pages are iteratively transferred over network until the remaining amount can be transferred within a predefined (but adjustable) time limit. In contrast to [KKL⁺07a], where two stop conditions are described, the actual current implementation has no stop conditions at all. If the rate of dirtying pages is too high and/or the time limit is too low, the migration process would literally take forever and the administrator or a management application has to increase the time limit or cancel the process. Compared to Xen, which indeed considers the case of insufficient convergence, this is a rather suboptimal implementation. An external entity needs to track the process, but with less information available than within the hypervisor itself.

3.5.3 Post-Copy Live Migration

Hines et al. [HDG09, HG09] have implemented and presented a different approach: In contrast to Xen's pre-copy algorithm, where the virtual machine memory is fully copied from the source to the destination host, they discuss a post-copy algorithm, where the CPU state is transferred to the destination and the VM is started immediately. Their algorithm then actively pushes the VM's memory pages to the destination host and concurrently transfers those pages faulted at the destination but not pushed yet. Their implementation requires the introduction of a new kind of page fault.

As the VM is almost immediately executed at the destination host, they avoid duplicate page transmissions, which increases total migration time but decreases downtime. They have studied the amount of page faults, which cause an on-demand page transfer over the network. The achieved downtime is much higher compared to pre-copy, allegedly because of their page fault detection implementation. However, there is no extensive study on how this influences service performance. As network latency is typically multiple orders of magnitude greater than (local) memory latency, performance during live migration could suffer severely for some workload types.

Hirofuchi et al. [HNIS11] have implemented post-copy live migration for KVM and have shown its potential, especially regarding energy efficiency. During the phases of intense research activity for this thesis, the source code was marked obsolete and not to be used in any way. Since recently, their current implementation is publicly available on the project page⁶. Post-copy live migration is a most interesting option to extend the experiments. However, for my thesis, their efforts were not available with sufficient quality to be able to conduct experiments. For the experiments carried out and evaluated within this thesis, using iterative pre-copy live migration was the only possible option.

⁶<http://grivon.apgrid.org/quick-kvm-migration>, last visited on 17.8.2012

3.5.4 Live Migration over Wide Area Networks

Studies investigating the impact of transferring memory pages over network on demand become even more important when considering the increasing effort put into researching live migration over WAN. Here, latencies are significantly higher compared to local networks. Another critical challenge is to ensure that network clients keep connected to the VM after it was migrated to the target location. Hence, some kind of network virtualization is additionally necessary. Currently common implementations of live migration require both servers involved to be within the same network broadcast domain, where an ARP announce packet is sufficient to inform clients about the virtual machine's new location. Here, the VM is migrated between to distinctive networks. As live migrating virtual machines between data centers to increase their energy efficiency is beyond the scope of this work, only a brief overview of the current research is given in the following.

Wood et al. [WRSV11] have interconnected multiple Cloud Computing platforms located at data centers in Texas and Illinois by VPNs. They have optimized live migration for network links with low bandwidth and high latency to decrease the time required of a migration. Travostino et al. [TDG⁺06] have migrated VMs across long distances between data centers. They have carried out a live migration from Amsterdam to San Diego with a downtime of just 2 s, using a logical fiber end-to-end link as network virtualization to ensure client connectivity. Harney et al. [HGM⁺07] have used mobile IPv6 to ensure connectivity, yielding a minimum downtime of 2.5 s. With increasing IPv6 availability, this approach is especially interesting. Hirofuchi et al. [HNO⁺09] have specialized on live storage migration, a particular challenge when migrating VMs across wide area networks.

Further, there are also increasing efforts in live migration of databases [DNAA] and storage [ZNS11, RSV07] for Cloud services.

3.5.5 Full System Trace and Replay

Liu et al. [LJL⁺09] have presented a completely different approach. They have implemented and evaluated live migration based on a full system trace and replay. First, a checkpoint of the VM is saved and transferred to the destination. During the creation of a checkpoint, the VM must be paused. Subsequently, the virtual machine's execution is traced on the source host and synchronized to target host. Thus, an execution log is transferred instead of dirty memory pages. Similar to the transfer of dirty pages, here, the log replay rate must be greater than the log grow rate for the process to converge. The authors mention a typical log grow rate of approximately 1 MiB/s and claim significantly decreased migration time, downtime, and data transfer involved. A log replay with a rate greater than 1 MiB/s is easily realizable even over today's wide area networks.

However, there is a severe constraint. The authors mention that it is very difficult to migrate VMs with multiple virtual CPUs using their approach. Furthermore, there are several open issues regarding the achievable quality of service. The VM must be paused during checkpointing, so it is crucial that creating a checkpoint is done as quickly as possible.

Generally, there are two possible locations where a checkpoint can be written to. If it is written to disk, then the hard drive's or storage subsystem's bandwidth can quickly become a bottleneck. If it is written to main memory, then the virtual machine's memory allocation will be temporarily doubled. Of course, the checkpoint could be directly transferred over the network to the destination host, without temporary storage. Then, the network bandwidth is clearly determining the amount of time required for checkpointing and the corresponding VM downtime. Additionally, the possible influence of the tracing overhead is still an open issue.

3.5.6 Increasing Live Migration Efficiency

The increasing size of virtual machines and the desire to migrate VMs over WAN have led to efforts in increasing live migration efficiency. A VM with a couple of GiB main memory may be migrated conveniently over Gbit LAN, but perhaps not over a WAN to a data center on another continent. Further, migration of VMs with dozens of GiB main memory is already difficult in the local case.

Hacking and Huidza [HH09] have presented a modification to KVM allowing to migrate large enterprise applications, e.g., SAP ERP. They build a dirty page transfer mechanism based on delta compression. They also use a warmup transfer phase, meaning that dirty pages are optimistically transferred in the background even before the decision to migrate the VM has been made. The warmup phase is initiated if the VMs load spikes over a threshold. If it decreases again, the warmup phase is canceled, if it grows further, the actual live migration is started. The proposed method of delta compression was further evaluated in [SHTE11].

Jin et al. [JDW⁺09] have implemented a page compression mechanism based on analyzing the memory page similarity for the Xen hypervisor. Both migration downtime and delay are reduced by a third. An approach by Zhang [ZHMM10], based on data deduplication using self-similarity and run length encoding performs similar. Riteau et al. [RMP11] have presented Shrinker, a system that not only efficiently migrates the main memory, but also the virtual disk images.

Chapter 4

Energy Efficient Dynamic Consolidation

4.1 Static Consolidation

Concerns regarding efficiency are one of several important reasons why modern data centers frequently use system virtualization to achieve static consolidation.

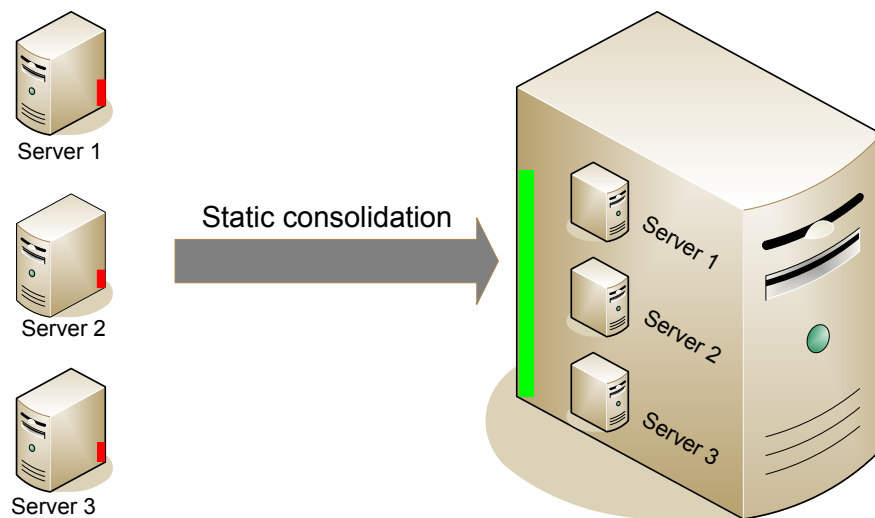


Figure 4.1: Static consolidation to increase the utilization of formerly idle servers.

This way, a set of idle servers can be easily consolidated to a smaller set of highly utilized servers, effectively shifting the operating region to an area of higher efficiency, thus reducing *direct equipment* power. The procedure is also often referred to as *physical-to-virtual* or *P2V consolidation*, and depicted in Figure 4.1.

This popular approach is a first step in the right direction, but further, more systematic solutions need to be investigated. On the one hand, there is a natural limit to efficiency when most infrastructures have already been consolidated using virtualization or have been deployed directly virtualized. On the other hand, business infrastructures typically have to bear workload with strong periodic components. Generally, static provisioning can achieve only little efficiency when the workload and hence the resource demand is dynamic. A brief overview of workload patterns commonly found in business infrastructures is given in the following Section.

4.2 Workload and Utilization Patterns

One of the first characterizations of web server workload was done in the 1990s by Arlitt and Williamson [AW96, AW97]. In their works, the focus was on file and transfer size distributions, file referencing behavior, self-similarity and aborted connections. They have found small, but non-zero and decaying autocorrelation in some of the data sets, supporting the assumption of self-similarity. Self-similarity of traffic had been previously studied for Ethernet networks by Leland et al. [LTWW94]. Arlitt and Jin [AJ00] have later examined the workload of the 1998 football world cup website. Although there is no time-series analysis, the hourly traffic trace clearly shows a repeated pattern of bursts at times when football matches were starting. However, the size of a bursts depends on a multitude of parameters, e.g., there is more interest in traditional and powerful football nations. In 2005, Arlitt and Williamsion have repeated [WAWB05] their examination with the result that there are no severe changes in workload characteristics.

More recently, Gmach et al. [GRCK07] have analyzed patterns of a six month long workload trace, consisting of more than hundred enterprise applications. Two thirds of the applications exhibit a weekly pattern, 17% even a daily pattern. Most of the 31 strong patterns relate to cyclic batch jobs. Interactive workload is mostly contributing to 76 medium patterns. The 32 weak patterns include constant or random demands with no cyclic behavior, and workload that was interrupted multiple times or even completely changed in nature. Furthermore, Bobroff et al. [BKB07] have studied hundreds of workload traces from highly diverse production systems to identify workload signatures that benefit most from dynamic VM placement. They have extracted three main categories of occurring signatures by considering three different features:

1. **Variability:** If an infrastructure's workload shows only little variability, then dynamic placement is generally unnecessary. A mapping once found to be optimal would remain valid for a long period of time. On the other hand, large variability renders optimal solutions void within short time, as the workload intensity changes over time. Then, dynamic workload placement is necessary to repeatedly find an optimal mapping.

2. **Autocorrelation:** Strong autocorrelation makes it easier to generate predictions with small residuals, as it expresses similarity of data samples from different time intervals. This is especially important when forecasts for multiple future intervals are desired. E.g., workload traces showing a daily pattern will typically have strong positive autocorrelation at a lag of 24 hours, and strong negative autocorrelation at a lag of 12 hours.
3. **Periodicity:** Additionally, a periodogram is used to discover strong, distinctive periodic components to decide the most suitable length of a pattern. Here, the power of a signal is plotted against its frequency. In the above mentioned example, there would be strong signal at a frequency of 24 hours. Additionally, there could be strong signals at 48 hours or seven days, if there are weekly periodic components. Methods from field of time series analysis explicitly take this seasonality into account.

A compilation of the research results presented in [BKB07] is presented in Table 4.1.

Category	Variability	Autocorrelation	Periodicity	Benefit
1	High	Strong	None	High
2	Low	Weak	None	Low
3	High	Strong	Strong	Very High

Table 4.1: Compilation of research findings from [BKB07] regarding workload signatures and their benefit from dynamic placement.

In my thesis, the primary focus lies on web servers. They are generally a promising candidate for yielding high benefit from dynamic placement. The workload is primarily generated directly by human users. Unless there is strong variation in the users' time zones, during the night hours, only little workload will occur. The vast majority of workload will emerge during the day hours, with a rising edge during the morning hours, a trailing edge during the evening hours and possibly various peaks at times related to the exact nature of the service and its users' habits. Hence, high variability is very probable to exist. Among others, the quality of forecasts will depend on the level of autocorrelation and periodicity.

With more and more formerly stand-alone applications being transformed into Cloud services, this pattern will even get stronger and more common. Unconsciously, millions of users will not only use their own device for everyday tasks (e.g., reading their mails, fixing appointments in their calendar or browsing their picture gallery) but also a growing amount of infrastructure hosting these Cloud services. To keep network latencies low, data centers sheltering Cloud services are placed as close to the target group as possible, so the assumption regarding time zones will hold.

In such a scenario, static reservation of resources, although in a virtualized approach, is a dead end regarding efficiency. Virtualization by itself can not change the fact that provisioning resources for day hours is a willful waste of resources during the night hours, as shown

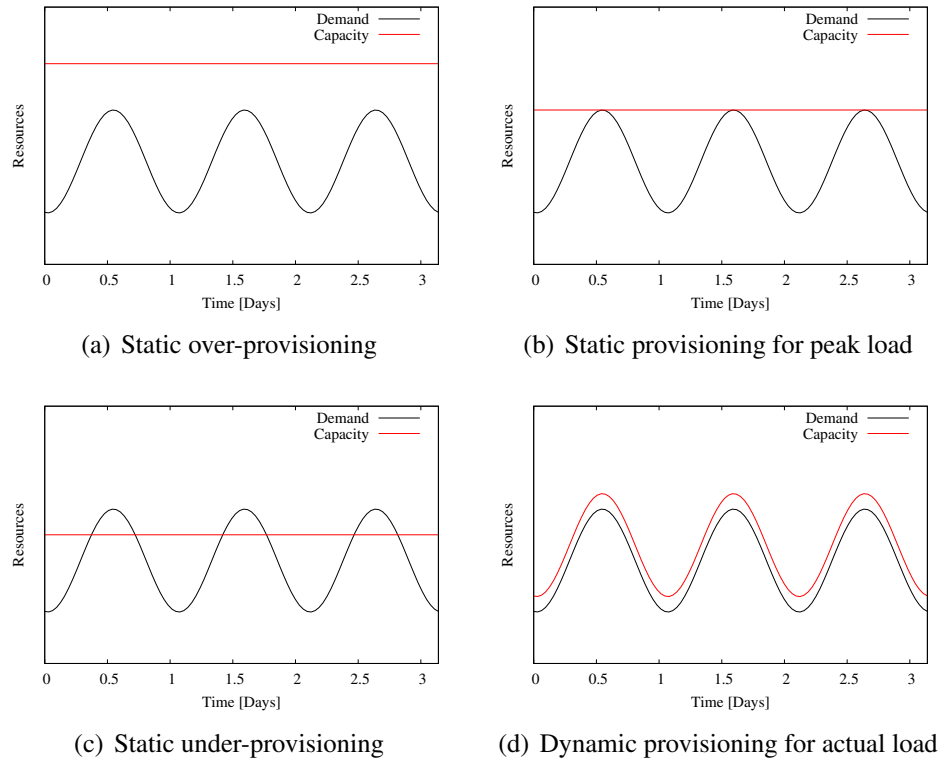


Figure 4.2: (a) Static over-provisioning wastes a significant amount of resources. (b) Provisioning for peak still wastes resources, and additionally risks resource shortage in case of unpredicted peaks. (c) Under-provisioning leads to loss of revenue, as QoS/QoE metrics will suffer. (d) Dynamic provisioning tries to minimize the gap between resource demand and online capacity. Plots (a), (b) and (c) as in [AFG⁺09].

in Figure 4.2(a). On the contrary, provisioning resources for night hours will typically result in catastrophic service levels and annoyed users during the day, as visible in Figure 4.2(c) As a consequence, infrastructures with focus on low response times are generally provisioned for peak workload as depicted in Figure 4.2(b), hazarding the resulting waste of energy and therefore money, but gaining the advantage of satisfied users, which is one of the most important criteria for success in business. Ideally, the amount of online, power consuming capacity would be tightly coupled to the amount of actually required resources, as visible in Figure 4.2(d). This approach is commonly called dynamic provisioning or dynamic consolidation.

4.3 Dynamic Consolidation

As virtualization is already common in current data centers for static consolidation, methods and models for taking the next step, dynamic consolidation, should be investigated. For this purpose, virtualization offers two key benefits, dynamic resource allocation and live migration of virtual machines. Both were briefly described in Section 3.4 and Section 3.5. These mechanisms provide flexibility exploitable for gaining higher energy efficiency and are technical building blocks for my thesis. However, currently resource allocations and live migrations need to be triggered manually. The absence of an automatism causes significant overhead and unnecessary costs. To dynamically scale the power consumption of a virtualized business infrastructure to the workload situation in an automatic way, several tasks need to be carried out by a management system:

1. The VMs' allocations must be automatically adapted to their near future resource requirements to avoid both resource wastage and shortage.
2. The VMs' distribution across the servers must be automatically adapted to the workload situation, aiming at consuming the least possible power given the workload intensity and potential service level constraints.
3. The servers must be automatically powered down and woken up with regard to the near future resource demand, physically implementing the power savings.

The first task is easily implemented using dynamic resource allocation, once the target allocation is known. For implementing the two latter tasks the ability to live migrate virtual machines between physical servers without any perceptible service interruption is pivotal. It enables us to dynamically shift workload between servers with regard to a cost model, where energy costs of a computing infrastructure play an important role (next to, e.g., response times). Of course, executing actions as described above is comparably easy once the decision *which* actions to carry out is taken. The main goal of this work is to create and evaluate models that contribute to the decision process. From this point of view, energy efficient dynamic consolidation of VMs under service level constraints basically is composed of three subproblems.

1. Workload and resource requirement predictions, see Section 4.4.
2. Energy efficient mapping of VMs to servers, see Section 4.5.
3. Implications regarding Quality of Service, e.g., live migration, see Section 4.6.

In the following, a short introduction to the different subproblems is given.

4.4 Workload and Resource Requirement Prediction

The approach of this work is to save energy by minimizing the gap between required resources and online, energy consuming resources. Unless the provisioning of additional resources is feasible within the same virtualized server the VM is currently running on, the time delay of provisioning needs to be considered. Adding a virtual CPU or main memory to a VM is done within fractions of a second, but live migrating a VM or even booting a server to be able to live migrate takes a considerable amount of time. In the ideal case, additional resources would be provisioned shortly before they are actually required. Independent from whether or not forecasting is used, it is necessary to create models which connect utilization metrics to response times or service levels. These models enable us to define utilization thresholds of physical servers that must not be exceeded in the mapping phase, and are further discussed in Section 4.6.

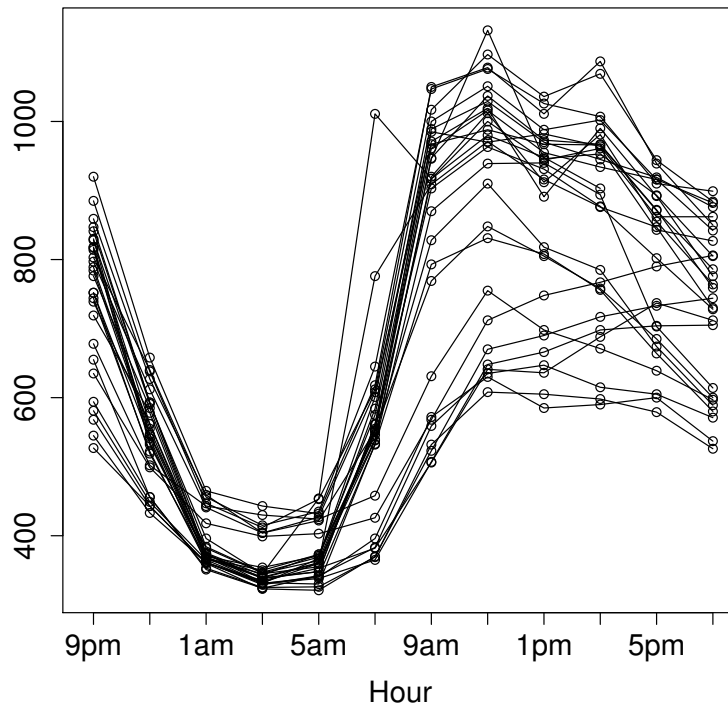


Figure 4.3: Time series of a VM's CPU utilization operated by the University of Vienna Central IT Department.

Figure 4.3 shows a CPU utilization time series plot of a VM in a business infrastructure, thankfully provided by the University of Vienna Central IT Department. It is a typical example for highly periodic patterns often found in business infrastructures, and shows periodicity at the frequencies of one day, and one week. Hence, at the same weekday and at the same hour, resource demands are very similar. Both can be explained by user behavior,

as most of them use online services during daylight hours on working days. In Figure 4.3, we can see a factor of approximately three between a typical per-day minimum and maximum. It is also clearly visible that the utilization rises quickly during the morning hours. Thus, quick reaction implemented by frequent control loop iterations or anticipatory action implemented by forecasting models is necessary to avoid resource shortage. In Chapter 5, a comparison is made between these two approaches.

Forecasting can also help in decreasing management overhead costs. Saving energy is primarily done by switching off not required servers, which implies considerable management overhead. E.g., during the phase of decreasing workload, VMs could be consolidated to less servers. VMs currently running on a server planned to be shut down need to be migrated to other servers before the shut down can happen. However, the decreasing phase will be followed by a phase of increasing workload intensity at some point in time. If the server's resources are required again, it needs to be booted and VMs have to be migrated back again. Naturally, this tedious procedure only makes sense if the workload intensity is not on the verge of rising again. The exact trade-off depends on a multitude of factors, e.g., the energy consumption during the outgoing and incoming live migrations, during server shutdown and boot, and of course the desired trade-off between quality of service and energy consumption. Additionally, powercycling a server puts heavy strain on mechanical and electrical components.

Especially with strongly periodic patterns of workload intensity, it is important to avoid flip-flop behavior of a management system. If we use forecasting models able to cope with seasonal behavior and to deliver useful predictions beyond the phase change, it is possible to avoid unnecessary overhead caused by shutting down a server shortly before the workload intensity is rising again. A practical example would be the arguable benefit of further consolidating an office's server infrastructure shortly before most employees typically arrive at the office and start their working day.

The field of time series analysis deals with such problems and offers several methods. In the scope of this work, two approaches are evaluated, the Holt-Winters method (triple exponential smoothing considering seasonal patterns) and auto-regressive moving-average methods (ARMA, ARIMA, SARIMA, etc.).

4.5 Energy Efficient Mapping of Virtual Machines to Physical Servers

Finding an energy efficiency optimal mapping of VMs to servers based on the current mapping is a multi-objective combinatorial optimization problem. For V VMs and R RMs (real machines, i.e. physical servers), there are R^V theoretically possible mappings, resulting in a huge search space even for low V and R . As mappings with at least a single overloaded RM are invalid in the context of this work, a vast amount of theoretically possible solutions found are practically irrelevant and the computation involved was in vain. Addition-

ally, practically possible solutions must be evaluated for their cost, e.g., in terms of energy, number of live migrations, server boots/shutdowns etc. An exhaustive search through R^V possible mappings is most likely intractable because of the computational demand involved, given practically interesting values of V and R . There are several well-known combinatorial optimization problems that are similar, but not identical to mapping VMs. Nevertheless, they need to be examined for their relevance to solving the problem at hand.

4.5.1 Knapsack Problem

The knapsack problem [KPP04, MT90], sometimes also denoted as rucksack problem, is an old, popular puzzle with a long history [GG66, WN67] in mathematical examination. A typical example is a thief who has a limited capacity to carry the stolen goods and therefore has to choose which items to take. The target function is to maximize the total value of the stolen goods, under the constraint that the knapsack's capacity is not exceeded. The problem can be generalized to multiple knapsacks with different or identical capacities, multiple dimensions etc. However, the problem is NP-complete, requiring algorithms that find approximate solutions in polynomial time.

A typical and well studied heuristic approach is to sort the items descending by their value per weight. Based on that, a common algorithm is to greedily pick items from the top of the sorted list until the capacity constraint would be violated, leading to the classification as a *greedy algorithm*. However, it is easy to find examples where this procedure gives results far from optimal. Hence, its output should be used only as a quick and intermediate result if the quality of the solution is important.

As mentioned above, the knapsack problem is not identical to the problem of mapping VMs. The most important difference in a business infrastructure scenario is that *all* VMs need a server to be hosted on at *all* times. In contrast to cluster-like infrastructures, where it may be indeed possible to queue or suspend jobs (here, VMs), it is prohibitive to not run a VM that provides, e.g., a web service to users or a customer's users.

As a result, all items, here VMs, must have the same value and sorting VMs based on this attribute does not help solving the problem. Furthermore, considering only a single dimension of weight and capacity is an approach too simple to be useful. For web services, at least CPU and memory resources need to be considered. Reasonable extensions are the network interface used for HTTP traffic and the disk, if primarily static files are requested by HTTP instead of dynamic pages. As a central storage for the VM's virtual disks is required for live migration of a VM, provided by, e.g., a SAN or an NFS share, the corresponding fiber-channel or network interface must be considered instead of a local disk. Of course, multidimensional knapsack problems (MDKP) are a well studied sibling of the knapsack problem, but the constraint that all VMs need a host at all times is still not adequately satisfiable.

Additionally, we have at least two knapsacks (servers), as otherwise, shifting workload is impossible. The multiple multidimensional knapsack problem (MMDKP) is typically referred to as bin packing problem, which is described in the following.

4.5.2 Bin Packing Problem

The bin packing problem is a step closer to the problem of mapping VMs to RMs than the knapsack problem. Here, we have a number of bins and a number of differently sized items. Each item needs to be packed into a bin, while minimizing the total number of required bins. Typically, items and bins have one to three dimensions, resembling different applications from cutting wooden logs (1d) or boards (2d) to packing parcels into trucks (2d or 3d). Generally, all bins have the same capacity. Due to the enormous search space, heuristic algorithms have to be used, often by first descending sorting the items by their size. Then, e.g., items are packed into the first bin that fits (First Fit Decreasing) or into the bin that fits best, i.e. after the item is packed, the corresponding bin's unused capacity is minimal (Best Fit Decreasing). For multiple dimensions, sorted lists for each dimension can be generated to fill bins in a dimension-balanced way (Balanced Fit).

Hence, the fact that we can not choose *what to pick*, but instead we have to choose *what to put where*, is appropriately considered. Also, multiple resource dimensions can generally be regarded, although different algorithms might be of interest depending on the number of dimensions. However, there is a subtle but very important difference when mapping VMs to servers and considering at least two dimensions. As visualized in Figure 4.4, the items must be aligned corner to corner, and not edge to edge as in typical bin packing problems, as the VMs requirements would not be accounted correctly to the corresponding server's utilization in all dimensions. This is commonly referred to as vector packing [BS96, KLMS84, Dyc90].

There is another major difference due to the scenario of this work. If all servers were of the same capacity and would consume the same amount of energy at the same level of utilization, the target function to minimize the number of used servers would be equivalent to minimize the power consumption. However, in a realistic scenario, we must assume servers to be heterogeneous with respect to both their resource capacities as well as power consumption, aggregated as heterogeneous energy efficiency. This assumption stems from the real world observations that a) most data center infrastructures grow over time, with different server models added in each extension phase and b) even if the whole infrastructure is deployed at once, it is often partitioned into different models or vendors to reduce the risk of being affected of widespread hardware production failures or to regard different requirements. Especially when dealing with virtualized infrastructures, where the virtualization layer abstracts from the underlying hardware layer in several ways, it is crucial to take the highly probable heterogeneity in hardware capacity and power consumption into account.

4.5.3 Virtual Machine Mapping Problem

In the following, the multi-objective combinatorial optimization problem involved in finding more energy efficient virtual machine mappings, denoted as Virtual Machine Mapping Problem, is outlined. It can be understood as an offline [VAD⁺08] packing problem, as the number and size of all items and bins is known at the beginning. Further, it is not necessary

to immediately decide where to put a specific item, rather we can choose which item to put where.

VMs In this case, the VMs are the *items* to be packed or mapped. We have V virtual machines, and the resource requirements for the VM i with $0 \leq i < V$ are represented by a D -dimensional vector $\vec{v}_i$, where $v_{i,d}$ represents the resource demand of VM i in component d , with $0 \leq d < D$. Consistently with the above described scenario that *all* VMs need to be packed, VMs have no profit or value, only weights in D dimensions. In the scope of this work, $D = 2$, representing CPU and memory demand, which are the most important resources for typical web server purposes. This could be extended to consider additional resources, e.g., network bandwidth, storage subsystems, etc.

RMs The physical servers or RMs (real machines, as opposed to virtual machines) are the *bins* sheltering the *items*. We have R RMs, and the resource capacities of the RM j are represented by a D -dimensional vector $\vec{r}_j$, with $0 \leq j < R$, where $r_{j,d}$ represents the resource capacity of RM j in component d , with $0 \leq d < D$.

Mapping VMs to RMs In order to describe a solution to the combinatorial problem, a binary mapping matrix X is used, with R rows and V columns, where $X_{ji} \in \{0, 1\}$ and $X_{ji} = 1 \iff v_i$ runs on r_j , 0 otherwise. A simple example to illustrate the concept of a mapping matrix is given as

$$X = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \quad (4.1)$$

The meaning of this example matrix is that r_0 hosts v_0 and v_3 , r_1 hosts v_1 and v_2 and r_2 hosts no VM and could be possibly turned off. As a VM must run at exactly one RM at a given time:

$$\forall i, 0 \leq i < V : \sum_{j=0}^{R-1} X_{ji} = 1 \quad (4.2)$$

Evaluating Mappings Given a valid solution to the combinatorial problem represented by a mapping matrix, the utilization of all servers caused by the mapping must be evaluated, with respect to all resources that are to be considered, here CPU and memory. The resource utilization is compared with configurable thresholds later on to identify and penalize overloaded servers. The CPU utilization is of additional interest as it is used for estimating the power consumption of a server.

Formalized, calculating the utilization of a server is done by multiplying the corresponding row in the mapping matrix with a vector representing the resource requirements of each

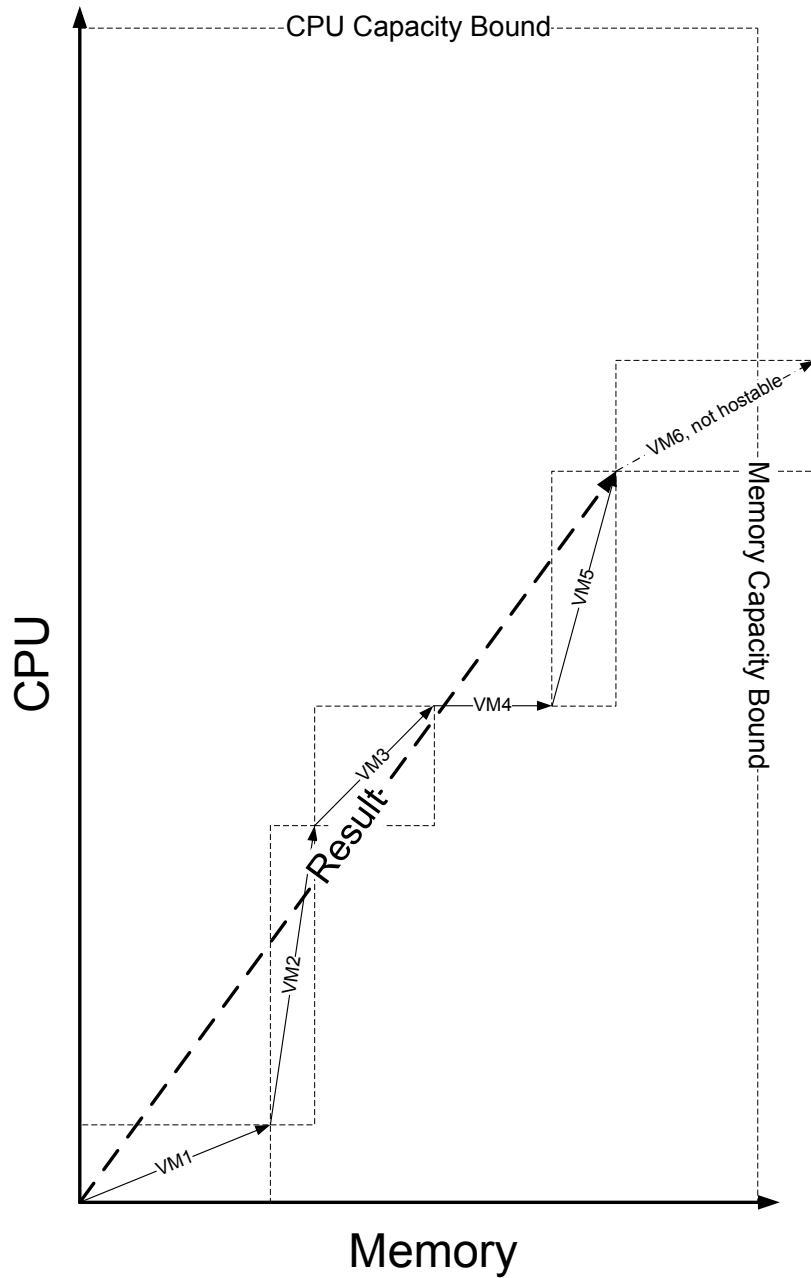


Figure 4.4: VM mapping, addition of vectors (VMs) until a capacity bound would be violated. Additionally, servers are heterogeneous regarding their CPU and memory capacities, and energy consumption.

VM in a specific resource category. E.g., based on the mapping matrix example of Equation 4.1 and an example VM memory requirement vector of

$$\vec{v}_{mem} = \begin{pmatrix} 300 \\ 650 \\ 1140 \\ 470 \end{pmatrix}, \quad (4.3)$$

the server memory utilization $\vec{r}_{mem}$ is calculated by

$$\begin{aligned} \vec{r}_{mem} &= X \cdot \vec{v}_{mem} = \\ &= \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 300 \\ 650 \\ 1140 \\ 470 \end{pmatrix} = \\ &= \begin{pmatrix} 1 \cdot 300 + 0 \cdot 650 + 0 \cdot 1140 + 1 \cdot 470 \\ 0 \cdot 300 + 1 \cdot 650 + 1 \cdot 1140 + 0 \cdot 470 \\ 0 \cdot 300 + 0 \cdot 650 + 0 \cdot 1140 + 0 \cdot 470 \end{pmatrix} = \\ &= \begin{pmatrix} 770 \\ 1790 \\ 0 \end{pmatrix} \end{aligned} \quad (4.4)$$

Target Function In the paragraphs above, a description of the combinatorial problem was given. As mentioned, a multi-objective combinatorial *optimization* problem needs to be solved, thus requiring the formulation of a target or cost function that is minimized. Here, the cost function is composed by several parts representing different categories and issues. Each part contributes to the total cost with a configurable weight (denoted as *penalty*), to be able to examine various priorities. E.g., the disposition to save energy in an extremely aggressive way with the possible short-time side-effect of overloaded servers will vary across data centers. To formalize, the target function is to minimize the total cost C of a mapping

$$C = \sum_k c_k \cdot w_k, \quad (4.5)$$

where c_k is the cost in category k and w_k is the weight for the corresponding costs in category k . Genetic algorithms typically use $\frac{1}{C}$ or $\frac{1}{C^2}$ as a measure of an individual's fitness. The exact target function and its parameters are described in detail in Chapter 5, where it is used in a cost model driven dynamic consolidation approach.

4.6 Quality of Service

4.6.1 Energy Costs, Quality of Service and Revenue

For business purposes, saving energy is generally not worth the effort if quality suffers. Depending on the business model, providers are earning money by customers buying products using an online shop or visiting the advertisement links on website. If the web server backing such a service is overloaded, users will be annoyed and either immediately stop using the service or will have less incentive to come back again. If the business model is IaaS (Infrastructure as a Service), customers will soon move their VMs to other IaaS-providers when their own customers experience bad quality of service.

A simple, figurative visualization of the importance of service quality is comparing the cost of a kWh of electricity and the amount of time a server can be run with that amount of energy. If we make the highly conservative assumption of a price of 1 Euro for a kWh, a server consuming 500 W in a data center with a PUE of 2.0, then this server can be run for an hour. If the server is switched off instead to save energy, a single order with a profit margin of 1 Euro lost due to bad service quality within an hour renders all savings void. The cheaper the electricity is, the more likely it is to lose savings by too aggressive measures.

There are several different problems involved in sustaining service quality. The most obvious perspective is how much resources a VM requires for meeting service levels. Depending on the exact hypervisor, mechanism available for resource allocation tuning are, e.g., memory ballooning and vCPU hot-plugging.

Furthermore, it is necessary to consider to which level a physical server can be utilized until its VMs and their services are suffering from low QoS. In contrast to number-crunching cluster infrastructures, it is not beneficial to clutter up servers with VMs as queuing issues will arise, causing high response times. It would be desirable to have a well reasoned resource utilization threshold, meaning a level of server utilization which should not be exceeded to assure good QoS. Thresholds can be easily considered by all classes of VM mapping algorithms. Deterministic heuristic algorithms profit from hard thresholds in terms of algorithm complexity, as every possible action within the algorithm must be stated explicitly to generate a solution. Other classes of algorithms, e.g., meta-heuristic genetic approaches, can try various actions (mutations, etc.) within a population of possible solutions and then evaluate the cost of their actions to pick the best solution and are hence far more flexible.

This problem of thresholds can be split into two subproblems. A hard utilization threshold is used in the mapping phase. A server is enforced to not accept a planned incoming migration, if a utilization threshold was exceeded. The VM in question must be assigned somewhere else. Nevertheless, a threshold can still be exceeded later on when the control loop has finished its iteration and waits until the next interval begins. In the meantime, the load can rise, potentially causing a threshold to be exceeded. At the next measurement iteration, it is detected.

This event must be penalized to appropriately regard the implications on QoS. However, a

simple 0/1 penalty, whatever the penalty weight exactly is, will be too simple. A violation by 2 percentage points of utilization will be most likely drastically less severe regarding QoS than a violation by 20 percentage points. Thus, a non-linear model should be used, with exact parameter values depending on the specific kind of service. A simple approach would be the use of a Markovian M/M/1 queue, with the infrastructure operator supplying the service rate and the maximum allowed response time. Then, an approximation of the portion of requests that could not be served within the maximum response time could be calculated. However, the exact consequences are depending on business economics circumstances and are not within the scope of my thesis. The most important factors are 1) the subjective Quality of Experience, meaning at which level of response times users will turn away and 2) the business model, meaning the amount of revenue lost by users turning away. However, these questions are out of the scope of my thesis, which provides general models taking the specific parameter values for a custom scenario as input.

As a result, a hard threshold is used in the mapping phase to inhibit overload situations here, but if overloads occur between measurement intervals, they are penalized by a non-linear model. This approach is used in Chapter 5.

The importance of thresholds has further aspects. Servers need to have resource reserves for peaks too shortly timed for evasive actions like live migration, outlined in the following Section. Furthermore, the amount of time between control loop iterations is highly important. If it is very long, substantial overload situations could be averaged out over time and left unnoticed. This problem is also investigated in Chapter 5.

4.6.2 Quality of Service During Live Migration

Live migration is another aspect of high importance for service quality. Its influence on service quality needs to be investigated, as it will be a frequently used management operation. Additionally, it is a resource intensive process, especially the publicly available iterative pre-copy implementation. The whole memory address space has to be scanned iteratively for dirty pages, which have to be transferred over network. During a dirty page's contents are copied to a network buffer, it can not be simultaneously written to because of physical reasons. Additionally, there might be locking mechanisms necessary to ensure data integrity at the software or operating system level. Thus, live migration consumes CPU, network and memory resources.

With dynamic consolidation, it will often occur that a server is experiencing a high level of utilization due to rising workload. Then, a number of VMs will be migrated to other servers. As described, this process consumes additional resources, probably causing even worse quality of service for the VMs currently hosted. In the worst case, both the VMs and the migration process do not get enough resources due to severe resource contention. Then, thrashing could occur, leading to slow migration progress and unnecessarily severe negative impact on quality of service. Typically, there will be room for choosing which VM or VMs to live migrate to other servers. The hypothesis to evaluate is that migrating more VMs that are less utilized could be better for service levels than vice versa.

As a consequence, the impact of live migration on quality of service is an important issue, and only little work has been done in the past, giving much room for improvement. Especially, the correlation between VM utilization metrics and cost of live migration has not been investigated yet.

Voorsluys et al. [VBVB09] have evaluated the cost of virtual machine live migration for a two-tier Web 2.0 scenario. The objective of their experiments is to quantify the slowdown and downtime experienced by the application under test during live migrations. While migrating a virtual machine, they track the HTTP response time SLA (90% and 99% limits). They come to the conclusion that live migration overhead is “acceptable but can not be disregarded” [VBVB09], depending on the SLA. The authors only consider a fully loaded VM. Different workload situations should be considered, leading to a more differentiated view. Furthermore, this thesis considers a different kind of workload, a different hypervisor and a different type of network storage. These differences in the experiments could lead to a more general perspective. In Chapter 6, this thesis identifies the most significant utilization metrics and presents models able to predict the service level during live migrations.

Akoush et al. [ASR⁺10] have investigated parameters affecting the time required for a live migration. They have evaluated different sets of parameters, e.g., page dirtying rates up to 300000 pages/second, typically equivalent to 1.2 GiB/s, link speeds from 100 Mbps to 10 Gbps, varying VM memory allocations, etc. They give theoretical lower and upper bounds regarding the migration delay. However, these bounds are very wide. Further, they use two simulation models to predict migration delays. The first is based on a average page dirty rate, which is only useful when the rate is fairly stable. The second is a history based page dirty rate, useful for variable page dirtying rates. Their efforts result in models able to predict the migration time and downtime within 90% accuracy for synthetic and real-world benchmarks. Nevertheless, their approach requires tracking of dirty pages, which makes resource intensive low level adaptations necessary within the hypervisor. Among other benchmarks, the authors have employed SPECweb, but have not studied quality of service. Especially, a link to utilization metrics has not been established. Duration and downtime of a live migration are not at all directly equivalent to the service level during the migration.

Breitgand et al. [BKR10] have examined a possible trade-off between two competing goals. The primary goal of iterative pre-copy migration is to minimize the time required for the final iteration. During this downtime, the VM is paused, and its remaining dirty pages are transferred to the destination server. On the other hand, it is important to achieve an acceptable quality of service during the pre-copy phase. Breitgand et al. [BKR10] have examined the scenario of an in-band live migration, meaning that the same network link is used for the live migration and virtual machine services. Obviously, during live migration these two facilities compete for resources and contentions are likely. The authors have carried out a simulation using real traces, using the portion of requests not served with a maximum response time as a cost function. Although simulated and calculated results are similar, an empirical proof was not presented. They come to the conclusion that quality of service during live migration is depending on the available network bandwidth. This result is comprehensible, but the underlying scenario and assumptions are rather exotic. Even

today, several networks exist in typical data centers. Common purposes next to workload are storage and management. It is reasonable to assume that live migration traffic is carried by a dedicated network. Furthermore, scanning the memory address space for dirty pages is a CPU intensive process, so there is at least a second resource likely to become a bottleneck during migration.

Liu et al. [LXJ⁺11] have presented a model able to predict the time an iterative pre-copy live migration takes to complete, based on measuring the page transfer rate and the page dirtying rate. Information about dirty pages is gathered by adapting the `Xen` hypervisor. They have evaluated migration delay, downtime, network traffic and energy consumption for various applications running inside a migrated VM, e.g., SPECweb2005. However, quality of service metrics, e.g., response times and service levels, are not part of their evaluation.

In Chapter 7, an evaluation carried out how estimating the remaining migration time as presented in [LXJ⁺11] can be used to optimize QoS during migration for a KVM hypervisor and a web server VM, without making changes to the hypervisor itself. First, their model's validity is verified for the experimental setup used throughout this thesis. Then, it is extended it by a pro-active service level management component, taking evasive action if a service level target is predicted to be violated during the remaining live migration.

Part II

New Approaches in Energy Efficient Data Centers

Chapter 5

MetaVisor - Cost Model Driven Dynamic Consolidation

5.1 Overview of Chapter

As discussed in Chapter 2, a vast majority of servers in data centers are underutilized for a significant amount of time, and operate at a very low rate of efficiency while still consuming huge amounts of energy. In this chapter, an investigation of how dynamic consolidation and workload forecasting can help to increase the energy efficiency of a virtualized, heterogeneous server infrastructure is presented. The evaluation is based on real utilization traces of a production system operated by the University of Vienna's central IT department. The traces contain the CPU utilization of more than 30 VMs over a period of four weeks. Due to the lack of variability in the memory utilization, we concentrate on the CPU utilization. These VMs offer all kinds of services to students, staff and other visitors. The utilization traces are used to investigate a business infrastructure scenario, where energy costs are just *one* of several parts of operational costs. A novel cost model using configurable penalties for the most important operational cost categories is presented. The total costs of a bin-packing related heuristic and a new genetic VM mapping algorithm used for dynamic consolidation (GA) and load balancing (LB) are compared. The benefit of forecasting the resource demand and shorter measurement intervals are shown. The flexibility of a meta-heuristic genetic algorithm is demonstrated. The GA and LB approaches are directly influenced by penalizing cost parameters. The cost model allows easy adaption by infrastructure operators to implement custom priorities and optimization goals.

5.2 Introduction and Related Work

The ecological and economical issues of data center power consumption were discussed in detail in Chapter 2. Despite many efforts in the last years, it is growing from year to year. Especially static consolidation of mostly idle servers using virtualization has become state of the art, but can increase energy efficiency only to a certain degree. An important reason is that business infrastructures generally have to bear highly variable workload, often with

strong periodic components. Static provisioning for dynamic resource demand has only a limited potential if quality of service is important. E.g., provisioning resources for day hours is a willful waste of resources during the night hours, and provisioning resources for night hours will typically result in catastrophic quality of service and annoyed users during the day. In both cases, the absence of an automatism causes unnecessary costs. Especially in environments with periodic workload variability (e.g., business infrastructures with 9-5 cycles), automatic resource allocation mechanisms are most beneficial, as presented in [BKB] and discussed in Section 4.2. Until more energy-proportional computing hardware is available off-the-shelf, the alternative is to increase the average utilization of servers to a region of higher efficiency, as shown in Section 2.2.1. This can be done by a management software which measures the utilization of the infrastructure, calculates a more energy efficient mapping under QoS constraints and executes the target distribution of VMs using live migration. Werner Vogels, CTO and Vice President of Amazon, has mentioned [Vog08] that achieving 40% utilization is a “major success for workloads that are not strictly CPU-bound” [Vog08], and achieving 50% is the “holy grail” [Vog08] of server consolidation.

Efforts to operate a data center infrastructure energy efficiently by a management software have become a popular topic in the literature. However, there is a multitude of different scenarios, goals, assumptions and constraints in these works. A short discussion of those works most related to my thesis is given in the following. The scenario of this approach is presented in detail in Section 5.4.

Bobroff et al. [BKB] have studied the potential of dynamically consolidating periodic workload regarding required online capacity and possible SLA violations. They analyze several different basic workload profiles and show that periodic workload is most suitable for dynamic consolidation, implemented by a Measure-Forecast-Remap algorithm. The mapping step is done by a first-fit bin packing heuristic, minimizing the number of online servers. This target function is only applicable to completely homogeneous server pools, and even under such restricted circumstances, bin packing heuristics often deliver results far from optimal. Some examples are mentioned below. Although energy costs are mentioned in [BKB], they are not part of the evaluation.

Beloglazov et al. [BB10] have presented an architecture for energy efficient Cloud computing, trying to minimize energy costs under QoS constraints. For mapping VMs to servers, they apply an adapted version of the Best Fit Decreasing heuristic. These heuristics were originally developed for tasks related to bin packing. They can happen to create solutions far from optimal, especially when a server infrastructure is assumed to be heterogeneous. Then, a solution using the minimum number of bins (servers) is not necessarily the solution consuming the least energy. A sorting criteria is also required for the servers to decide which of them are to be filled first. In [BB10], a VM is mapped to the server that shows the least increase in energy consumption. We interpret that this criteria is also used for the first VM, so that the increase in energy consumption is then dominated by the chosen server’s idle power. However, such measures can not avoid that the final mapping is obviously not consuming the least energy. E.g., it could have been better to choose a different server, although it is showing a bigger power consumption increase in the first step, but offering a smaller slope in

the following due to a different CPU model, finally leading to a lower energy consumption. There are many examples for such power consumption profiles in the SPECpower_ssj2008 benchmark results, as discussed in Section 2.2.3. Kumar et al. [KTK⁺09] have developed and evaluated vManage, which places workload under consideration of power, thermal and performance aspects using stabilized first fit and best fit heuristics.

Stillwell et al. [SSVC10] have evaluated several bin packing related heuristics for a *static* workload allocation problem. Their examination is very detailed, but are not applicable to the problem of dynamic workload allocation, where demands change over time. Furthermore, servers are assumed to be homogeneous, hence the examination at cluster infrastructures. Borgetto et al. [BDPS09] have proposed the novel metric of energetic yield, which is the gap between the amount of resources a job would need, and the amount it actually got. The aim is to minimize this gap during workload placement. Servers are homogeneous too, which is realistic for clusters, but not for business infrastructures. Entropy [HLM⁺09] is an open-source implementation of dynamic consolidation of virtual machines. It is focusing on HPC clusters, where broader assumptions can be made and higher packing rates of VMs are possible due to the typical on/off characteristic of number-crunching tasks. pMapper [VAN08a, VAN08b] and Snooze [FRM⁺10] are further examples for systems focusing on cluster infrastructures. The work by Hyser et al. [HMGW] also examines infrastructures with identical hosts. It aims for load balancing using a simulated annealing algorithm. In this Chapter, load balancing is implemented by a meta-heuristic genetic algorithm and penalizing utilization variance.

Genetic algorithms [Gol89] are a frequently used approach to bin packing optimization [FD92], especially for multi-objective optimization [FF⁺93]. The idea of genetic algorithms can be seen as a descendant and representative of evolutionary algorithms [FOW66]. Xu and Fortes [XF10] have evaluated the ability of a genetic algorithm extended with fuzzy logic to trade-off between resource wastage, power consumption and thermal dissipation. They compare their results to several bin packing related heuristics, which are clearly outperformed by the genetic approach. However, they only use a single mutator, and do not consider that servers could be overloaded between two control loop intervals. Finally, there have been increasing efforts [LWHC07, MBL⁺09, YCP05, PJS10] in implementing genetic algorithms using a general purpose GPU framework, e.g., CUDA.

The isolation between guests and the hypervisor and management problems stemming from that is a topic frequently found in the literature. Wood et al. [WSVY07] have proposed a black-box and a gray-box strategy for VM live migration. Briefly, the former treats VMs as a black-box, while the latter introspects the VM and its applications regarding status and utilization metrics. The goal is to quickly detect and resolve hotspots. Here, the gray-box strategy helps to detect memory pressure better, as the hypervisor is unaware of the amount of free memory within the VM. Jones et al. [JADAD06] have presented Geiger, a system to infer information about the buffer cache and virtual memory system from a guest to the hypervisor. Wood et al. [WTLS⁺09] have also investigated the potential of memory page deduplication, often denoted as *page sharing*. For long time, this feature was exclusive to VMware ESX. Nowadays, the Linux kernel and KVM can provide this, here called KSM

(Kernel Samepage Merging). Chen et al. [CIL⁺07] have presented an approach to transform high-level service level objectives to low-level system utilization thresholds using queuing network models.

Nathuji and Schwan [NS07] have presented VirtualPower, a hypervisor extension that allows VMs to have a software CPU power state, in addition to the hypervisor's conventional power states a CPU offers in hardware. This way, both can be coordinated to save energy. Basically, this allows to use the capping abilities of DVFS also in a virtualized setup. Zhang et al. [ZBG⁺05] have even advocated “friendly” virtual machines that self-adapt their allocation to their demands. Stoess et al. [SLB07] have developed a low-level, fine grained energy account system for hypervisors to allow power capping for guests. Weng et al. [WWLL09] have evaluated different virtual CPU scheduling algorithms for the Xen hypervisor and propose a hybrid approach to consider both high-throughput and concurrent VM workloads.

Kusic et al. [KKH⁺08] try to reduce the power consumption of a heterogeneous infrastructure by pro-active limited lookahead control. Depending on the workload, they can save 20% to 30% of power with little service level violations. Extending the controller by risk-awareness further increases the power savings and decreases service level violations. However, instead of live migrating virtual machines, they implement workload placement by starting and stopping VMs on different servers. Using control theory to ensure QoS for multi-tier applications was also studied by Padala et al. [PSZ⁺07] and Kalyvianaki et al. [KCH09].

Burge et al. [BRW07] have examined the potential of saving power by request scheduling in heterogeneous data center scenario. They employ economic models considering the varying patience of customers, job length, consumed energy, job revenue, cancellation costs, etc. They come to the conclusion that heterogeneity in data centers can actually be exploited. They also show that even very simple heuristics, e.g., shutting down a server that has been idle for the last n minutes, can save a significant amount of energy. However, they focus on the decision where and when to deploy a customer's job. Once it is deployed, it can not be migrated elsewhere. With finite job lengths that are typical for cluster infrastructures, this is a justifiable assumption. In business infrastructures, most jobs are typically so long that they have to be assumed infinite. Steinder et al. [SWC08] have investigated a similar scenario.

Zhao and Huang [ZH09] have implemented a load balancing algorithm using live migration for Eucalyptus[NWG⁺09], an open-source Cloud Computing platform offering Infrastructure as a Service (IaaS). They use a distributed load balancing algorithm to avoid centralization. However, they do not consider the memory capacity of the servers at all. OpenNebula[MVML09] is a platform similar to Eucalyptus. Choi et al. [CKSC08] have proposed a machine learning framework that learn from experience when and where to migrate a virtual machine in case of overload. As their approach requires all possible migrations to be evaluated, scalability problems for bigger infrastructures are likely to occur. Breitgand et al. [BE11] have formulated and evaluated an extension to the general assignment problem [Kuh55], called Elastic Services Placement Problem and aiming at Cloud Computing

platforms. The goal is to fulfill service level agreements, but the energy consumption and resulting costs are left for future work. The problem was also formulated [CP09] as generalization of the knapsack problem, called multi-dimensional, multiple-choice, multiple knapsacks problem. However, the target function is to minimize the number of used physical machines, which is equivalent to minimize energy consumption only for completely homogeneous infrastructures.

5.3 Open Issues and Own Contribution

As discussed in detail in Section 5.2, there are several open issues in the area of energy efficient dynamic consolidation. Most of the existing efforts are focusing on number-crunching cluster infrastructures, where a different set of assumptions and research questions is of relevance than for business infrastructures. If servers are assumed heterogeneous, then heuristics are often the only mapping algorithm that is evaluated. Especially, realistic assumptions for a business infrastructure scenario and a trade-off between multiple categories of operational costs, e.g., energy, QoS, and management overheads, are insufficiently researched.

In this chapter, we present *MetaVisor*, a prototypical infrastructure management software that automatically manages a set of hypervisors and their VMs using cost model driven dynamic consolidation. The goal is to minimize energy consumption while avoiding underprovisioning, resulting in service degradation and SLA violations. The amount of online, power consuming resources should at all times be as close as possible to the amount of actually required resources, as depicted in Figure 5.1. We compare two implementations of dynamic consolidation with a load balancing policy.

To reflect a business infrastructure scenario, servers are assumed to be heterogeneous in terms of CPU performance, amount of main memory and energy efficiency, as data centers often evolve over the time and shelter servers of different generations. The heterogeneous energy efficiency of RMs can actually be taken into account. VMs are also assumed to be heterogeneous in terms of required CPU and memory resources. Furthermore, their resource requirements can vary over time, possibly even periodically. In general, at least CPU and memory utilization should be taken into account when mapping VMs, so balancing a server's resource components is an important artifact of every mapping heuristic. The scenario is outlined more detailed in Section 5.4.

The integration of *MetaVisor* into data center infrastructures is described in Section 5.5. It explores the current mapping of an infrastructure using `libvirt` and synchronously gathers utilization data of both servers and VMs using *MetaStat*, a custom developed utilization logger. It detects main memory pressure and queuing issues within VMs and updates resource allocations (number of vCPUs, amount of main memory) accordingly. If activated and enough historic data is available, forecasts are created to allow anticipatory action.

Moreover, status costs (energy consumption and overload situations) and reconfiguration costs, here most importantly live migration and server state change overheads, must be considered. Therefore, we define a cost model in Section 5.6, where several cost categories

contribute with configurable weights (*penalties*) to a sum of weighted costs, which we try to minimize. This allows several trade-offs, e.g., between energy costs and SLA violations, or between energy costs and live migration overhead, etc.

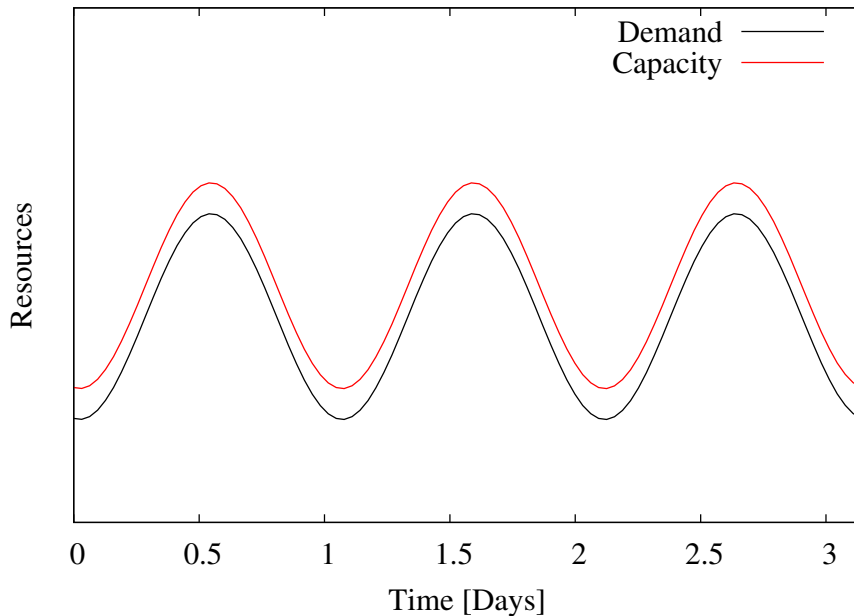


Figure 5.1: Dynamic provisioning for actual load to increase the energy efficiency of a computing infrastructure.

The problem of mapping VMs to physical servers in a way that minimizes power consumption, but does not lead to severe Quality of Service violations, is a multi-objective combinatorial optimization problem. It has to be solved frequently as the VMs' resource demands are usually dynamic. Due to the computational complexity, exact solutions are practically intractable. We evaluate a greedy heuristic stemming from the problem of vector packing and a flexible meta-heuristic genetic algorithm, as described in Section 5.7. If the resource demand decreases, the infrastructure is consolidated by live migrating VMs to be able to shut down servers. If the demand rises, the infrastructure is scaled to the resource demand by waking up servers, before live migrations are taking place.

Ideally, this happens pro-actively by making forecasts about the workload development. The field of time series analysis offers several methods. To quickly and reliably examine longer time spans and different scenarios, the control loop is evaluated in a simulation mode *MetaSim*, using real utilization traces. A GUI frontend for *MetaSim* showing a simulation test run is visible in Appendix A. The utilization traces are briefly presented in Section 5.8. We evaluate two forecasting approaches, the Holt-Winters method (triple exponential smoothing considering seasonal patterns) and seasonal auto-regressive moving-average methods (ARMA, ARIMA, SARIMA, etc.). They are further discussed in Section 5.9.

A proper base of comparison is required to evaluate algorithms implementing dynamic consolidation. We regard it reasonable to collate dynamic consolidation against load balancing, as they represent opposed paradigms. A load balancing policy always uses all available resources, and ignores energy costs in favor of very low SLA violation probability. We implement load balancing by a meta-heuristic genetic algorithm that penalizes utilization variance. Effectively, the whole server pool is used at all times and overload situations are extremely improbable.

5.4 Scenario, Assumptions and Constraints

In the following, the most important assumptions and constraints for designing the *MetaVisor* infrastructure management software to increase energy efficiency are discussed.

Periodicity and Variability of Workload. Focusing on a web server scenario, where the overwhelming part of workload is created by human users, it is realistic to assume high variability and often, strong periodicity of workload intensity. Especially the CPU resource demand will show correlation to the workload intensity. Of course, there will be spontaneous peaks and bursts, e.g., caused by new hyperlinks from popular web sites or search engine crawlers indexing a website. Additionally, seasonal or long-term trends may exist.

Heterogeneous Servers. As already outlined in Section 4.5.2, data centers used for business purposes are typically sheltering servers from different generations and vendors. Completely homogeneous data centers are very seldom and mostly used for number-crunching cluster installations which are deployed at once. Cluster infrastructures are not within the scope of this work, which is focusing on existing, heterogeneous business infrastructures. There, servers are typically heterogeneous with respect to both their resource capacities as well as power consumption, aggregated as heterogeneous energy efficiency. Additionally, the heterogeneity is generally exploitable for increasing the correlation between workload intensity and power consumption. E.g., imagine a server with rather low CPU resources, but high memory resources. Such a server could be used for sheltering VMs during the night times where only very little workload occurs. Although CPU power saving mechanisms are getting more and more sophisticated, the less CPU cores such a machine has, the less power is generally required.

Static Offline Server Power Consumption. Servers switched off for saving power are assumed to consume a static, small amount of power. To allow a remotely triggered boot, only a small amount of power is needed for receiving the wake-on-LAN magic packet. However, the specific amount of power can vary depending on the exact server, which is appropriately considered, as all consumption data is configurable on a per-server level. In

the simulation mode *MetaSim*, server data from the SPECpower_ssj2008 benchmark is used, and their standby power consumption is unavailable. Hence, an estimation of 10 W standby (ACPI S4) power consumption is used.

Storage and Network Infrastructure Consumption. Although a central storage system, e.g., a SAN or NFS-share, is required to be able to live migrate virtual machines between physical servers, the consumption of this equipment is out of consideration. The same holds true for network equipment like switches. The reasons for this are that on the one hand, these devices have a highly static power consumption behavior, with only marginal influence of the workload intensity [HDP09] and thus allowing the approximation of a static power consumption. Hence, the additional consumption caused by these devices can be easily added, independently of the workload intensity. On the other hand, such devices are existing anyways within data centers and independently from the aim of this work. Switches are required for connecting servers with each other and their clients, and some kind of central storage for virtual machine disk images is already used within virtualized data centers with high probability, as it allows better storage redundancy, easier management, quicker fail-over and therefore higher availability, and more straightforward backups.

Linearly Correlated Server Power Consumption. As discussed in Section 2.2.1, a linear correlation between the CPU utilization and the dynamic power range of a server is a first approximation to estimate the power consumption of a server given a specific utilization. The exact values for P_{idle} and P_{peak} are of course depending on the specific server.

$$P = P(U_{CPU}) = P_{idle} + (P_{peak} - P_{idle}) \cdot U_{CPU} \quad (5.1)$$

Negligible Virtual Machine Arrival and Departure Rates. In contrast to related efforts focusing on VM deployment issues, virtual machine arrival and departure rates are considered to be negligible here. In business infrastructures, there may be a new project or customer from time to time that requires new virtual machines, or virtual machines used for testing purposes are not needed any more and shut down. Nevertheless, there is no steady come and go of virtual machines.

Equally Important VMs. In contrast to cluster infrastructures, where jobs could possibly be queued before submission or suspended during execution, here, all VMs need to have a hosting physical server at all times. The job lengths are assumed to be infinite. For the evaluation in this chapter, all VMs have the same priority.

Utilization Thresholds. Servers are not allowed to be fully utilized to avoid queuing issues. In fact, they are required to have reserves for short peaks, where even live migration would be too slow as a useful countermeasure. Additionally, live migration itself requires

a considerable amount of CPU and network resources. Hence, utilization thresholds need to be configurable, giving the relative amount of a server's resources that should be used by VMs, denoted as quota capacity, or usable capacity. If it is (predicted to be) exceeded, the mapping of VMs must be automatically rearranged.

Management Action Overhead. Management action, e.g., live migration, server boot and shutdown impose additional costs. These need to be considered and compared with the benefit of taking an action, e.g., the potential energy savings.

5.5 Prototype Architecture

MetaVisor is a prototypical implementation of a cost model driven dynamic consolidation approach. It is written in Java 1.6 and heavily uses `libvirt`¹, a light-weight virtualization API written in C with drivers for a growing set of hypervisors, through Java bindings. The focus is on Linux systems, Debian Squeeze AMD64 is used for RMs and VMs, and KVM as hypervisor.

One of its design principles is to minimize the need for manual user interaction. To add a hypervisor host (RM), only the following input is required: A FQDN or IP address, a MAC address for WakeOnLAN, the standby, idle, and peak power consumption, and the SPECpower_ssj2008 benchmark result (*specpoints*) used as energy efficiency sorting criteria necessary for the greedy heuristic.

On each RM, a `libvirt` daemon and `sshd` with public key authentication are necessary. On all VMs and RMs, a custom developed utilization logging software *MetaStat* is required. It has a small footprint, uses `pidstat` and `sar` for measuring the utilization, and ensures highly synchronous measurements, as it is triggered by network broadcasts.

For development, VMs with Apache2, PHP5-FastCGI and MediaWiki were used as web-tiers and VMs with MySQL5 and dumps of the Latin Wikipedia are used as database-tier. For workload generation, a custom developed software *MetaBench* was used. It allows to automatically create and execute a trace of HTTP requests based on a list of template URLs and a time series of workload intensity. As storage for VMs, an NFS share over GBit Ethernet was used.

In Figure 5.2, the integration of *MetaVisor* into a data center infrastructure is illustrated. *MetaVisor* runs on a management server (or VM) outside the set of RMs/VMs it manages. The added power consumption is insignificant for business scenarios as *MetaVisor* only needs little computing power and existing infrastructures will typically consist of at least dozens of servers, networking equipment, and a SAN. A SAN is necessary for doing VM live migrations, as described above, and is assumed existing in any professional data center. RMs can even be completely disk-less by booting from the SAN (e.g., FiberChannel,

¹<http://libvirt.org>, last visited on 5.9.2012

iSCSI). A screenshot of *MetaGUI*, a visualization frontend of *MetaVisor*, showing an empirical test run is visible in Appendix A.

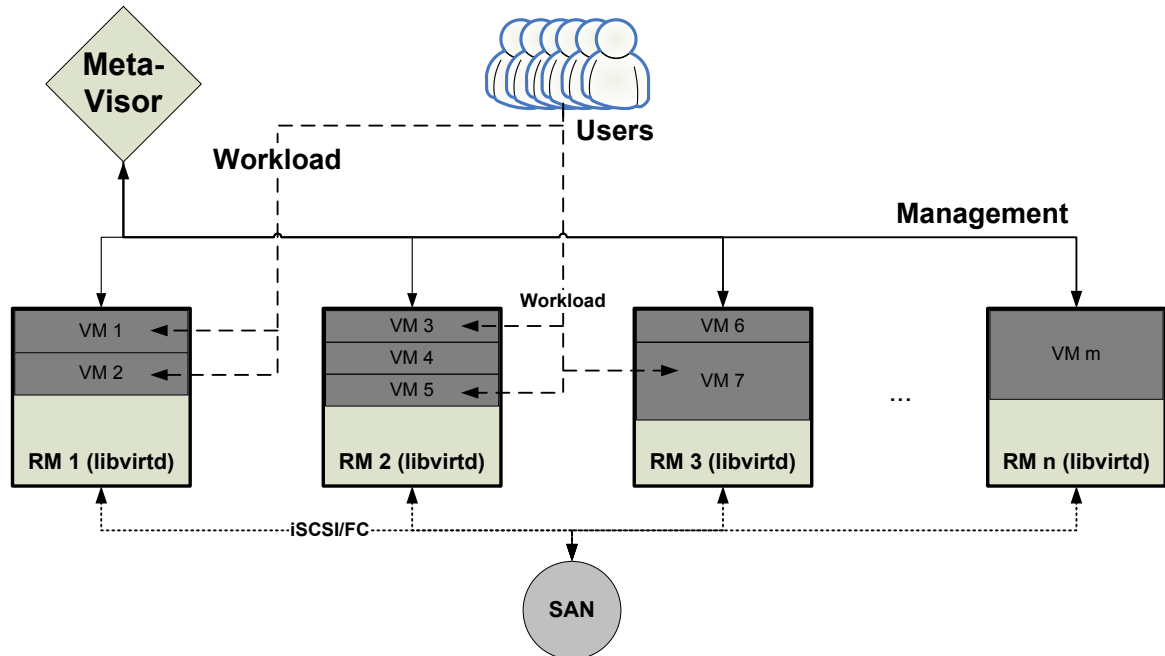


Figure 5.2: *MetaVisor*'s integration into a data center infrastructure.

In the following, the main control loop of *MetaVisor* is briefly described. The below-mentioned steps are executed in an endless loop with configurable iteration frequency.

Step 1: Resource Identification

Provided with a list of RMs, *MetaVisor* automatically explores these RMs using `libvirt`. For each RM, it identifies the amount of main memory, the CPU architecture, the number of CPU cores and their frequency, and the total CPU resources. Additionally, it detects the VMs currently running on a RM for further processing.

Step 2: Resource Utilization Logging

After running RMs and their VMs are identified, *MetaVisor* collects utilization data from all RMs and VMs. To start the utilization measurement synchronously on all RMs and VMs, *MetaVisor* sends a network broadcast with a unique signature (a kind of *magic packet*), additionally containing the measurement interval length. Upon receiving this packet, all instances of *MetaStat* start measuring the utilization using `pidstat` and `sar`. Dozens of metrics are collected from both hypervisor and the VM perspective, e.g., assigned main memory, used amount of swap, consumed CPU time, load averages, block, memory and network device activity. In the meantime, the control loop sleeps and wakes up when the

utilization logging should be finished. Then, all data is gathered in parallel over the network and is written into a MySQL database. Where applicable, relative values are computed, e.g., average network packets per second. Further, the storing of utilization data is crucial for doing forecasting.

Step 3: Optional Resource Demand Forecasting

When the current resource utilization is known, optional forecasting takes place if activated and enough historic data is available. Currently available methods are Holt-Winters and SARIMA/ARIMA/ARMA models, they are further described in Section 5.9. The respective functions of the GNU R forecasting library are called from the Java VM using `rJava` and `JRI`.

For CPU resources, a time series of the amount of CPU cycles (including `steal` and `iowait`) a VM has used is built. Based on this, a forecast is created for the next interval. However, there is a subtle problem. Consider a VM with 1 vCPU, consuming 100% of the cycles the corresponding physical CPU core offers. It is probable that the VM is experiencing severe queuing issues, and would need a second vCPU. Then, it would be able to consume the amount of CPU cycles two physical cores offer. E.g., after getting a second vCPU, 130% of the original amount of cycles are consumed, or 65% of the cycles if normalized to the two vCPUs. Hence, a forecast, howsoever sophisticated, has severe troubles to reliably capture an important constraint: The amount of CPU resources a VM can consume is capped by the number of assigned vCPUs.

To find a remedy, *MetaVisor* measures the UNIX `load average` inside the VM to identify queuing issues. Currently, the `load5` is chosen as a trade-off between reacting too fast (`load1`) and reacting to slow (`load15`). The UNIX `load average` is the exponentially weighted number of processes in the run queue for the last 1, 5 and 15 minute(s)². The run queue consists of running and runnable processes. In general, when the length of the run queue becomes greater than the number of (v)CPU cores, runnable processes have to wait until CPU resources are free again.

If the `load5` is greater than n_{vCPU_s} , the number of currently assigned vCPUs, then *MetaVisor* reacts drastically by setting the future vCPU assignment to

$$n'_{vCPU_s} = \lceil load_5 \cdot n_{vCPU_s} \rceil$$

and multiplies the originally calculated forecast with `load5` too. To draw an example, a VM which had 1 vCPU, a `load5` of 1.7, consumed cycles at an average rate of 2500 MHz, and was forecast to consume 2600 MHz, will get 2 vCPUs, and the forecast is updated to $1.7 \cdot 2600 \text{ MHz} = 4420 \text{ MHz}$. After the next measurement interval, more meaningful information is again available for predicting future demands, as the VM could then, e.g., have consumed 4100 MHz with 2 vCPUs.

²a.k.a. `load1`, `load5`, `load15`

For main memory, there is a similar problem. A VM can not consume more main memory than its assignment allows. Again, the VM is introspected, and the memory utilization is analyzed. A target allocation is computed to assure the VM's percentage of free memory is between a configurable lower and an upper bound, with the constraint that a VM's memory allocation can not grow beyond its initially configured maximum. Buffers and caches are assumed to be required, and are not counted as free memory. The amount of used swap memory is subtracted from the amount of free memory, as a VM should never need to use swap memory. We have made the observation that soon, a steady state of memory allocation is reached, which is consistent with the utilization trace data discussed in Section 5.8. Reasons for this are the typical UNIX strategy to use the main memory as much as possible and the infinite job lengths in a web and database scenario. Hence, forecasting is not done for main memory due to the lacking demand variability in the investigated scenario.

Step 4: Mapping VMs to RMs

Once the VMs' future resource allocations are available, *MetaVisor* tries to efficiently map the VMs onto the RMs using either a greedy heuristic or a meta-heuristic genetic algorithm. Both are described in detail in Section 5.7.

Step 5: Live Migration

When the target mapping is available, it is established by carrying out live migrations. *MetaVisor* performs the required live migrations using `libvirt`. If necessary, RMs are switched on using WakeOnLAN beforehand. The duration of a live migration depends on a) the size of a VM's memory allocation, b) the rate of a VM changing the content of its memory pages, denoted as *page dirtying rate* c) the involved RMs' resources available for performing the migration and d) the bandwidth between the involved RMs. These questions are further examined in Chapter 6 and Chapter 7. Currently, live migrations are done in sequence. For further minimizing the total duration of establishing a target mapping, performing migrations in parallel is an interesting option. All migrations having no common involved RM could be easily combined to a single step.

Step 6: Resource Allocation Execution

When the target mapping is established, *MetaVisor* executes the VMs' new resource allocations by using `libvirt` for setting a number of vCPUs and an amount of main memory. As a further improvement, executing the new resource allocation before a possible live migration in case the VM's new resource allocation is smaller than its current is planned for future work.

Step 7: RM Shutdown

At last, *MetaVisor* shuts down RMs that are hosting no VMs, and are not required. This is done by sending a shutdown command over a public-key authenticated SSH session.

5.6 Cost Model

We consider several aspects of a mapping to define its total cost. In the following, we briefly describe the cost categories under consideration and the configurable weighting parameters for each category of cost, denoted as *penalties*. E.g., the energy consumption can be weighted differently depending on saving goals, energy prices, etc. In Section 5.11, we will present results for simulation runs with different parameter values. The heuristic BFF mapping is evaluated using these penalties, the genetic mapping algorithm directly tries to find solutions that minimize the total weighted cost C_{total} , which is simply calculated by

$$C_{total} = \sum_{i=0}^{n-1} c_i \cdot p_i, \quad (5.2)$$

where c_i is the raw cost value in category i , e.g., the amount of kWh consumed, and p_i the penalty for category i . Although the penalties are basically dimensionless, for most categories it is reasonable to think of them as an amount of money, e.g., a thousandth of a Euro, representing the cost of an action or state. E.g., for each kWh consumed by the server infrastructure, an energy penalty of 600 could represent a price of 30 cents for a kWh, and a power usage effectiveness (PUE) factor of two. As mentioned during the rest of this Chapter, there are many interesting extensions to this cost function possible.

5.6.1 Energy Consumption

The energy consumption is of course one of the most important cost categories, as it is the primary aim of this work to achieve a reduction. In order to calculate the energy consumption (kWh) of the total infrastructure, we first estimate the power consumption for each server j , on average. We use a simple linear approximation studied by Fan et al. [FWB07].

$$P_j = P(U_{j,CPU}) = P_{j,idle} + (P_{j,peak} - P_{j,idle}) \cdot U_{j,CPU} \quad (5.3)$$

The idle and peak power consumptions are taken from SPECpower_ssj2008 benchmark results for the simulation. For future work, this could be extended by interpolating the function at steps of 10% CPU utilization, as the respective power consumptions are available in the SPECpower_ssj2008 benchmark.

Based on Equation 5.3, we use each server's average CPU utilization in a measurement interval and the interval length to estimate the server's energy consumption E_j in kWh. For switched off servers, we consider $P_{standby}$, the standby power consumption instead of

Equation 5.3 to calculate E_j . We then simply calculate the total infrastructure energy consumption during the interval by

$$E = \sum_{j=0}^{R-1} E_j \quad (5.4)$$

Finally, the energy consumption E is multiplied with the penalty `energyPenalty`.

5.6.2 Live Migration

The number of VM live migrations is contributing to the total costs. Live migration is a resource intensive task, so the energy consumption of the servers involved will be temporarily higher. Additionally, the service running in the VM may exhibit worse response times during a live migration as shown in Chapter 6. For the evaluation presented here, the live migration cost has a 0/1 behavior, so every live migration is contributing with the fixed penalty denoted as `migrationPenalty`. As shown in the rest of this work, a non-linear penalty function is an interesting candidate for future work

5.6.3 Server Boots and Shutdowns

The number of server boots and shutdowns are also part of the total costs. Booting or shutting down a server takes time and consumes additional energy. Further, it is a phase of heavy strain on mechanical and electronic parts. Each server boot contributes with `bootPenalty` to the total costs, each server shutdown with `shutdownPenalty`.

5.6.4 Server Overload and Utilization Thresholds

In contrast to number-crunching infrastructures like HPC clusters, it is not possible to fully utilize servers with VMs. If they are sheltering web, database, mail etc. services, massive queuing issues will arise, leading to catastrophic quality of service. Additionally, servers need to have resource reserves for short time peaks where live migration is not an option. Further, resource reserves are required for the tedious process of live migration. Hence, we use configurable utilization thresholds t_d for both CPU and memory resources of each server, denoted as warning thresholds, e.g., `cpuUtilizationWarningLevel`. If at least one resource component is exceeding the warning threshold, the server is regarded as overloaded. Hence, servers are allowed to be utilized up to a resource quota, and a server j is overloaded if

$$U_{j,d} > t_d$$

for least a single d , $d = 0 \dots D - 1$, where $U_{j,d}$ is the utilization of server j in the resource d , so $0 < U_{j,d} \leq 1$, and where $0 < t_d \leq 1$ is a configurable constant specifying the quota capacity of a RM. As mentioned in Section 4.6, heuristic mapping algorithms profit from hard utilization thresholds in terms of complexity. If a server is exceeding thresholds, VMs

have to be migrated to other servers. However, the cost of an overload situation should be considered non-linearly, as exceeding a threshold by 2% will generally have less impact on QoS than exceeding it by 20%. Hence, we apply a non-linear cost function when evaluating the overload cost of a mapping. The exact function will most likely be different for each specific scenario as it depends on a multitude of factors. Additionally, it could even be heterogeneous for different VMs. However, this is out of the scope of my thesis, where I assume that a continuous overload cost function has been provided. For case study evaluation purposes, servers are interpreted as a Markovian M/M/1 queue, so that ρ can be interpreted as the CPU utilization and

$$P(T \leq x) = F_T(x) = 1 - e^{-\mu(1-\rho)x}, \quad (5.5)$$

gives the theoretical probability that the response time T of a request is lower than or equal to a limit x for a given service rate μ [BGdMT06]. Thus, $F_T(x)$ can be interpreted as service level percentile. Here, we are interested in the opposite, meaning the probability that the response time is greater than a limit x :

$$P(T > x) = 1 - F_T(x) = e^{-\mu(1-\rho)x}, \quad (5.6)$$

As exact values are depending on the specific infrastructure, we evaluate a case study using $x = 1$ and $\mu = 10$. The corresponding cost function is visible in Figure 5.3. Of course, μ is heterogeneous across servers, but service times or rates are unavailable within SPECpower_ssj2008 results. To generate heterogeneity, μ is multiplied with a factor relative to the maximum throughput of a SPECpower_ssj2008 system under test.

For each VM hosted on a server with an exceeded threshold, $P(T > x)$ is used to calculate the cost of overload. The scenario is that the infrastructure is used to earn money, either directly by an online shop, or indirectly by ads. If the attempt to save energy was too aggressive or a forecast went wrong, visitors may abstain from placing an order as they experience a bad QoS. As not the server itself, but its VMs suffer from low QoS, we multiply $P(T > x)$ with the number of VMs the server was hosting.

Additionally, we consider the amount of time, here the interval length, a VM was hosted on a server exceeding a utilization threshold. We do not know the exact amount of time a VM was hosted on an overloaded server, as the interval length determines the granularity of information. We simply treat the respective server as having exceeded thresholds for the whole interval length, and for each VM hosted on it, the penalty applies for each minute of the interval. The idea is to penalize exceeded thresholds in configurations with higher interval length stronger. Reasons for this are: a) it takes longer then to detect and react to overloads, and b) it is more probable to be unable to even detect them, when overloads are averaged out over a long period of time. Finally, the value is multiplied with the configurable `rmUtilizationPenalty`.

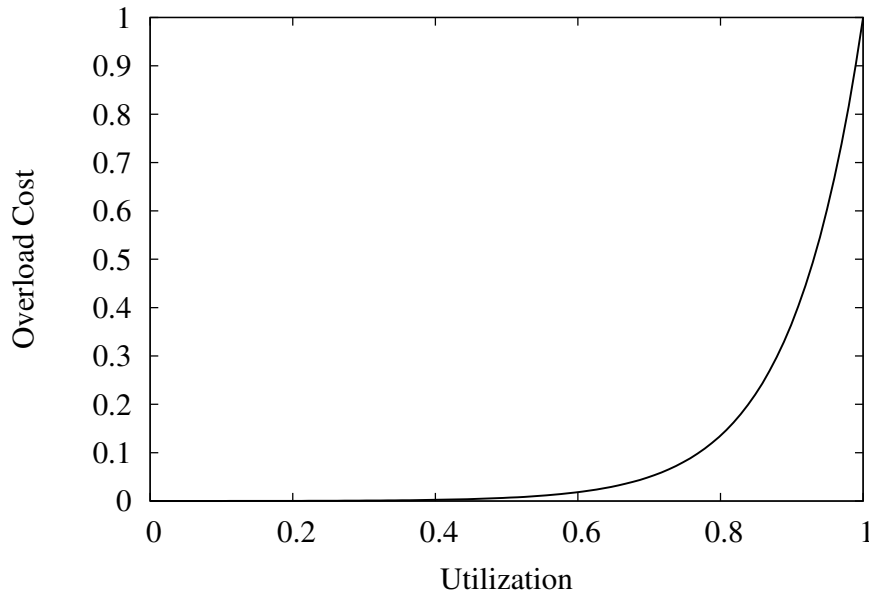


Figure 5.3: Non-linear overload cost function.

5.6.5 Utilization Variance

The variance of server utilization is considered to penalize an imbalance of utilization across servers. This cost category is especially important for the genetic algorithm presented and evaluated in this work. We use a fitness function, reciprocal to the cost function, to evaluate population individuals representing possible solutions. Then, penalizing utilization imbalance with a heavy weight leads to a preference of solutions that balance the workload as much as possible across all available servers. Hence, the GA will effectively do load balancing using the whole existing infrastructure. Depending on the live migration penalty, in this case adversary to the variance penalty, the load balancing will be carried out more or less aggressively.

Load balancing is perhaps the most interesting scenario for comparison with the approach of this work, as it means to always use all existing resources instead of always use only the estimated required resources. The sum of CPU utilization variance and memory utilization variance is multiplied with `variancePenalty`, representing the utilization variance cost. It is considered in the fitness evaluation of the individual solutions, but not shown in the result plots in Section 5.11, as the utilization variance costs are virtual, not real.

5.7 Virtual Machine Mapping Algorithms

The VMs' distribution across the servers must be automatically adapted to the workload forecasts, aiming at consuming the least possible power given the workload intensity and potential service level constraints. VMs can be live migrated to consolidate a set of idle servers to a smaller set of well utilized servers, thus saving energy by switching off servers. On the other hand, VMs can be live migrated from servers which are predicted to be overloaded soon to other servers that were automatically booted to scale the infrastructure size to the resource demand.

Finding an optimal mapping of VMs to servers based on a current mapping is a multi-objective combinatorial optimization problem. For V VMs and R RMs (physical servers), there are R^V theoretically possible mappings, resulting in a huge search space even for low V and R . Mappings with at least a single overloaded RM are invalid in the context of this work. Thus, a vast amount of theoretically possible solutions found are practically irrelevant and the computation involved was in vain. Additionally, practically possible solutions must be evaluated for their cost, as described in the previous Section.

An exhaustive search through R^V possible mappings is most likely intractable because of the computational demand involved, given practically interesting values of V and R . We examine two algorithms to map VMs to servers: A heuristic approach derived from vector packing (see also Section 4.5), denoted by Balanced-First-Fit (BFF), and a meta-heuristic genetic algorithm (GA) flexibly adaptable to various scenarios. The adaption is possible by using a customizable cost model feeding into the GA's target function.

5.7.1 Balanced First Fit Heuristic

The first mapping algorithm we compare with load balancing is a greedy heuristic originally used for bin and vector packing tasks. There are several changes necessary to use such heuristics for mapping VMs to heterogeneous server, considering at least two resource components, CPU and memory. Probably most important, instead of the items (the VMs), we have to sort the bins (the servers) by some criteria to decide which of them are to be filled first. Further, we always want to map all VMs, so sorting VMs is only reasonable for e.g., balancing the resource utilization of the servers. Additionally, there is a crucial difference regarding multiple dimensions of items and bins here. In contrast to bin packing or multidimensional, multiple knapsack problems, items are not to be aligned edge by edge, but must be aligned corner by corner, best denoted by *vector packing*.

As our goal is to save energy, we use the SPECpower_ssj2008 benchmark results, denoted as *specpoints*, as an energy efficiency sorting criteria, where higher values represent a higher efficiency. The SPECpower_ssj2008 benchmark was described in Section 2.2.3, the *specpoints* give an overall ratio of throughput per power, measured at CPU utilizations from 10% to 100% in steps of 10. All of these results contribute equally to the final result. The necessity for this sorting criteria is a specific problem of heuristics. For each definition of efficiency, we can easily find situations where another definition would be better. E.g., for

phases of low workload, a definition putting more weight on low CPU utilizations would be better. The general problem is that the efficiency of a server is depending on its utilization, but its utilization must here be determined by having an efficiency metric. These are just briefly mentioned reasons why we compare a heuristic VM placement algorithm with a genetic algorithm, which is far more flexible within still reasonable computation time.

We show a detailed description of our implementation in Listing 1 using pseudo-code, and give a brief picture of its main concepts in the following. We start with the current mapping to minimize reconfiguration costs. Then, we split the problem into three sub-parts.

1. First, all online servers are examined for utilization levels that are exceeding a threshold. If so, we remove VMs in a resource balanced way until the server is not exceeding any thresholds, and denote all removed VMs as *homeless*.
2. Second, we try to map these VMs to other servers, beginning with the most energy efficient. Again, we try to add VMs in a way that balances CPU and memory utilization of a server. During phases of high workload, it is possible that not all VMs have found a new place, as all servers might have reached their threshold level of utilization. Then, we force a mapping by ignoring the threshold, and just consider the actual resource capacity.
3. Third, we try to consolidate the infrastructure. In a dry-run, we first check if *all* VMs of a host could be migrated to other, more energy-efficient, already running hosts. We only want to migrate several VMs if as a result, the respective host can be shut down. A reason for a negative result is, e.g., that the more efficient servers do not have enough resources left. Additionally, we use a hysteresis parameter denoted by `vmConsolidationInertia`, which gives the amount of time a VM should stay on a server, unless the server is exceeding a utilization threshold. By adjusting the value, we can consolidate more or less aggressively. If all VMs could be placed on more efficient server, we implement the results from the dry-run.

Algorithm 1 Balanced First Fit Heuristic Pseudocode

```

1: map ← getCurrentMapping()
2: for all s in map.getOnlineServers() do
3:   if s.warningThresholdExceeded() then
4:     cpuSortedVMs ← sortVMsCpuDescending(s.getVMs())
5:     memSortedVMs ← sortVMsMemDescending(s.getVMs())
6:     while s.warningThresholdExceeded() do
7:       if s.hasHigherCpuUtil() then
8:         v ← cpuSortedVMs.poll()
9:         memSortedVMs.remove(v)
10:        map.remove(v)
11:      else
12:        v ← memSortedVMs.poll()

```

```
13:         cpuSortedVMs.remove(v)
14:         map.remove(v)
15:     end if
16: end while
17: end if
18: end for
19: homelessVMs ← map.getHomelessVMs()
20: cpuSortedVMs ← sortVMsCpuDescending(homelessVMs)
21: memSortedVMs ← sortVMsMemDescending(homelessVMs)
22: for all v in homelessVMs do
23:     for s in serversByDescendingEfficiency do
24:         if s.desiredThresholdExceeded() then skip
25:         end if
26:         if s.hasHigherCpuUtil() then
27:             for v in memSortedVMs do
28:                 if s.canHost(v) then
29:                     cpuSortedVMs.remove(v)
30:                     memSortedVMs.remove(v)
31:                     homelessVMs.remove(v)
32:                     map.setRunOn(v, s);
33:                 end if
34:             end for
35:         else
36:             for v in cpuSortedVMs do
37:                 if s.canHost(v) then
38:                     cpuSortedVMs.remove(v)
39:                     memSortedVMs.remove(v)
40:                     homelessVMs.remove(v)
41:                     map.setRunOn(v, s);
42:                 end if
43:             end for
44:         end if
45:     end for
46: end for
47: for all v in homelessVMs do
48:     forceMapping(v)
49: end for
50: for all s in map.getAllServers() do
51:     for all v in map.getVMs(s) do
52:         if v.getSameHostTime() < vmConsolidationInertia then break
53:         end if
54:         for all sDest in map.getMoreEfficientServers(s) do
55:             if sDest.canHost(v) then
56:                 tmpMap.setRunOn(v, sDest);
```

```

57:         tmpVM2RM.put(v, sDst);
58:     end if
59: end for
60: end for
61: if tmpMap.getVMs(s).isEmpty() then
62:     for all v:sDst in tmpVM2RM do
63:         map.setRunOn(v, sDst);
64:     end for
65: end if
66: end for

```

5.7.2 Genetic Algorithm

The meta-heuristic genetic algorithm is fundamentally different from the heuristic algorithm. Here, we can directly use the cost function configurable by the parameters described in Section 5.6 to evaluate individual solutions of a population. To be specific, the reciprocal of the costs, $f = \frac{1}{C}$, is used as *fitness* value.

At the end of each generation, individuals are then selected to be part of the next generation depending on their fitness. We primarily use the *elitism selection* mode here, where the best n individuals are selected. Another available option is the so-called *roulette wheel selection*, where individuals are selected with a probability in relation to their fitness.

We have implemented a *single point crossover* operation, where two randomly selected parent individuals create two offspring, where one offspring gets all “genes” left of the cutting point from one parent, and right from the cutting point from the other parent. For the other offspring, it is vice versa.

$$\begin{aligned}
 X_{Father} &= \left(\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right); \quad X_{Mother} = \left(\begin{array}{cc|cc} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{array} \right) \\
 X_{Son} &= \left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array} \right); \quad X_{Daughter} = \left(\begin{array}{cc|cc} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{array} \right)
 \end{aligned}$$

There are several *mutation* operations. The most important is perhaps the `migrateVm` mutation, where a VM is set to run on another server:

$$\begin{aligned}
 X_{old} &= \left(\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right) \\
 X_{new} &= \left(\begin{array}{cccc} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right)
 \end{aligned}$$

Further, a `consolidateRm` mutation is implemented. Here, all VMs of a server are set to run on other, already online servers:

$$X_{old} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$X_{new} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Additionally, a `swapRm` mutation is used, where all VMs of two servers are exchanged:

$$X_{old} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$X_{new} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Finally, a `swapVm` mutation is available, where two servers each migrate a VM between them:

$$X_{old} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$X_{new} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

The process which servers and/or VMs to take as operands is random for all operations, which are not carried out if servers would be overloaded. With the parameters listed in Table 5.1, we can influence the amount of these operations within each generation, the population size and the number of generations. During the evaluations, it was noticed that the number of generations necessary to create a good mapping depends heavily on the current mapping and the level of required optimization. If there is only little change in the workload situation, the fitness of the population reaches a steady state after a few generations. If there is a drastic change, the fitness increases over more than hundred generations. Hence, we define a maximum number of generations, and stop early if a better solution was not found for n consecutive generations.

Several populations, then typically called *demes*, can be created to allow multi-threaded computation. Here, an exchange rate can be specified, influencing the amount of solutions that are exchanged between threads after each generation to increase diversity in the "gene" pool.

5.8 Utilization Trace Data

The utilization traces thankfully provided by the University of Vienna's central IT department consist of CPU utilization traces of 33 VMs running in a production system, thus bearing real workload. The CPU utilization is measured in MHz, giving the amount of CPU cycles spent by the VMware hypervisor for a specific VM. The traces represent a period of four weeks, with intervals of two hours, where a utilization sample gives the average utilization within an interval. The infrastructure is driven by VMware, offering only little flexibility regarding utilization trace export. Trace length and resolution are not arbitrarily configurable. Traces representing a period of time longer than four weeks are not possible with a resolution of two hours. On the other hand, a trace length of four weeks is only possible with a resolution of two hours. Already, an interval length of two hours is rather long, thus we will evaluate shorter interval times using interpolation.

Interestingly, the infrastructure's memory utilization showed little to no variability at all. We explain this by the memory management behavior of UNIX-like operating systems, which use otherwise not required memory space for buffering and caching and quickly reach a steady state. On the other hand, Windows operating systems have the peculiarity to zero-fill [Wal02] the whole memory during startup, thus effectively prohibiting a hypervisor³ from assigning memory pages to a guest on demand. Hence, we concentrate exclusively on CPU utilization and assume the VM memory allocation as heterogeneous, but fixed over time.

Figure 5.4 shows a typical CPU utilization trace and its decomposition into a trend, seasonal and random component. We can clearly see the differences between day times and night times, as well as between weekdays and weekends. During weekdays, there is almost a factor of three between minima and maxima. In Figure 5.5, a diagnostic time series plot of the same utilization trace is shown. Here, each day is represented by a line, giving an impression of a possible pattern within days. Note that our time series starts at 9 pm, so in Figure 5.5, midnight is between the second and third tick. We can clearly see a pattern that during morning hours, utilization rises quickly to stabilize during day hours, slowly decreases at afternoon until it quickly declines at evening. Comparing Figure 5.4 and Figure 5.5 yields the observation that on weekends, the pattern is qualitatively similar, but quantitatively weaker.

In Figure 5.6, an autocorrelation plot of the same trace is depicted. A lag of 1.0 represents 1 period of the time series, which is consistently with Figure 5.5 24 hours. As expected, there is a strong negative autocorrelation at lags around 0.5 (12 hours) and even stronger, but positive autocorrelation at lags around 1.0. The blue dashed lines in Figure 5.6 mark the range of statistically significant autocorrelation.

Of course, the trace briefly presented above is just one of over 30 traces, nevertheless it is a qualitatively typical sample, representing a common pattern within the traces. However,

³http://www.linux-kvm.org/page/FAQ#Is_dynamic_memory_management_for_guests_supported.3F, last visited on 4.9.2012

not all of the traces show such a strong negative/positive autocorrelation at 12/24 hour lags. Figures 5.7 and 5.8 show the autocorrelation of all VMs at a lag of 12 hours and 24 hours, respectively. Interestingly, although most VMs show strong positive autocorrelation at a lag of 24 hours, not all of them show a strong negative autocorrelation at a lag of 12 hours. Some VMs even experience positive autocorrelation at a lag of 12 hours. This gives evidence of heterogeneity regarding the VMs' usage patterns, which will be considered in Section 5.9.

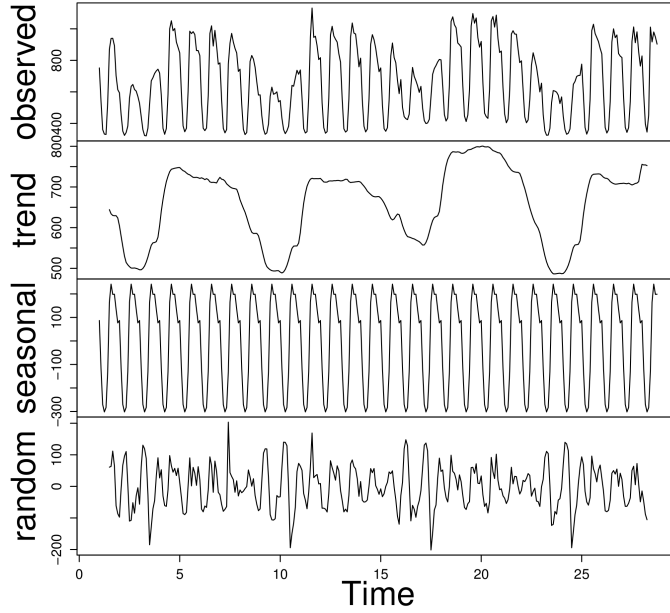


Figure 5.4: Decomposition of a typical CPU utilization trace into a trend, seasonal and random component.

5.9 VM workload forecasting

As discussed in the previous Section, the VM utilization traces show significant heterogeneity, high variability and strong autocorrelation. As a consequence, a mapping that is fine for a specific interval can lead to overload and severe QoS violations in the following interval. Hence, mappings should be calculated pro-actively to anticipate overloads. For the problem at hand, methods from the field of time series analysis are natural candidates for evaluation.

We have evaluated two forecasting methods, the seasonal Holt-Winters method (triple exponential smoothing considering seasonal patterns) and seasonal auto-regressive integrated moving-average (SARIMA) methods. For both, we use the implementations of GNU R. We generate a new model for each VM, to consider the obvious heterogeneity in the traces. We do this in each interval, as we have seen complete changes in behavior over time.

For both methods, we use the model selection abilities of GNU R to estimate the best

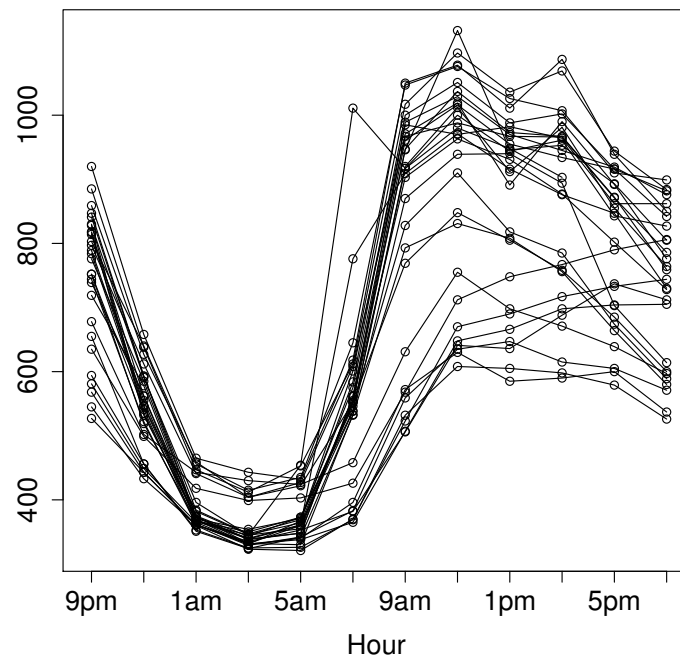


Figure 5.5: Diagnostic time series plot of the same trace as in Figure 5.4.

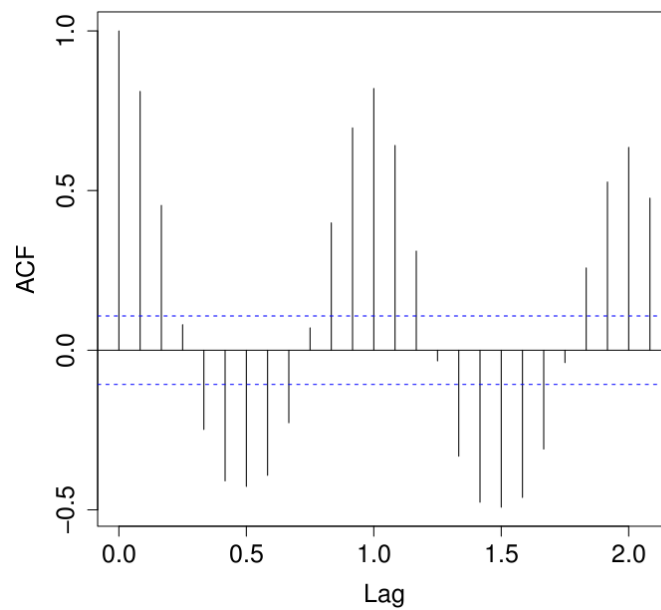


Figure 5.6: Autocorrelation plot of the same trace as in Figure 5.4.

parameter values. For Holt-Winters, the α , β and γ parameters and the type of seasonality, which can be additive or multiplicative, are determined automatically.

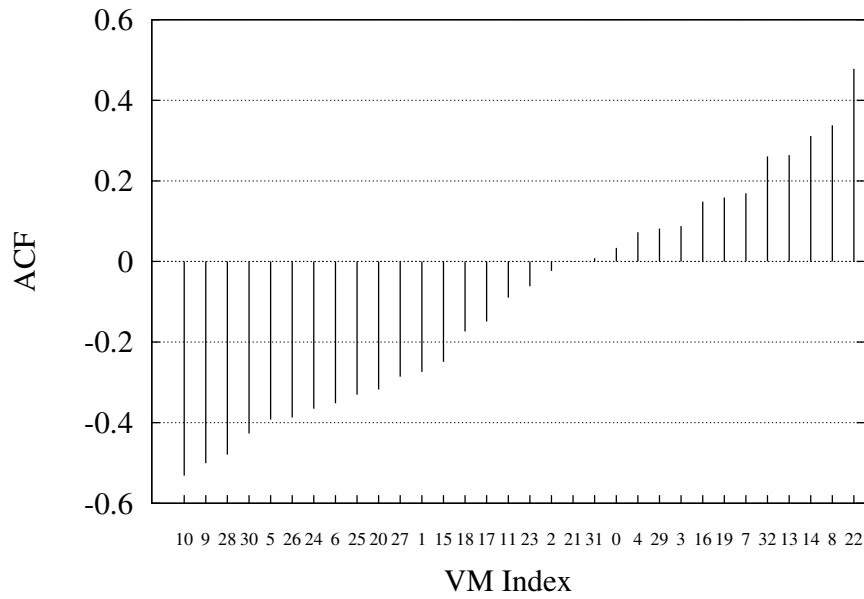


Figure 5.7: Autocorrelation of all VMs at lag 0.5, meaning 12 hours.

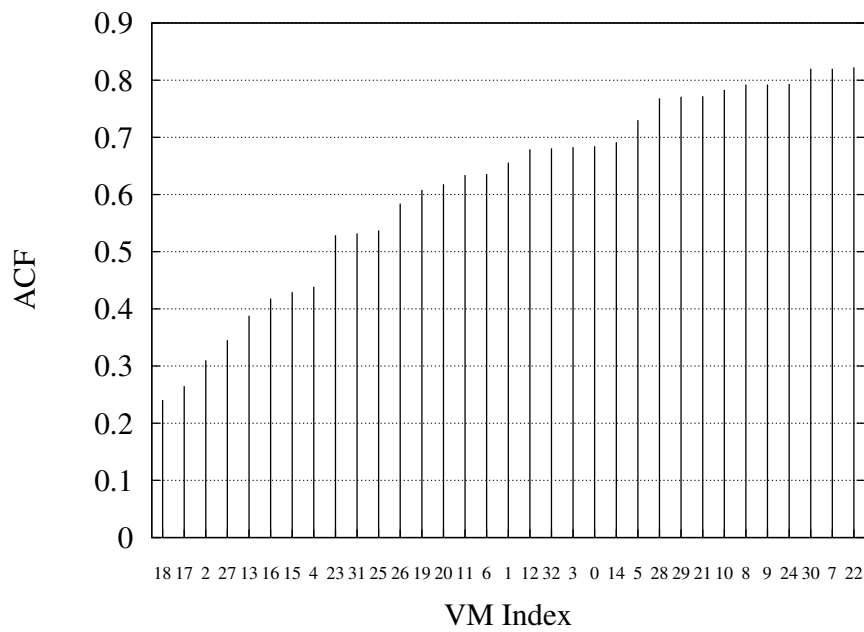


Figure 5.8: Autocorrelation of all VMs at lag 1, meaning 24 hours.

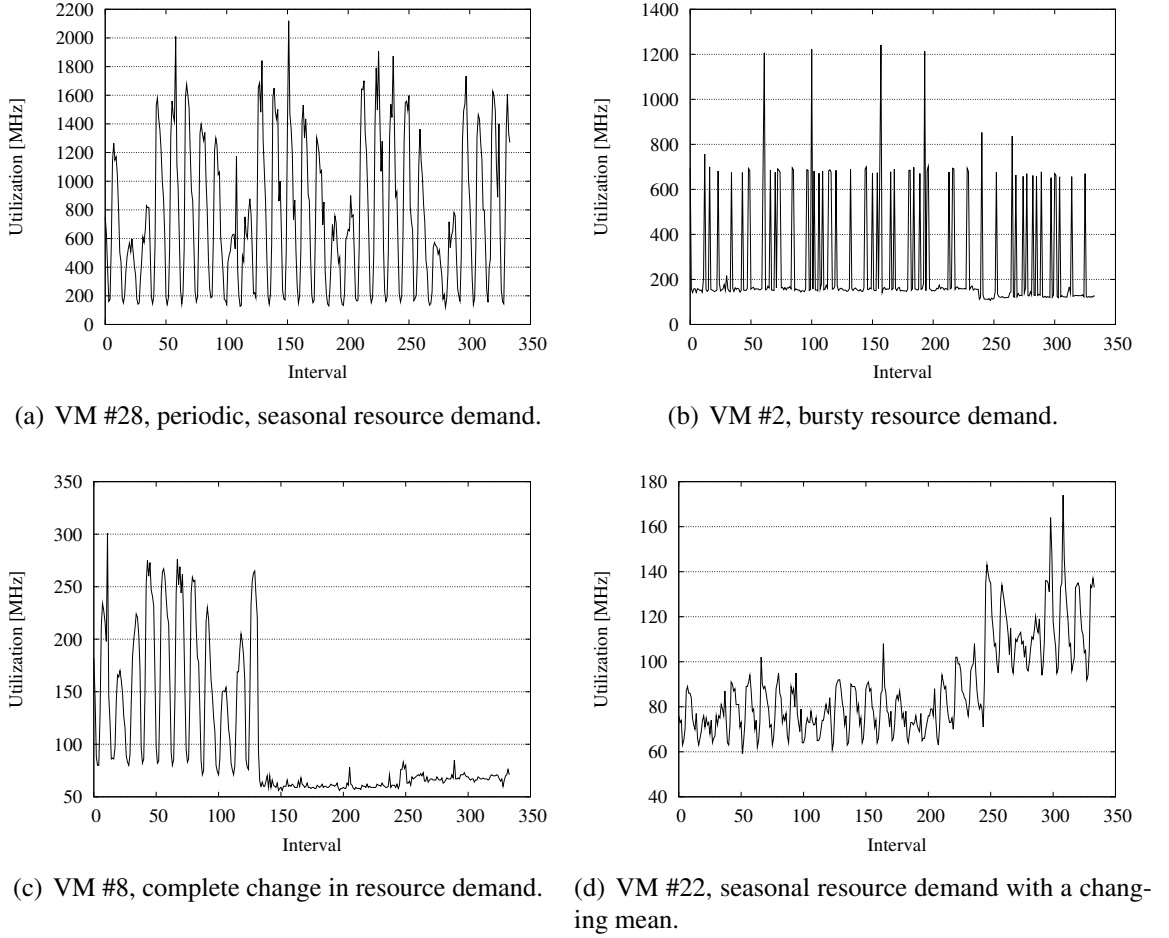


Figure 5.9: Several typical examples of VM utilization traces.

For the class of seasonal auto-regressive integrated moving-average methods, we also apply model selection, but the maximum number of seasonal and non-seasonal auto-regressive (p, P) and moving-average terms (q, Q) was restricted to 2 in order to avoid extremely long computation times. The order of first-differencing (d) and seasonal-differencing (D) was automatically determined by GNU R. A stepwise model selection process was used, again to speedup computation.

Additionally, we have created linear interpolations of the traces with a resolution of 3600 s, 1800 s, and 900 s, to be able to evaluate more frequent control loop iterations. As mentioned, the original traces have a resolution of 7200 s due to data export restrictions in the VMware software. For all of these interpolations, we have created forecasts too.

The traces have a length of four weeks. For evaluating the dynamic consolidation approach, the last three weeks were used both for simulation runs with and without forecast-

ing. Generally, the forecasting back window, meaning the period of time used to create the forecasting model, is seven days. The back window then slides with each interval. Hence, we use the first week of the trace for building the forecast models for the first measurement interval, and do not include these costs of learning in the results, as we are interested in the steady state of the system. For SARIMA at interpolations of 1800 s and 900 s, it had to be shortened to two days, as otherwise, the computation was not finished after days.

Especially important, we have taken the conservative approach of using the 95th percentile upper bound of the confidence interval as actual forecast. Then, the probability that the observation will be greater than the predicted value, leading to SLA violations, should be 5% at maximum. From another perspective, the VM should get its required amount of resources 95% of the time, which easily translates to a simple service level agreement. In Figure 5.10, several typical forecasting traces are shown. We can see that all example traces can be forecast well, even for an interval length of 7200 s. However, the number of available samples is small in this case, and the forecast's upper bound is generally over-estimating the demand significantly. Nevertheless, under-estimations are extremely rare. If we had shorter measurement intervals, e.g., 900 s as also shown in Figure 5.10, and hence more samples available, the over-estimation is noticeably reduced.

5.10 Evaluation

In order to evaluate the combined potential of both workload forecasting and dynamic consolidation regarding energy efficiency, we have developed *MetaSim*, a simulation mode for *MetaVisor*. The simulation uses the *MetaVisor* control loop, and allows us to use utilization traces from several data sources, here plain text files, to validate the control loop. Further, long time spans and multiple scenarios can be evaluated within comparably short time. We denote the (simulated) time between control loop iterations as *interval length* and each iteration as *interval*. To create a simulation run, four components are necessary:

1. The simulation input data, comprised of a number of VMs and their resource demands, and a list of servers and their characteristics, e.g., resource capacities, power consumption etc. The configuration is done using XML, see Appendix B for the base configuration. Multiple VMs were built from each of the traces to be able to simulate a larger infrastructure. The amount of VMs built from each trace, a scaling factor and the amount of main memory were varied to ensure heterogeneity. The CPU resource demand and capacity time series resulting from this configuration and used in the simulations is shown in Figure 5.11. As we can see, there is much potential for shifting the typical operating region of servers closer to the warning level threshold, here 60% of the nominal capacity.
2. The configuration of the cost model, cf. Section 5.6. The base configuration is visible in Table 5.1 and also shown in Appendix B.

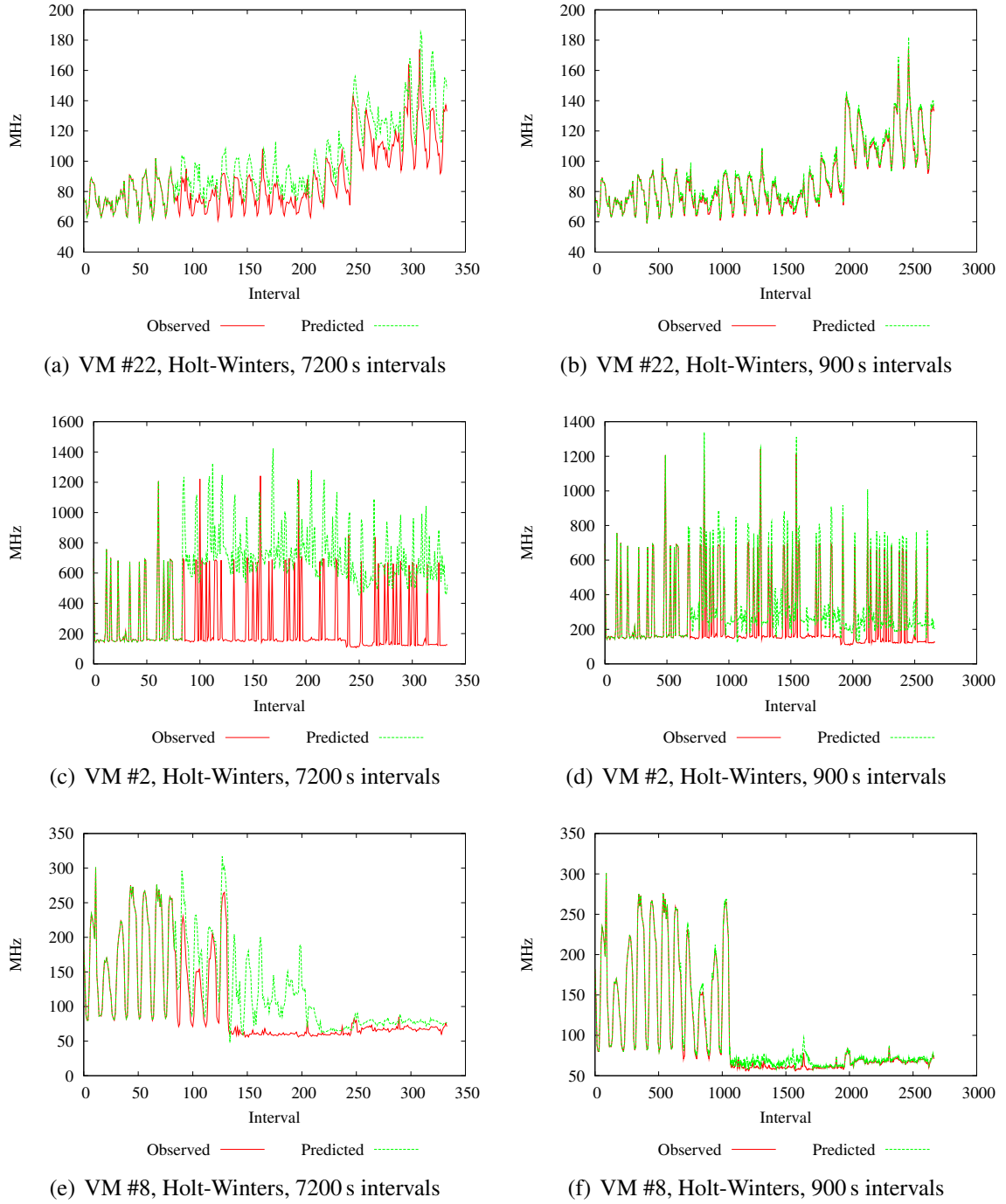


Figure 5.10: Several examples for increased forecasting precision when the trace data is interpolated. Due to using the 95th percentile upper bound as forecast, the demand is over-estimated in the worst-case. Under-estimations occur extremely seldom.

3. A VM mapping implementation, denoted as *decision module*, cf. Section 5.7.
4. Several, partly optional parameters influencing the simulation itself or its evaluation, cf. Table 5.1.

In the first interval an initial mapping is created, using the chosen decision module. The initial mapping is not evaluated, to give each decision module the chance to warm-up by creating a reasonable initial state according to its algorithm. Subsequently and in each control loop iteration, the following steps are carried out:

1. VM utilizations are updated, as they will have changed since the last observation. If forecasting is used, then the forecast utilizations are updated too. Based on the mapping from the last interval which was not recalculated by a decision module yet, server utilizations (optionally forecasts too) are updated accordingly. E.g., this can lead to the detection of server overload if the mapping chosen in the last interval was too optimistic or aggressive, or forecasting went wrong.
2. Before recalculating the mapping, it is evaluated regarding servers exceeding a threshold, and the utilization cost function is used to determine overload costs.
3. A new mapping is calculated by a chosen decision module, optionally based on forecasts. Several operations are possible here, e.g., VMs are migrated to another server,

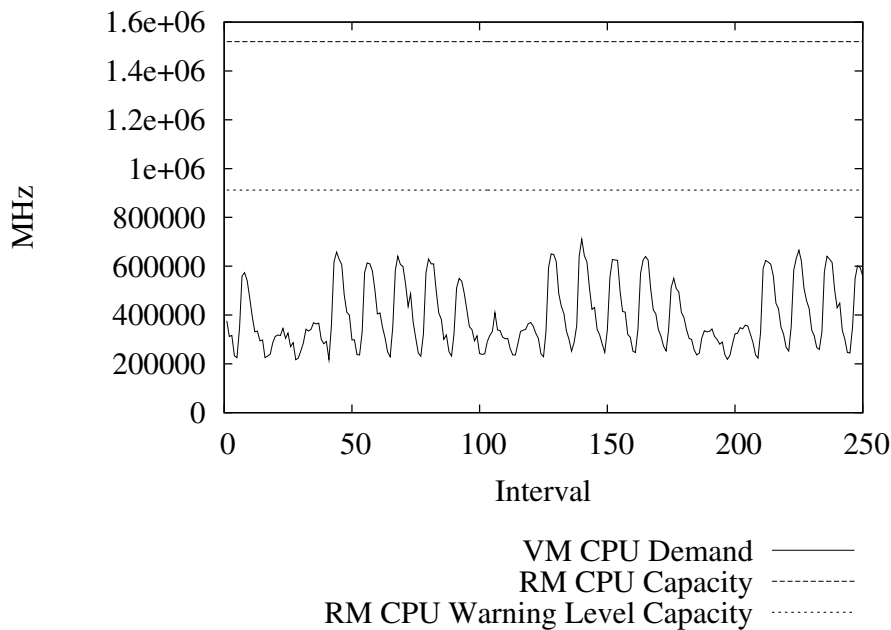


Figure 5.11: The time series of total VM CPU demand, server capacity and quota capacity within the warning level used in the simulations.

servers are powered down or woken up, and related information like energy consumption is updated accordingly.

4. Finally, the new mapping is evaluated considering several aspects, feeding into a cost function we have elaborated on in Section 5.6.

5.11 Results

In the following, results are presented for the described cost model driven dynamic consolidation. We start evaluation the decision modules in combination with optional forecasting and more frequent control loop intervals.

5.11.1 Interval Length and Forecasting

As already mentioned, the original trace data has an interval length of two hours, which is rather long. In a first evaluation, we have linearly interpolated the trace data into resolutions of 3600 s, 1800 s and 900 s, comparing them with the original resolution of 7200 s. The resulting total weighted costs for all combinations of interpolation, mapping decision module and optional forecasting method are shown in Figure 5.12, the parameters used to yield these results are listed in Table 5.1. We demonstrate the performance and flexibility of our cost model and the described genetic algorithm. A first proof of its flexibility is that is easily adaptable to carry out load balancing by adjusting a single parameter, the `variancePenalty`. In this evaluation, a starting set of parameter values is chosen. The consequences of changing these parameters are evaluated in several case studies presented in the successive Subsections.

In the following, the resulting total weighted costs and their inner structure shown in Figure 5.12 are interpreted. We can see with the parameters used here, the most important cost categories are energy and overloads. It is also clearly visible that energy costs are rather independent from the interval length. Load Balancing (LB) performs equally for all interpolations and is the base of comparison for all other results.

With an interval length of 7200 s, both the genetic algorithm (GA) and the greedy balanced first fit heuristic (BFF) are causing massive overload costs when forecasting is not used. This is a direct consequence of including the interval length into the overload cost function, representing that a less frequently iterating control loop takes longer to detect overload situations, and more damage will be done until detection. However, with forecasting enabled, a significant amount of total costs, approximately 40%, can be saved compared to LB. Holt-Winters and ARIMA are performing very similar. As the computational effort for the ARIMA model selection is significantly higher, Holt-Winters should be favored over ARIMA in comparable scenarios. GA is performing similar to BFF, only the combination GA-ARIMA is performing slightly worse.

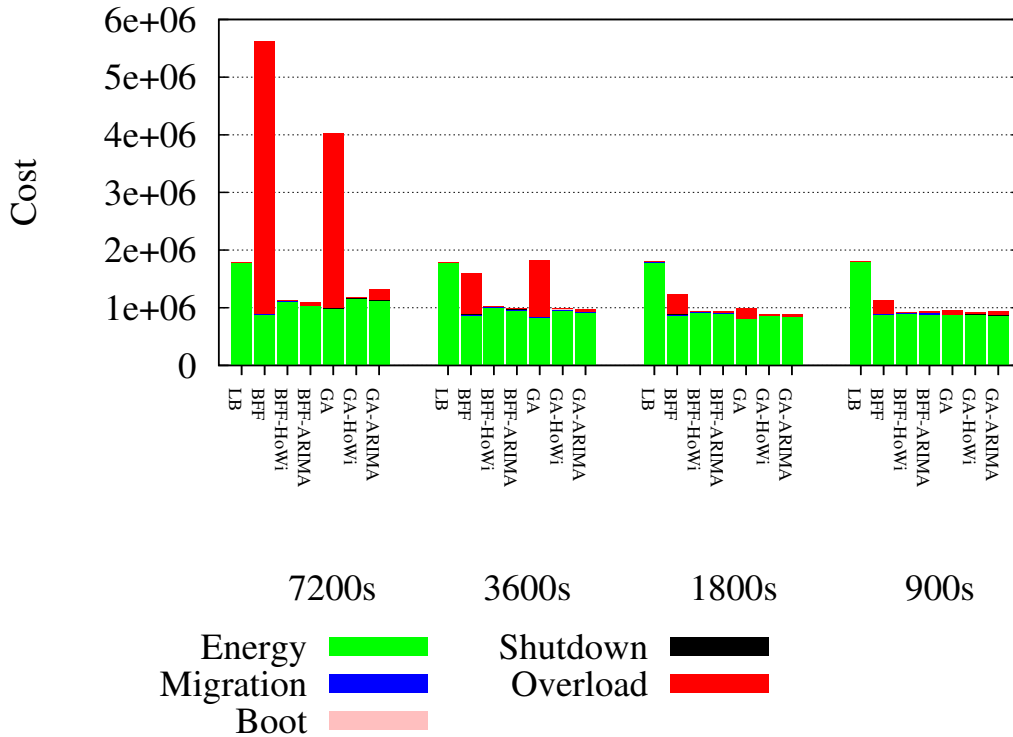


Figure 5.12: Total weighted costs and their inner structure for all combinations of interpolation, mapping algorithm and forecasting method.

With an interval length of 3600 s, BFF starts to reduce the total cost by a small amount even without any forecasting done. GA without forecasting is now performing equal to LB. With forecasting enabled, both BFF and GA benefit compared the 7200 s interval length, GA is now even performing better than BFF.

At an interval length of 1800 s, the difference between enabled and disabled forecasting is becoming smaller. GA now saves approximately 40% even without forecasting. However, both BFF and GA further benefit from forecasting, and GA with forecasting is decreasing the cost most. Between Holt-Winters and ARIMA, there is no significant difference. At this interval length, the sample back window for ARIMA had to be shortened to two days (96 samples), as otherwise, the model selection for the whole trace of three weeks (1008 intervals) was not finished within days of CPU time. For real world scenarios, howsoever good the quality of the resulting forecasts would be, such run times would render them irrelevant.

At an interval length of 900 s, the costs of all approaches are further becoming smaller, however the gain is very little. Again, the back window of ARIMA had to be shortened to two days, containing 192 samples.

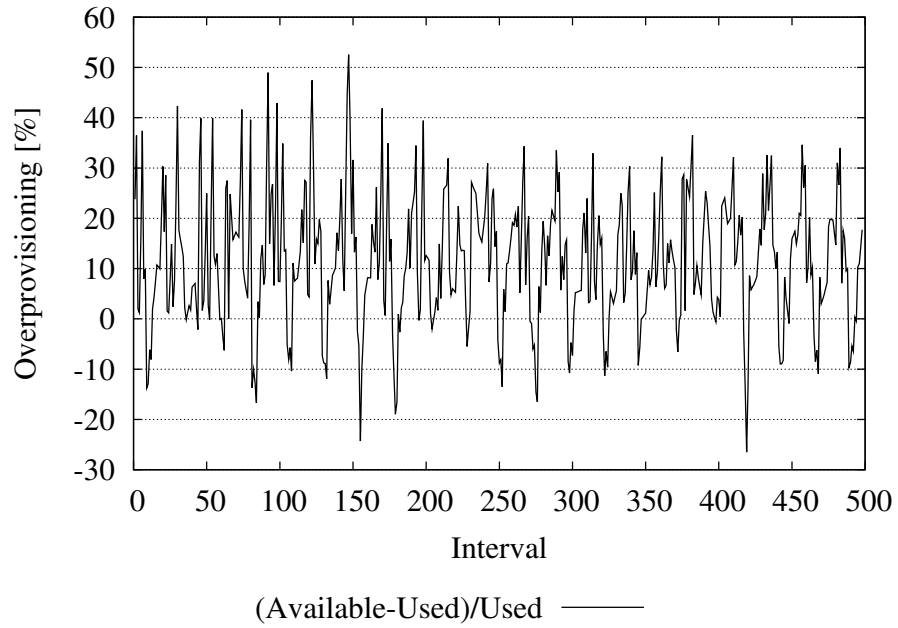


Figure 5.13: Provisioning efficiency for an interval length of 3600 s and the BFF heuristic without forecasting.

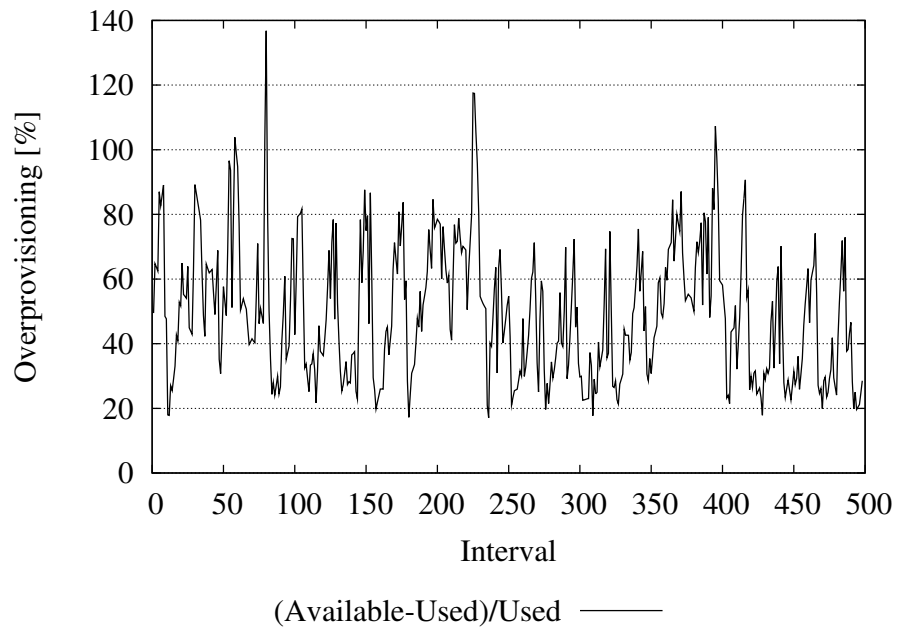


Figure 5.14: Provisioning efficiency for an interval length of 3600 s and the BFF heuristic with Holt-Winters forecasting.

Relevance	Parameter	Value
All	cpuUtilizationWarningLevel	0.6
	memoryUtilizationWarningLevel	0.8
	utilizationCostFunctionMu	10
	utilizationCostFunctionAllowedResponseTime	1
	energyPenalty	600
	migrationPenalty	1
	rmUtilizationPenalty	10
	bootPenalty	1
	shutdownPenalty	5
Load Balancing	variancePenalty	100000
BFF	rmIdleTimeoutSeconds	900
	vmConsolidationInertiaSeconds	600
GA and LB	numberOfThreads	4
	numberOfGenerations	200
	maxGenerationsOfFitnessNotIncreased	10
	sizeOfPopulation	800
	crossoverRate	0.5
	exchangeRate	0.1
	migrateVmRate	0.3
	swapRmRate	0.1
	swapVmRate	0.1
	elitism	true
Optional Forecasting	requiredPeriodsForForecasting	3

Table 5.1: Parameters used for yielding results shown in Figure 5.12.

In Figure 5.13, we can see the provisioning efficiency of the BFF heuristic for an interval length of 3600 s, without forecasting. 0% over-provisioning is the ideal case where the online quota capacity is exactly fulfilling the demand. 100% over-provisioning represents that twice the amount of required quota capacity is online. Similarly, -100% denotes that twice the resources are required than are actually online. It is clearly visible in Figure 5.13 that under-provisioning occurs frequently due to the lack of forecasting. Figure 5.14 shows the same simulation run, but with Holt-Winters forecasting enabled. Under-provisioning does not happen at all, however significant over-provisioning is common. If we increase the control loop frequency by decreasing the interval length to 900 s as shown in Figure 5.15, we can decrease over-provisioning substantially, without risking under-provisioning. If over-provisioning occurs, then the impact is also shorter because the control loop can react more quickly.

As a result, we can see in Figure 5.12 that cost model driven dynamic consolidation

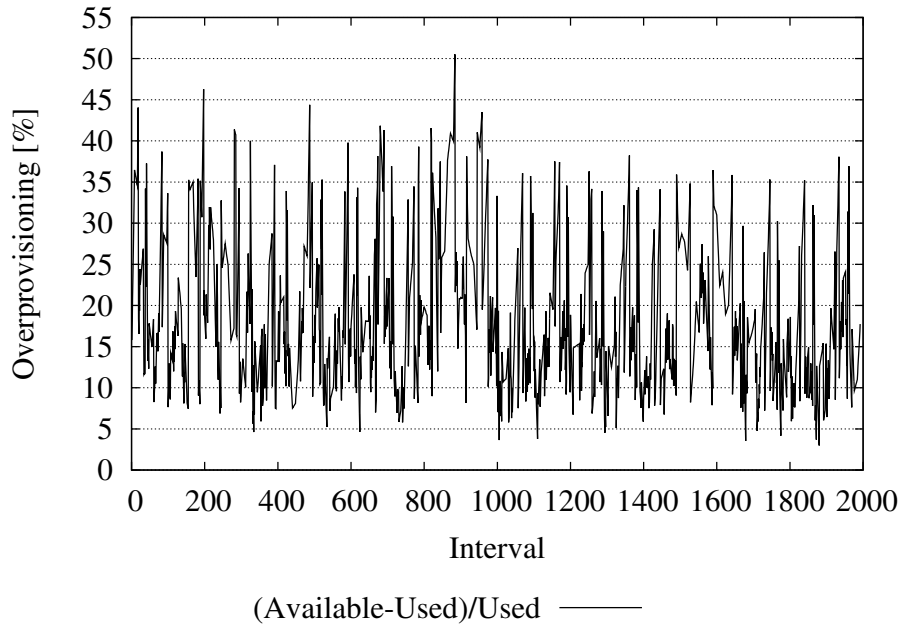


Figure 5.15: Provisioning efficiency for an interval length of 900 s and the BFF heuristic with Holt-Winters forecasting.

can easily save 40% of total costs even with infrequent control loop iterations, but with forecasting enabled. The difference between Holt-Winters and ARIMA models regarding forecast quality is negligible, although the runtime of ARIMA is significantly longer. The more frequent control loop iterations are carried out, the smaller the difference between enabled and disabled forecasting becomes. Generally, BFF and GA are performing equally in this case scenario. However, in contrast to the GA, the BFF heuristic is unable to react to changing cost model parameter values. E.g., the BFF will deterministically create the same solution for a given status quo, independent from energy price.

In the following, the computationally more intensive approach of using a genetic algorithm will be justified by demonstrating its flexibility to adapt to a change in conditions. For the following parameter influence evaluations, an interval length of 3600 s with Holt-Winters forecasting was used, as it provides a good trade-off between runtime and quality of the result. Further, the simulation trace was shortened to one week instead of three weeks to speed up the computation and to be able to evaluate more parameter values within the same time.

5.11.2 Overload Cost Parameter Influence

For every infrastructure that should be consolidated dynamically, one of the most important questions will be how sensitive it is to overload situations. An infrastructure that is serving

weblogs will most likely lead to a different perspective on overload situations than online shop operators or even providers of business critical services will have. The genetic algorithm decision module is able to adapt to the specific circumstances of an infrastructure. To demonstrate this, the influence of changing the overload penalty on the resulting total costs is shown in Figure 5.16.

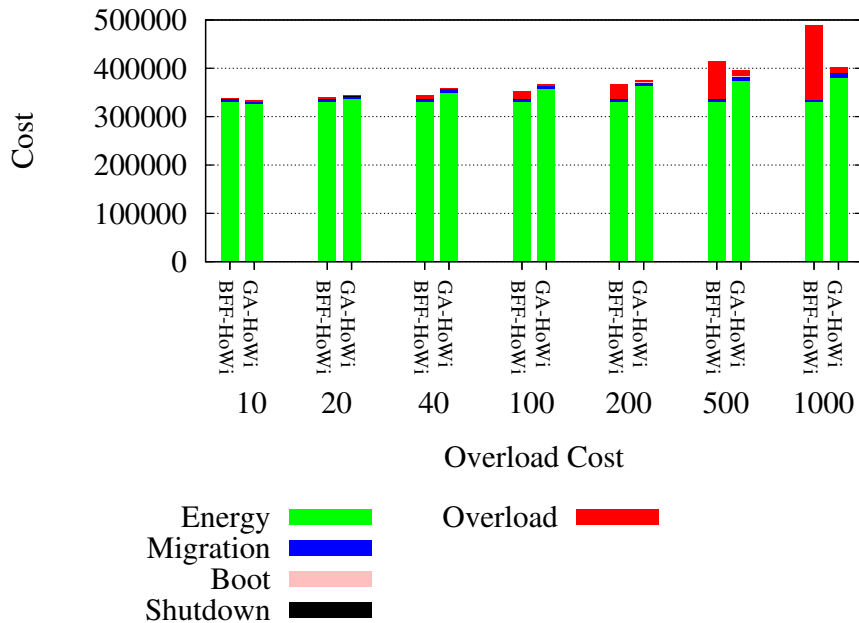


Figure 5.16: Influence of changing the overload penalty on BFF/GA solution quality.

First, the simulation was done again with the same parameters as in Table 5.1 to have a proper base of comparison. The trace length was shortened, so the results are not comparable in absolute numbers to Figure 5.12. We can clearly see in Figure 5.16 that with increasing value of the overload penalty, the accumulated sum of overload costs severely increases for BFF. This is expectable, as it is unable to act based on the cost model, but is only evaluated with it. On the other hand, GA is able to react to the changed circumstances by increasing the amount of energy used, leading to a sum of overload costs that is growing only very little with the increasing overload penalty. Of course, the results have to be understood as case studies, the exact values have to be determined for each infrastructure. Furthermore, it is probable that the VMs are heterogeneous in terms of overload sensitivity. On the one side, there can be technical reasons, e.g., an authentication service that is running into time-outs if users do not (here: *can* not) respond within a fixed amount of time. On the other side, there can be economic reasons, e.g., there are services that make more revenue and therefore have higher priority, or a customer has paid much money for a better SLA. This heterogeneity is implementable quite easily into the genetic decision module.

5.11.3 Migration Cost Parameter Influence

Another aspect to investigate is how the cost of live migrating VMs is contributing to the total costs and how the genetic algorithm can adapt to changing circumstances. Again, a base of comparison with the same parameters as in Table 5.1 was simulated first. Then, the penalty of a migration was set to 50 and 100 cost units, respectively. The results are visible in Figure 5.17.

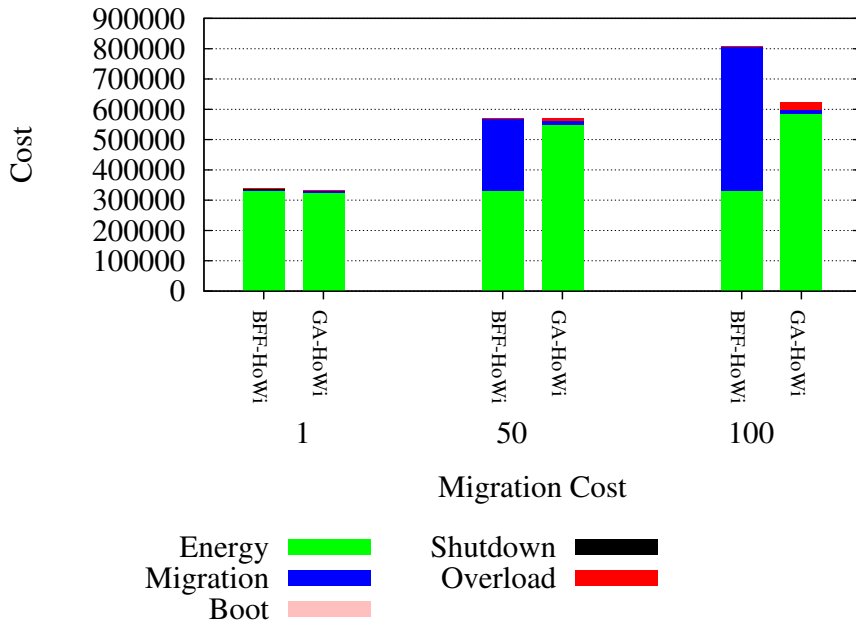


Figure 5.17: Influence of changing the migration penalty on BFF/GA solution quality.

Again, we can clearly see that GA is outperforming BFF by adapting to the cost model. It is never performing worse than BFF, but reacts by increasing the amount of energy spent and slight overload situations tolerated in order to avoid costly live migrations. For a migration penalty of 100 units, it is saving approximately 20% compared to BFF. It is important to note here that the live migration penalty is currently implemented in a simple 0/1 approach, as each live migration is contributing with a fixed amount of penalty. As shown in Chapter 6 and Chapter 7, this should be changed to a non-linear model consisting of several important utilization metrics. E.g., a VM that is already suffering from queuing issues, should not be migrated unless there are no alternatives. Other, less utilized VMs should be quickly migrated, and the freed resources should be assigned to the overloaded VM. With more and more variables added to an optimization problem, the design and development of heuristics becomes increasingly complex, and the solution space is better explored using stochastic processes. Nevertheless, the heuristic is outperformed by the GA already for a simple 0/1 penalty model. Further, the above mentioned ideas regarding heterogeneity do apply here

too. A VM providing critical services and thus sensitive to any implementation of live migration should be migrated with less priority than a VM hosting a blog. Again, the genetic approach is comparably easily extensible to such demands.

5.11.4 Energy Cost Parameter Influence

As the last parameter case study, the influence of the energy price was evaluated. In addition to the same value as in Table 5.1, the energy penalty was doubled and quadrupled. We can see in Figure 5.18 that the GA is also able to react to a changed energy price. For a penalty per kWh used of 2400 units, the GA can save approximately 10%. It is important to note here that due to the other parameters and enabled forecasting, energy costs are basically the only cost category relevant for the total costs. E.g., if there was a significant amount of costs stemming from live migrations for both decision modules in the base scenario, the GA could try to migrate more often in order to save energy more aggressively. Again, the BFF heuristic would have to be changed explicitly in order to consider this.

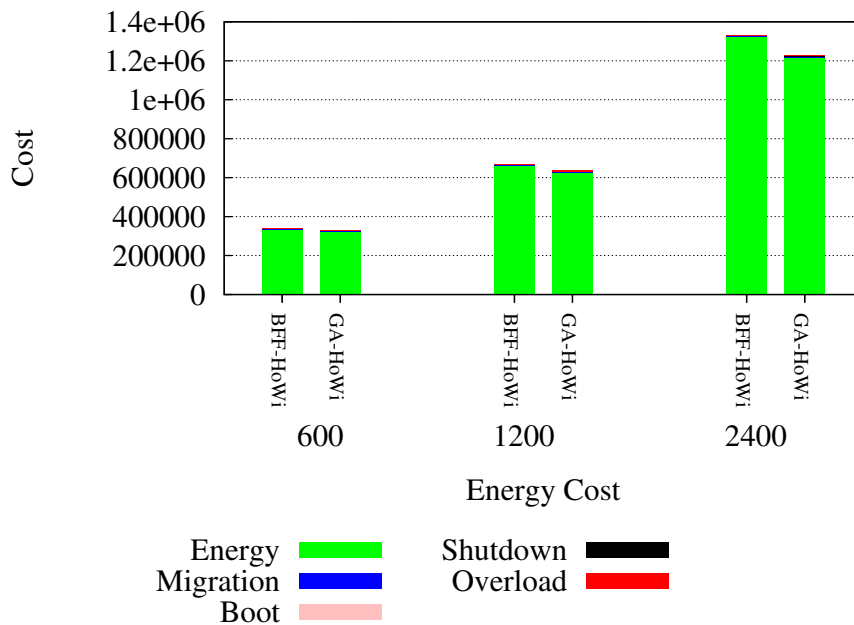


Figure 5.18: Influence of changing the energy penalty on BFF/GA solution quality.

5.11.5 VM Consolidation Inertia

After evaluating the influence of the cost model parameter values, the two hysteresis parameters of the BFF heuristic are evaluated in the following. They have the purpose of flip-flop

avoidance. First, the influence of the VM consolidation inertia parameter is investigated. It is basically a minimum amount of time a VM is guaranteed to run on a specific host after it was migrated there. As long as neither the host is exceeding the utilization threshold nor the amount of time was reached, it is not allowed to be migrated again. In Figure 5.19, results are shown for parameter values ranging from 0 s, effectively deactivating the hysteresis, to 7200 s. The RM Idle Timeout was set to 0 s, meaning servers can be immediately shut down if they host no VMs. The control loop interval was set to 300 s, and Holt-Winters was used as forecaster.

Interestingly, there is no significant difference visible. However, this can be explained by the BFF implementation and by the high accuracy forecasts. During the consolidation phase of the algorithm, a server is either fully consolidated, meaning that *all* of its VMs can be run on other, more energy efficient servers and the respective server can be shut down, or not at all. Thus, if a *single* VM has not reached the minimum required time being on the same host, the server is not consolidated. Due to the forecasting, anticipatory live migrations are frequently carried out, as also visible as migration costs in Figure 5.19 and indirectly visible as extremely low overload costs. Hence, the mapping of the servers is frequently changing, and the chance that at least a single VM has not reached the minimum required time being on the same host, is high. Only for an inertia setting of 7200 s, there is a small increase of costs visible, stemming from the energy cost. Hence, increasing the parameter value at some point leads to increased energy consumption, but if forecasting is done with high accuracy, overload costs can not be decreased by increasing the parameter value.

The assumption was that if servers are consolidated less aggressively, they have more resource reserves that help to avoid thresholds being exceeded. However, in the scenario at hand, this does not further improve the already very high cost saving of 50%. Interestingly, as shown in Figure 5.20, there is also no clear benefit visible if no forecasting is done. For consolidation inertia parameters of four and eight hours, the energy consumption is increased, but neither migration costs nor overload costs are noticeably decreased. We can conclude that in the investigated scenario, forecasting is a far better approach to minimize overload costs. An evaluation that is of high interest for future work is if and how this changes if migration costs are built from using a non-linear model. Then, more aggressive consolidation could lead to more frequent migrations in costly, high workload intensity situations, when the intensity is rising again.

5.11.6 RM Idle Timeout

The second parameter relevant for the BFF heuristic is the RM Idle Timeout. Similar as above, it is a minimum amount of time before an action is allowed to happen. Here, the action is turning off a server that has not been hosting VMs for the last n seconds. The VM Consolidation Inertia was set to 0 s to avoid interference between these two parameters. The control loop interval was set to 300 s, and Holt-Winters was used as forecaster. In Figure 5.21, results are shown for parameter values ranging from 0 s, effectively allowing immediate shut down, to 7200 s. The other parameter were set as listed in Table 5.1. Under

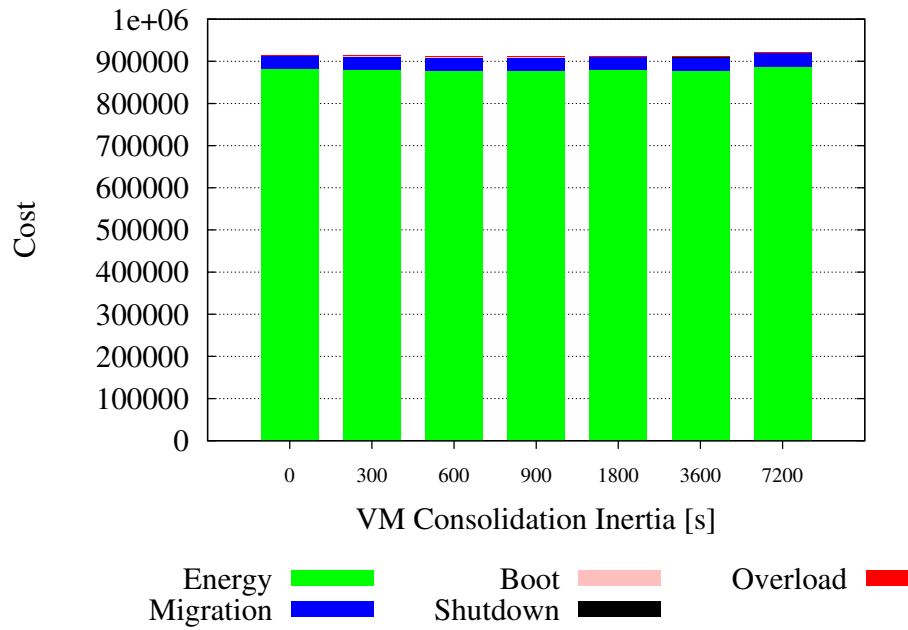


Figure 5.19: Influence of the VM consolidation inertia parameter on the total weighted costs when Holt-Winters forecasting used.

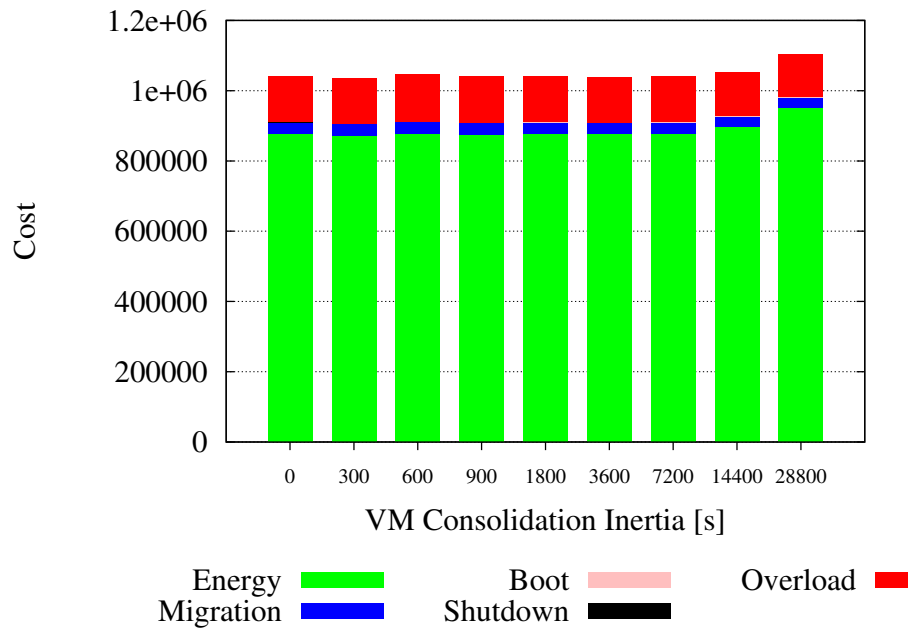


Figure 5.20: Influence of the VM consolidation inertia parameter on the total weighted costs when no forecasting is done.

these circumstances, there is no benefit in letting idle servers run longer. Only if boot and shutdown penalties were significantly higher, we can expect different results.

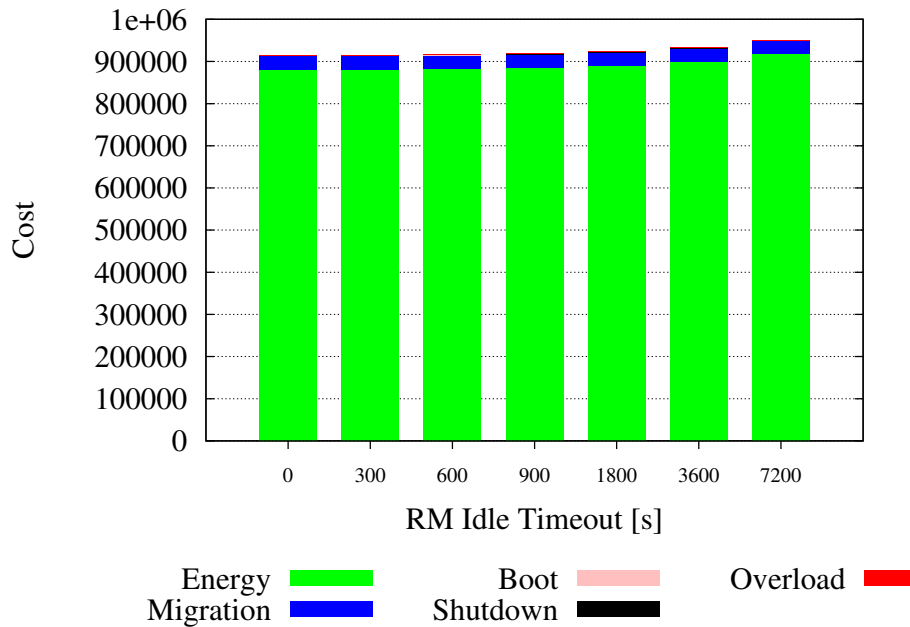


Figure 5.21: Influence of the RM idle timeout parameter on the total weighted costs.

5.11.7 Performance Evaluation

Finally, runtime performance evaluations were carried out on several hardware platforms, described in Table 5.2.

Platform:	Low Power	Desktop	Server
CPU	AMD E-350	AMD Phenom II X4 955	Intel Xeon E5-2670
CPU Frequency	1.6 GHz	3.2 GHz	2.6 GHz
CPU Cores	2	4	8
CPU L2-Cache	2x512 KiB	4x512 KiB	8x256 KiB
CPU L3-Cache	N/A	6 MiB shared	20 MiB shared
CPU TDP	18 W	125 W	115 W
Memory	2 GiB	16 GiB	64 GiB

Table 5.2: Description of platforms used in the performance evaluations.

In Figure 5.22, results are shown for the runtime (wall time) of the BFF heuristic on a

low power platform and a high end desktop platform. We can clearly see that the heuristic is very quickly computable even on a low power platform and with activated forecasting. Note that 26 servers and more than three hundred VMs are simulated, meaning that gathering of utilization data will most likely take noticeably longer than calculating a new mapping. Using a high-end desktop CPU offers a speedup of approximately 3. The BFF heuristic is single-threaded, and is not a candidate for coarse-grained parallelization. The other cores of the quadcore CPU are left unused, but can nevertheless be important for fetching utilization data from a database multi-threaded. Furthermore, forecasting can be easily parallelized as each VM trace represents an isolated problem.

In Figure 5.23, results are shown for the genetic algorithm. Here, activating forecasting causes only insignificant overhead, as the genetic algorithm dominates the time required to compute a solution. Again, a speedup of approximately three is visible between the platforms. Note that these run times have to be interpreted as worst-case scenarios, as the number of generations was fixed to 100. In most control loop iterations, the current mapping is generally fine, only a few modifications, e.g. live migrations, are necessary to optimize it. Then, the population's fitness reaches a steady state very quickly, as the distance in the search space between the current solution and a potentially better solution is comparably small. Hence, we can stop the genetic algorithm if a better solution has not been found within n consecutive generations. If forecasting however predicts a severe change in workload intensity, then a viable solution that also anticipates overload takes more time to be found. Its distance in the search space is comparably big and a lot of mutations, crossovers etc. are necessary to create it. Then, a steady state regarding population fitness is typically reached after dozens of generations. Hence, the number of generations necessary to create a good solution depends heavily on the status quo, and the process should be run until the quality of the solution does not increase further for several generations. This avoids both unnecessary long run times in the first case described above, and far from optimal solutions in the second case. However, this performance optimization is not deterministic and was not used in this runtime evaluation in order to ensure comparability.

Furthermore, on both platforms, the evaluation was carried out single-threaded, so a further speedup due to parallelization needs to be investigated. Results are shown in Figure 5.24 for the desktop and server platform. Again, the performance optimization described above was not used for comparability reasons. The quadcore CPU is evaluated with up to four threads, the dual octocore platform is tested with up to 16 threads and with disabled Hyper-Threading. On the quadcore CPU, the speedup is stagnating at three threads at a level of approximately 2.2. On the dual octocore platform, the speedup is stagnating at six threads at a level of 2.5. As a result, we see that there is performance potential in parallelizing the genetic algorithm, however, the results could be better. A further examination is necessary to identify bottlenecks. Natural candidates are increased cache miss rates at the shared L3-cache due to the increased rate of random accesses to memory, synchronization overheads that need to be minimized (e.g., exchange of individual solutions between threads) and increased logging overhead by more frequent invocation to blocking functions.

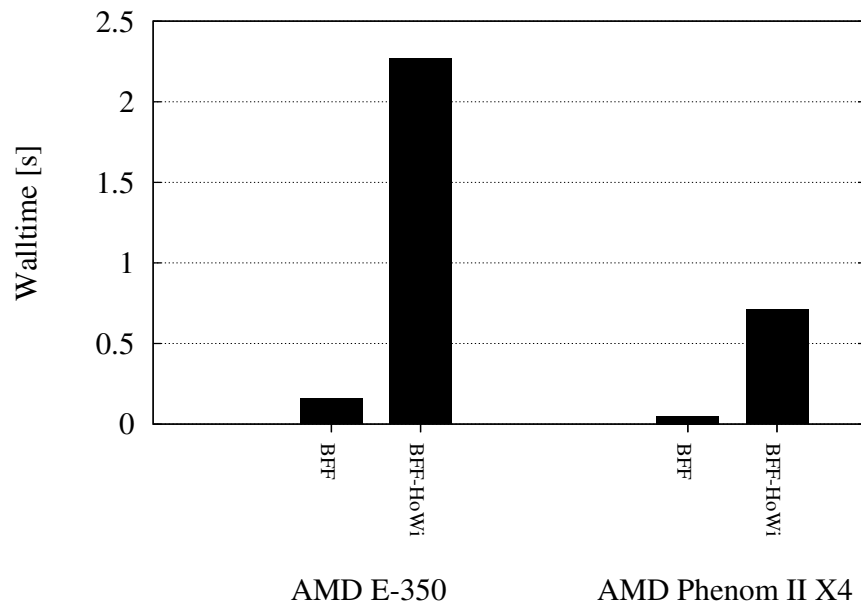


Figure 5.22: Runtime per interval of BFF with/without Holt-Winters forecasting on a low power CPU and a high-end desktop CPU.

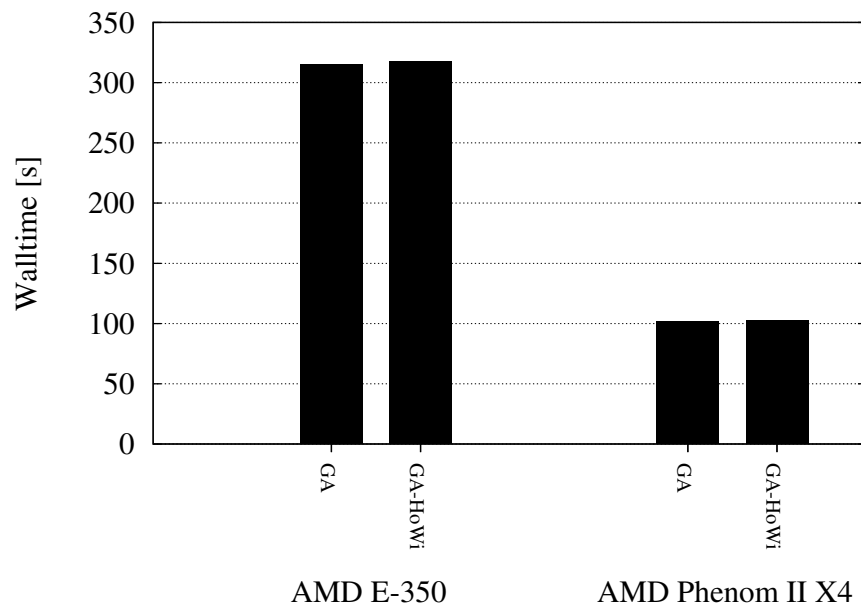


Figure 5.23: Runtime per interval of GA with/without Holt-Winters forecasting on a low power CPU and a high-end desktop CPU.

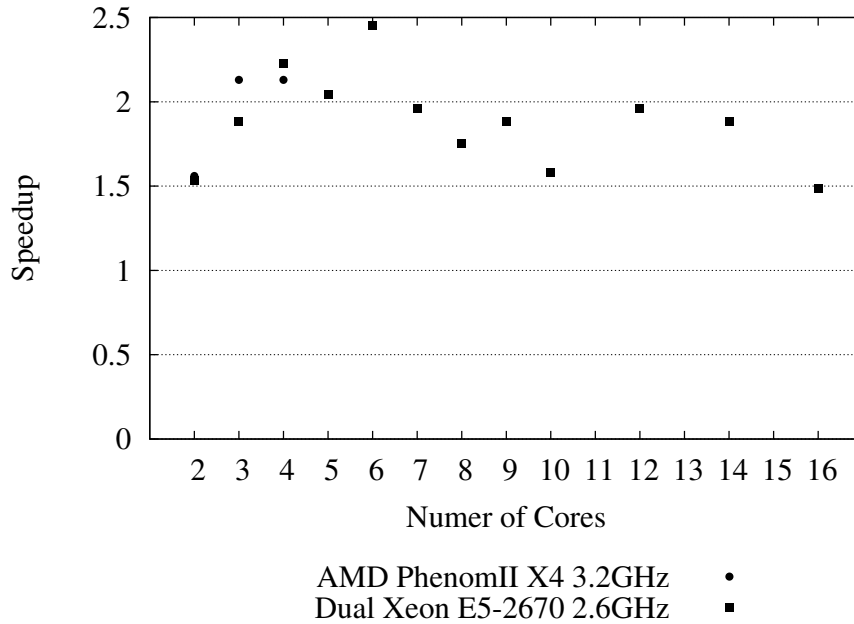


Figure 5.24: Speedup achieved by parallelizing the genetic algorithm.

5.12 Discussion

In this Chapter, we have presented results how dynamic consolidation and workload forecasting can be combined to reduce energy consumption of virtualized heterogeneous data centers, based on real utilization traces. We have investigated a business infrastructure scenario, where the cost of energy is just *one* of several parts of operational costs. We have presented a cost model feeding into a meta-heuristic genetic algorithm's fitness function. This allows easy adaption by infrastructure operators to specific optimization demands. We have used a case study set of parameter values to compare the total weighted costs of load balancing with a greedy heuristic and a genetic algorithm, combined with Holt-Winters and ARIMA forecasting.

As a result, the total operational costs of an infrastructure can be reduced by 40% to 50%, depending on the configuration and the scenario. The greedy heuristic and the genetic algorithm deliver similar results for most evaluations of the starting set of cost parameters.

For long interval lengths, i.e., infrequent control loop iterations, forecasting significantly helps to decrease the costs. It can further increase the provisioning efficiency for shorter interval lengths. Holt-Winters and SARIMA deliver forecasts of similar quality, with SARIMA being computationally much more demanding.

We have shown the flexibility of the genetic algorithm, which can be turned from dynamic consolidation to load balancing by simply penalizing utilization variance across the physical servers. In unison, we have shown the inflexibility of the greedy heuristic, which can not be

neglected, although its runtime is significantly shorter than those of a genetic algorithm.

Further, the genetic algorithm is easily adaptable to different circumstances, e.g. a different energy price, live migration overhead or overload costs. Especially, it offers flexibility for considering heterogeneous overload costs and live migration overheads. Generally, not all VMs will be equally sensitive to live migration, and VMs will react differently to overload situation depending on the of service they provide.

The genetic algorithm is coarse-grained parallelizable, however the speedups still give room for improvement. The greedy heuristic is computable within very short time even on low power CPU platforms, but lacks the flexibility to adapt to changing cost parameters.

5.13 Future Work

An important topic for future work will be to evaluate different cost scenarios. For live migration penalties, a similar approach to the non-linear overload cost function could be used. As shown in Chapter 6, the consequences of live migration on QoS heavily depend on the VM's utilization while it is migrated. Here, a continuous function defining the penalty of an migration happening at a specific level of utilization could replace the current, simple 0/1 penalty.

Additionally, the cost function could be extended. Possible extensions are heterogeneous overload costs for VMs, meaning heterogeneous revenue losses, thus implementing priorities. Furthermore, heavily penalizing the co-existence of two specific VMs on a server, thus implementing that some VMs should never run on the same host due to performance reasons, customer isolation, etc., is an interesting part of future work.

The computational demand of the genetic algorithm could be decreased by the storing the recalculated mappings of all control loop intervals within a sliding back-window. This could be done with an insignificant amount of required database storage. The length of the back-window should be configured with respect to the resource demand periodicity, e.g., one day. In a new interval, all known mappings would be updated with the current or forecast resource demand, and added to the population of solutions. The idea is that mappings once found to be optimal could be very close to the optimal solution in an interval of similar resource demand. Then, adding historic mappings would drastically decrease the amount of generations and hence time required to reach a high level of fitness.

Finally, an investigation about the feasibility and the potential of using a GPGPU to perform and evaluate a large number of mutations concurrently during a genetic algorithm generation should be carried out.

Chapter 6

Predicting Quality of Service During Iterative Pre-Copy Live Migration

6.1 Overview of Chapter

The ability to live migrate virtual machines (VMs) between physical servers without any perceptible service interruption is pivotal for building more energy efficient computing infrastructures in the future. Nevertheless, energy efficiency is not worth the effort if quality metrics (e.g., QoS, QoE) are severely decreased. Dynamic consolidation using live migration offers a solution to the lacking energy proportionality of today's server hardware.

In this chapter, we identify the most significant utilization metrics to predict the service level during live migrations for a web server scenario. We show important correlations, give reasons and draw conclusions for systems using live migration for yielding higher energy efficiency. We also give reasons for extending the current hypervisors' capabilities regarding VM utilization collection and reporting. We present the effects of live migration on service levels for different workload scenarios.

In particular, we demonstrate that live migration should be done preventively. This anticipates disproportional high service level degradation due to live migration. We examine the most important utilization metrics for predicting the service level by both stepwise and exhaustive regression. As a result, we can explain 90% of the service level variance during live migration with a single variable, using more variables yields 95%.

6.2 Introduction and Related Work

System virtualization became very popular during the last years. It is now widely adopted and commonly found within data centers. Interestingly, even modern data centers which already use system virtualization do not fully profit from the key benefits of virtualization and dynamic consolidation described in Chapter 4. Common reasons for data centers not utilizing dynamic resource allocation and live migration are fears concerning the quality of feasibility and possible side effects, the yet unclear potential of saving energy and money, and a manifold lack of automation in typical virtualization solutions:

1. The VMs' allocations are not automatically adapted to their resource requirements.
2. The VMs' distribution across the servers is not automatically adapted to the workload.
3. The servers are not automatically powered down and woken up with regard to the resource demand.

To solve the two latter issues, live migration is an important mechanism: It provides the possibility to establish a more energy efficient target distribution of virtual machines, as it can be used for dynamic consolidation (turning off servers) and scaling (using more or faster servers). Therefore, we consider the ability to live migrate virtual machines between physical servers without any perceivable service interruption to be pivotal for building more energy efficient computing infrastructures in the future. It enables us to dynamically shift workload between servers with regard to a cost model, where energy costs of a computing infrastructure play an important role (next to, e.g., response times).

However, only little work has been done yet to study the influence of live migration on service quality. The most related work to the efforts presented in this chapter were done by Voorsluys et al. [VBVB09]. They have evaluated the cost of virtual machine live migration for a two-tier Web 2.0 scenario. The objective of their experiments is to quantify the slow-down and downtime experienced by the application under test during live migrations. While migrating a virtual machine, they track the HTTP response time SLA (90% and 99% limits). They come to the conclusion that live migration overhead is "acceptable but can not be disregarded" [VBVB09], depending on the SLA. The authors only consider a fully loaded VM. We suggest that different workload situations should be considered, leading to a more differentiated view. Especially, we need to build models predicting the service level during live migrations. Furthermore, this thesis considers a different kind of workload, a different hypervisor and a different type of network storage. These differences in the experiments could lead to a more general perspective. A broader overview about related work was given in Section 4.6.2.

6.3 Open Issues and Own Contribution

As mentioned above, only little work has been done yet to study the influence of live migration on service quality and to build models predicting the service level during live migrations. This is an important field of research, as there is a significant increase of work being done enhancing energy efficiency of data centers. Using live migration to dynamically consolidate workload is an important part of those efforts, so we need to be able to identify and quantify possible side effects. In this chapter, we present the following contributions:

1. We show the most significant correlations between utilization metrics and the service level, give reasons and draw conclusions for systems using live migration.

2. We present the effects of live migration on service levels for different workload scenarios (low, medium, high).
3. We demonstrate that live migration should be done preventively, before the VM is even highly loaded. This anticipates disproportional high service level degradation.
4. We identify the most significant utilization metrics to predict the service level during live migrations.
5. Using stepwise and exhaustive regression approaches, we can explain 90% of the service level variance with a single explanatory variable, and 95% using 12 variables.
6. Based on our measurements and analysis, we recommend that more efforts should be made to export specific VM utilization data by hypervisors. This would ease the task of building systems using live migration as a mechanism for establishing more energy efficient VM target distributions in a scenario where response times are critical.

6.4 Experimental Setup

6.4.1 Infrastructure

For carrying out our experiments, we have used the following infrastructure, depicted in Figure 6.1: A central node is used for sheltering the virtual machine disk images and exporting them over GBit Ethernet by `open-iscsi`¹. Additionally, it acts as a management node that

- periodically live migrates a virtual machine from server A to server B and vice versa using `libvirt`²,
- logs the system utilization of all physical servers and virtual machines involved,
- tracks the HTTP response time of virtual machines,
- logs the power consumption of the servers participating in the live migration and
- logs the time the live migration takes to complete (*migration delay*).

As operating system, we use Debian Lenny AMD64 for the complete infrastructure. As virtualization layer, we use `qemu-kvm` running on `Linux 2.6.33.2`. VMs with `Apache2`, `mod-php5` and `MediaWiki` are used as web-tiers and a VM with `MySQL5`

¹<http://www.open-iscsi.org>

²<http://libvirt.org>

and database dumps of the Latin Wikipedia is used as database-tier. For HTTP workload generation, we have used `siege`³.

All servers involved have two GBit Ethernet network interfaces plugged into separate GBit switches. The first one is used exclusively for workload (here, HTTP and SQL traffic), the second one for storage, management and live migration traffic. The two servers used for benchmarking the forth- and back-migrations are built with identical components (AMD Athlon II, DDR3 RAM). The MySQL VM is hosted on an additional server (Phenom II X4 955, 8 GB DDR3) to keep network latencies comparable independent from the server the Apache VM currently runs on.

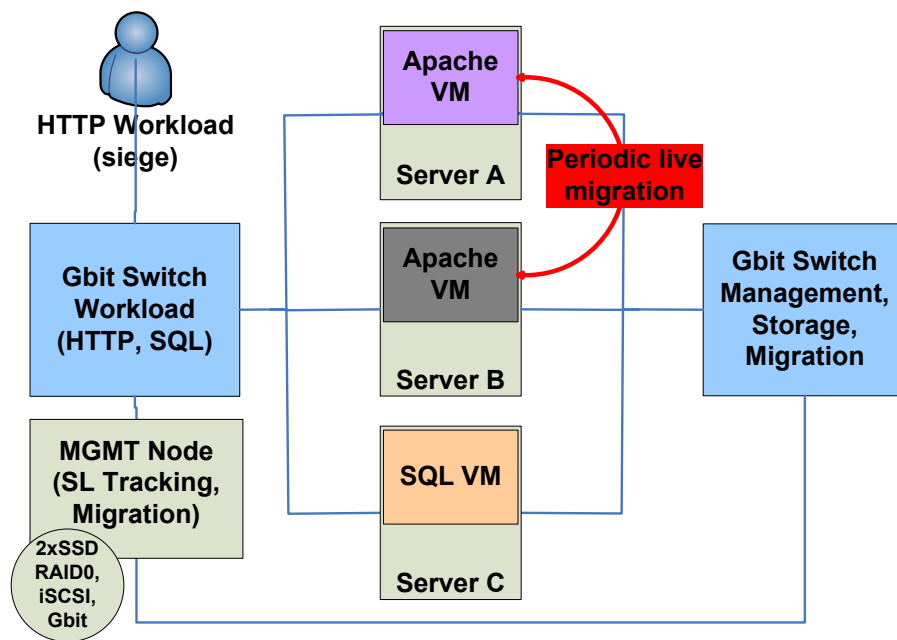


Figure 6.1: Experimental infrastructure setup.

6.4.2 Methodology

The processes described in the following Subsection are executed concurrently.

Workload Generation

After a short cache warm-up phase of several minutes, we increase the workload the VM has to bear from 50 to 600 concurrent virtual users in steps of 50, using `siege`. Each step lasts 120 minutes. Subsequently, we decrease the concurrency from 600 to 50 virtual users, in steps of 50, each step lasting 120 minutes. The think time of virtual users is evenly

³<http://www.joedog.org/index/siege-home>

distributed between 0 and 120 s. With 600 concurrent users, the virtual machine's CPU utilization is hitting 90% and severe queuing issues arise. `siege` randomly takes URLs from a text file sheltering 1000 URLs. This way we can assure that a significant amount of requests can not be completely served from cache, but must be computed.

Service Level Tracking

Every second, the central node mentioned in Section 6.4.1 sends a HTTP request, measures the response time, the HTTP status code and tracks the SLA satisfaction. The maximum allowed response time is one second, which is quite low, but resembles the fact that users are more and more accustomed to very low response times by, e.g., Google. After each request, the last 300 s of measurements are considered for calculating the service level rate.

Additionally, the service downtime during the live migration process is measured by repeatedly trying to establish a TCP connection to port 80 at intervals of 50 ms. In this period of time, already established connections are stalling, but new connections are not possible.

Utilization Logging

For measuring the utilization of servers and VMs, we use a custom developed software *MetaStat* that runs on all servers and VMs. The *MetaVisor* software on the central node uses network broadcasts to synchronously trigger all servers and VMs to start a new measurement interval, here 900 s long. This way, we can fairly assume that all nodes involved measure the same window of time. In our experiments we have observed that the amount of non-overlapping regions is less than 0.25 s, resulting in less than 0.03% for a total interval time of 900 s. To gather utilization data, we use `sar` for measuring server and VM utilization and `pidstat` to measure the individual `kvm` processes. The results are gathered over network and written to a database. In summary, we measure more than hundred utilization metrics, comprised of data about the VM and the two involved servers. To be able to study the influence of different VM memory assignments on live migration, our custom software makes sure that the ratio of the VM's free memory is in the range of 25% as a lower and 50% as an upper limit. On violation, it changes the VM's memory assignment accordingly using `libvirt`.

6.5 Data Overview

After the test run was completed, we have exported the measurement data from our database to plot the history of basic utilization metrics and our target variables. At first, we were interested in the effects of our workload generation.

Figure 6.2 shows the history of the VM's CPU utilization during our experiments. It is clearly visible that our incremental/decremental approach with equal steps causes a linear slope regarding VM CPU utilization. To find potential queuing issues, we used the `UNIX`

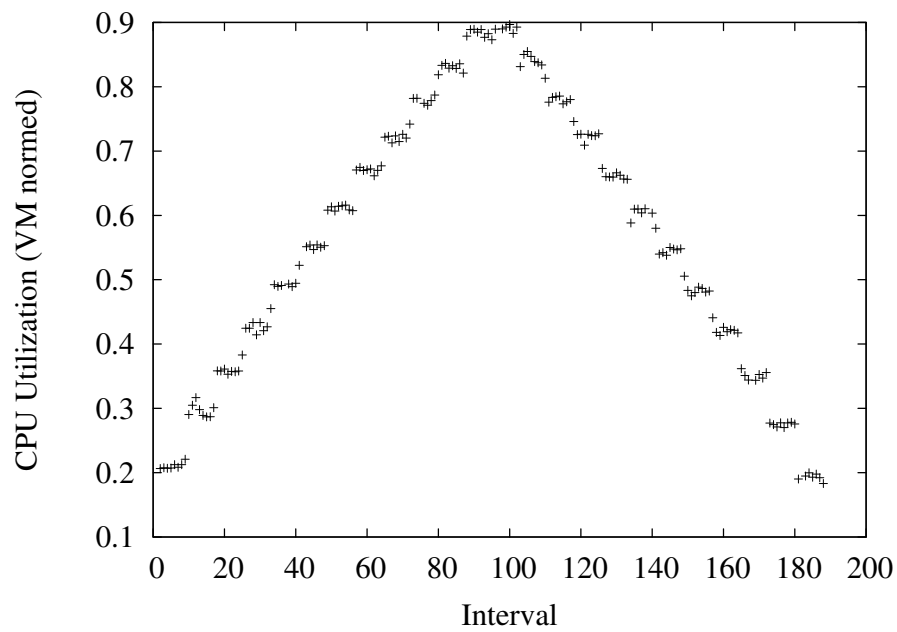


Figure 6.2: Effect of increasing/decreasing HTTP workload in equal steps on the VM's CPU utilization.

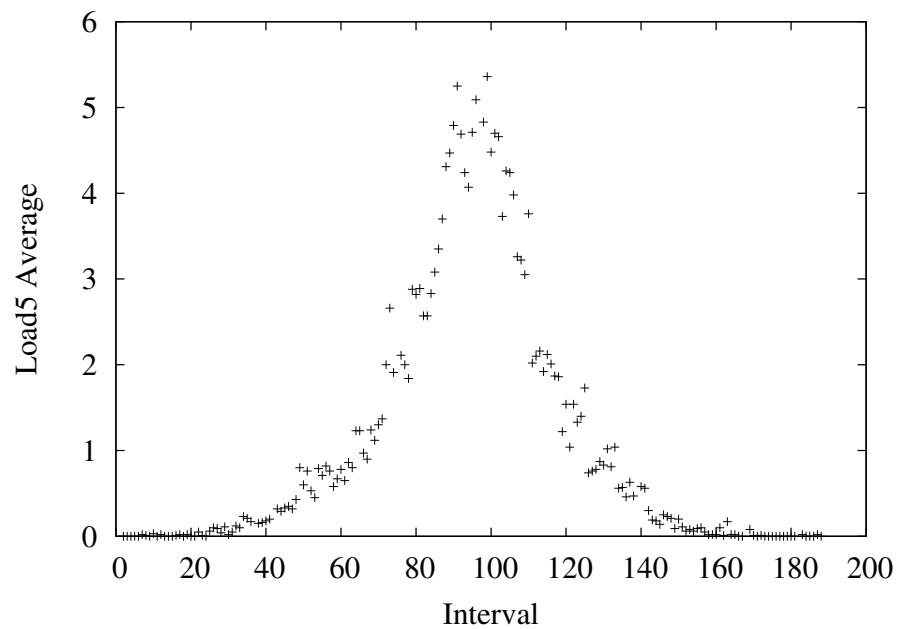


Figure 6.3: Effect of increasing/decreasing HTTP workload in equal steps on the VM's UNIX load average.

load average as a first meaningful approach. The UNIX load average is the exponentially weighted number of processes in the run queue for the last 1, 5 and 15 minute(s)⁴. The run queue consists of running and runnable processes. In general, when the length of the run queue is greater than the number of CPU cores, runnable processes have to wait until CPU resources are free again. Figure 6.3 shows the history of the VM's load5. As the VM has only a single virtual CPU, queuing issues will start to arise above a load of 1. Load averages that are multiples of the number of CPU cores are typically a sign for severe queuing issues and service quality degradation.

Comparing Figure 6.3 and Figure 6.2 results in the assumption that the load5 is getting too high when the CPU utilization rises above 60-70%. Figure 6.7 shows the correlation between CPU utilization and load5 and supports the assumption. Furthermore, we can interpret the UNIX load as approximation to $\bar{Q}$, the average number of jobs in a Markovian M/M/1 queue, and the VM's CPU utilization as ρ , the system utilization. The theoretical relation for exponentially distributed service times [BGdMT06] is

$$\bar{Q}_{M/M/1} = \frac{\rho^2}{1 - \rho}, \quad (6.1)$$

and we can see in Figure 6.7 that the measured data follow the trend of the theoretical solution very closely. As the UNIX load is exponentially averaged by definition and the service times are not necessarily exactly exponentially distributed, a systematic deviation from the theoretical solution is expectable. We can extend (6.1) to an M/G/1 queue [BGdMT06], resulting in

$$\bar{Q}_{M/G/1} = \frac{\rho^2}{1 - \rho} \cdot \frac{(1 + c_B^2)}{2}, \quad (6.2)$$

where c_B is the coefficient of service time variation. For exponentially distributed service times we have $c_B^2 = 1$ and (6.2) degenerates to (6.1). For deterministic service times we have $c_B^2 = 0$, resulting in

$$\bar{Q}_{M/D/1} = \frac{\rho^2}{2(1 - \rho)}, \quad (6.3)$$

dividing the mean queue length by two if we have deterministic service times instead of exponentially distributed service times. As visible in Figure 6.7, the M/M/1 line ($c_B^2 = 1$) is too high, the M/D/1 line ($c_B^2 = 0$) is too low for higher values of utilization. Using simple linear regression to find the coefficient between our measured values and the theoretical value of $\bar{Q}_{M/M/1}$ for a given utilization results in $c_B^2 = 0.42$. As shown in Figure 6.7, the M/G/1 line with appropriate c_B^2 fits the measured data best.

To investigate the potential role of queuing issues on the service quality level, we analyzed the HTTP response time history. Figure 6.5 shows that there is a massive increase of response times during the test run phases with higher workload. Figure 6.4 shows the HTTP response time in a service level history representation. It is clearly visible that the service

⁴a.k.a. load1, load5, load15

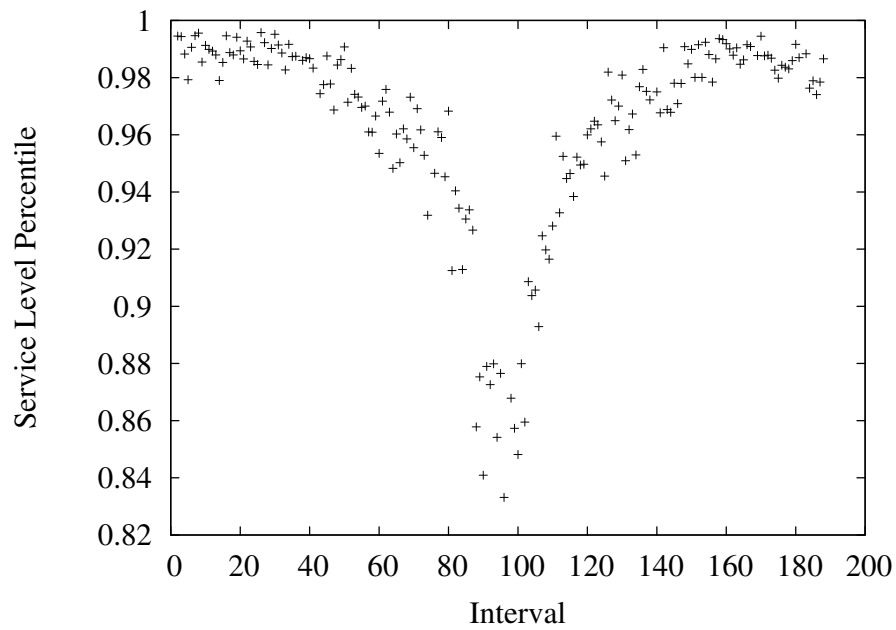


Figure 6.4: Service levels decrease significantly during phases of high workload. The maximum allowed response time is 1 s, which is extremely conservative.

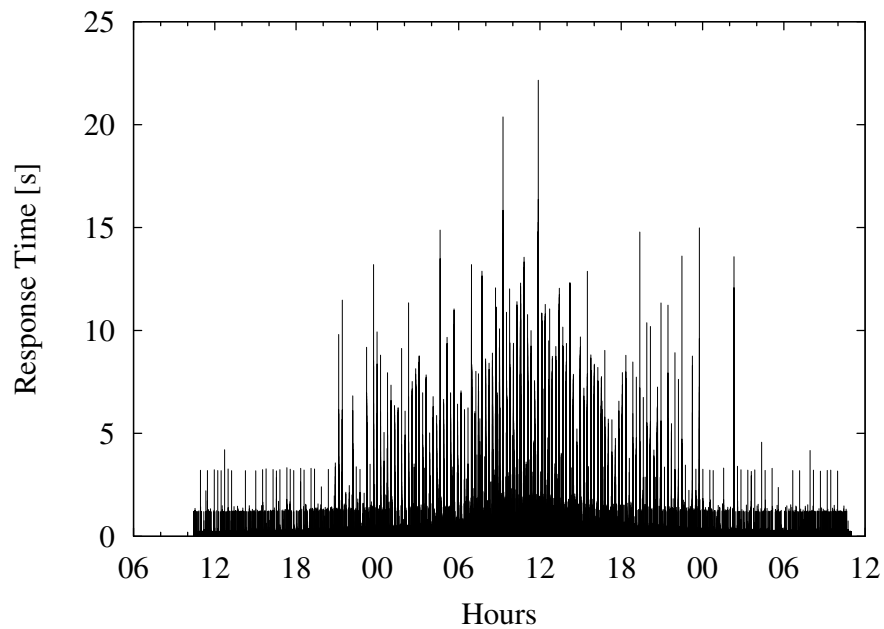


Figure 6.5: HTTP response times are massively increased during phases with higher workload.

level ratio (the ratio of all HTTP requests within the last 300s that were completed within 1 s) decreases significantly during our experiment and rapidly reaches values that are by all means unacceptable. Note that our approach of 1 s maximum response time is an extremely conservative one.

Comparing Figure 6.4 with Figure 6.3 yields the assumption that the service level is negatively influenced by a rising load average, as queuing issues become important.

Figure 6.8 shows the correlation between the load average and the service level ratio and supports the assumption. Furthermore it shows that there might be a second, non-linear correlation factor since we can see a change of tendency at a load average of approximately 4. Therefore, it is reasonable to additionally include the squared value into the pool of possible explanatory variables. Again, we can interpret the `UNIX load` as approximation to $\bar{Q}_{M/M/1}$ as described above, and the service level ratio as

$$P(T \leq x) = F_T(x) = 1 - e^{-\mu(1-\rho)x}, \quad (6.4)$$

which gives the theoretical probability that the response time T is lower than or equal to a limit x for a given service rate μ . Figure 6.8 shows that for $x = 1\text{ s}$ and $\mu = 13$, the empirically determined service rate, the measured data follow the theoretical solution closely within variance for higher values of `load`. For lower `load` values, the theoretical probability is too high. As described in Section 6.8.1, we have experienced random response time peaks that are slightly beyond 1 s even for low load, possibly caused by a configuration of our software stack that needs additional adjustment. Of course, such experimental effects are not covered by the idealization of a Markovian chain.

Figure 6.9 shows the history of service downtime during live migrations. We can conclude that there is a positive correlation with the workload the VM has to bear, but we can also see high variance. It might be necessary for future work to decrease the interval for service downtime tracking from currently 50 ms to a lower, but still technically practicable value.

We can also see high variance in Figure 6.10, which shows the migration delay history. During phases of low workload, the variance is small and the delay is positively correlated with the emerging workload. From interval 40 to 140 the delay varies from approximately 20s to 40s which is astonishing and problematic regarding stepwise regression models as AIC or LEAPS, as the data are heteroscedastic (i.e., the variance is not constant).

We suspect that the dynamic variance stems from KVM's live migration implementation. Dirty memory pages are iteratively transferred over network until the remaining amount can be transferred within a predefined (but adjustable) time limit. If the page dirtying rate is too high or the time limit is too low, the migration process would literally take forever and an administrator or a management application has to increase the time limit or cancel the process. Compared to Xen, which considers stop conditions, this is a rather suboptimal implementation. In Chapter 7, we present and evaluate a modified version of KVM, which includes stop conditions.

Figure 6.11 makes it clearly visible that there is an almost perfect linear correlation between the time a live migration takes to complete and the energy consumption of the two servers involved during this time. This can be explained by three circumstances:

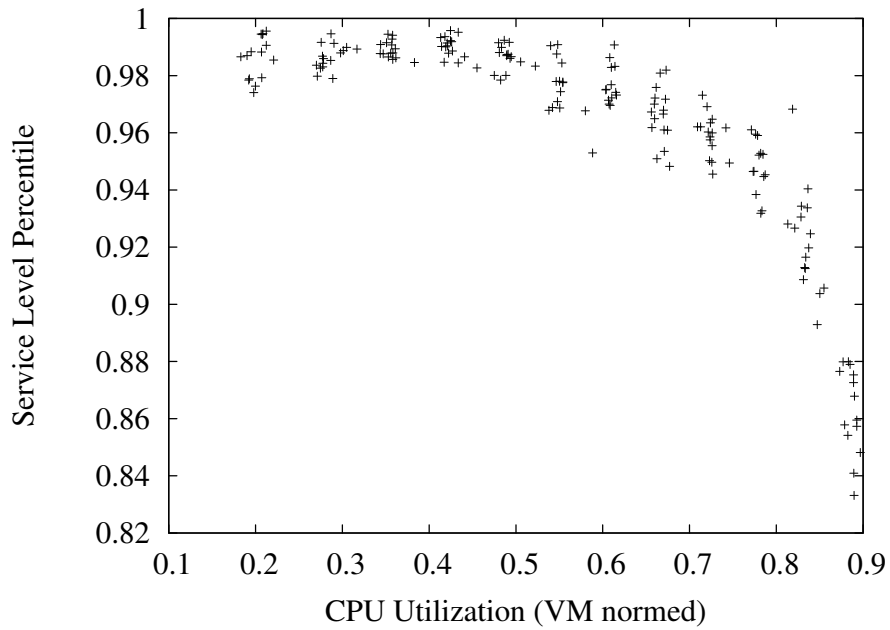


Figure 6.6: HTTP response times are massively increased during phases with higher work-load.

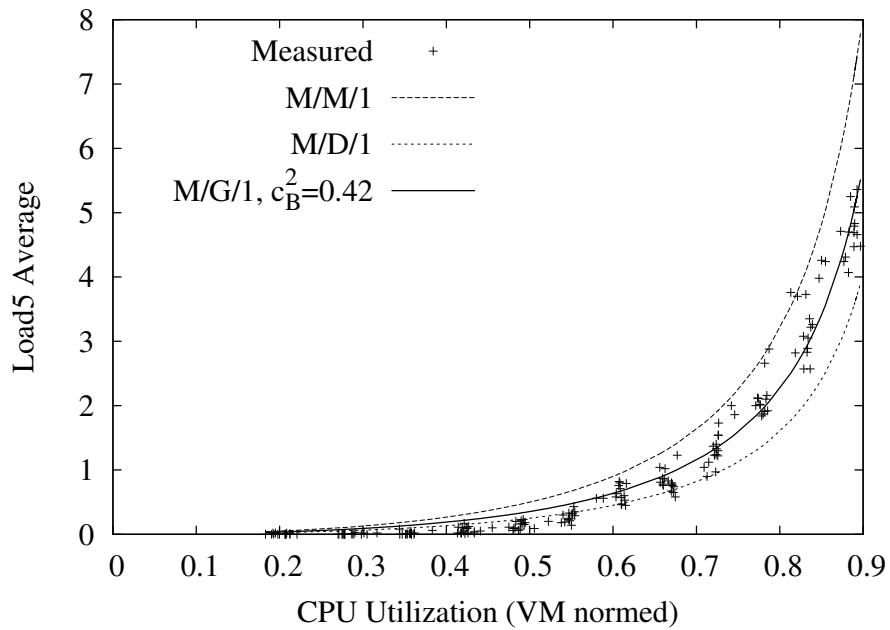


Figure 6.7: Queuing issues are rapidly increasing at 60-70% CPU utilization. The measured data follow the trend of the theoretical solution very closely.

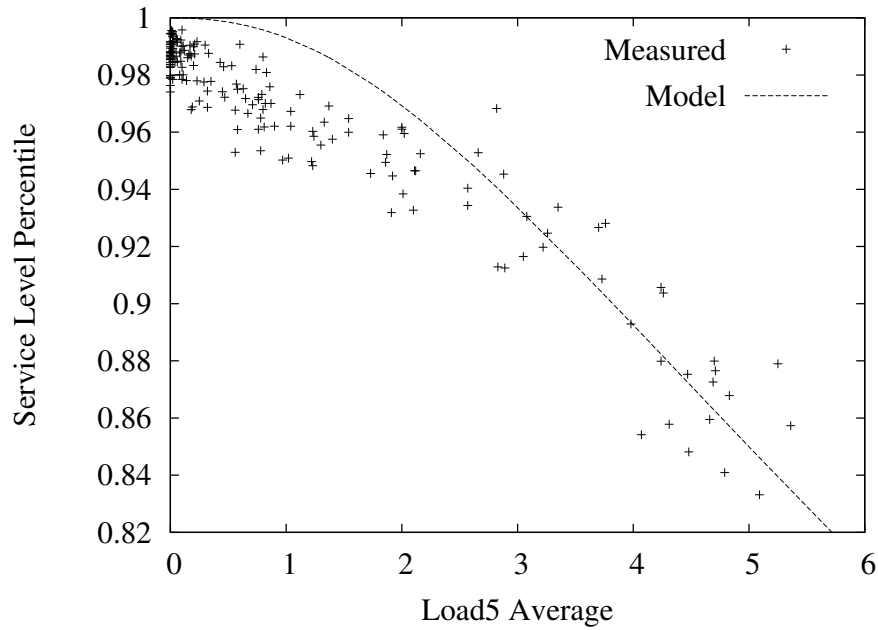


Figure 6.8: Higher load average is an indicator for queuing issues which are causing decreased service level ratios. The measured data follow the theoretical solution closely for higher load values.

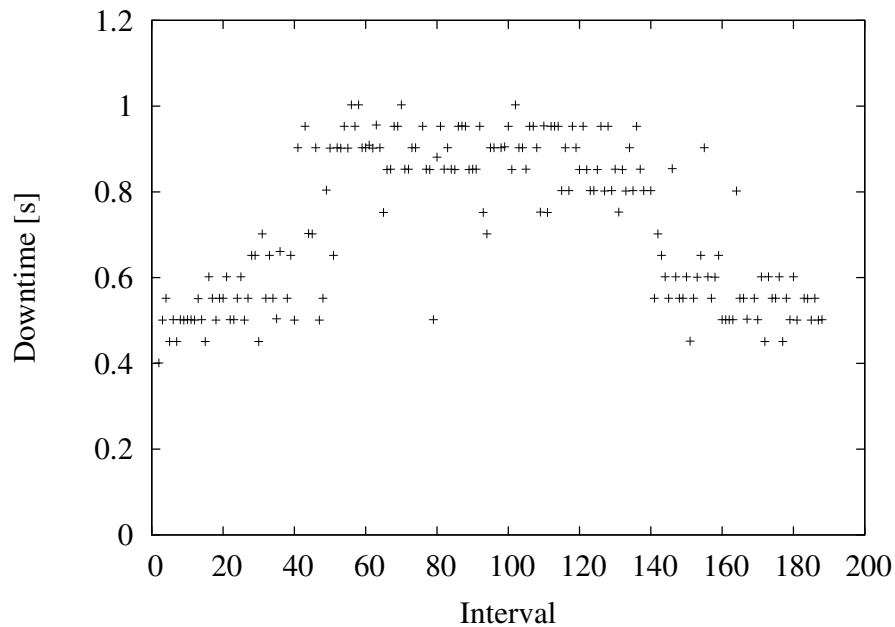


Figure 6.9: The amount of time the VM's service was not reachable during live migrations is positively correlated with the amount of workload, but also highly variant.

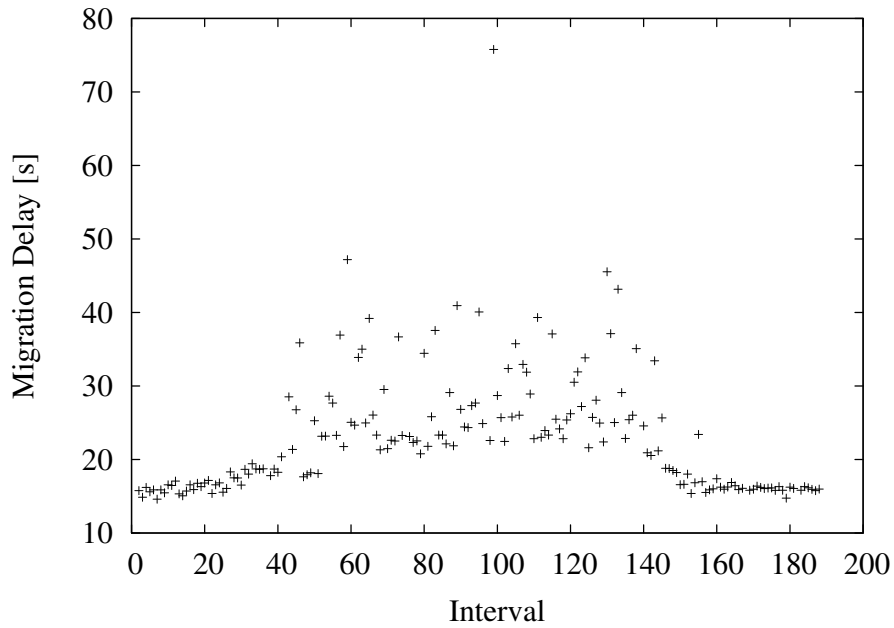


Figure 6.10: During phases of higher workload, we can see high variance in the amount of time a live migration takes.

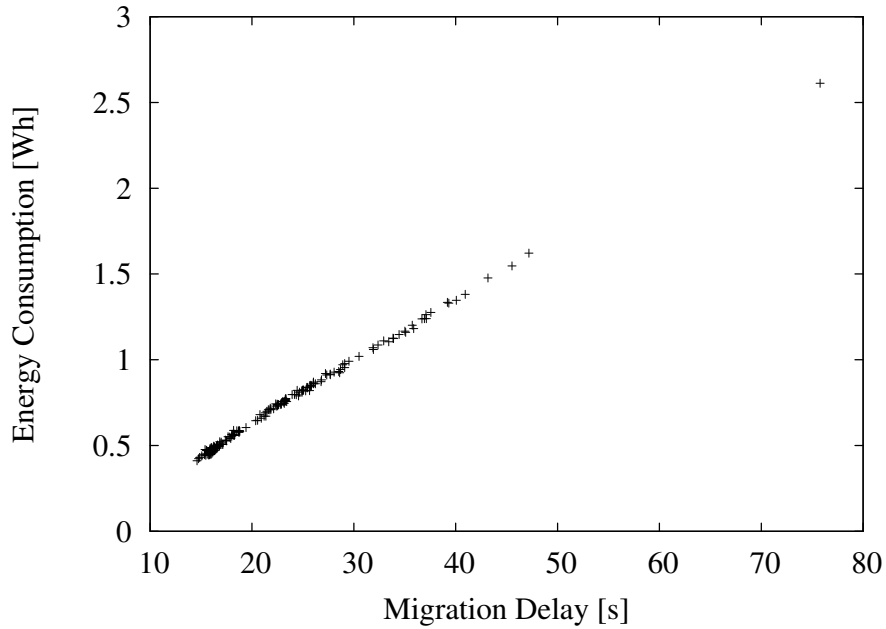


Figure 6.11: There is an almost perfect linear correlation between energy consumption and migration delay due to our experimental setup.

1. Migration delay and energy consumption are both highly positively correlated with the VM's utilization.
2. We have not limited the network bandwidth used for live migration. For migrations with different bandwidth limits, we expect less linearity in the results, as there is a base amount of energy consumption [BH07] regardless of the utilization. Due to the non-proportionality of energy consumption, we suggest that live migrations should be done with no bandwidth limit to be finished as soon as possible [DHKC09].
3. The pair of servers participating in the migration was always the same. For migrations between different pairs of servers, we expect more variance in the correlation because of the heterogeneity of pairs regarding energy consumption and efficiency.

6.6 Data Analysis

After we have gained an overview over the measurement data, we analyze them quantitatively. We are especially interested in determining the most influencing utilization metrics to be able to estimate, in this case, the service level during a planned live migration, by using only a few and therefore the most influencing utilization metrics.

The measured data consists of our response variable `slaRatio` and 131 explanatory variables originating from the VM and the two servers involved. There are three additional response variables (`migrationDelay`, the time a migration takes to complete; `wh`, the energy consumption during a migration; `downtime`, the service downtime during the final memory transfer) that are out of the scope of this work. As visible in Figure 6.11, decreasing the time required for a live migration directly decreases the involved energy consumption too. Further, a trade-off between migration delay, downtime and service quality is presented in Chapter 7.

The task of model building is to explain each response as a function of the explanatory variables. The desired models should be parsimonious (number of explanatory variables should be low) and, in relation to this number, the model quality should be as good as possible.

6.6.1 Model Quality

For yielding a high model quality, we took the following approach: After removing incomplete measurement data, we removed all explanatory variables with zero variance. E.g., the servers involved did not use swap space at any time, therefore including the respective variable is unnecessary and slows down computations.

An important measure for model quality is R^2 , the squared correlation coefficient, which explains how much better our model is with respect to the simple data average, with $0 \leq R^2 \leq 1$, where a larger value is better. We also consider the adjusted squared correlation coefficient

$$R_{adj}^2 = R^2 - \frac{J(1 - R^2)}{K - J - 1}, \quad (6.5)$$

where J is the number of included explanatory variables, K the number of data sets and $K - J - 1$ the degrees of freedom. R_{adj}^2 is important as it can decrease as a result of including additional variables. Apart from R_{adj}^2 , several statistical requirements and metrics need to be considered for achieving reliable results:

The model needs to be linear in its regression coefficients, so if there is a non-linear correlation between an explanatory and a response variable, data must be transformed accordingly. Studying the raw data, we have seen various samples for possible non-linear correlations, e.g., as discussed above for Figure 6.8. Therefore, we have additionally included the squared values of the explanatory variables (marked by `_SQ`) so that non-linear correlations can be detected during model selection.

The residuals have a population mean of zero, follow a normal distribution and are homoscedastic, which means that their variance is constant. To see whether the residuals are normally distributed, we plotted the data on quantile-quantile-plots (Q-Q-Plots). On such a plot we put the quantiles of the theoretical normal distribution on the x-axis, and the empirically measured quantiles of the data on the y-axis. If the resulting plot is nearly linear (a line) then we can conclude that the data follows a normal distribution. The Q-Q-plots of our residuals indeed mainly follow a straight line, but show deviation at the tails.

There is no significant multicollinearity between explanatory variables. If this assumption does not hold, an explanatory variable X_j is a linear combination of other explanatory variables and therefore redundant and causing unreliable results. Typically, the variance inflation factor

$$VIF_j = \frac{1}{1 - R_j^2}, \quad (6.6)$$

is used as an index for detecting severe multicollinearity [Lud07]. R_j^2 is here the squared correlation coefficient of a regression with X_j as response of all other explanatory variables. As we had collected a lot of related utilization metrics, we had to choose carefully which explanatory variables to include and which not. E.g., as only two servers and one VM are involved, the amounts of free server memory are perfectly negatively correlated.

Block device transactions and storage network traffic are highly and positively correlated, as we use `open-iSCSI`. Incoming and outgoing workload network traffic are also highly and positively correlated, as each HTTP request is followed by a response.

Additionally, lots of variables (e.g., CPU, memory, network metrics) have a common reason for their variance, which is the varying workload, causing high correlation. Finally and due to virtualization, we have measured utilization both from the hypervisor's perspective and inside the VM, as it is not guaranteed that these two perspectives necessarily and always give identical results. In our scenario, with a single VM and no background load, we experienced highly correlated results.

As with all experimental measurements, there are data outliers that are seldom but have large influence on the prediction model, although they are not helping to explain the general case. To diminish the influence of outliers, we have chosen to remove data with a minimum distance from the mean of 3σ , which is a typical trade-off between removing too much data (2σ) and removing almost no data at all (4σ).

6.7 Model Selection

We have used two different approaches for model selection, a stepwise (AIC) and an exhaustive method (LEAPS).

6.7.1 Akaike Information Criterion (AIC)

The Akaike Information Criterion [Aka74] is defined by

$$AIC = 2k - 2\ln(L), \quad (6.7)$$

where k is the number of independent parameters of a model, L the maximum likelihood, and lower values of AIC are better. It can be used in a stepwise model selection approach to find a trade-off between number of parameters (model size) and goodness of fit (model quality), as it is a mathematical formulation of the principle of parsimony in model building [Aka74]. We have used the stepwise AIC function of GNU R to build parsimonious models explaining our response variable. Starting with an empty model, in each step a variable can be both added or removed, until the AIC can not be decreased further. Of course, this procedure does not guarantee to find a global optimum, but typically gives a near-optimum result within reasonable time.

6.7.2 LEAPS

For comparison with the stepwise AIC approach, we have used an exhaustive all-subsets-regression implemented in the LEAPS package of GNU R to find the best of all possible models for a given range of model size. Evaluating all possible subsets is a computationally intensive task even if the number of variables is limited, but we wanted to know if the effort leads to higher model quality compared to stepwise approaches.

6.8 Qualitative Results

In the following, we present results for three different amounts of workload in a web server scenario. The Figures 6.12-6.14 show typical traces of the response time before, during and after several live migrations for the particular load scenario. The solid black line shows the response time (left y-axis) of an HTTP request which was started at a time depicted by the

x-axis. The maximum allowed response of 1 s is shown as a pink dashed line. The blue and green peaks indicate the time a live migration was started respectively finished. The red line (right y-axis) shows the service level percentile, the amount of all HTTP requests during the last 300 s that were finished within 1 s, and is computed after each request. The dashed lines (right y-axis) show SLA ratios that are relevant for the respective SLA trace. As we had suspected, this produces a more general view than [VBVB09] about the influence of live migration on service levels. The impact of live migrations on the service level clearly depends on the amount of workload the respective VM has to bear.

6.8.1 Low Load

When the virtual machine was facing only little load (100 concurrent users), we have experienced slightly increased response times during live migration, but the maximum response time of one second is only seldom violated.

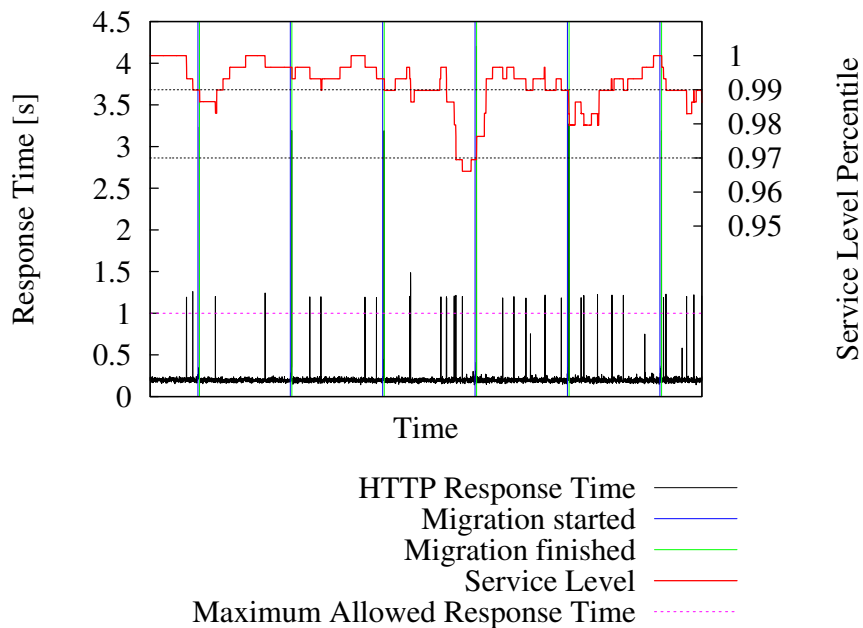


Figure 6.12: Influence of live migration on HTTP response time and service level during low load.

We can see in Figure 6.12 that the response time is very stable apart from live migrations and seldom outliers that violate the maximum allowed response time only slightly. We think that the peaks during this phase of low load are possibly caused by a configuration of our software stack that needs additional adjustment. Furthermore, cache misses and Apache/PHP process spawns are still frequent due to our randomized access pattern.

During live migrations, the maximum allowed response time (1 s) is almost always satisfied. After live migrations, the response time decreases quickly to the value before the

migration. The SLA ratio generally satisfies the 97% limit. With a slightly increased but still rather low maximum allowed response time of 1.5 s, not a single response time violation would occur. As a result of live migrations, the service level decreases, if at all, only little.

6.8.2 Medium Load

Figure 6.13 shows a typical trace for a medium load scenario (300 concurrent users). Apart from live migrations, the 95% limit is clearly satisfied, and those response times that do violate the allowed maximum are violating it mostly only slightly. However, during live migrations, the service level frequently drops below the 90% limit.

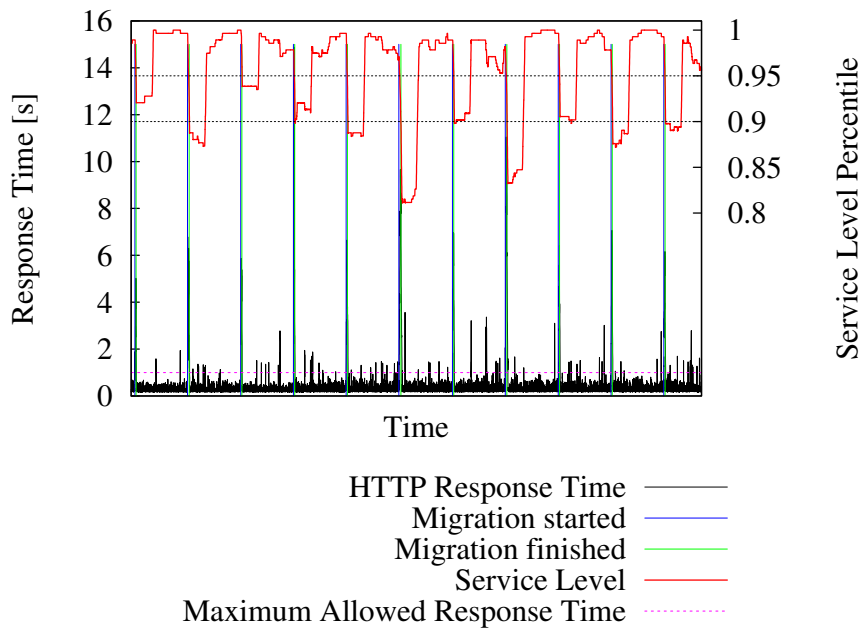


Figure 6.13: Influence of Live Migration on HTTP Response Time and service level during medium load.

6.8.3 High Load

Figure 6.14 shows a typical trace for a high load scenario (500 concurrent users). The variation in the response time apart from live migrations is rising, and response time violations are more severe in these phases. The 95% limit is only barely satisfied. This is to be expected as the VM is highly utilized and queuing effects become important. As a result, a VM with tight response time targets should not be allowed to come this far. It should have gotten more resources (in this case, virtual CPUs) earlier. From a service quality perspective, it is better to migrate several low to medium utilized virtual machines, as these VMs

inhibit a significantly less drastic impact on quality of service during live migration. This is also clearly visible in Figure 6.6. Subsequently, the freed resources can be assigned to the high utilized virtual machine.

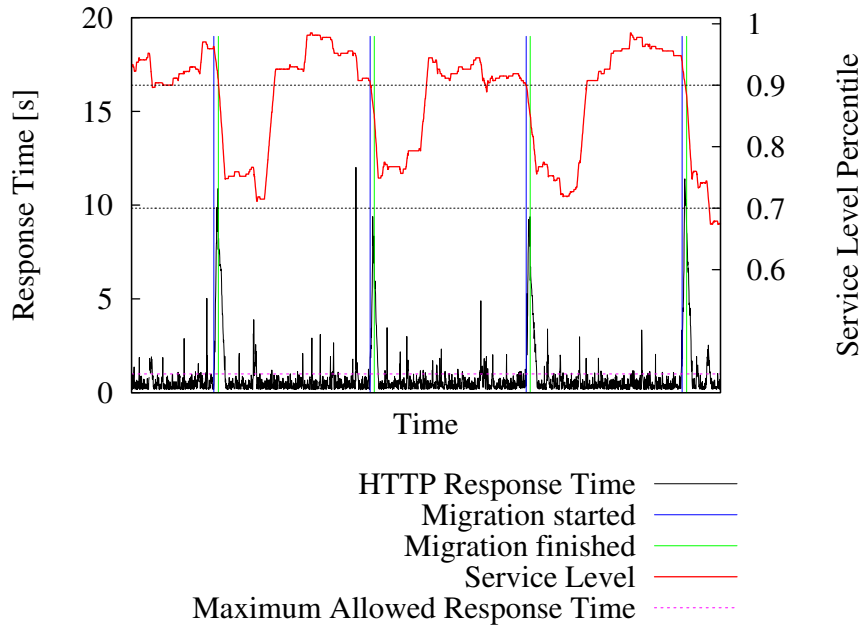


Figure 6.14: Influence of Live Migration on HTTP Response Time and service level during high load.

The result of not taking such an dynamic resource assignment is visible in Figure 6.14. Because of live migrations, the maximum allowed response time is often and heavily violated, resulting in unacceptable low service levels. The service level typically decreases by 20-25% percentage points. This would be clearly unacceptable for any professional web service.

6.9 Regression Analysis

6.9.1 Results for AIC

Using the stepwise AIC approach described in Section 6.7.1 we identified the most important explanatory variables regarding the service level. The result was a model with twelve explanatory variables plus the intercept. The most influencing and significant variables are shown in Table 6.1. We can see that a high number of squared values was selected and take this as evidence for our assumption of existing non-linear correlations. The selected model achieves $R^2_{adj} = 0.9569$ with $\sigma = 0.0130$, the residuals' standard deviation, which are good values for non-synthetic experimental measurements.

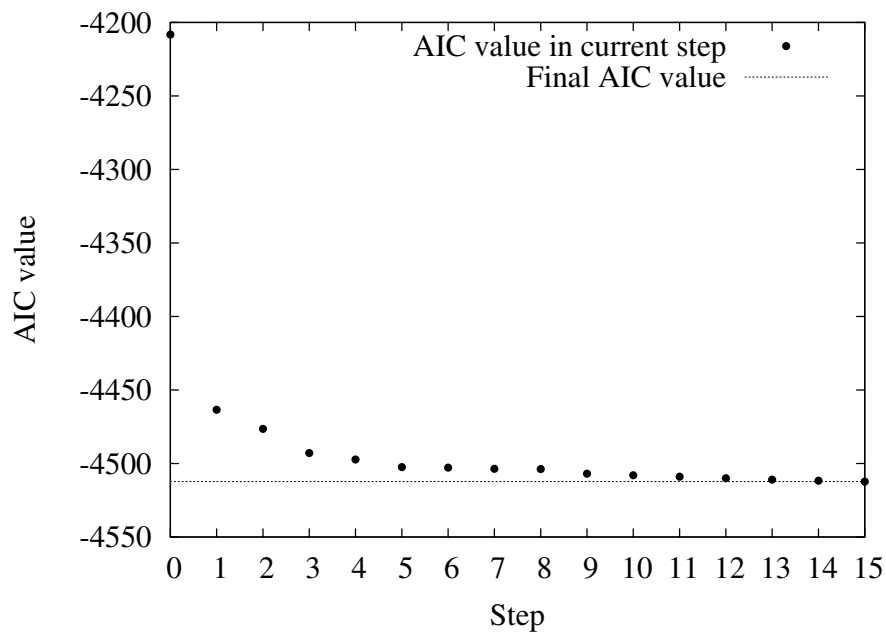


Figure 6.15: The AIC value quickly improves during the first steps and then slowly converges to the final value.

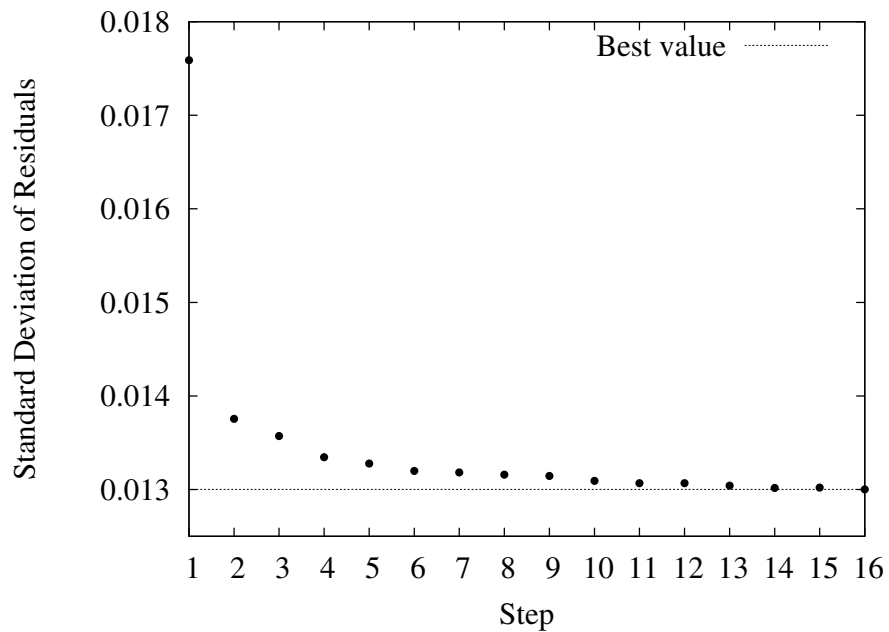


Figure 6.16: The residuals' standard deviation quickly improves during the first steps and then slowly converges to the final value, when using stepwise AIC.

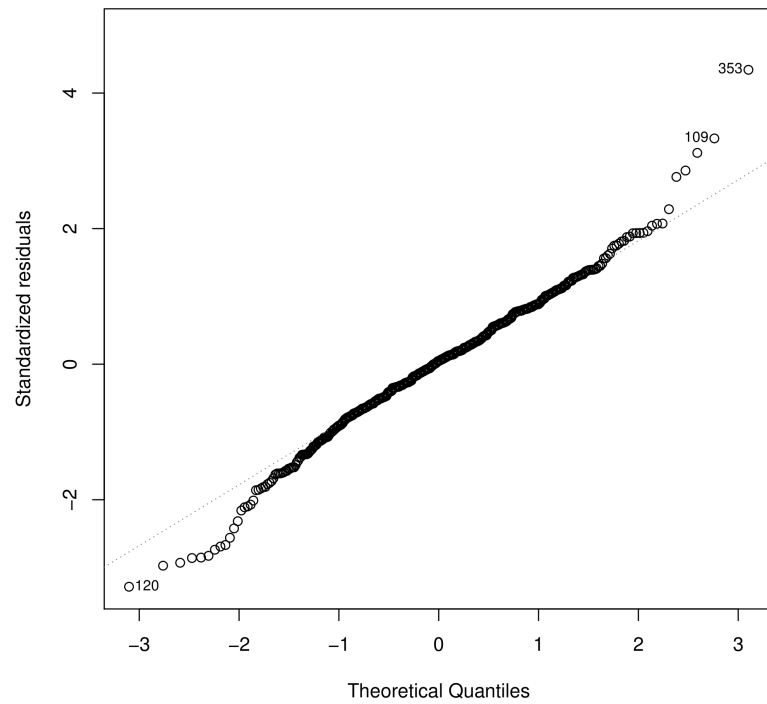


Figure 6.17: QQ-Plot of standardized residuals for the model produced by stepwise AIC.

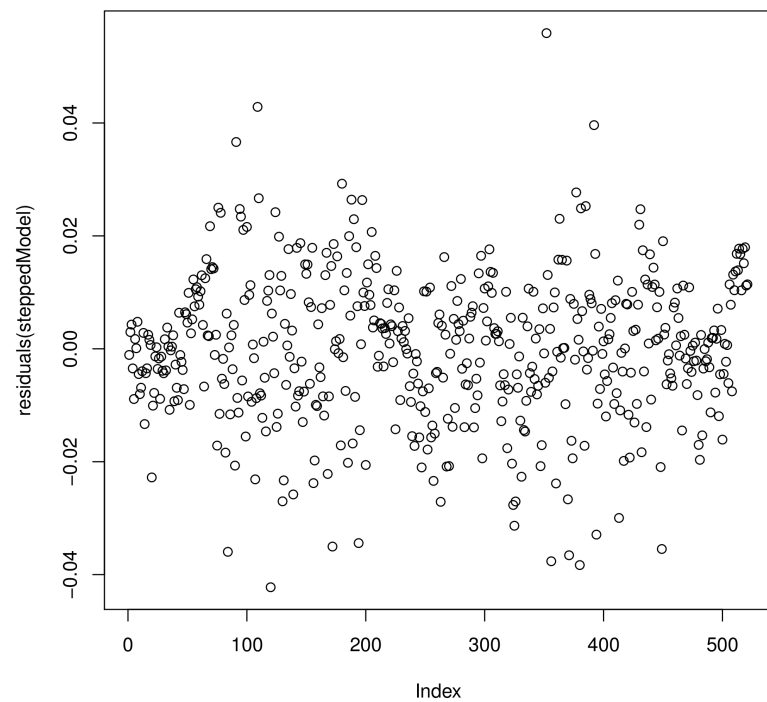


Figure 6.18: There is no visible correlation between the residuals' variance and the time series.

Figure 6.15 shows how the final model results from a process that minimizes the AIC value. We can see that the AIC value improves quickly during the first steps and then converges to the final value only slowly. As a consequence, models with less variables than the final one could suffice, depending on requirements regarding model quality. Possible solutions are a) limiting AIC to a number of steps, b) fixing the number of variables as it is possible with LEAPS.

In Figure 6.16, the residuals' standard deviation after each step is visible. We can see that the regression's error is low even during the first steps and is further minimized step by step.

Figure 6.17 clearly shows that the residuals of the selected regression model are following a normal distribution (depicted by the dotted line) very closely, only in very marginal areas there are a few outliers.

In Figure 6.18, the residuals are plotted against the measurement index and show that their variance does not significantly depend on the position in the time series. In order to examine the generalizability of the models produced, we have performed a leave-one-out cross-validation. When having n data samples, this procedure creates n different models with $n - 1$ training samples and a single test sample, so that each data sample is used as a test sample for a model that was created without using the respective sample as training data. Applying this method yields a prediction error of 0.000179. As a consequence, our approach delivers highly generalizable models.

6.9.2 Results for LEAPS

For comparison with the stepwise AIC approach, we have used an exhaustive all-subsets-regression (LEAPS) to find the best of all possible models for a maximum number of twelve explanatory variables (plus the intercept), which is consistent with the final model complexity of stepwise AIC. Evaluating all possible subsets is a computationally intensive task even if the number of variables is limited, but we wanted to know if the effort leads to higher model quality compared to our stepwise approach using AIC.

As a result, the LEAPS approach yields a model using only `load5` which explains 90% of the variance ($R_{adj}^2 = 0.9$, $\sigma = 0.0247$). Figure 6.19 shows that more complex models with a higher number of explanatory variables result in better values of R_{adj}^2 , but the increase is rather small as the model with 12 variables yields $R_{adj}^2 = 0.94$ with a residuals' standard deviation of $\sigma = 0.0159$.

In Figure 6.20, the value of σ for each model size is visible, and gives an impression similar to Figure 6.16, although the regression's error does not improve that quickly and only reaches slightly higher values. Considering the significantly higher computing demands of an exhaustive approach compared to a stepwise and the slightly better quality of the stepwise approach, we conclude that the stepwise AIC regression approach is preferred for the described scenario.

A further, significant utilization metric our results showed is the CPU utilization of the VM. Figure 6.6 shows the relation between the CPU utilization of the KVM process and

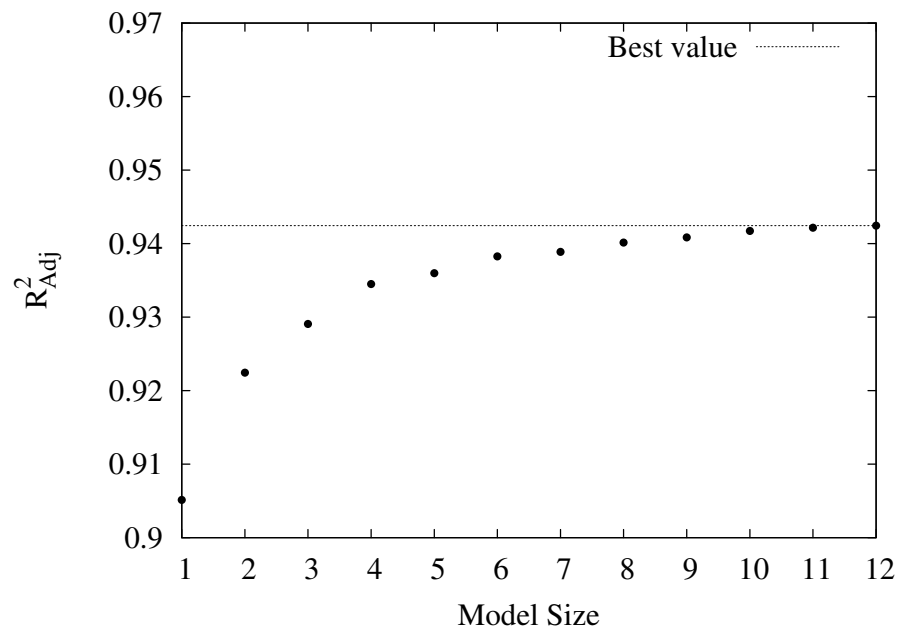


Figure 6.19: Higher model complexity yields better model quality when using LEAPS, but the increase is rather small.

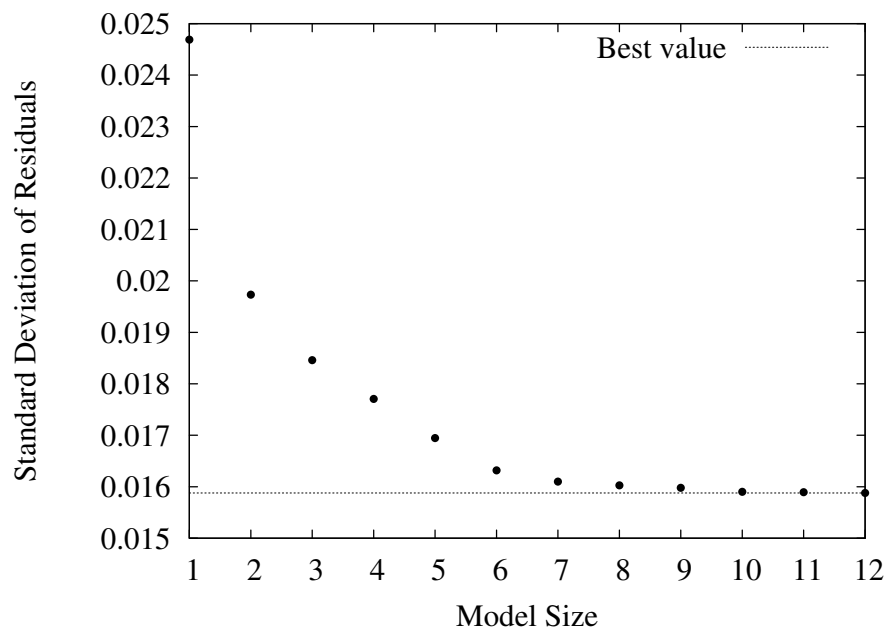


Figure 6.20: Higher model complexity yields better regression error when using LEAPS.

the service level. Again, there is a clearly visible correlation, although not linear, but polynomial. The explanation is that the higher the CPU utilization gets, the higher the amount of dirty memory pages per memory transfer iteration typically gets. This assumption will not hold for extremely CPU intensive computations with very little memory requirements (e.g., CPU stress tests), but for most types of real life workload it will. We can conclude that for most scenarios, a pro-active migration (when the CPU utilization has not exceeded, e.g., 60-70%) will be beneficial if the impact on service level is to be minimized.

Variable	Meaning	Estimate	Std. Error	$Pr(> t)$
Intercept		2.395e+00	5.069e-01	3.00e-06
wp01_load5	The UNIX load5 of the VM.	-1.871e-02	2.627e-03	3.67e-12
wp01_swapUsed	The amount of swap used by the VM.	-7.656e-07	8.809e-08	< 2e-16
wp01_residentSize_SQ	The squared amount of resident memory used by the qemu-kvm process.	-4.652e-14	1.652e-14	0.00506
src_host_cpu_proc_s	The number of tasks created per second on the source host.	-2.475e+00	9.166e-01	0.00716
src_host_cpu_proc_s_SQ	The squared number of tasks created per second on the source host.	1.091e+00	4.133e-01	0.00856
wp01_cpu_util_vmnorm_SQ	The squared CPU utilization measured inside the VM.	-1.328e-01	3.187e-02	3.63e-05
wp01_cpu_util_vmnorm	The CPU utilization measured inside the VM.	9.517e-02	2.316e-02	4.64e-05
wp01_load5_SQ	The squared UNIX load5 of the VM.	-1.140e-03	1.918e-04	5.22e-09
wp01_freeMemRatio_SQ	The squared ratio of free memory inside the VM.	1.976e-02	9.462e-03	0.03727

Table 6.1: The most influencing and significant variables of the final model produced by stepwise AIC.

6.10 Discussion

We draw the following conclusions from our measurements and analysis regarding service level compliance for a virtualized web server infrastructure:

The impact of live migration on service levels depends not only on the SLA, but also on the amount of workload the respective VM has to bear. Tighter service level agreements can be fulfilled for low workload situations. For high workload situations, the service level decreases massively because of live migrations, therefore such situation should be avoided.

As a consequence, live migrations should be triggered before a virtual machine is heavily loaded, if service level agreements are to be fulfilled or can not be relaxed generally. For medium workload situations, the service level typically stays within the 95% limit despite live migrations. Arguably, the 95% limit could suffice in such situations. It would be only temporary to avoid further, massive decrease of service levels, by migrating the VM to a server with more resources.

Multiple live migrations within short time would cause even stronger deviations in the service level ratio. As a consequence, we recommend that a VM which has been migrated recently should exhibit an inertia effect, not only to avoid senseless flip-flop migrations, but to stabilize the service level. Systems and algorithms using live migration to dynamically shift workload should consider this.

If possible, it is better to migrate several low to medium utilized virtual machines, as these VMs inhibit a significantly less drastic impact on quality of service during live migration. Subsequently, the freed resources can be assigned to the high utilized virtual machine.

It is possible to predict the service level variance during a live migration to 90% using only a single variable, the `UNIX load5 average`. Extending the model to 12 variables using LEAPS yields $R^2_{adj} = 0.94$, the stepwise AIC approach results in a model with 12 variables and $R^2_{adj} = 0.95$ and a residual standard error of $\sigma = 0.013$. The models selected are of comparable quality and are robust to statistical tests.

As a consequence, systems using live migration as a mechanism to realize a more energy efficient target distribution need to consider the `UNIX load average`, if service levels during live migrations are important. Gathering `load` information of VMs currently needs to be done by VM introspection, as typical hypervisors do not collect and export this information. There are related efforts by `qemu-kvm` and `libvirt` developers to pass-through the VMs' memory utilization, as the amount of free memory can not be reliably observed directly by the hypervisor.

Therefore, we recommend that such efforts should be extended to additionally export `load` information. Further utilization metrics can be collected too, but for the described scenario, the `UNIX load average` is the most important one and should not be disregarded.

6.11 Future Work

We plan several extensions to the status quo of our work. In our setup, we had only a single web server VM that was under load. It would be interesting to investigate the influence of additional VMs, for an idle, a utilized and a mixed scenario. When hosting more than one VM on a server, positive and negative effects of `Linux`' and `qemu-kvm`'s *kernel samepage merging* (KSM) should be evaluated too. KSM is basically a deduplication mechanism of identical memory pages across processes with copy-on-write.

Furthermore, an examination of the influence of live migrating to the database VM could lead to important results, as migrating database VMs is generally seen as a taboo because failures can affect multiple VMs relying on the database service. We also plan to experiment with different hypervisors and services to validate our results more generally.

An additional area of future work are the live migration parameters `qemu-kvm` provides, e.g., enforcing bandwidth limits on the memory transfer, and increasing the maximum allowed downtime during the final memory transfer.

Finally, we plan to identify the most influencing utilization metrics for the other response variables we have measured (delay of the live migration, energy consumption during the live migration and service downtime during the final memory transfer) and to create parsimonious models that predict the response with high precision.

Chapter 7

Service Level Management for Iterative Pre-Copy Live Migration

7.1 Overview of Chapter

In this chapter, we extend the efforts presented in Chapter 6, where we have identified the most significant utilization metrics to predict the service level during live migrations for a web server scenario. Considering only the `UNIX load average`, we were able to explain 90% of the service level variance during live migration. We have demonstrated that live migration should be done preventively to anticipate disproportional service level degradation due to live migration. However, our experiments have focused on the KVM implementation of iterative pre-copy live migration, and a fixed set of parameters.

In this chapter, we present results for a low level approach by patching KVM to implement Xen's live migration stop conditions, and a high level approach by monitoring the progress and service level state of a live migration and estimating the impact on the service level during the remaining migration time. We compare these approaches with an unpatched, vanilla KVM version and different sets of parameters used for live migration.

We show that iterative pre-copy live migration should not be implemented without either stop conditions at the hypervisor level or a management facility for infrastructures with QoS targets. Both approaches result in significantly better service levels and prevent the VM from thrashing and complete service disruption.

7.2 Introduction and Related Work

As further described in Section 3.5.2, iterative pre-copy live migration is depending on the assumption that the amount of dirty pages quickly converges to a minimum. Then, service interruptions during the final iteration are imperceptible. If this assumption of convergence does not hold, the live migration may run indefinitely. A clear case of lacking convergence is when the page dirtying rate is exceeding the page transfer rate. Another problematic situation occurs if the page dirtying rate is getting close to the transfer rate, although not necessarily exceeding it. Then, the migration process will converge only very slowly, caus-

ing severe service degradation due to the lengthy and resource intensive migration process. As a consequence, stop conditions are necessary to ensure that the process does finish eventually. In Xen, three stop conditions [ASR⁺10] are defined, as mentioned in Section 3.5.2. For KVM, the situation is different. In contrast to [KKL⁺07a], where two stop conditions are described, the actual current implementation has no stop conditions at all.

Liu et al. [LXJ⁺11] have presented a model able to predict the time an iterative pre-copy live migration takes to complete, based on measuring the page transfer rate and the page dirtying rate. Information about dirty pages is gathered by adapting the Xen hypervisor. They have evaluated migration delay, downtime, network traffic and energy consumption for various applications running inside a migrated VM, e.g., SPECweb2005. However, quality of service metrics, e.g., response times and service levels, are not part of their evaluation.

7.3 Open Issues and Own Contribution

In this chapter, we extend the results discussed in Chapter 6 by evaluating different implementations of iterative pre-copy live migration and their effects on web service levels. Especially, we have implemented a low level solution by patching KVM to implement Xen's stop conditions, and a high level solution by using a model able to estimate the remaining migration time and the consequences on the service level.

We evaluate how estimating the remaining migration time can be used to optimize QoS during migration for a KVM hypervisor and a web server VM, without making changes to the hypervisor itself. After examining the model presented by Liu et al. [LXJ⁺11] regarding validity for our experimental setup, we extend it by a pro-active service level management component, taking evasive action if a service level target is predicted to be violated due to the live migration.

We compare these approaches regarding their effects on service levels during live migration with the current KVM implementation and different migration parameters. Due to an iterative migration process that is not converging, trying to reach unachievable low downtimes causes unacceptably low service levels, as the migration process itself is increasing the resource pressure onto the VM for a too long period of time.

We show how to achieve acceptable response times and service levels during live migration also for higher workload intensities by trading off migration delay and migration downtime. Both of our approaches result in significantly better service levels and prevent the VM from thrashing and complete service disruption. As a result, iterative pre-copy live migration should not be implemented without either stop conditions or a management facility taking pro-active evasive action for infrastructures with QoS targets.

7.4 Experimental Setup

7.4.1 Infrastructure

For carrying out our experiments, we have used the same infrastructure as described in Section 6.4.1. However, there were slight modifications:

- Instead of open-iscsi, NFS was used to export the virtual machine disk images over GBit Ethernet.
- Instead of Debian Lenny AMD64, the newer version Debian Squeeze AMD64 was used for the complete infrastructure.
- The virtualization layer was updated too, we have used `qemu-kvm-1.0` running on `Linux 3.2.0`.
- Instead of Apache, `lighttpd` with `php5-fastcgi`, `xcache` was used to host the MediaWiki software.

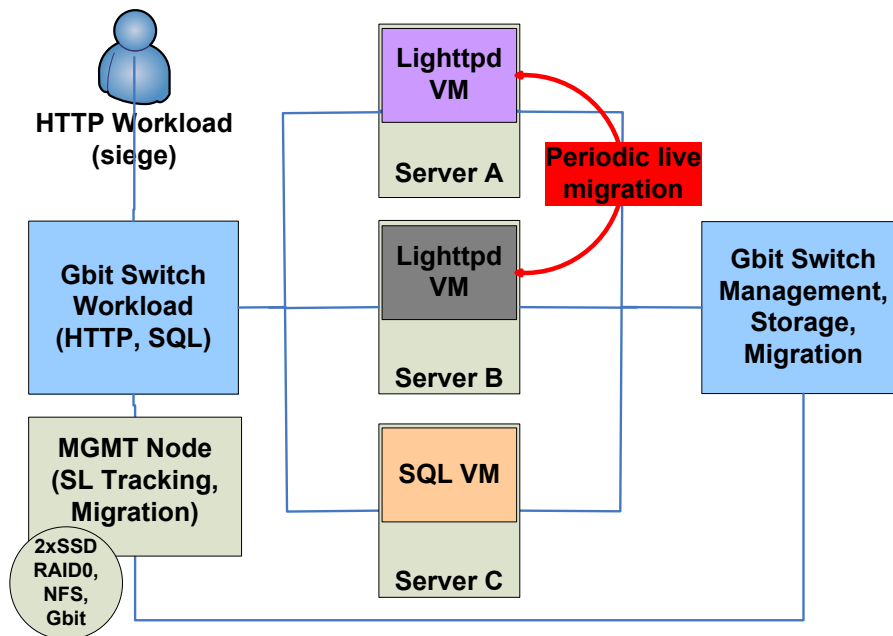


Figure 7.1: Experimental infrastructure setup.

7.4.2 Methodology

Furthermore, we have employed a methodology similar to Section 6.4.2. The generation of workload and the logging of utilization data was identical. For the tracking of the service level, we have set the maximum allowed response to 1.5 s in order to eliminate seldom violations probably caused by regular PHP session cleanups. More importantly, we have increased the measurement resolution of the service downtime during the live migration process. The service downtime is measured by repeatedly trying to establish a TCP connection to port 80 with a socket timeout of 10 ms, instead of previously 50 ms. During the downtime, already established connections are stalling and new connections are deferred. If the downtime is not longer than socket timeouts, clients should just notice a delay during the connection establishment. The most important methodical change is that we have logged the progress of the live migration, described in the following.

Migration Progress Logging

During the live migration, we track its progress every second using `libvirt`, which provides macroscopic progress information. Most important are 1) the amount of memory transferred over network, denoted as *processed memory*, which is not to be mistaken for the amount of clean memory pages, and 2) the amount of remaining memory, in the following defined as *dirty pages*. The iterative pre-copy live migration mechanism copies *dirty* pages to the destination host and marks them to be *clean*, while the VM is still running. Thus, pages can be written concurrently and become *dirty* again, therefore they must be copied over the network again in the next iteration. Based on these two progress metrics, we estimate the *dirty page rate*, which we define as the amount of memory that changed its status from clean to dirty per second during the interval i , by

$$D_i = \frac{(p_i - p_{i-1}) - (d_{i-1} - d_i)}{t_i}, \quad (7.1)$$

where p_i is the amount of processed memory at the end of interval i , p_{i-1} the respective amount at the end of interval $i - 1$, d_i and d_{i-1} are the respective amounts of dirty memory, and t_i is the length of interval i in seconds. Note that the amount of processed memory is continuously growing throughout the whole migration with every page that is transferred over the network. Further, we assume $d_i < d_{i-1}$, meaning that the process is at least slowly converging. The basic idea of our approach is to compare the increase of processed memory with the decrease of dirty pages. If no memory page that was already transferred over the network would be written again by the VM, then

$$p_i - p_{i-1} = d_{i-1} - d_i \quad (7.2)$$

or, equivalently

$$(p_i - p_{i-1}) - (d_{i-1} - d_i) = 0, \quad (7.3)$$

as each processed page would perfectly equate to a dirty page less. However, the VM is active during all but the last pre-copy iteration and can cause memory pages change their state from clean to dirty. Thus, the increase of processed memory does not equate to the decrease of dirty memory, yielding

$$(p_i - p_{i-1}) - (d_{i-1} - d_i) > 0, \quad (7.4)$$

and we use this difference to estimate the amount of memory that changed its state from clean to dirty in the last interval. The alternative to this estimation would be to modify the hypervisor to frequently dump a huge bit vector signaling the state of each memory page and to analyze their development over time. Of course, such efforts would result in a more detailed progress information, e.g., the set of pages that become dirty again most frequently would be detectable. However, we use the macroscopic progress information provided by `libvirt` as a first step. We also calculate the transfer rate within each interval i ,

$$R_i = \frac{(p_i - p_{i-1})}{t_i}, \quad (7.5)$$

which is simply the amount memory transferred in the interval i . Naturally, if the page dirtying rate is getting close to the transfer rate, then the live migration process is not progressing, as the amount of dirty pages is not converging towards zero.

We use the transfer and dirty page rate in Section 7.5.5 to estimate how long a migration will take to complete and the corresponding impact on the service level. As live migration is a resource-intensive task, the migrating VM is facing additional resource pressure. If it is already highly loaded, QoS will severely suffer from a migration converging too slowly or even not at all.

7.5 Results

7.5.1 Unpatched KVM

In a first experiment, we have evaluated an unpatched default version of KVM. There are no stop conditions ensuring that the live migration eventually finishes, if the amount of dirty pages does not converge towards zero. Additionally, there is a hard coded bandwidth throttle built into KVM's live migration mechanism. In `qemu-kvm-1.0/migration.c`, line 42 says:

```
#define MAX_THROTTLE (32 << 20) /* Migration speed throttling */
```

`MAX_THROTTLE` is then used as a hard limit regarding migration speed, its unit is Byte/s. Thus, KVM limits the migration speed by default to 32 MiB/s, or 256 Mbit/s. On the one hand, this is far more than 100 Mbit Ethernet is able to transfer, so TCP congestion mechanisms will be triggered anyway. On the other hand, Gbit Ethernet offers approximately four times the bandwidth, so three quarters of its bandwidth are never used. Until recently,

it was impossible to override this hard coded limit externally, without changing the source code. In `qemu-kvm-1.0/migration.c`, line 120, the default value for the maximum allowed downtime is set:

```
static uint64_t max_downtime = 30000000;
```

The value is in nanoseconds, so the maximum allowed downtime during the final iteration is by default 30 ms. However, we have found in our measurements that the true downtime is multiple times larger than the value set here. In our previous works, we have observed live migrations that never finish, when the load becomes too high. To find a remedy, we had to set `MAX_THROTTLE` to 1 Gbit/s and to increase the maximum allowed downtime to make convergence more likely. Here, we compare the default implementation with a patched version implementing Xen's stop conditions and a high level approach by monitoring the progress and service level state of a live migration and estimating the impact on the service level during the remaining migration time.

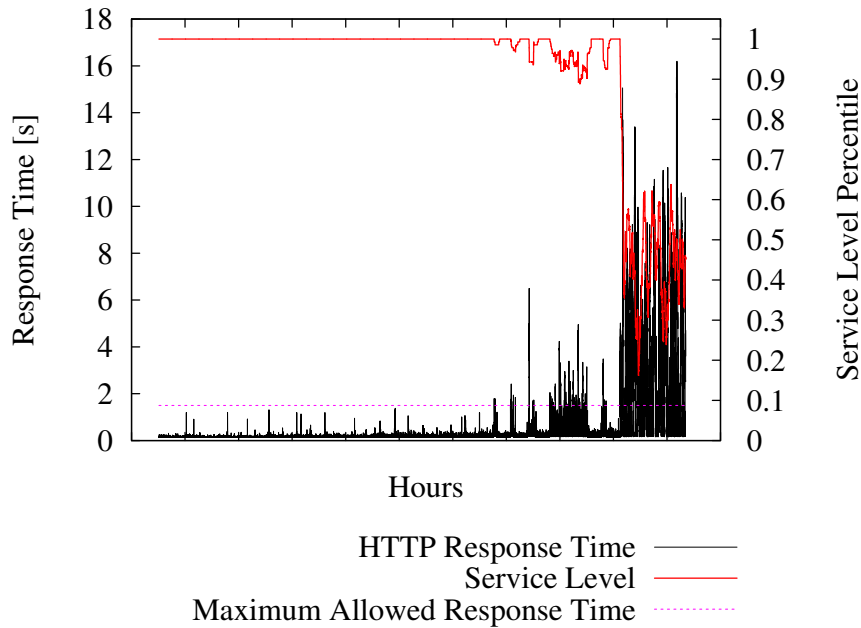


Figure 7.2: Influence of Live Migration on HTTP Response Time and service level during rising load for unpatched KVM.

In Figure 7.2, the development of the service levels during rising load is shown for KVM's default implementation. Due to the limited page transfer rate of 32 MiB/s, the service level suddenly decreases as the migration does not converge. This happens at a CPU utilization of approximately 30%, and the experiment was stopped as migrations started to take an hour to complete. In Figure 7.9, we can see the migration delay increasing exponentially when the page dirtying rate gets closer to the fixed page transfer rate. Obviously, the migration does not converge, and is eventually finished only because the VM's service severely suffers from

the migration, meaning it can do less and less page dirtying. In a real scenario, users would quickly stop using the service due to a catastrophic quality of service, as response times massively increase and network timeouts occur regularly. Having a single, longer downtime is most likely better than a potentially everlasting unresponsive VM.

7.5.2 Bandwidth Increased KVM

Our first step to change KVM's clearly improvable default settings was to increase the bandwidth used for dirty page copying to 1 Gbit/s in order to actually utilize our network's available bandwidth. In Figure 7.3, the result is shown. In contrast to the default bandwidth setting, the system is able to migrate a higher utilized VM too, but the service level is rather unstable during migrations and shows random outliers. As visible in Figure 7.10, above a CPU utilization of approximately 40-50%, the service level starts to decrease below 90%. As the utilization increases further, the service level regularly drops below 80%.

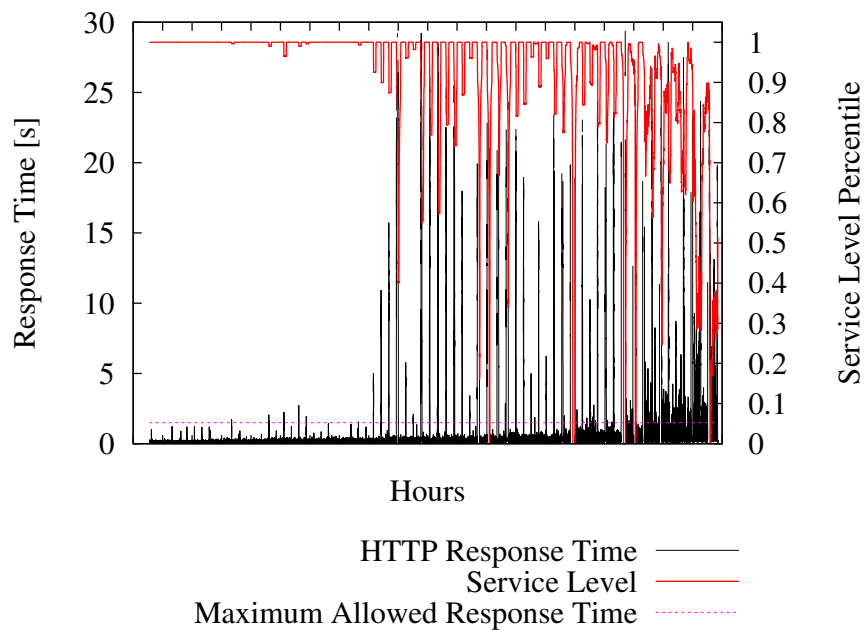


Figure 7.3: Influence of Live Migration on HTTP Response Time and service level during rising load for unpatched KVM with 1 Gbit/s migration speed.

7.5.3 Bandwidth Increased KVM, Strict Downtime

Next, we were interested in KVM's migration performance for 1 Gbit/s of bandwidth, but a strict downtime during the final iteration of 4 ms. Without any stop conditions, the migration will only complete if the system is able to converge to a point where the remaining

dirty pages are transferable within 4 ms. In Figure 7.4, the development of HTTP response time and the service level is shown. We can clearly see how response times suddenly and massively increase, and the service level drops to zero. With further increased load, our system is even unable to make any HTTP connections. The situation is so severe that the VM is not becoming responsive again, even minutes after live migrations. Eventually, the experiment is aborted as `siege` is running into regular time-outs when trying to connect to the VM. Consequently, trying to achieve a low downtime during the final iteration has a drastically negative impact onto the service level. In the following, we present and evaluate two approaches that trade-off migration delay and migration downtime with respect to the service level.

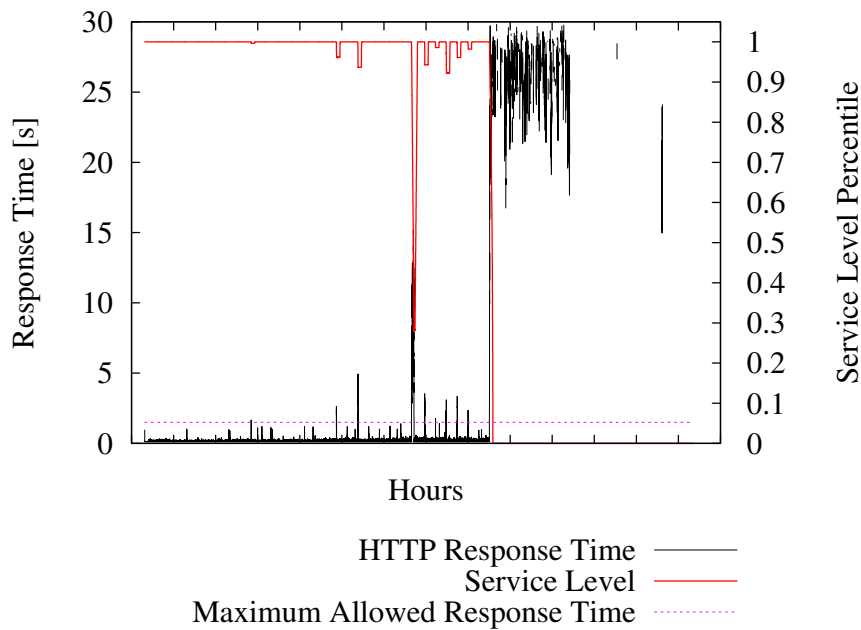


Figure 7.4: Influence of Live Migration on HTTP response time and service level during rising load for unpatched KVM with 1 GBit/s migration speed and strict downtime target.

7.5.4 Xenalike KVM

In a low level approach, we have patched KVM to implement Xen’s live migration stop conditions. For the patch, see Appendix C. The final iteration is forced when either

1. the VM’s memory address space was scanned 29 times for dirty pages, meaning 29 iterations were carried out,
2. more than three times of the VM’s allocated memory was transferred.

Then, independently from the amount of remaining dirty pages, the VM is paused and the final iteration starts. Subsequently, the VM is unpaused at the destination host. In between, the VM's services are of course unavailable, possibly affecting service levels. However, if a longer downtime prevents that a non-converging migration puts more and more resource pressure onto the VM, the impact on response times and service levels could be actually positive. In Figure 7.5, the result is shown for the patched version of KVM. We can clearly see how the service level is slowly and smoothly decreasing during rising load, as visible in Figure 7.10. Only at very high CPU utilization of almost 90%, the service level drops below 90%, meaning that more than 10% of the HTTP requests in the last 300 s could not be served within 1.5 s. Figure 7.8 shows how the downtime is increased with higher CPU utilization, avoiding massively decreased service levels.

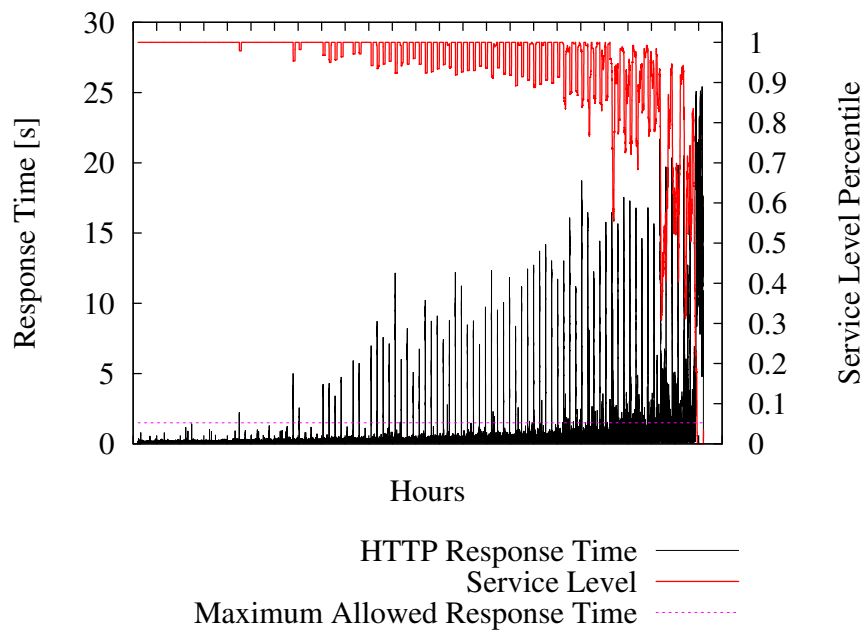


Figure 7.5: Influence of Live Migration on HTTP Response Time and service level during rising load for patched KVM implementing Xen-like stop conditions.

As a result, forcing the final iteration of a non-converging live migration is clearly beneficial for the service level, especially for higher workload intensity. Instead of clinging firmly to an unachievable low downtime, the downtime target is relaxed in favor of ending the resource pressure the migration puts on the VM as soon as possible.

7.5.5 Service Level Managed KVM

Although the patch implementing Xen's stop conditions in KVM is very small, we were finally interested in evaluating a high level approach. As mentioned above, Liu et al. [LXJ⁺11]

have presented a simple model able to predict the amount of time an iterative pre-copy live migration takes to complete, based on the page transfer and dirtying rates. We have first tested if the model is valid for our experimental setup, using the unpatched KVM version with 1 Gbit/s bandwidth and 30 ms downtime target, as it has no stop conditions and is able to migrate a high utilized VM with a high page dirtying rate, despite significant response time increase. A comparison of the measured migration delay and the model is visible in Figure 7.6, showing very small residuals. Hence, it is possible to predict the migration delay without any low level hypervisor modifications as used in [LXJ⁺11].

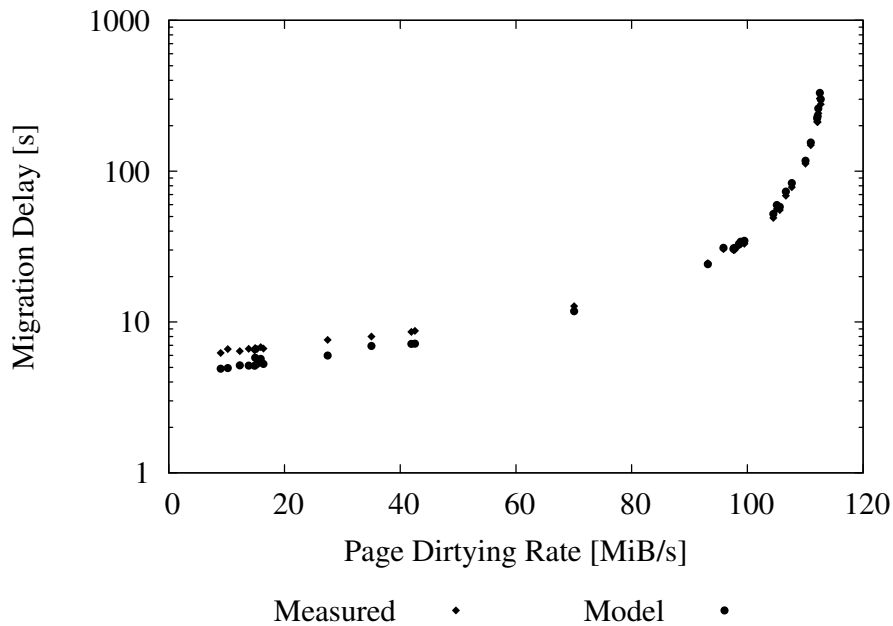


Figure 7.6: Influence of the rate of page dirtying on the time required to complete a live migration. Note the logarithmic scaling on the y-axis.

As we were already monitoring the progress of the live migration, we have extended our monitoring facility by a *service level management component*. Every second, we were not only monitoring the VM's response time and calculating page transfer and dirtying rates, but also predicting the remaining time the migration will take to complete. We have defined a *service level target* of 80% for the migration process, meaning that as a result of the migration, the service level is not allowed to decrease below 80%. As there are several possible events that could lead to a violation of this service level target and to be robust against measurement outliers, we have also defined an *action threshold* and a corresponding *counter*. If the counter reaches the threshold value, here 3, the migration is forced. As the monitoring facility is active every second, we regard a value of 3 as a trade-off between robustness and quick action. Events incrementing the counter are:

1. The dirty page rate is greater than the transfer rate, and the remaining time is not

computable. The migration will not converge, not even slowly.

2. If the current response time is greater than the allowed maximum, we assume it will not decrease during the rest of the migration, meaning that all requests during the remaining estimated migration time will violate the maximum allowed response time too. From that, we calculate the service level at the end of the migration, and if it is violating the service level target, the action counter is incremented. Here, we also estimate the time required for the final iteration based on the current amount of dirty pages and the transfer rate, and consider its negative impact on the service level, as connections will be stalled or even run into time-outs. Hence, we force the migration sooner, by considering the unavoidable final iteration and its service level impact.

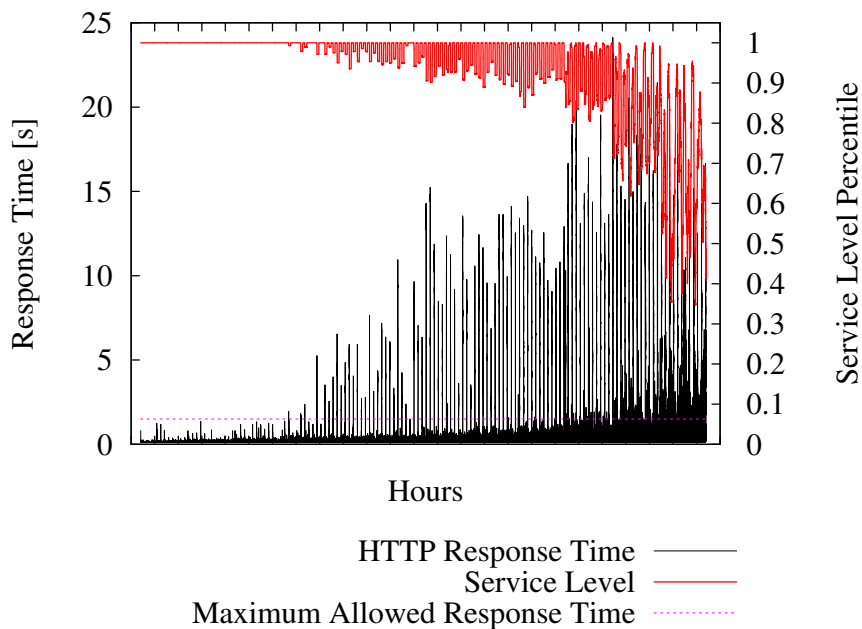


Figure 7.7: Influence of Live Migration on HTTP Response Time and service level during rising load for service level managed KVM.

Additionally, if the service level is already violating the target of 80%, we force the migration immediately, disregarding the action threshold. In Figure 7.7, the influence of the live migration on the HTTP response time and service level during rising load is shown when using the described service level management approach. Comparing the result to the patched version of KVM shown in Figure 7.5 yields the conclusion that both approaches are of comparable quality, although different in their implementation. A high level monitoring facility, although more complex, may be preferable over low level efforts such as adapting and maintaining source code, as it offers higher flexibility.

7.5.6 Comparison

In Figure 7.8, the development of the HTTP service downtime during the final pre-copy iteration is visible for all experimental scenarios described above. We can see how the unpatched, vanilla KVM achieves a downtime of 0.5 s. However, the system was not able to live migrate the virtual machine during phases of higher workload intensities. It is also clearly visible that the true downtime is multiple orders greater than the maximum allowed downtime setting of 30 ms. Here, a factor of up to 20 was observed, although we had configured very aggressive ARP announce timing settings for the virtual Ethernet bridge the VM connects to and the service downtime was measured with a minimum resolution of 10 ms.

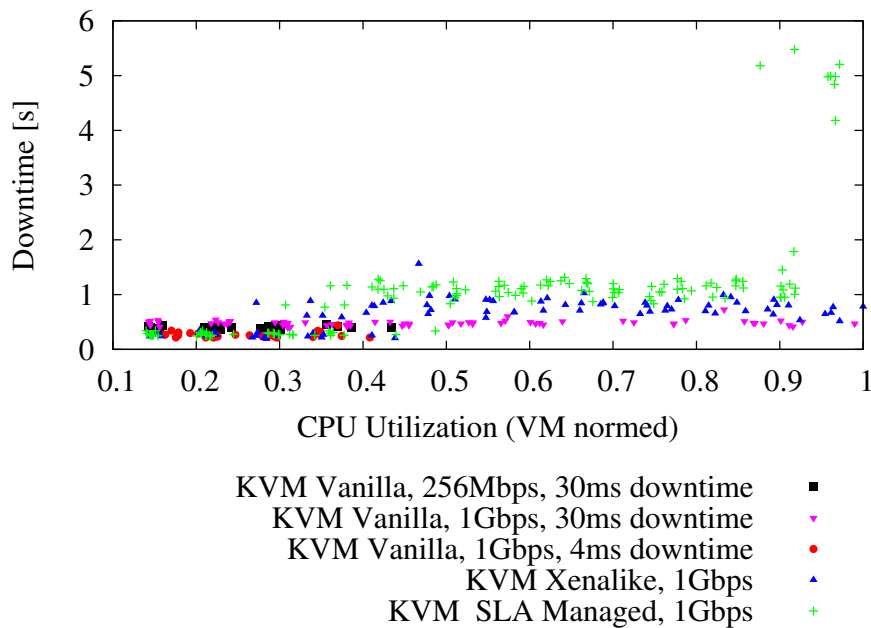


Figure 7.8: Comparison of the service downtime development with increasing CPU utilization.

Decreasing the maximum allowed downtime to 4 ms leads to actual downtimes of approximately 0.3 s, as visible in Figure 7.8. However, even with a migration speed of 1 Gbit/s, the system is unable to converge to a state where transferring the remaining dirty pages would take such a short time, even when the VM only has to bear medium workload intensity. Our patched version of KVM, denoted by Xenalike, is also set to 4 ms of maximum allowed downtime, again resulting in approximately 0.3 s true service downtime. In contrast to the experiments described above, the true downtime is increased from approximately 0.3 s to 0.6 s-1 s when the workload intensity, and consequently, the page dirtying rate, rises. Then, the stop conditions force the migration's final iteration, causing a slightly higher downtime, but significantly decreasing the migration delay and its negative impact on response times.

The SLA managed approach shows downtimes of up to 5 s for very high CPU utilization, as the service level is already too low and the migration is forced. A small benefit regarding the service level is visible in Figure 7.10.

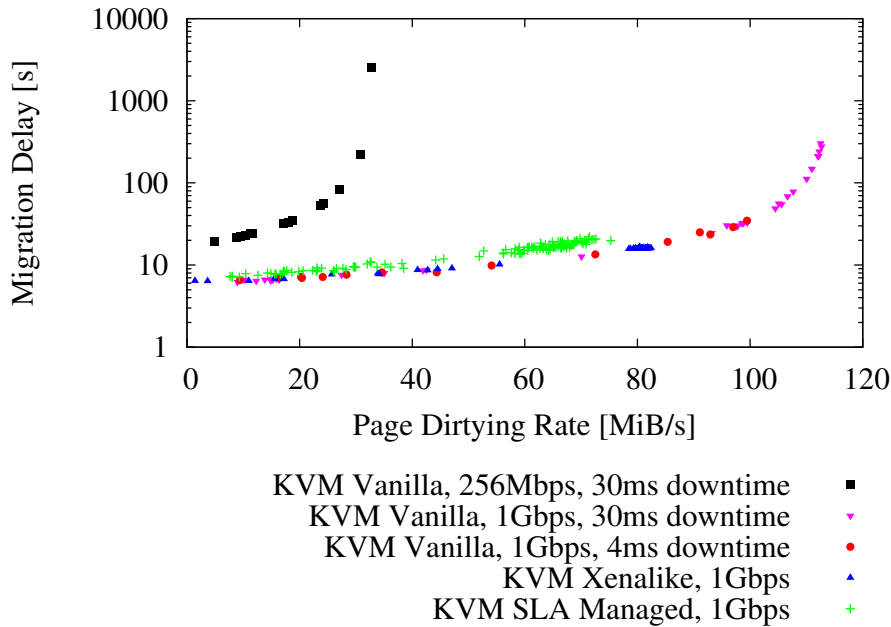


Figure 7.9: Influence of the rate of page dirtying on the time required to complete a live migration.

In Figure 7.9, the time a migration takes to complete, denoted as migration delay, depending on the page dirtying rate is shown for all experiment scenarios. Note the logarithmic scaling on the y-axis. We can see how the default, unpatched KVM is unable to migrate within acceptable time if the page dirtying rate comes close to 32 MiB/s, with catastrophic impact on the service level as mentioned above. With a migration bandwidth of 1 Gbit/s, the system's breaking point is deferred, but the development is similar. Page dirtying rates closer to 1 Gbit/s or 125 MiB/s would further increase the time required until the migration has converged to a minimum of dirty pages. The Xenalike stop conditions and the SLA managed approach both prevent the migration from reaching a state where convergence to the configured downtime targets would take too long and severely decrease service levels.

A plot showing a direct comparison of the service level during live migration, depending on the workload intensity, here visualized by the VM's CPU utilization, is depicted in Figure 7.10. There, our patched version of KVM is showing a slow and linear decrease of service levels while the CPU utilization rises until a level of approximately 75%. Subsequently, the service level decreases non-linear, but the system is always able to finish the migration. The service level managed KVM approach achieves similar service levels, for very high CPU utilization above 90%, sometimes even better than the patched KVM approach, due to the fact that the service level is already too low and the migration is forced

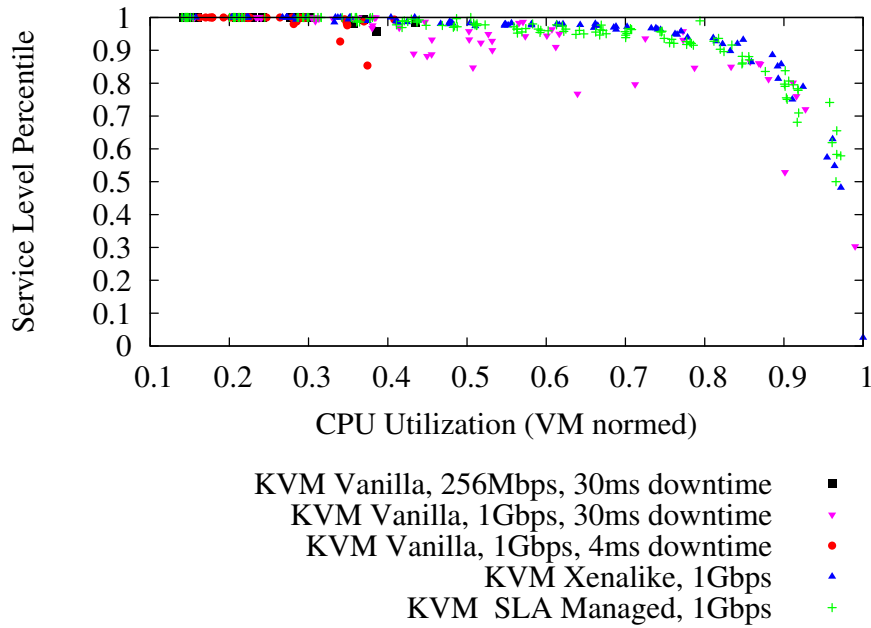


Figure 7.10: Comparison of the service level's development with increasing CPU utilization.

without further monitoring. This is also visible in Figure 7.8, where this implementation shows downtimes of 5 s for CPU utilization higher than 90%. All other approaches evaluated show significantly decreased service levels, suddenly beginning at a CPU utilization of approximately 35%. If the migration bandwidth or the downtime target are too low, unpatched KVM is even unable to migrate the VM without thrashing its service.

If we compare Figure 7.10 with Figure 7.8, we can clearly see that the overall performance of our patched KVM and service level managed KVM approaches are significantly the best. Instead of clinging firmly to an unachievable low downtime target, resulting in disproportionally long migration delays and catastrophic service levels, the final pre-copy iteration is forced if the system does not converge within a reasonable amount of time and effort. As a consequence, the service downtime, during which existing connections and connection establishments are stalling, increases. Nevertheless, the effects on quality of service are significantly less severe, as shown in Figure 7.5 and Figure 7.7, where it is clearly visible that these approaches are able to achieve acceptable service levels for higher workload intensities than the existing implementations.

7.6 Discussion

In this chapter, we have evaluated the influence of live migration on service levels for a web server scenario, using the KVM hypervisor. We have shown that the default implementation

and settings are clearly giving room for improvement, as migration leads to catastrophic service levels. Increasing the migration bandwidth just defers the system's breaking point, but offers no general solution. Therefore, we have implemented a low level solution by patching KVM to implement Xen's stop conditions, and a high level solution by using a model able to estimate the remaining migration time and the corresponding impact on the service level due to the resource pressure caused by the dirty page scanning and copying during the migration. Due to an iterative migration process that is not converging, trying to reach unachievable low downtimes causes unacceptably low service levels, as the migration process itself is increasing the resource pressure onto the VM for a too long period of time.

Both of our approaches result in significantly better service levels and prevent the VM from thrashing and complete service disruption, achieving acceptable service levels during live migration also for higher workload intensities by trading off migration delay and migration downtime. Hence, they enable service providers to migrate also during phases of high workload intensity without causing disproportional high service level degradation. As a result, we have shown that iterative pre-copy live migration should not be implemented without either stop conditions or a management facility taking pro-active evasive action for infrastructures with QoS targets.

7.7 Future Work

For future work, we are especially interested in adapting our experiments regarding web service QoS to implementations of post-copy live migration. Here, one of several challenges will be the mechanism pushing memory pages from the source to the destination host. In addition to on-demand pushing and a background transfer, the quality of detecting the VM's *working set* will be important to minimize service level degradation, as it allows to prioritize the transfer of those pages which will be demanded by the VM's applications soon. However, this would require an observation phase before the actual post-copy live migration starts.

Chapter 8

Security and Privacy of Energy Efficient Data Centers

8.1 Overview of Chapter

Virtualized data centers where several virtual machines (VMs) are hosted per server have become state of the art in data centers. As a consequence of energy efficiency concerns, the exact combination of VMs running on a specific server will most likely change over time. Using virtualization often comes with the illusion of perfect isolation and therefore high privacy and security. There are many examples that such claims are in fact mostly illusory.

In this chapter, we present experimental results how to use the energy/power consumption logs of a power monitored server as a side-channel that allows us to recognize the exact combination of VMs it currently hosts to a high degree. For classification, we use a maximum log-likelihood approach, which works well for comparably small training and test set sizes. We also show to which degree a specific VM can be recognized, regardless of other VMs currently running on the same server, and show false negative/positive rates.

To cross-validate our results, we have used a Kolmogorov-Smirnov test, resulting in comparable quality of recognition within shorter time. In order to clarify whether our approach is generalizable and yields reproducible results, we have set up a second experimental infrastructure in Lyon, using a different hardware platform and power measurement device. We have obtained similar results and have experimented with different CPU frequency scaling governors, yielding comparable quality of recognition.

As a result, energy consumption data of servers must be protected carefully, as it is potentially valuable information for an attacker trying to track down a VM to mount further attack steps.

The Chapter is structured as follows: in Section 8.2, we compare our work to related efforts and give examples for possible combination of approaches in the future. In Section 8.4, we describe our experimental platforms and methodology. For reasons of clarity, there are subsections for each of the two platforms we have carried out our experiments. In Section 8.5, we explain the methodology of the different evaluations we have conducted. In Section 8.6, we show and interpret the results we have obtained in our experiments. In order

to be clear, Section 8.6 is organized in subsections to represent the respective experimental platforms. Finally, we discuss the results and draw conclusions in Section 8.7.

8.2 Introduction and Related Work

Virtualized data centers where several virtual machines (VMs) are hosted per server are state of the art. They are expected to become even more popular in the future with the new Cloud Computing paradigm emerging. At the same time, an increasing number of new vulnerabilities are discovered, bugs that could not be exploited in the past are suddenly exploitable, and whole classes of attacks (e.g., timing attacks, power consumption attacks, etc.) find new scenarios of application [CPK10].

As a consequence of energy efficiency or load balancing concerns within Cloud Computing infrastructures, the exact combination of VMs running on a specific server will most likely change over time by using live migration [CFH⁺05, HLM⁺09]. Using virtualization often comes with the illusion of perfect isolation and therefore high privacy and security. There are many examples that such claims are in fact mostly illusory.

Garfinkel et al. [GR05] discuss the arising security challenges of virtualized computing environments in general and propose to move the security tasks from the VMs to a highly secure hypervisor. The main problem is that virtualization adds a layer of complexity to computing environments, and the more complex a setup is, the harder it is to secure it. The addition of the hypervisor layer increases the range of possible attacks compared to non-virtualized environments [JG11].

Especially highly dynamic infrastructures, where creating a (virtual) machine is almost as simple as copying a file, are hard to keep safe with static security architectures. E.g., it might be hard for intrusion detection systems to notice malicious network anomalies if there is no stable and trainable network state due to the flexibility virtualization provides.

In contrast to direct attacks aiming at the software stack (e.g., the virtualization layer), this chapter presents an indirect attack, in this case using energy consumption logs. Such data is likely to exist in Cloud Computing infrastructures to be able to monitor the infrastructure state and to compute energy efficient workload mappings. In the future, it could be also used for accounting, as energy becomes a precious and expensive resource.

Kortchinsky [Kor09] attacks the virtualization layer and describes how a vulnerability in a graphic driver makes it possible to break out of a VMware guest and become root at the hypervisor level. The exploit only works on virtualized systems and in way that may become familiar in the future: The bug allows to access memory beyond the graphic memory boundary. In a non-virtualized system, this is clearly a bug, but most probable no vulnerability as physically non-existent memory can not be read or written. In a virtualized system, graphic memory is just a region of main memory, so access beyond the graphic memory boundary becomes meaningful and dangerous. Such a kind of attack is a possible step after a malicious user has found the server a specific VM is hosted on.

Cloud environments should guarantee the security of both user data and application pro-

cessing [SGR09]. As physical resources are shared between the guest virtual machines, flaws in the isolation of VMs lead attackers to have access to data and applications belonging to other users. This kind of threat is based on a two-step attack: targeted VM localization and then the attack itself.

The localization phase has been proved to be feasible by Ristenpart et al. [RTSS09]. They carry out an attack on an infrastructure level. They are able to get enough information about the internals of Amazon EC2 to deliberately place a VM on the same host as a target VM. Additionally, they discuss how this ability can be used to start cross-VM side-channel attacks. The efforts described in this chapter are complementary in a way that the approaches may be combined to a faster or more accurate attack.

Once the targeted VM is localized, attacks relying on flaws and vulnerabilities are possible. For example, Ormandy [Orm08] found a bug in the hypervisor code concerning the emulated I/O ports that allow a malicious user to write to an arbitrary out-of-bounds memory location. Another vulnerability in the guest virtual device driver allowed a guest VM to crash the host computer and the VMs hosted by it¹. Numerous similar vulnerabilities have been found [JG11, Orm08].

Xensploit [Jon08] is an implementation of a man-in-the-middle attack on live migration, targeting Xen and VMware. As the memory pages are transferred without any encryption, Xensploit is able to change the object code of a running `ssh` daemon to allow access without any authentication. Such attacks could be prevented by encrypting the memory pages network transfer, although this would cause computational overhead on the nodes involved in the live migration. However, changing memory contents of a VM during live migration is not the only conceivable attack scenario.

Oberheide et al. [Jon08] differentiate possible attacks into the categories data plane, control plane and migration module and give example attacks. E.g., a malicious hypervisor node could make false claims about its available resources, energy efficiency etc. An algorithm trying to balance or consolidate workload would then be triggering the live migration of a large number of VMs to this malicious node which in turn gains privileged access to the VMs.

8.3 Open Issues and Own Contribution

In this chapter, we investigate the potential of extracting valuable information from raw energy consumption logs with impact on user privacy and security. We present experimental results, showing that it is indeed possible to track down the server hosting a VM in question, by using the servers power footprint. We regard our work as an additional, indirect attack vector, to gain more information about a most probably complex computing infrastructure, usable in a preparation stage before launching actual attack steps. E.g., even if the VM an attacker is aiming at is highly secured with a packet filter, intrusion detection system

¹<http://www.vmware.com/security/advisories/VMSA-2009-0006.html>

etc., all of such protection measures are defending the VM against attacks from the network or Internet uplink. Nevertheless, the VM is still completely dependent on the hypervisor's integrity and users have to put trust into the hypervisor not being malicious or compromised. Technologies like vTPM [PSvD06] are still in its early days, and just encrypting the VM's file systems will most probably not be a reliable remedy. The data have to be decrypted at some point in time for computational tasks. Unless the VM is exclusively used as a storage server for encrypted data, the decrypted data will reside at least temporarily in the server's main memory, accessible by the hypervisor.

As a consequence, if the VM in question is not an adequate or convenient target for direct attacks because of whatever reason, there is still the hypervisor. Once compromised, it would allow privileged and yet, for the VM's owner, hardly detectable access to the VM. The most dangerous attackers are those trying to leave no traces at all and stay undetected, so if attacking a VM would cause too much attention (e.g., because of an IDS) and the hypervisors in a Cloud infrastructure are insufficiently secured or have known vulnerabilities, they will be a convenient prey. Nevertheless, the fact that there will be at least dozens of hypervisors and only one of them is hosting the VM in question is still an open issue. The more hypervisors are required to be cracked to gain access to the VM, the more likely it is that the attack is detected even before fully carried out. However, if the attacker can retrieve power consumption data, e.g., because of an insufficiently secured SNMP access to a metered rack PDU, it could be possible to extract information about which server the VM is running on. The attacker would then concentrate on this single machine to cause only a minimum of attention.

This chapter presents an approach where attackers can use monitoring data such as power consumption of physical resources to locate a given VM. To the best of the author's knowledge, no research work has dealt with the privacy of power monitoring data in Cloud environments yet. However, this issue has been studied in the research fields concerning smart grid [SLP11] and home metering [KFB11]. Customers are indeed aware of what frequent data collected by home smart meters can reveal about their appliance usage.

8.4 Experimental Details

In the following, we describe our experimental setups and methodology of measurements. In a first step, we have examined the feasibility of our approach in Vienna. In order to clarify whether our approach is generalizable and yields reproducible results, we have set up a second experimental infrastructure in Lyon, using a different hardware platform and power measurement device. In both of our experimental infrastructures, we have used four different `qemu-kvm/Linux` VMs in our experiments:

- A MySQL database VM, denoted by `sql`, hosting a dump of the Latin Wikipedia for the following `wp` VM
- an Apache web server VM with PHP5 and MediaWiki, denoted by `wp`

- an FTP server VM, denoted by `ftp`
- and a VM doing I/O operations using `bonnie++`, denoted by `bonnie`.

E.g., the string `sql_ftp_bonnie` denotes that all VMs except the Apache VM were running. The string `wp` denotes that only the Apache VM was running. During the training phase, we have measured the energy consumption of a server for all 15 possible combinations (one combination of four VMs, four combinations of three VMs, six combinations of two VMs, and four combinations of a single VM running) of VMs running while a client sent a fixed trace of workload. The traces for `wp` and `ftp` were changing in concurrency of client requests, `sql` was acting as a database tier for `wp`. For `bonnie`, the number of benchmark runs within an interval was changed to simulate a varying I/O load.

8.4.1 Experimental Infrastructure in Vienna

In our setup in Vienna, we have run the VMs on an AMD Phenom II X4 955 (3.2 GHz) CPU with 8 GB RAM. The VM disk images were hosted on an additional node with two SSDs in software-RAID0, connected over Gbit Ethernet.

To measure the power consumption, we have used a Hioki 3334 wattmeter device that allows several thousands measurements per second and integrates them to deliver active power consumption (W) respectively energy consumption (Wh) results. Based on the energy consumption, we have calculated the average power consumption within an interval of 6 s.

Each combination was traced for 40 minutes of workload, so we have approximately 400 data samples for each combination. During the whole experiment, the `ondemand` governor of the Linux CPU frequency system (for AMD platforms, `powernow_k8` driver) was activated to scale the frequency of all cores to the actual demand. The PhenomII X4 CPU is able to scale the frequency on a per-core level, so all of its four cores can be set to a different frequency. Available frequencies are 800 MHz, 2.1 GHz, 2.5 GHz and of course 3.2 GHz.

8.4.2 Experimental Infrastructure in Lyon

In order to clarify whether our approach is generalizable and yields reproducible results, we have set up a second experimental infrastructure in Lyon. The hardware platform was comprised of six identical servers with Dual-Socket Quadcore Intel Xeon E5506 (2.13 GHz) and 12 GB RAM. On one of these servers, the VM disk images were hosted with five SSDs in hardware-RAID0, connected over Gbit Ethernet. The remaining five servers, all of them were identical, were used to run the VM combinations in parallel to speed up measurements.

To measure the power consumption, we have used a OmegaWatt multi-channel energy consumption logger which was able to measure all of the servers simultaneously and independently. The OmegaWatt device has no ability to integrate values, so its precision is not comparable to the Hioki 3334 used in Vienna. On the other hand, devices similar to

the OmegaWatt logger are much more likely to be found in data centers than high precision devices intended for lab use, so we consider this as a more realistic attack ground. We have polled the OmegaWatt device every second, so for approximately 40 minutes of workload, we have obtained approximately 2400 power consumption samples.

In contrast to the PhenomII X4 CPU that was used in Vienna, the Xeon E5506 CPUs are not able to scale frequencies on a per-core level. Here, all cores of a CPU (socket) are set to the same frequency by the driver (`acpi_cpufreq`). E.g., if the server is only slightly loaded and only one core is required to run at a higher frequency than the minimum, all of the socket's CPU cores (here, four) will be automatically set to the same frequency by the `ondemand` governor.

We have carried out our experiments with three different scaling governors:

1. `ondemand`, to be able to compare the results to those obtained in Vienna,
2. `performance`, where all cores are fixed to the default frequency and scaling is in effect deactivated, and
3. `powersave`, where all cores are fixed to the minimum frequency.

With the `powersave` and `performance` governors, the range of possible power consumption should be smaller, which could make a difference for our evaluations. The available frequencies when using a Xeon E5506 CPU with the `ondemand` governor are: 2128 MHz, 1995 MHz, 1862 MHz, 1729 MHz and 1596 MHz. Additional to the lack of per-core scaling support, the Xeon CPU has a much smaller scaling range ($\frac{f_{min}}{f_{max}} = 0.75$) compared to the PhenomII ($\frac{f_{min}}{f_{max}} = 0.25$). The total power consumption of a CPU is the sum of dynamic and static power [KAB⁺03], and typically denoted by the approximation

$$P = ACV^2f + VI_{leak},$$

where A is the fraction of gates actively switching and C the capacitance of all gates. A smaller range of frequency will therefore possibly lead to smaller range of power consumption. As there are other metrics influencing the power consumption (especially V^2 is contributing heavily) and the CPU is not the only device causing power consumption, we were interested if a different CPU frequency scaling governor makes a significant difference in our scenario.

8.5 Evaluation

For evaluation, all consumption traces were split into training and test sample sets, for different test sample set sizes ranging from 5 to 50 in steps of 5. To minimize the effect of outlier results, the training/test sample set split was randomized and repeated 1000 times for each test sample set size. For each of these runs, we had to compute the probability density

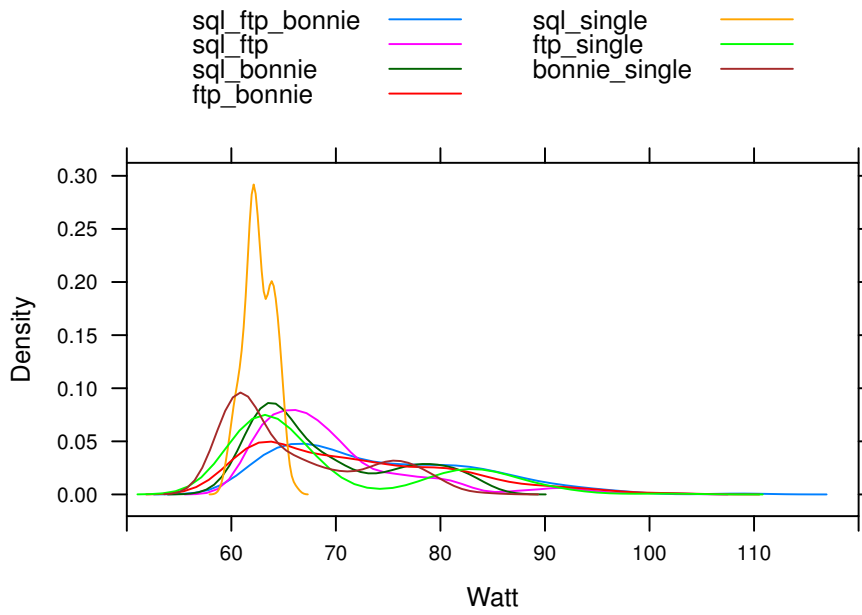


Figure 8.1: Vienna: Probability density of power consumption, combinations **not** containing the Apache VM.

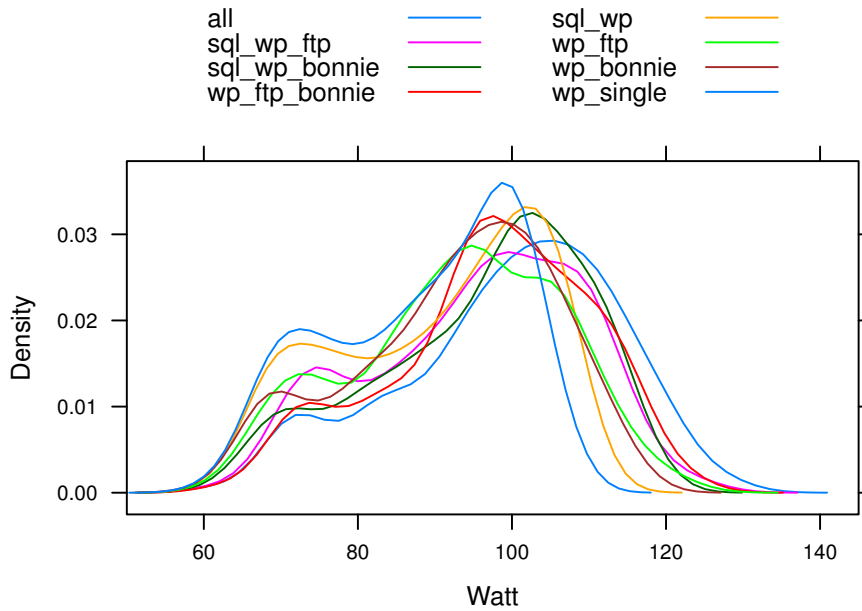


Figure 8.2: Vienna: Probability density of power consumption, combinations **containing** the Apache VM.

functions from scratch for each VM combination. To recognize the origin of a test sample set and therefore the combination of VMs, we used a maximum log-likelihood approach. For a given test sample set, we computed the log-likelihood for each possible origin to identify the most probable origin. To consider different perspectives, we have evaluated four different match conditions:

1. **Exact match:** The recognized combination is exactly matching the true combination, so the state of all VMs was correctly recognized.
2. **Subset match:** The true combination is subset of the recognized combination, so all running VMs were correctly recognized, but there are false positives, meaning at least one VM was falsely recognized as running.
3. **Set match:** This evaluation mode gives the degree of correctness of the classification. As we have four VMs and each of them can be running or not, we can interpret both the true and the recognized state as a quadruplet of $\{yes, no\}$. The degree of correctness is then the intersection of both quadruplets and possible values range from (0 of 4) to (4 of 4). For plotting purposes, we have scaled the range to $[0;1]$.
Based on the set match evaluation, we have calculated the false positive/negative recognition rates. A false positive in this context is that a VM was recognized running but is in fact not, and a false negative that a VM that was in fact running was not recognized so.
4. **VM match:** The last evaluation mode represents the perspective of an attacker trying to track down a specific VM. In such a case, false positives/negatives are not important as long as the true state of a specific VM is recognized correctly to identify the server a VM is running on quickly. Once this server is found, further attack steps could be mounted.

8.6 Results

8.6.1 Results of Experiments in Vienna

Figure 8.1 and Figure 8.2 show the power consumption density for each VM combination. Two plots are used for better visibility. We can see in Figure 8.2 that the power consumption of all combinations containing the *Apache VM (wp)* is very similar, and that they differ heavily from all combinations that do **not** contain the *Apache VM*, which are depicted in Figure 8.1.

Maximum Log-Likelihood

We can see in Figure 8.3 that for comparably small training and test set sizes, a lot of exact combinations can be recognized correctly to a high degree using the described log-

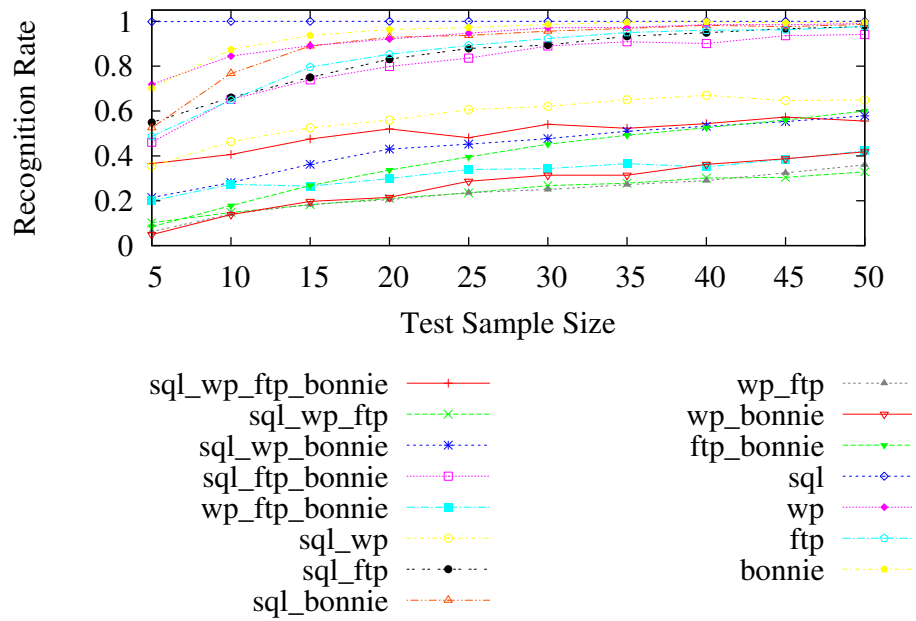


Figure 8.3: Vienna, Exact Match Condition: Recognition Rates are very high for combinations **not** containing wp.

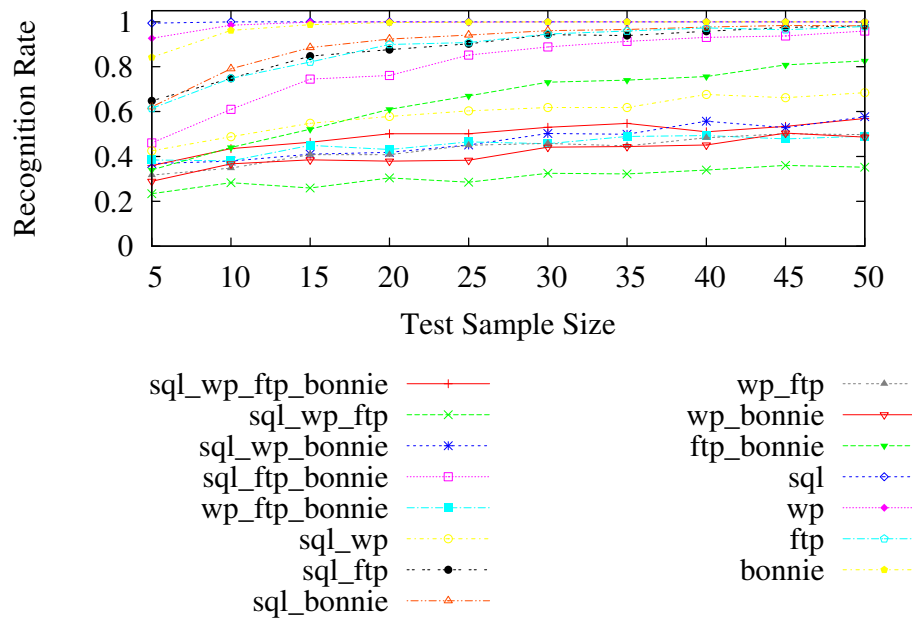


Figure 8.4: Vienna, Subset Match Condition: Recognition Rates are generally increased.

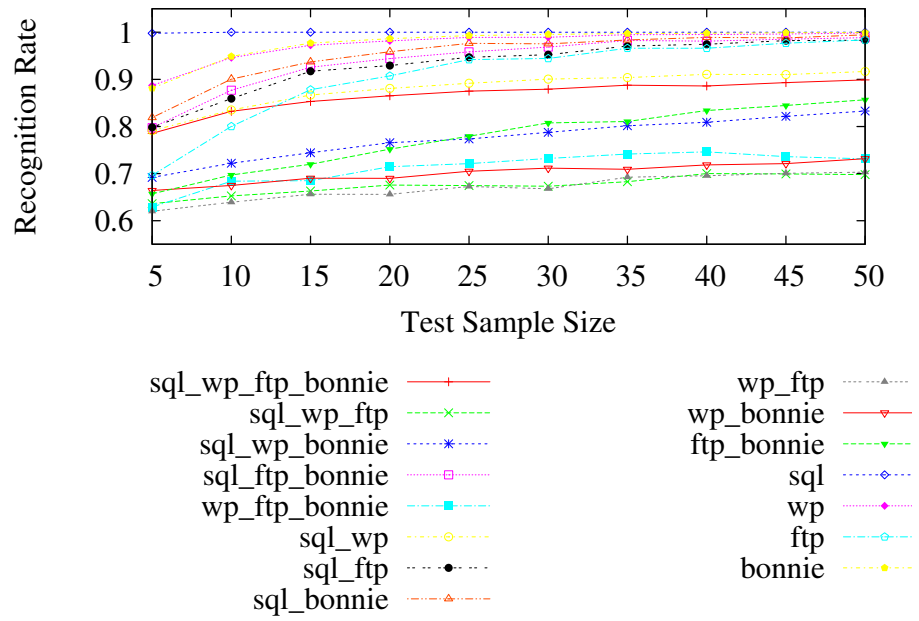


Figure 8.5: Vienna, Set Match Condition: Average degree of combination recognition.

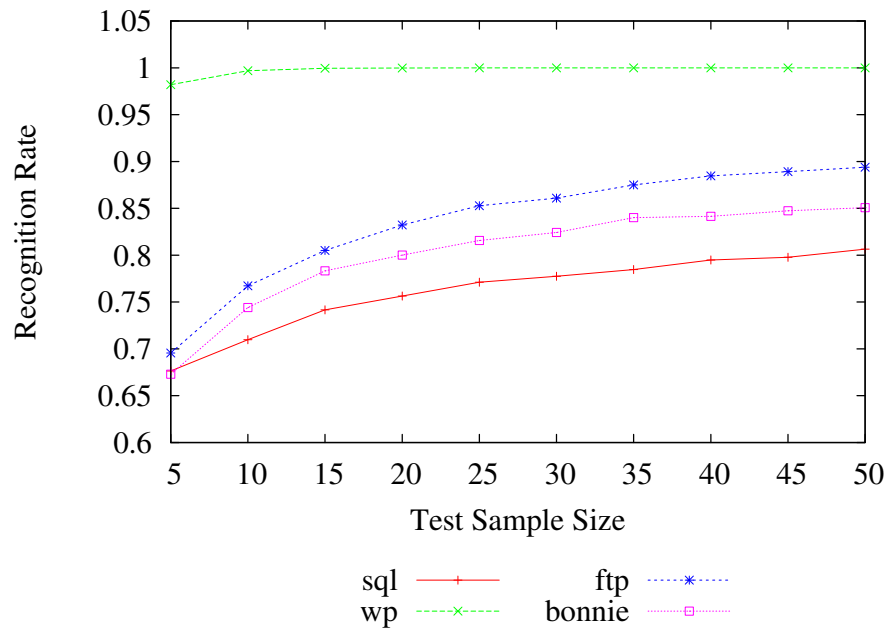


Figure 8.6: Vienna, Per VM Recognition Rates: wp is extremely well recognizable.

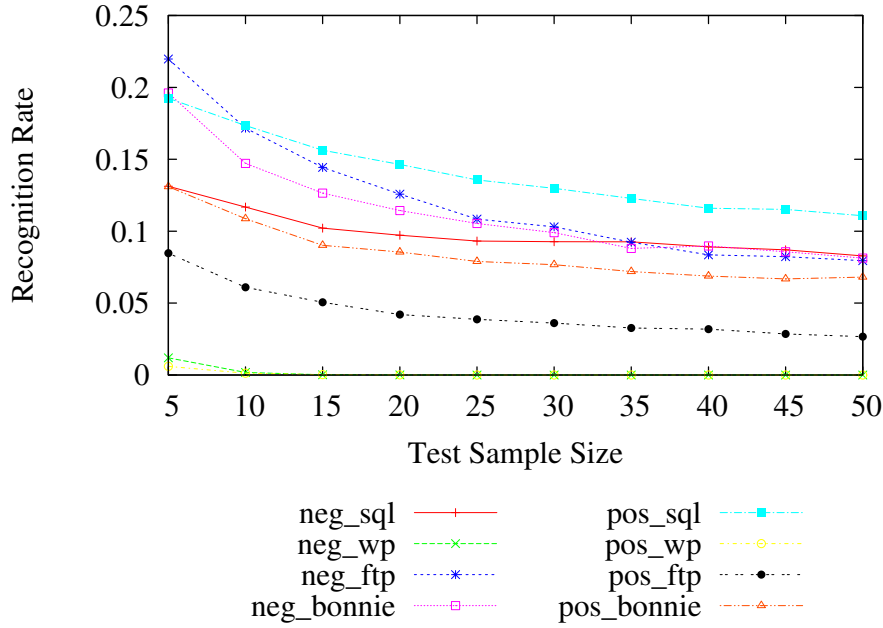


Figure 8.7: Vienna, False Positive/Negative Recognition Rates.

likelihood approach. Most of the combinations yielding lower values of recognition rates have a running wp VM (Apache, PHP) in common.

As visible in Figure 8.2, all combinations containing wp

1. have a very similar power consumption trace and
2. have a higher average power consumption than all combinations **not** containing wp.

The consequences of these facts are that

1. it is hard to recognize which VMs are running in combination with wp, but
2. it is easy to recognize the state of wp correctly, which is clearly visible in Figure 8.6.

In Figure 8.4, we can see that if we relax the match condition from an exact match to a subset match, the match rate is increased, as false positives do not decrease the recognition rate, but only false negatives. For, e.g., the combinations ftp_bonnie and wp ftp, the increase is clearly visible.

In Figure 8.5, the average degree of classification correctness is shown. Please note that the y-axis is scaled differently for better visibility. We see even higher recognition rates and that on average, the combinations with wp are classified falsely by a single VM that is not recognized correctly (3 of 4 correct).

Figure 8.6 shows the degree to which our approach is able to recognize the state of a specific VM correctly, regardless of the combination of VMs it is running with. This is

perhaps the most interesting evaluation mode considering our scenario of privacy risks in a Cloud Computing infrastructure. Even for small numbers of test sample sizes, we can recognize whether a VM is running or not to a high degree.

In Figure 8.7, false positive and negative rates are visible and show how the respective gaps to a recognition rate of 1 in Figure 8.6 are composed by false positives and negatives. E.g., we can see both high false positive and negative rates for `sql`, which is expectable due to the fact that `sql` has a very narrow density peak in a low region of power consumption. As a consequence, the weak `sql` signal will be hidden easily by additional, stronger signals, and it is hard to recognize the presence or absence of `sql` correctly. On the other hand, `wp`, the strongest of all signals, yields almost no false positives/negatives at all, but as mentioned above, the strong `wp` signal is hiding the weaker ones and makes it hard to recognize other VMs running in combination with `wp`.

Kolmogorov-Smirnov Test

Finally, we wanted to verify the described maximum log-likelihood approach by using a Kolmogorov-Smirnov test for classification. Figure 8.11 shows the results for recognizing the state of a specific VM correctly, regardless of the combination of VMs it is running with. If we compare the results to Figure 8.6, we can see that the Kolmogorov-Smirnov test yields recognition rates approximately 5-10% lower than the log-likelihood approach. If we take into account that the log-likelihood approach requires frequent generation of density functions, which is a computationally highly intensive task and can easily take dozens of minutes for a fixed test sample set size and a sufficient number of repetitions, the quality of results obtained by using a much faster Kolmogorov-Smirnov test for classification could be acceptable too. Especially, if VMs are frequently live-migrated to other servers due to, e.g., load balancing or energy efficiency concerns, obtained results could be obsolete if the computation involved takes too long.

8.6.2 Results of Experiments in Lyon

In order to clarify whether our approach is generalizable and yields reproducible results, we have repeated and extended our experiments on a second infrastructure in Lyon, using a different hardware platform and power measurement device described in Section 8.4.

In Figure 8.16, we can see the VM recognition rate when using the `ondemand` governor. The result should be compared to Figure 8.6, where the `ondemand` governor was used too, but on a different hardware platform etc., as described above. We can see that `wp` is achieving similarly high recognition rates. Interestingly, `ftp` is yielding slightly worse rates, but `sql` and `bonnie` significantly higher rates. If we compare the false positive and negative rates (Figure 8.7 and Figure 8.15), we can see that

- `ftp` has similar false negative rates, but much higher false positive rates,
- `sql` has similar false negative rates, but much lower false positive rates, and

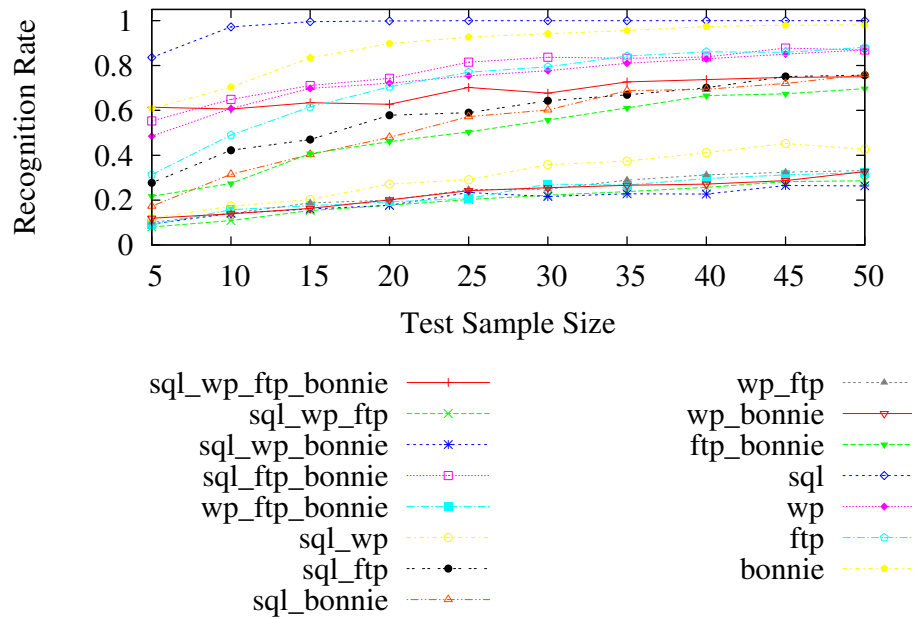


Figure 8.8: Vienna, Exact Match Condition using a Kolmogorov-Smirnov test: Recognition Rates are very high for combinations **not** containing wp.

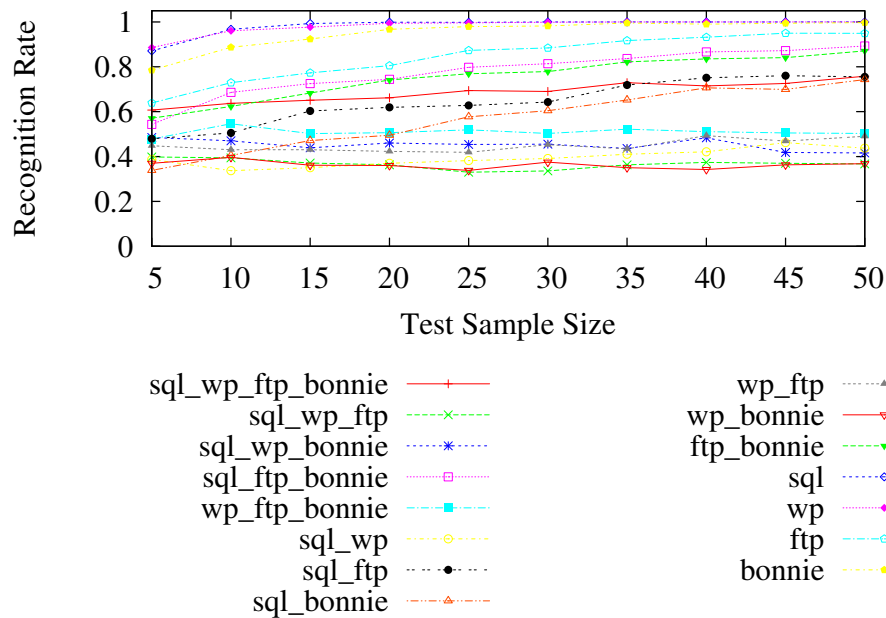


Figure 8.9: Vienna, Subset Match Condition using a Kolmogorov-Smirnov test: Recognition Rates are generally increased.

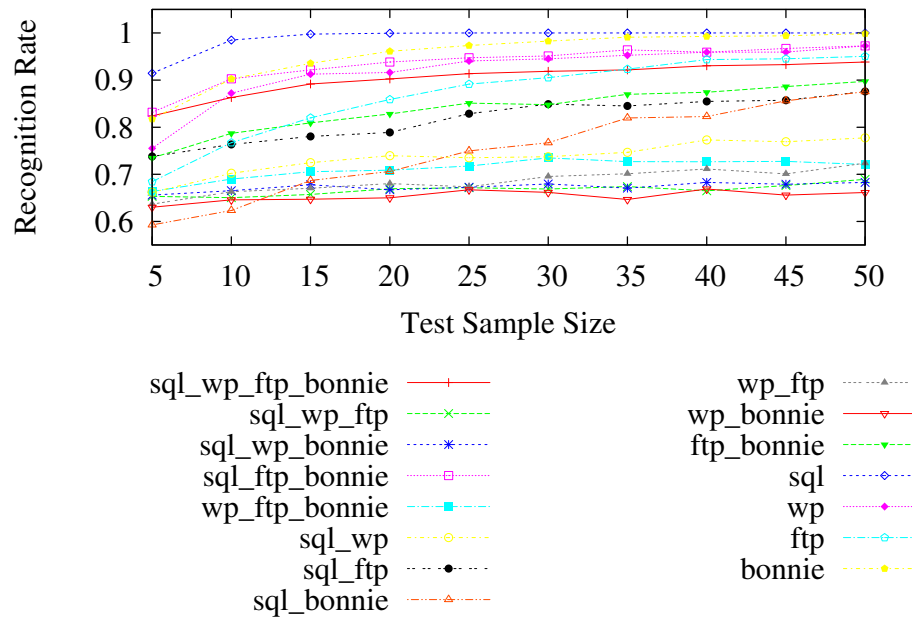


Figure 8.10: Vienna, Set Match Condition using a Kolmogorov-Smirnov test: Average degree of combination recognition.

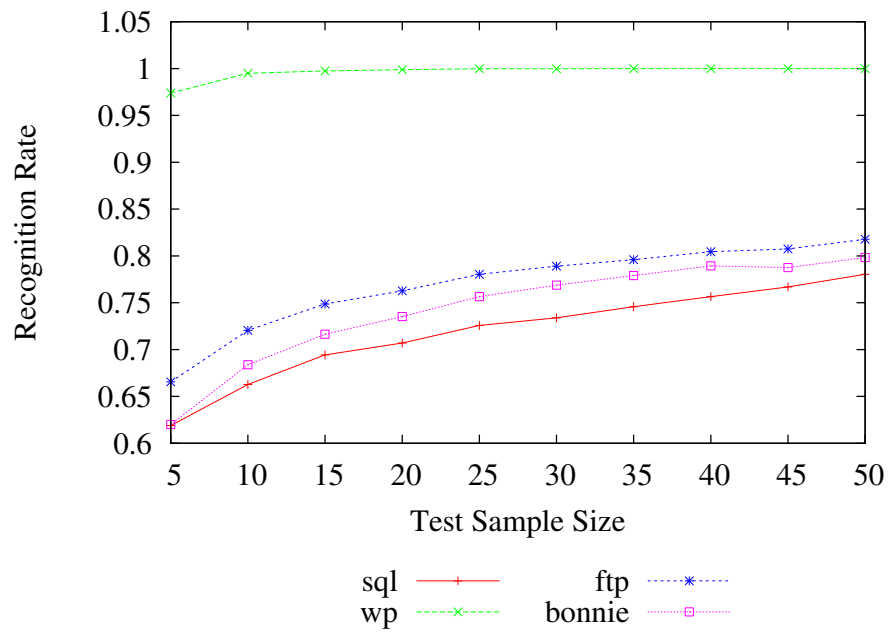


Figure 8.11: Vienna, Per VM Recognition Rates using a Kolmogorov-Smirnov test for validation purposes, results are of comparable quality to Figure 8.6.

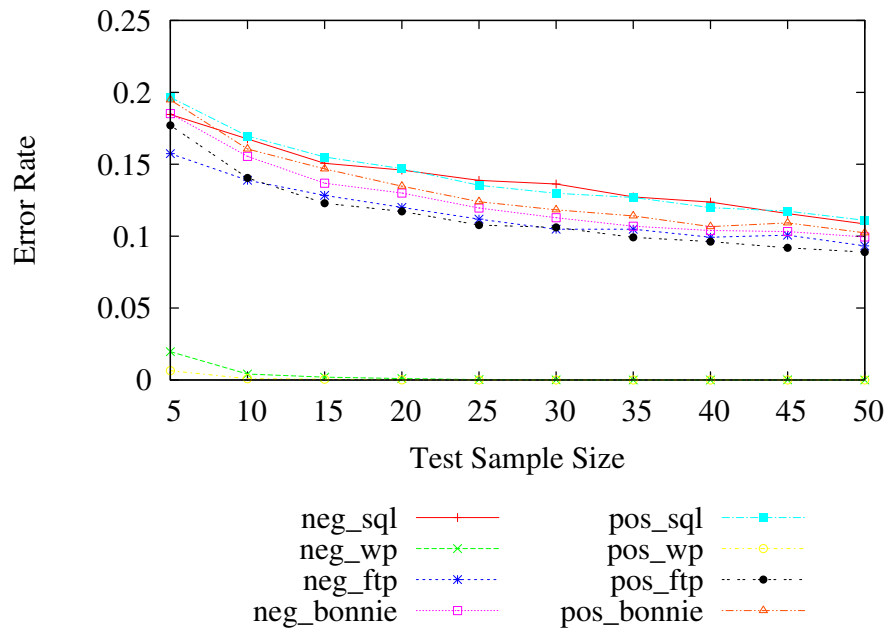


Figure 8.12: Vienna, using a Kolmogorov-Smirnov test: False Positive/Negative Recognition Rates.

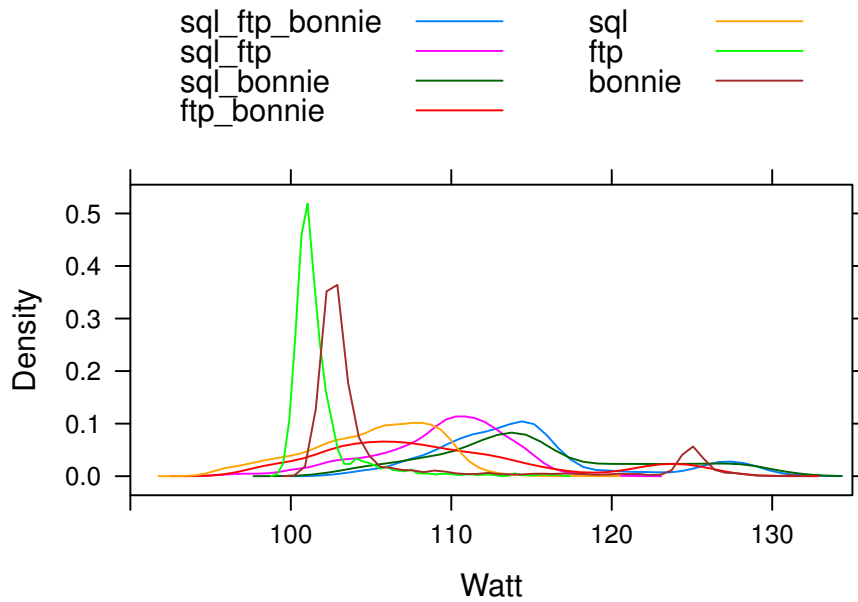


Figure 8.13: Lyon, ondemand governor: Probability density of power consumption, combinations **not** containing the Apache VM.

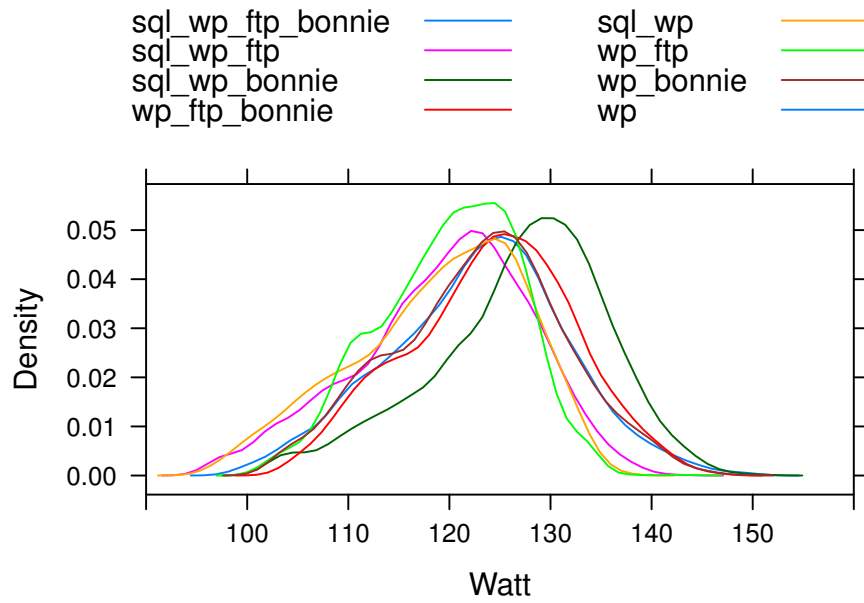


Figure 8.14: Lyon, ondemand governor: Probability density of power consumption, combinations **containing** the Apache VM.

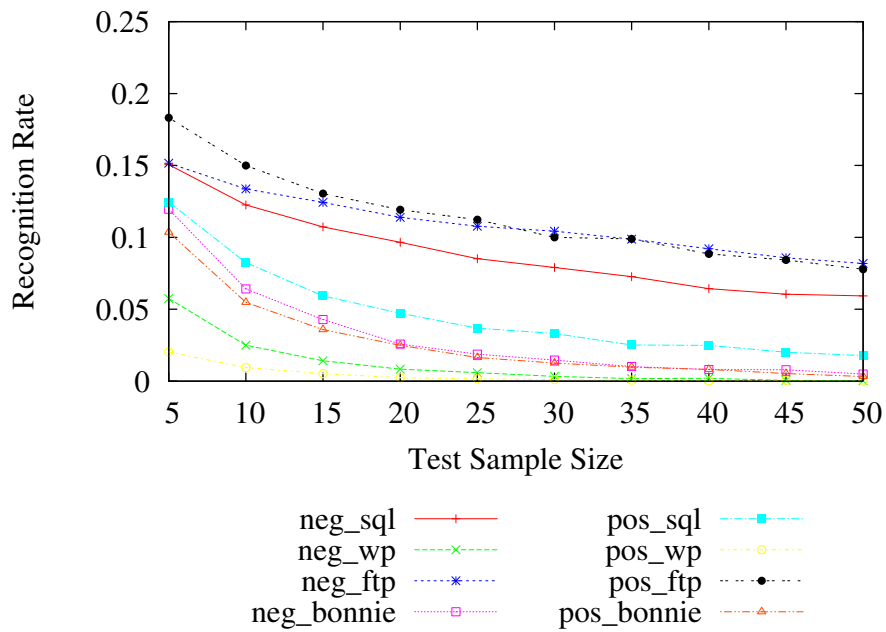


Figure 8.15: Lyon, ondemand governor: False Positive/Negative Recognition Rates.

- `bonnie` has much lower both false negative and positive rates.

Looking at the power consumption probability densities when these VMs run alone (cf. Figure 8.13 and Figure 8.1) reveals that `ftp` is now the VM with the lowest and most narrow power footprint, and therefore the weakest signal, easily hidden by stronger ones if running in combination. The much higher false positive rate of `ftp` is a direct consequence. Vice versa, `sql` is not the weakest signal anymore, resulting in lower false positive rates. Finally, `bonnie` has a much more characteristic probability density when running alone, with two unique peaks. In consequence, all combinations containing `bonnie` are highly distinctive from all combinations not containing `bonnie`.

After we had empirical proof that our approach is generally working on a different hardware platforms too, we have investigated the potential impact of different CPU frequency scaling governors. In Figure 8.18, which should only be compared to Figure 8.16 and Figure 8.17, the results for the `performance` governor, where the frequency is fixed to the default frequency and scaling is in effect deactivated, are visible. The `sql` VM is yielding slightly worse rates compared to Figure 8.16, but despite that, no significant difference can be extracted.

Finally, in Figure 8.17, the results for a `powersave` governor scenario, where the frequency is fixed to the minimum frequency, are shown and are not significantly differing from the `performance` scenario.

As a consequence, we can conclude that using the mentioned CPU frequency scaling governors the Linux kernel provides does not significantly influence the VM recognition rates and our approach is generalizable, including infrastructures using CPU frequency scaling due to energy efficiency concerns.

8.7 Discussion

We have presented experimental research on how to use the energy/power consumption logs of a power monitored server as a side-channel that allows us a) to recognize the exact combination of VMs it currently hosts to a high degree, and b) to track down the server hosting a specific VM. The discussed results show that our approach is suitable to reduce the complexity of attacking a Cloud infrastructure when aiming at a specific VM, allowing the attacker to avoid being too conspicuous by having knowledge about the hypervisor a VM is running on.

Using the described maximum log-likelihood approach, a lot of exact combinations can be recognized correctly. If the match condition is relaxed to a subset match, the recognition rate is increased. To track down a specific VM, the state of the respective VM needs to be recognizable regardless of combination of VMs it may run in. Having similar consumption densities leads to a situation where it is hard to recognize the exact combination, but it is easy to recognize the VM that is causing the similarity.

We have compared the maximum log-likelihood against Kolmogorov-Smirnov, yielding recognition rates approximately 5-10% lower than the maximum log-likelihood approach.

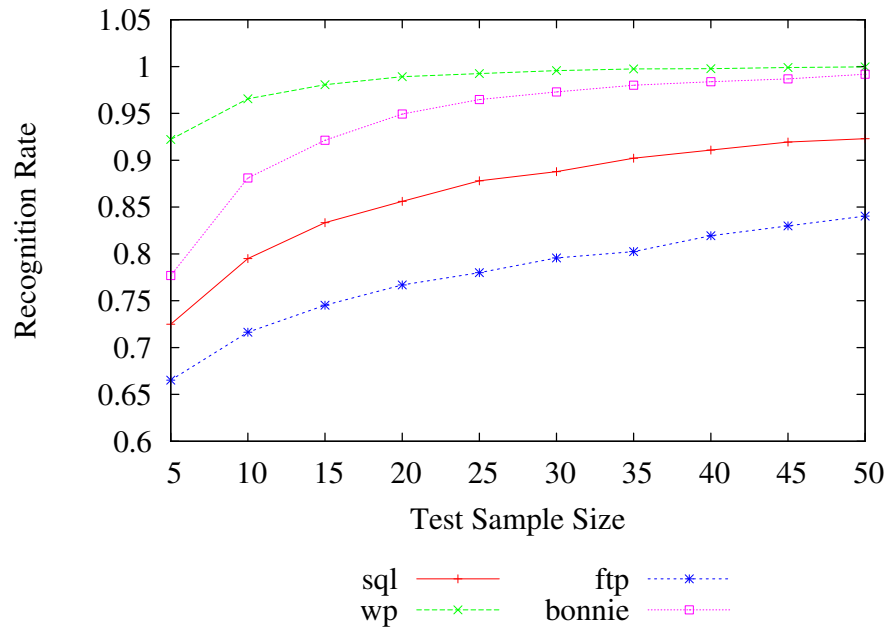


Figure 8.16: VM recognition rates for the ondemand governor scenario.

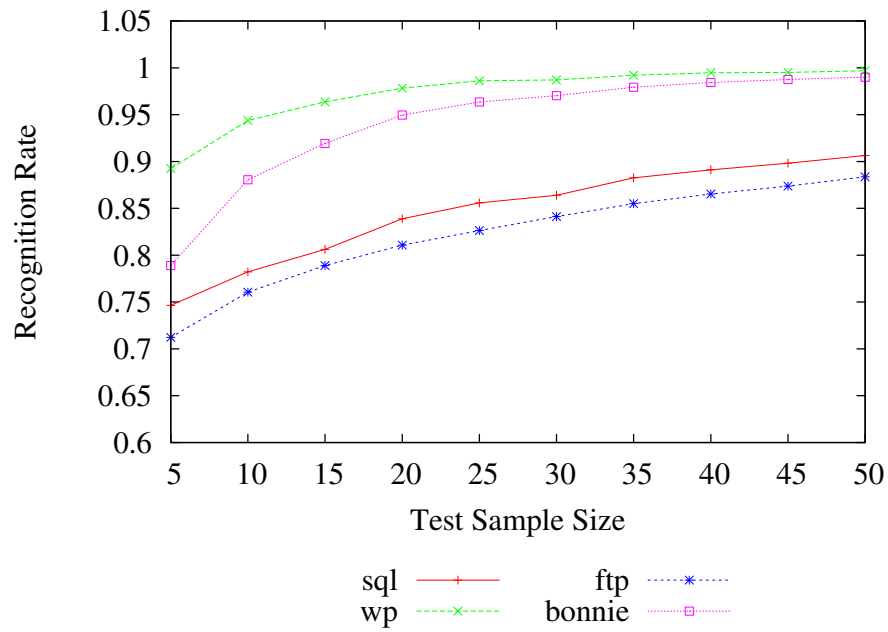


Figure 8.17: VM recognition rates for the powersave governor scenario.

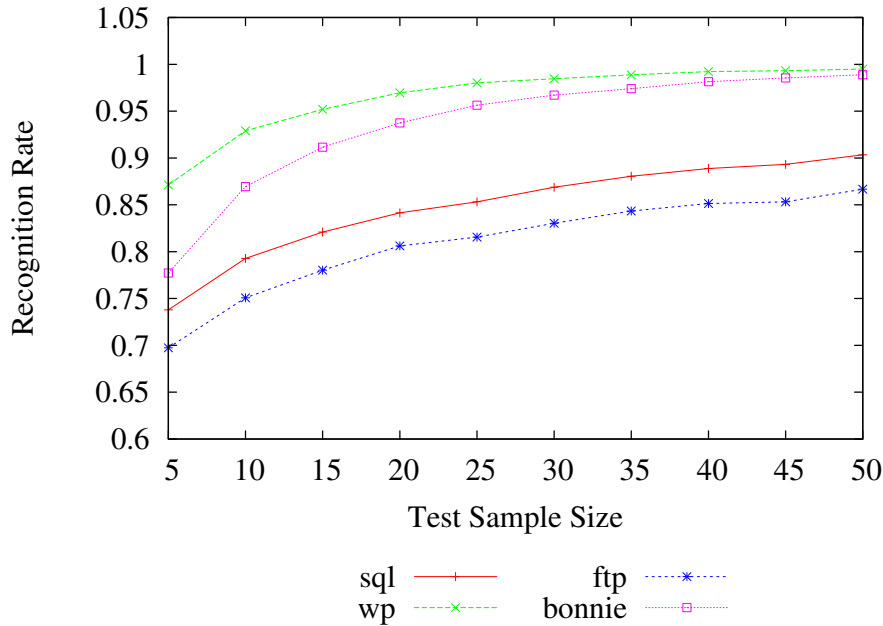


Figure 8.18: VM recognition rates for the `performance` governor scenario.

Taking into account the much faster runtime of Kolmogorov-Smirnov and the fact that results can become obsolete due to live migration of VMs, we think that the quality of results obtained from using Kolmogorov-Smirnov could be acceptable too.

In order to clarify whether our approach is generalizable and yields reproducible results, we repeated and extended our experiments on a second infrastructure in Lyon, using a different hardware platform and a power measurement device likely to be found in data centers. We have obtained similar results and have experimented with different CPU frequency scaling governors, yielding comparable quality of recognition.

As we have shown, power and energy consumption data of servers must be protected carefully, as it is possible to extract valuable information that an attacker could use to track down a VM before starting further attack steps. It is especially astonishing how few samples are required in our scenario for correct classification.

8.8 Future Work

We plan to extend our work by adding random noise activity and therefore power consumption to investigate at which level of noise the recognition rate is suffering significantly. We also want to add more types of VMs and use different workload patterns too see if there is a level of complexity which is decreasing the recognition rate respectively increasing the time required for computing reliable results. Additionally, there could be a point of complexity where Kolmogorov-Smirnov should be favored due to its lower runtime.

Finally, we plan to put effort into a more sophisticated attack scenario that does not require a training phase based on knowledge of the true combination of VMs running, but only information about the network address and kind of service (e.g., HTTP) of a VM, and access to an insufficiently secured metered rack PDU (e.g., by SNMP). Then, an attacker could send workload in varying intensities to the VM to cause changes in the power consumption of the server the VM is running on, trying to find the highest correlation between his actions and the consumption reactions. The assumption is that after removing noise, the server with the highest correlation in power consumption is, with high probability, the server hosting the VM in question. In such a scenario, the intensity and duration of injecting workload required to get reliable results is of high relevance, as a brute force approach would most probably rise suspicion.

Chapter 9

Conclusions and Future Work

In this thesis, a cost model driven approach to dynamic server consolidation aiming at increasing the energy efficiency of heterogeneous business computing infrastructures was proposed and evaluated. By using virtualization and virtual machine live migration, workload encapsulated in virtual machines can be rearranged across the infrastructure according to a cost model. A configurable cost model was defined in order to trade-off energy cost savings with quality of service. In this cost model, the cost of energy is one of several cost categories. Especially, overload situations leading to decreased quality of service and management overheads, e.g., live migration, are important cost categories.

As a result for the investigated scenario, 40%-50% of total operational costs can be saved by forecasting the resource demand and pro-actively creating a new, optimal mapping of virtual machines to physical servers. A greedy balanced first fit heuristic and a flexible meta-heuristic genetic algorithm, directly influenciabale by a configurable cost model were evaluated and compared with load-balancing. The greedy heuristic and the genetic algorithm deliver similar results for most evaluations of the starting set of cost parameters.

The flexibility of the genetic algorithm was shown by changing the cost parameters in several case studies, resulting in the genetic algorithm's adaption to the altered circumstances and definition of "optimal". Additionally, it can be turned from dynamic consolidation to load balancing by simply penalizing utilization variance across the physical servers.

Forecasting is essential when control loop iterations are carried out with low frequency, and can further increase the provisioning efficiency for shorter interval lengths. Holt-Winters and seasonal ARIMA deliver forecasts of similar quality, with seasonal ARIMA being computationally much more demanding.

For future work, extending the cost model to support heterogeneous VM overload costs as well as non-linear and heterogeneous VM live migration cost is especially interesting. Additionally, a virtual cost category similar to utilization variance could be introduced to penalize and effectively inhibit the co-existence of pairs of VMs which should not be hosted on the same server due to performance or security aspects. Further, other approaches in solving multi-objective combinatorial optimization problems could be investigated, e.g., simulated annealing.

Iterative pre-copy live migration is a resource intensive management action due to the dirty page scanning and copying during the migration. As it is a key mechanism for the

approach of this thesis, its influence on quality of service was empirically investigated and models created that allow to predict this influence. It is possible to predict the service level variance during a live migration to 90% using only a single variable, the UNIX `load5` average. Extending the model to 12 variables yields an $R_{adj}^2 = 0.95$.

Qualitatively, tighter SLAs can be fulfilled for low workload situations. For high workload situations, the service level decreases massively because of live migrations, therefore such situation should be avoided. As a consequence, live migrations should be triggered before a virtual machine is heavily loaded, if SLAs have to be fulfilled and can not be temporarily relaxed. The service level decreases non-linearly with increased utilization, hence the above mentioned cost model should be extended to capture this relationship. A further part of future work is to examine different types of services for their sensitivity to live migration.

Iterative pre-copy live migration clearly depends on the assumption of dirty page convergence to zero. Currently, not all hypervisors have stop conditions to enforce termination if this assumption does not hold. Due to an iterative migration process that is not converging, trying to reach unachievable low downtimes causes unacceptably low service levels, as the migration process itself is increasing the resource pressure onto the VM for a too long period of time.

On the one hand, a low level solution by patching KVM to implement Xen's stop conditions was evaluated. On the other hand, a high level solution by using a model able to estimate the remaining migration time and the corresponding impact on the service level was evaluated. Both approaches result in significantly better service levels and prevent the VM from thrashing and complete service disruption. They achieve acceptable service levels during live migration also for higher workload intensities by trading off migration delay and migration downtime.

The most interesting option for future work is to extend the evaluation to post-copy live migration, which is fundamentally different from iterative pre-copy live migration. An implementation for KVM became publicly available at the end of this thesis.

Finally, an examination was carried out regarding the possible consequences of collecting and storing energy consumption data of servers on security and privacy. Experimental results were presented how to use the energy consumption logs of a power monitored server as a side-channel that allows a) to recognize the exact combination of VMs it currently hosts to a high degree, and b) to track down the server hosting a specific VM.

A maximum log-likelihood approach was compared with a Kolmogorov-Smirnov test. The latter performs slightly worse than the former, but offers substantially lower runtimes. Having similar power consumption probability densities leads to a situation where it is hard to recognize the exact combination, but it is easy to recognize the VM that is causing the similarity. A verification whether the approach is generalizable was done by repeating and extended the experiments on a second infrastructure in Lyon. As a result, power and energy consumption data of servers must be protected carefully, as it is possible to extract valuable information that an attacker could use to track down a VM before starting further attack steps.

For future work, a more sophisticated attack scenario based on a service URI and real-time access to power logs of a number of servers should be investigated. Then, an attacker could send workload in varying intensities to the VM to cause changes in the power consumption of the server the VM is running on, trying to find the highest correlation between his actions and the reactions in power consumption.

Bibliography

- [AA06] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XII, pages 2–13, New York, NY, USA, 2006. ACM.
- [AFG⁺09] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.
- [AJ00] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30–37, 2000.
- [AK08] G. Ananthanarayanan and R.H. Katz. Greening the switch. In *Proceedings of the 2008 conference on Power aware computing and systems*, pages 7–7. USENIX Association, 2008.
- [Aka74] H. Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, December 1974.
- [AKS00] E.R. Altman, D. Kaeli, and Y. Sheffer. Welcome to the opportunities of binary translation. *Computer*, 33(3):40–45, 2000.
- [AMD07] AMD. AMD I/O Virtualization Technology (IOMMU) Specification. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf, February 2007. (last visited: 05/04/2008).
- [ASR⁺10] S. Akoush, R. Sohan, A. Rice, A.W. Moore, and A. Hopper. Predicting the performance of virtual machine migration. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 37–46. IEEE, 2010.
- [AW96] M.F. Arlitt and C.L. Williamson. Web server workload characterization: The search for invariants. In *ACM SIGMETRICS Performance Evaluation Review*, volume 24, pages 126–137. ACM, 1996.
- [AW97] M.F. Arlitt and C.L. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking (ToN)*, 5(5):631–645, 1997.

- [BB10] A. Beloglazov and R. Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 577–578. Ieee, 2010.
- [BDF⁺03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.
- [BDPS09] D. Borgetto, G. Da Costa, J.-M. Pierson, and A. Sayah. Energy-aware resource allocation. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 183–188, oct. 2009.
- [BE11] David Breitgand and Amir Epstein. Sla-aware placement of multi-virtual machine elastic services in compute clouds. In Nazim Agoulmine, Claudio Bartolini, Tom Pfeifer, and Declan O’Sullivan, editors, *Integrated Network Management*, pages 161–168. IEEE, 2011.
- [BGdMT06] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi. *Queueing Networks and Markov Chains*. Wiley & Sons, 2nd edition, 2006.
- [BH07] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40:33–37, December 2007.
- [BH09] L.A. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108, 2009.
- [BKB] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007*.
- [BKB07] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007*, 2007.
- [BKR10] D. Breitgand, G. Kutiel, and D. Raz. Cost-aware live migration of services in the cloud. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, page 11. ACM, 2010.
- [Bri07] K.G. Brill. The invisible crisis in the data center: The economic meltdown of moore’s law. *white paper*, Uptime Institute, 2007.
- [BRW07] J. Burge, P. Ranganathan, and J.L. Wiener. Cost-aware scheduling for heterogeneous enterprise machines (cash’em). In *Cluster Computing, 2007 IEEE International Conference on*, pages 481–487, sept. 2007.
- [BS96] J.E. Beck and D.P. Siewiorek. Modeling multicomputer task allocation as a vector packing problem. In *System Synthesis, 1996. Proceedings., 9th International Symposium on*, pages 115–120. IEEE, 1996.

-
- [CAB⁺07] Z. Chen, P. Ananthanarayanan, S. Biswas, B. Campbell, H. Chen, S. Desai, D. Go, R. Goel, V. von Kaenel, J. Kassoff, et al. A 25w soc with dual 2ghz power cores and integrated memory and i/o subsystems. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 104–105. IEEE, 2007.
- [Car11] Careglio, D. and Da Costa, G. and Kat, R., Pierson, J.M. Hardware leverages for energy reduction in large scale distributed systems, 2nd Year, 2011.
- [CFH⁺05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.
- [Cha05a] Charlie Demerjian. Intel Vanderpool holds promise, some pitfalls. <http://www.theinquirer.net/en/inquirer/news/2005/02/25/intel-vanderpool-holds-promise-some-pitfalls>, February 2005.
- [Cha05b] Charlie Demerjian. AMD's Pacifica revealed in full. <http://www.theinquirer.net/en/inquirer/news/2005/06/06/amds-pacifica-revealed-in-full>, June 2005.
- [CIL⁺07] Yuan Chen, S. Iyer, Xue Liu, D. Milojicic, and A. Sahai. Sla decomposition: Translating service level objectives to system level thresholds. In *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, page 3, june 2007.
- [CJ06] Sean Campbell and Michael Jeronimo. An Introduction to Virtualization. http://softwarecommunity.intel.com/isn/downloads/virtualization/pdfs/An_Introduction_to_Virtualization.pdf, 2006. (last visited: 03/20/2008).
- [CKSC08] Hyung Won Choi, Hukeun Kwak, Andrew Sohn, and Kyusik Chung. Autonomous learning for efficient resource utilization of dynamic vm migration. In *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, pages 185–194, New York, NY, USA, 2008. ACM.
- [CMR⁺10] J. Chang, J. Meza, P. Ranganathan, C. Bash, and A. Shah. Green server design: Beyond operational energy to sustainability. *Memory*, 4(10):50, 2010.
- [CP09] Paolo Campegiani and Francesco Lo Presti. A general model for virtual machines resources allocation in multi-tier distributed systems. *Autonomic and Autonomous Systems, International Conference on*, 0:162–167, 2009.
- [CPK10] Yanpei Chen, Vern Paxson, and Randy H. Katz. What's new about cloud computing security? Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley, Jan 2010.

- [DdAG⁺10] G. Da Costa, M.D. de Assunção, J.P. Gelas, Y. Georgiou, L. Lefèvre, A.C. Orgerie, J.M. Pierson, O. Richard, and A. Sayah. Multi-facet approach to reduce energy consumption in clouds and grids: the green-net framework. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 95–104. ACM, 2010.
- [DHKC09] Stephen Dawson-Haggerty, Andrew Krioukov, and David E. Culler. Power optimization – a reality check. Technical Report UCB/EECS-2009-140, EECS Department, University of California, Berkeley, Oct 2009.
- [DNAA] Sudipto Das, Shoji Nishimura, Divyakant Agrawal, and Amr El Abbadi. Live database migration for elasticity in a multitenant database for cloud platforms.
- [DO91] Fred Douglass and John Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software - Practice and Experience*, 21:757–785, 1991.
- [Dou89] Fred Douglass. Experience with process migration in sprite. In *Proceedings of USENIX Workshop on Distributed and Multiprocessor Systems*, pages 59–72, 1989.
- [Dyc90] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.
- [ENE07] S. ENERGY. Report to congress on server and data center energy efficiency public law 109-431. *Public law*, 109:431, 2007.
- [EP07] Christoph Egger and Andre Przywara. Virtualisierungs-Turbo. *Linux Magazin, Technical Review Virtualisierung*, 2006/07.
- [FD92] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 1186–1192. IEEE, 1992.
- [FF⁺93] C.M. Fonseca, P.J. Fleming, et al. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the fifth international conference on genetic algorithms*, volume 1, page 416. Citeseer, 1993.
- [FOW66] L.J. Fogel, A.J. Owens, and M.J. Walsh. Artificial intelligence through simulated evolution. 1966.
- [FRM⁺10] Eugen Feller, Louis Rilling, Christine Morin, Renaud Lottiaux, and Daniel Leprince. Snooze: A scalable, fault-tolerant and distributed consolidation manager for large-scale clusters. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pages 125–132, Washington, DC, USA, 2010. IEEE Computer Society.
- [FWB07] 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*, pages 13–23. ACM, 2007.

-
- [GG66] PC Gilmore and RE Gomory. The theory and computation of knapsack functions. *Operations Research*, pages 1045–1074, 1966.
- [GKL08] L. Gray, A. Kumar, and H. Li. Characterization of specpower_ssj2008** benchmark. In *SPEC Benchmark Workshop*, 2008.
- [GMT⁺06] S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, and B. Myatt. Best practices for data centers: Lessons learned from benchmarking 22 data centers. *Proceedings of the ACEEE Summer Study on Energy Efficiency in Buildings in Asilomar, CA. ACEEE, August*, 3:76–87, 2006.
- [Gol73] R.P. Goldberg. Architectural principles for virtual computer systems. Technical report, DTIC Document, 1973.
- [Gol89] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley, 1989.
- [GR05] Tal Garfinkel and Mendel Rosenblum. When virtual is harder than real: security challenges in virtual machine based computing environments. In *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, pages 20–20, Berkeley, CA, USA, 2005. USENIX Association.
- [GRCK07] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, pages 171–180. IEEE, 2007.
- [Gri07] The Green Grid. The Green Grid Metrics: Describing Data Center Power Efficiency. <http://www.thegreengrid.org/en/Global/Content/white-papers/Green-Grid-Metrics>, 2007. (last visited: 08/29/2011).
- [Hav05] Havard Bjerke. Virtualization hardware extensions. <http://project-xen.web.cern.ch/project-xen/xen/hardware.html>, December 2005.
- [HDG09] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev.*, 43(3):14–26, 2009.
- [HDP09] H. Hlavacs, G. Da Costa, and J.M. Pierson. Energy consumption of residential and professional switches. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 1, pages 240–246. IEEE, 2009.
- [HG09] Michael R. Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 51–60, New York, NY, USA, 2009. ACM.
- [HGM⁺07] Eric Harney, Sebastien Goasguen, Jim Martin, Mike Murphy, and Mike Westall. The efficacy of live virtual machine migrations over the internet. In *VTDC '07: Proceedings of the 2nd international workshop on Virtualization technology in distributed computing*, pages 1–7, 2007.

- [HH09] Stuart Hacking and Benoît Hudzia. Improving the live migration process of large enterprise applications. In *VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, pages 51–58, New York, NY, USA, 2009. ACM.
- [HLM⁺09] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 41–50, New York, NY, USA, 2009. ACM.
- [HMGW] C. Hyser, B. McKee, R. Gardner, and B.J. Watson. Autonomic virtual machine placement in the data center.
- [HNIS11] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi. Making vm consolidation more energy-efficient by postcopy live migration. In *CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 195–204, 2011.
- [HNO⁺09] Takahiro Hirofuchi, Hidemoto Nakada, Hirotaka Ogawa, Satoshi Itoh, and Satoshi Sekiguchi. A live storage migration mechanism over wan and its performance evaluation. In *VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, pages 67–74, New York, NY, USA, 2009. ACM.
- [Int07] Intel. Intel Virtualization Technology for Directed I/O architecture specification. [http://download.intel.com/technology/computing/vptech/Intel\(r\)_VT_for_Direct_IO.pdf](http://download.intel.com/technology/computing/vptech/Intel(r)_VT_for_Direct_IO.pdf), 2007. (last visited: 04/21/2008).
- [JADAD06] Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Geiger: monitoring the buffer cache in a virtual machine environment. *SIGARCH Comput. Archit. News*, 34(5):14–24, 2006.
- [JDW⁺09] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan. Live virtual machine migration with adaptive, memory compression. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–10. Ieee, 2009.
- [JG11] W. Jansen and T. Grance. Guidelines on Security and Privacy in Public Cloud Computing. Draft from the National Institute of Standards and Technology, USA, 2011.
- [Jon08] Evan Cooke Farnam Jahanian Jon Oberheide. Empirical exploitation of live virtual machine migration. <http://www.blackhat.com/presentations/bh-dc-08/Oberheide/Whitepaper/bh-dc-08-oberheide-WP.pdf>, 2008.
- [KAB⁺03] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore’s law meets static power. *Computer*, 36(12):68–75, 2003.

-
- [KCH09] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, pages 117–126, New York, NY, USA, 2009. ACM.
- [Kek06a] David M. Auslander Cullen E. Bash; Chandrakant D. Patel Keke Chen. Local temperature control in data center cooling: Part i, correlation matrix. <http://www.hpl.hp.com/techreports/2006/>, 2006.
- [Kek06b] David M. Auslander Cullen E. Bash; Chandrakant D. Patel Keke Chen. Local temperature control in data center cooling: Part ii, statistical analysis. <http://www.hpl.hp.com/techreports/2006/>, 2006.
- [Kek06c] David M. Auslander Cullen E. Bash; Chandrakant D. Patel Keke Chen. Local temperature control in data center cooling: Part iii, application. <http://www.hpl.hp.com/techreports/2006/>, 2006.
- [KFB11] G. Kalogridis, Z. Fan, and S. Basutkar. Affordable Privacy for Home Smart Meters. In *Workshop on Smart Grid Security and Communications (in conjunction with ISPA)*, pages 77–84, 2011.
- [KKH⁺08] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. In *ICAC '08: Proceedings of the 2008 International Conference on Autonomic Computing*, pages 3–12, Washington, DC, USA, 2008. IEEE Computer Society.
- [KKL⁺07a] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.
- [KKL⁺07b] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Proceeding of the Linux Symposium, Volume One*, 2007.
- [KLMS84] R.M. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 289–298. ACM, 1984.
- [Koo07] J.G. Koomey. Estimating Total Power Consumption by Servers in the US and the World. Technical report, Lawrence Berkeley National Laboratory and Stanford University, 2007.
- [Koo08] J.G. Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3:034008, 2008.
- [Koo11] Jonathan Koomey. Growth in Data center electricity use 2005 to 2010. <http://www.analyticspress.com/datacenters.html>, 2011. (last visited: 08/29/2011).

- [Kor09] Kostya Kortchinsky. Cloudburst: Hacking 3d (and breaking out of vmware). In *Black Hat Technical Security Conference*, 2009.
- [KPP04] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.
- [KTK⁺09] Sanjay Kumar, Vanish Talwar, Vibhore Kumar, Parthasarathy Ranganathan, and Karsten Schwan. vmanage: loosely coupled platform and virtualization management in data centers. In *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*, pages 127–136, New York, NY, USA, 2009. ACM.
- [Kuh55] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.
- [LJL⁺09] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. Live migration of virtual machine based on full system trace and replay. In *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 101–110, New York, NY, USA, 2009. ACM.
- [LTWW94] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of ethernet traffic (extended version). *Networking, IEEE/ACM Transactions on*, 2(1):1–15, 1994.
- [Lud07] Thomas Kneib Stefan Lang Ludwig Fahrmeier. *Regression*. Springer, 1st edition, 2007.
- [LWHC07] Jian-Ming Li, Xiao-Jing Wang, Rong-Sheng He, and Zhong-Xian Chi. An efficient fine-grained parallel genetic algorithm based on gpu-accelerated. In *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pages 855–862, sept. 2007.
- [LXJ⁺11] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th international symposium on High performance distributed computing, HPDC '11*, pages 171–182, New York, NY, USA, 2011. ACM.
- [MBL⁺09] Ogier Maitre, Laurent A. Baumes, Nicolas Lachiche, Avelino Corma, and Pierre Collet. Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation, GECCO '09*, pages 1403–1410, New York, NY, USA, 2009. ACM.
- [MCR06] J. Moore, J.S. Chase, and P. Ranganathan. Weatherman: Automated, online and predictive thermal mapping and management for data centers. In *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pages 155–164, June 2006.
- [MCRS05] Justin Moore, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.

-
- [MDP⁺00] Dejan S. Milojevic, Fred Douglass, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. *ACM Computing Surveys*, 32:2000, 2000.
- [MPL09] D. Molaro, H. Payer, and D. Le Moal. Tempo: disk drive power consumption characterization and modeling. In *Consumer Electronics, 2009. ISCE'09. IEEE 13th International Symposium on*, pages 246–250. IEEE, 2009.
- [MT90] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.
- [MVML09] Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. Elastic management of cluster-based services in the cloud. In *ACDC '09: Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pages 19–24, New York, NY, USA, 2009. ACM.
- [NS07] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 265–278, New York, NY, USA, 2007. ACM.
- [NWG⁺09] Daniel Nurmi, Rich Wolski, Chris Grzegorzczak, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.
- [Orm08] T. Ormandy. An Empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments. White Paper, 2008.
- [PC11] Jean-Marc Pierson and Henri Casanova. On the utility of dvfs for power-aware job placement in clusters. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par 2011 Parallel Processing*, volume 6852 of *Lecture Notes in Computer Science*, pages 255–266. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-23400-2_24.
- [PG74] G.J. Popek and R.P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.
- [PJS10] Petr Pospichal, Jiri Jaros, and Josef Schwarz. Parallel genetic algorithm on the cuda architecture. In Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Anikó Ekárt, Anna Esparcia-Alcazar, Chi-Keong Goh, Juan Merelo, Ferrante Neri, Mike Preuß, Julian Togelius, and Georgios Yannakakis, editors, *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 442–451. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-12239-2_46.
- [Pla09] Platts. Day-ahead market prices. MegaWatt Daily, McGraw-Hill, 2006–2009.
- [PS05] C.D. Patel and A.J. Shah. Cost model for planning, development and operation of a data center. *HPL 2005-107 (R. 1)*, HP Lab. Technical Report, 2005.

- [PSvD06] R. Perez, R. Sailer, and L. van Doorn. vTPM: virtualizing the trusted platform module. In *Conference on USENIX Security Symposium*, 2006.
- [PSZ⁺07] Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 289–302, New York, NY, USA, 2007. ACM.
- [QNC06] Benjamin Quetier, Vincent Neri, and Franck Cappello. Selecting a virtualization system for grid/p2p large scale emulation. In *Proc of the Workshop on Experimental Grid testbeds for the assessment of large-scale distributed applications and tools (EXP-GRID'06)*, Paris, France, 19-23 june, 2006.
- [QWB⁺09] A. Qureshi, R. Weber, H. Balakrishnan, J. Gutttag, and B. Maggs. Cutting the electric bill for internet-scale systems. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 123–134. ACM, 2009.
- [RG05] Mendel Rosenblum and Tal Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38:39–47, 2005.
- [RMP11] P. Riteau, C. Morin, and T. Priol. Shrinker: improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing. *Euro-Par 2011 Parallel Processing*, pages 431–442, 2011.
- [RRM09] P. Ranganathan, S. Rivoire, and J. Moore. Models and metrics for energy-efficient computing. *Advances in Computers*, 75:159–233, 2009.
- [RRT⁺08] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 48–59. ACM, 2008.
- [RSV07] K. K. Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. Live data center migration across wans: a robust cooperative context aware approach. In *INM '07: Proceedings of the 2007 SIGCOMM workshop on Internet network management*, pages 262–267, New York, NY, USA, 2007. ACM.
- [RTSS09] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.
- [SGR09] N. Santos, K. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Conference on Hot topics in cloud computing (HotCloud)*, Berkeley, CA, USA, 2009. USENIX Association.
- [SHTE11] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *Proceedings of the 7th*

- ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 111–120. ACM, 2011.
- [SLB07] Jan Stoess, Christian Lang, and Frank Bellosa. Energy management for hypervisor-based virtual machines. In *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2007. USENIX Association.
- [SLP11] D. Seo, H. Lee, and A. Perrig. Secure and Efficient Capability-Based Power Management in the Smart Grid. In *Workshop on Smart Grid Security and Communications (in conjunction with ISPA)*, pages 119–126, 2011.
- [Smi88] J.M. Smith. A survey of process migration mechanisms. *ACM SIGOPS Operating Systems Review*, 22(3):28–40, 1988.
- [SML10] J. Sahoo, S. Mohapatra, and R. Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222–226. IEEE, 2010.
- [SSB⁺08] R. Sharma, A. Shah, C. Bash, T. Christian, and C. Patel. Water efficiency management in datacenters (part i): Introducing a water usage metric based on available energy consumption. *Hewlett-Packard Laboratories Technical Report, HPL-2008-206*, 2008.
- [SSVC10] Mark Stillwell, David Schanzenbach, Frédéric Vivien, and Henri Casanova. Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and Distributed Computing*, 70(9):962–974, 2010.
- [SWC08] Malgorzata Steinder, Ian Whalley, and David Chess. Server virtualization in autonomic management of heterogeneous workloads. *SIGOPS Oper. Syst. Rev.*, 42(1):94–95, 2008.
- [TDG⁺06] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. De Laat, J. Mambretti, I. Monga, B. Van Oudenaarde, S. Raghunath, and P. Yonghui Wang. Seamless live migration of virtual machines over the man/wan. *Future Generation Computer Systems*, 22(8):901–907, 2006.
- [Tre08] T. Treutner. Virtual home environments. Diploma Thesis, 2008.
- [TvS08] Andrew S. Tanenbaum and Maarten van Steen. *Verteilte Systeme - Prinzipien und Paradigmen*. Pearson Studium, 2nd edition, 2008.
- [VAD⁺08] B. Vöcking, H. Alt, M. Dietzfelbinger, R. Reischuk, C. Scheideler, H. Vollmer, and D. Wagner. *Taschenbuch der algorithmen*. Springer, 2008.
- [VAN08a] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

- [VAN08b] Akshat Verma, Puneet Ahuja, and Anindya Neogi. Power-aware dynamic placement of hpc applications. In *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, pages 175–184, New York, NY, USA, 2008. ACM.
- [VBVB09] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, pages 254–265, Berlin, Heidelberg, 2009. Springer-Verlag.
- [VKUR10] Akshat Verma, Ricardo Koller, Luis Useche, and Raju Rangaswami. Srcmap: energy proportional storage using dynamic consolidation. In *Proceedings of the 8th USENIX conference on File and storage technologies, FAST'10*, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association.
- [VMw07] VMware. A performance Comparison of Hypervisors. http://www.vmware.com/pdf/hypervisor_performance.pdf, 2007. (last visited: 04/21/2008).
- [Vog08] Werner Vogels. Beyond server consolidation. *Queue*, 6(1):20–26, January 2008.
- [Wal02] C.A. Waldspurger. Memory resource management in vmware esx server. *ACM SIGOPS Operating Systems Review*, 36(SI):181–194, 2002.
- [WAWB05] A. Williams, M. Arlitt, C. Williamson, and K. Barker. Web workload characterization: Ten years later. *Web content delivery*, pages 3–21, 2005.
- [WCC⁺08] John Paul Walters, Vipin Chaudhary, Minsuk Cha, Salvatore Guercio Jr., and Steve Gallo. A comparison of virtualization technologies for hpc. *Advanced Information Networking and Applications, International Conference on*, 0:861–868, 2008.
- [WN67] H.M. Weingartner and D.N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, pages 83–103, 1967.
- [WRSV11] T. Wood, KK Ramakrishnan, P. Shenoy, and J. Van der Merwe. Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 121–132. ACM, 2011.
- [WSVY07] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, NSDI'07, pages 17–17, Berkeley, CA, USA, 2007. USENIX Association.
- [WTLS⁺09] Timothy Wood, Gabriel Tarasuk-Levin, Prashant Shenoy, Peter Desnoyers, Emmanuel Cecchet, and Mark D. Corner. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 31–40, New York, NY, USA, 2009. ACM.

- [WWLL09] Chuliang Weng, Zhigang Wang, Minglu Li, and Xinda Lu. The hybrid scheduling framework for virtual machine systems. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 111–120, New York, NY, USA, 2009. ACM.
- [Xen06] XenSource. Xen: Enterprise Grade Open Source Virtualization, Inside Xen 3.0, A XenSource Whitepaper. http://xen.xensource.com/files/xensource_wp2.pdf, 2006. (last visited: 04/19/2008).
- [XF10] Jing Xu and J.A.B. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCoM)*, pages 179–188, dec. 2010.
- [YCP05] Qizhi Yu, Chongcheng Chen, and Zhigeng Pan. Parallel genetic algorithms on programmable graphics hardware. In Lipo Wang, Ke Chen, and Yew Ong, editors, *Advances in Natural Computation*, volume 3612 of *Lecture Notes in Computer Science*, pages 444–444. Springer Berlin / Heidelberg, 2005. 10.1007/11539902_134.
- [Zay87] E. Zayas. Attacking the process migration bottleneck. In *ACM SIGOPS Operating Systems Review*, volume 21, pages 13–24. ACM, 1987.
- [ZBG⁺05] Yuting Zhang, Azer Bestavros, Mina Guirguis, Ibrahim Matta, and Richard West. Friendly virtual machines: leveraging a feedback-control model for application adaptation. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 2–12, New York, NY, USA, 2005. ACM.
- [ZH09] Yi Zhao and Wenlong Huang. Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud. In *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, pages 170–175, aug. 2009.
- [ZHMM10] X. Zhang, Z. Huo, J. Ma, and D. Meng. Exploiting data deduplication to accelerate live virtual machine migration. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, pages 88–96. IEEE, 2010.
- [ZNS11] Jie Zheng, Tze Sing Eugene Ng, and Kunwadee Sripanidkulchai. Workload-aware live storage migration for clouds. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, pages 133–144, New York, NY, USA, 2011. ACM.

Appendix A

MetaVisor and MetaSim GUI

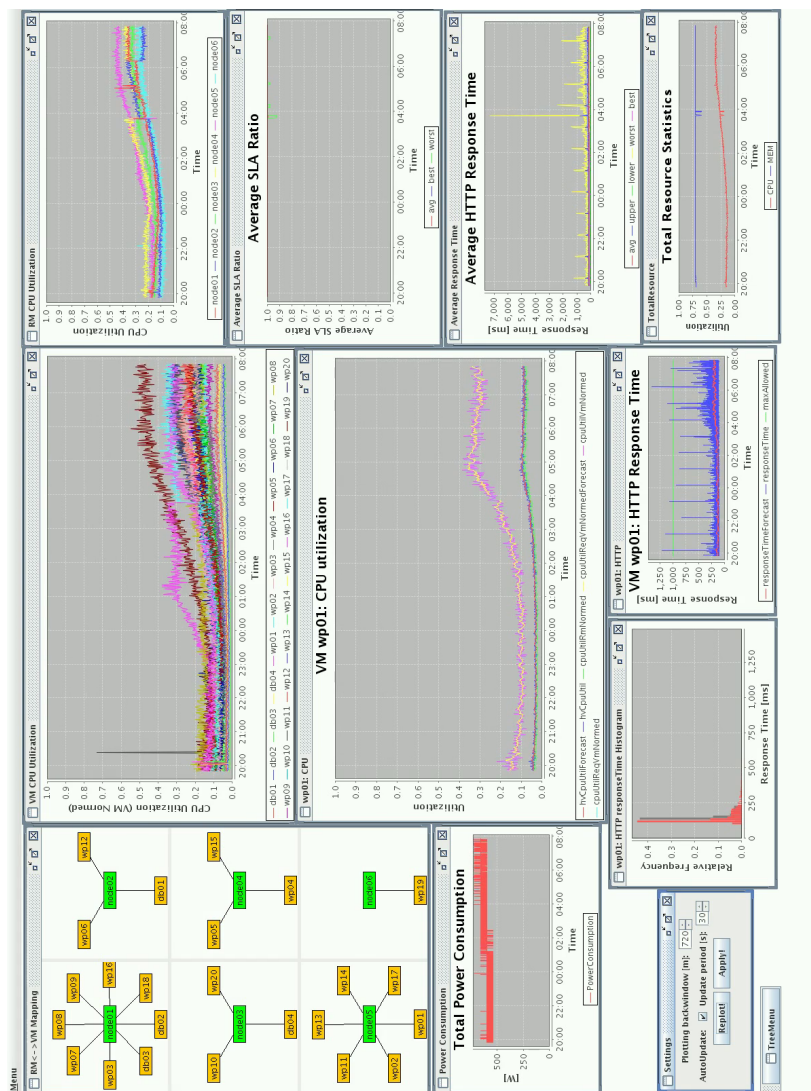


Figure A.1: MetaVisor GUI showing an empirical test run.

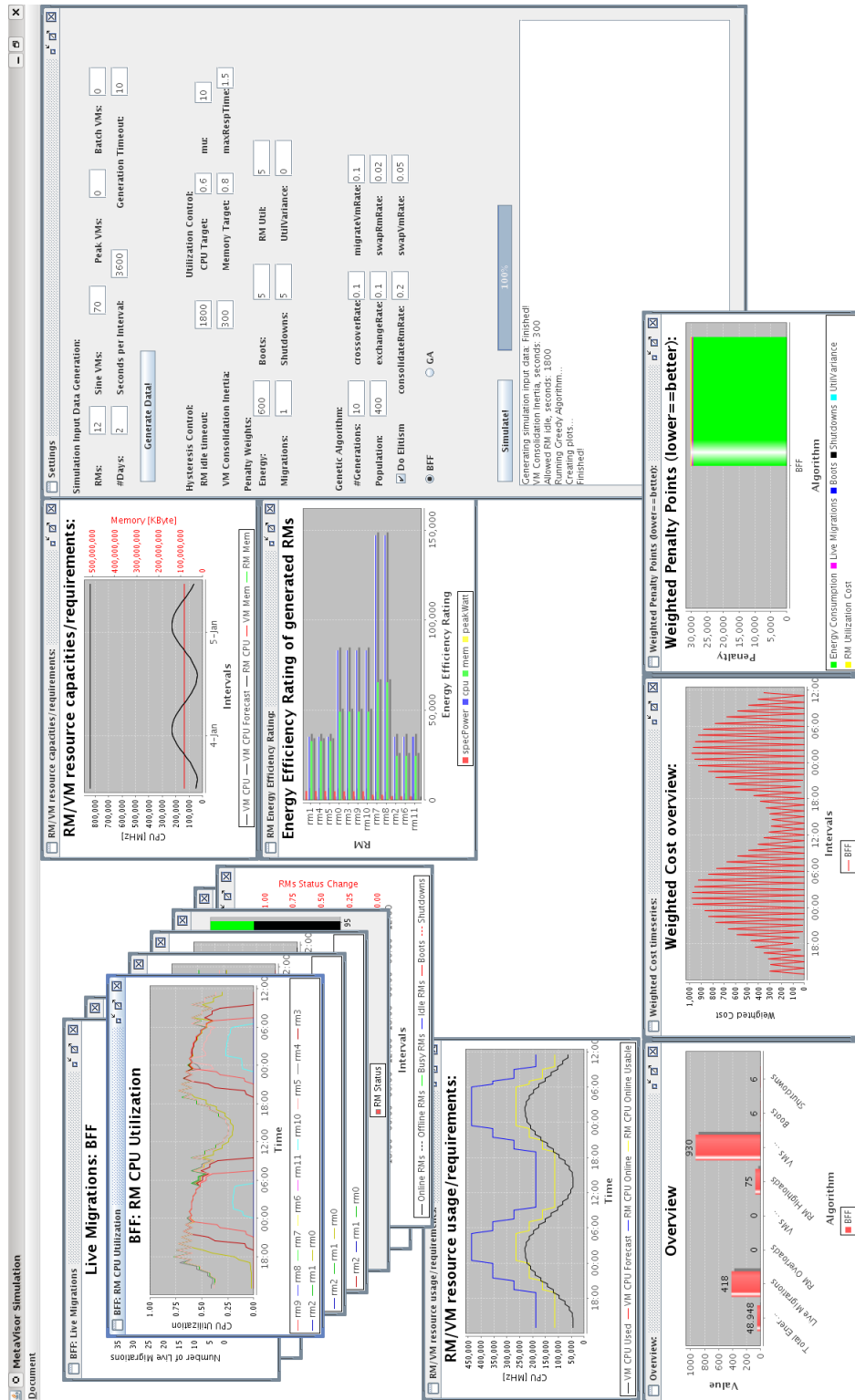


Figure A.2: MetaSim GUI showing a simulation test run.

Appendix B

Basic Configuration for the MetaSim Simulation.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <simulation>
3   <type>file</type>
4   <scenario id="zidspec">
5     <plot>true</plot>
6     <VMs>
7       <skipDays>7</skipDays>
8       <readDays>21</readDays>
9       <vm>
10        <amount>25</amount>
11        <cpuUtilizationTraceFile>traces / zid / 0. dat</cpuUtilizationTraceFile>
12        <memory>512</memory>
13        <scaling>2</scaling>
14      </vm>
15      <vm>
16        <amount>25</amount>
17        <cpuUtilizationTraceFile>traces / zid / 1. dat</cpuUtilizationTraceFile>
18        <memory>1024</memory>
19        <scaling>4</scaling>
20      </vm>
21      <vm>
22        <amount>4</amount>
23        <cpuUtilizationTraceFile>traces / zid / 2. dat</cpuUtilizationTraceFile>
24        <memory>768</memory>
25        <scaling>5</scaling>
26      </vm>
27      <vm>
28        <amount>15</amount>
29        <cpuUtilizationTraceFile>traces / zid / 3. dat</cpuUtilizationTraceFile>
30        <memory>512</memory>
31        <scaling>2</scaling>
32      </vm>
33      <vm>
34        <amount>20</amount>
35        <cpuUtilizationTraceFile>traces / zid / 4. dat</cpuUtilizationTraceFile>
36        <memory>512</memory>
37        <scaling>2</scaling>
38      </vm>
39      <vm>
40        <amount>5</amount>
41        <cpuUtilizationTraceFile>traces / zid / 5. dat</cpuUtilizationTraceFile>
42        <memory>512</memory>
43        <scaling>5</scaling>
44      </vm>
45    </VMs>
```

```
46     <amount>10</amount>
47     <cpuUtilizationTraceFile>traces / zid / 6. dat</ cpuUtilizationTraceFile>
48     <memory>512</memory>
49     <scaling>1</ scaling>
50 </vm>
51 <vm>
52     <amount>15</amount>
53     <cpuUtilizationTraceFile>traces / zid / 7. dat</ cpuUtilizationTraceFile>
54     <memory>768</memory>
55     <scaling>1</ scaling>
56 </vm>
57 <vm>
58     <amount>10</amount>
59     <cpuUtilizationTraceFile>traces / zid / 8. dat</ cpuUtilizationTraceFile>
60     <memory>768</memory>
61     <scaling>3</ scaling>
62 </vm>
63 <vm>
64     <amount>5</amount>
65     <cpuUtilizationTraceFile>traces / zid / 9. dat</ cpuUtilizationTraceFile>
66     <memory>768</memory>
67     <scaling>3</ scaling>
68 </vm>
69 <vm>
70     <amount>25</amount>
71     <cpuUtilizationTraceFile>traces / zid / 10. dat</ cpuUtilizationTraceFile>
72     <memory>1024</memory>
73     <scaling>3</ scaling>
74 </vm>
75 <vm>
76     <amount>4</amount>
77     <cpuUtilizationTraceFile>traces / zid / 11. dat</ cpuUtilizationTraceFile>
78     <memory>1024</memory>
79     <scaling>2</ scaling>
80 </vm>
81 <vm>
82     <amount>8</amount>
83     <cpuUtilizationTraceFile>traces / zid / 12. dat</ cpuUtilizationTraceFile>
84     <memory>768</memory>
85     <scaling>2</ scaling>
86 </vm>
87 <vm>
88     <amount>5</amount>
89     <cpuUtilizationTraceFile>traces / zid / 13. dat</ cpuUtilizationTraceFile>
90     <memory>384</memory>
91     <scaling>1</ scaling>
92 </vm>
93 <vm>
94     <amount>15</amount>
95     <cpuUtilizationTraceFile>traces / zid / 14. dat</ cpuUtilizationTraceFile>
96     <memory>768</memory>
97     <scaling>5</ scaling>
98 </vm>
99 <vm>
100    <amount>4</amount>
101    <cpuUtilizationTraceFile>traces / zid / 15. dat</ cpuUtilizationTraceFile>
102    <memory>768</memory>
103    <scaling>2</ scaling>
104 </vm>
105 <vm>
106    <amount>8</amount>
107    <cpuUtilizationTraceFile>traces / zid / 16. dat</ cpuUtilizationTraceFile>
108    <memory>512</memory>
```

```

109     <scaling>1</scaling>
110 </vm>
111 <vm>
112     <amount>13</amount>
113     <cpuUtilizationTraceFile>traces / zid / 17. dat</cpuUtilizationTraceFile>
114     <memory>384</memory>
115     <scaling>1</scaling>
116 </vm>
117 <vm>
118     <amount>20</amount>
119     <cpuUtilizationTraceFile>traces / zid / 18. dat</cpuUtilizationTraceFile>
120     <memory>512</memory>
121     <scaling>1</scaling>
122 </vm>
123 <vm>
124     <amount>4</amount>
125     <cpuUtilizationTraceFile>traces / zid / 19. dat</cpuUtilizationTraceFile>
126     <memory>768</memory>
127     <scaling>3</scaling>
128 </vm>
129 <vm>
130     <amount>20</amount>
131     <cpuUtilizationTraceFile>traces / zid / 20. dat</cpuUtilizationTraceFile>
132     <memory>1024</memory>
133     <scaling>3</scaling>
134 </vm>
135 <vm>
136     <amount>8</amount>
137     <cpuUtilizationTraceFile>traces / zid / 21. dat</cpuUtilizationTraceFile>
138     <memory>512</memory>
139     <scaling>3</scaling>
140 </vm>
141 <vm>
142     <amount>5</amount>
143     <cpuUtilizationTraceFile>traces / zid / 22. dat</cpuUtilizationTraceFile>
144     <memory>512</memory>
145     <scaling>3</scaling>
146 </vm>
147 <vm>
148     <amount>2</amount>
149     <cpuUtilizationTraceFile>traces / zid / 23. dat</cpuUtilizationTraceFile>
150     <memory>512</memory>
151     <scaling>2</scaling>
152 </vm>
153 <vm>
154     <amount>15</amount>
155     <cpuUtilizationTraceFile>traces / zid / 24. dat</cpuUtilizationTraceFile>
156     <memory>512</memory>
157     <scaling>1</scaling>
158 </vm>
159 <vm>
160     <amount>15</amount>
161     <cpuUtilizationTraceFile>traces / zid / 25. dat</cpuUtilizationTraceFile>
162     <memory>512</memory>
163     <scaling>4</scaling>
164 </vm>
165 <vm>
166     <amount>5</amount>
167     <cpuUtilizationTraceFile>traces / zid / 26. dat</cpuUtilizationTraceFile>
168     <memory>384</memory>
169     <scaling>2</scaling>
170 </vm>
171 <vm>

```

Appendix B Basic Configuration for the MetaSim Simulation.

```
172     <amount>3</amount>
173     <cpuUtilizationTraceFile>traces / zid / 27. dat</cpuUtilizationTraceFile>
174     <memory>384</memory>
175     <scaling>2</scaling>
176 </vm>
177 <vm>
178     <amount>15</amount>
179     <cpuUtilizationTraceFile>traces / zid / 28. dat</cpuUtilizationTraceFile>
180     <memory>512</memory>
181     <scaling>3</scaling>
182 </vm>
183 <vm>
184     <amount>6</amount>
185     <cpuUtilizationTraceFile>traces / zid / 29. dat</cpuUtilizationTraceFile>
186     <memory>768</memory>
187     <scaling>3</scaling>
188 </vm>
189 <vm>
190     <amount>12</amount>
191     <cpuUtilizationTraceFile>traces / zid / 30. dat</cpuUtilizationTraceFile>
192     <memory>1024</memory>
193     <scaling>4</scaling>
194 </vm>
195 <vm>
196     <amount>5</amount>
197     <cpuUtilizationTraceFile>traces / zid / 31. dat</cpuUtilizationTraceFile>
198     <memory>768</memory>
199     <scaling>5</scaling>
200 </vm>
201 <vm>
202     <amount>4</amount>
203     <cpuUtilizationTraceFile>traces / zid / 32. dat</cpuUtilizationTraceFile>
204     <memory>1024</memory>
205     <scaling>3</scaling>
206 </vm>
207 </VMs>
208
209 <RMs>
210     <rm>
211         <amount>2</amount>
212         <cpuPerformance>2.540</cpuPerformance>
213         <cpuMHzPerCore>2600</cpuMHzPerCore>
214         <numberOfCores>32</numberOfCores>
215         <memory>49152</memory>
216         <powerStandby>10</powerStandby>
217         <powerIdle>110</powerIdle>
218         <powerPeak>446</powerPeak>
219         <specPowerPoints>4894</specPowerPoints>
220     </rm>
221     <rm>
222         <amount>4</amount>
223         <cpuPerformance>1.343</cpuPerformance>
224         <cpuMHzPerCore>2200</cpuMHzPerCore>
225         <numberOfCores>16</numberOfCores>
226         <memory>32768</memory>
227         <powerStandby>10</powerStandby>
228         <powerIdle>53</powerIdle>
229         <powerPeak>245</powerPeak>
230         <specPowerPoints>5032</specPowerPoints>
231     </rm>
232     <rm>
233         <amount>4</amount>
234         <cpuPerformance>0.959</cpuPerformance>
```

```

235     <cpuMhzPerCore>1800</cpuMhzPerCore>
236     <numberOfCores>16</numberOfCores>
237     <memory>32768</memory>
238     <powerStandby>10</powerStandby>
239     <powerIdle>57</powerIdle>
240     <powerPeak>178</powerPeak>
241     <specPowerPoints>4186</specPowerPoints>
242 </rm>
243 <rm>
244     <amount>2</amount>
245     <cpuPerformance>2.242</cpuPerformance>
246     <cpuMhzPerCore>2300</cpuMhzPerCore>
247     <numberOfCores>64</numberOfCores>
248     <memory>65536</memory>
249     <powerStandby>10</powerStandby>
250     <powerIdle>122</powerIdle>
251     <powerPeak>580</powerPeak>
252     <specPowerPoints>2968</specPowerPoints>
253 </rm>
254 <rm>
255     <amount>4</amount>
256     <cpuPerformance>1.26</cpuPerformance>
257     <cpuMhzPerCore>2200</cpuMhzPerCore>
258     <numberOfCores>16</numberOfCores>
259     <memory>49152</memory>
260     <powerStandby>10</powerStandby>
261     <powerIdle>52</powerIdle>
262     <powerPeak>240</powerPeak>
263     <specPowerPoints>4752</specPowerPoints>
264 </rm>
265 <rm>
266     <amount>4</amount>
267     <cpuPerformance>0.900</cpuPerformance>
268     <cpuMhzPerCore>2933</cpuMhzPerCore>
269     <numberOfCores>12</numberOfCores>
270     <memory>24576</memory>
271     <powerStandby>10</powerStandby>
272     <powerIdle>132</powerIdle>
273     <powerPeak>304</powerPeak>
274     <specPowerPoints>2075</specPowerPoints>
275 </rm>
276 <rm>
277     <amount>2</amount>
278     <cpuPerformance>1.696</cpuPerformance>
279     <cpuMhzPerCore>2100</cpuMhzPerCore>
280     <numberOfCores>48</numberOfCores>
281     <memory>32768</memory>
282     <powerStandby>10</powerStandby>
283     <powerIdle>468</powerIdle>
284     <powerPeak>852</powerPeak>
285     <specPowerPoints>1297</specPowerPoints>
286 </rm>
287 <rm>
288     <amount>4</amount>
289     <cpuPerformance>1.1231</cpuPerformance>
290     <cpuMhzPerCore>2500</cpuMhzPerCore>
291     <numberOfCores>32</numberOfCores>
292     <memory>32768</memory>
293     <powerStandby>10</powerStandby>
294     <powerIdle>418</powerIdle>
295     <powerPeak>792</powerPeak>
296     <specPowerPoints>927</specPowerPoints>
297 </rm>

```

Appendix B Basic Configuration for the MetaSim Simulation.

```
298
299     <cpuUtilizationWarningLevel>0.6</cpuUtilizationWarningLevel>
300     <memoryUtilizationWarningLevel>0.8</memoryUtilizationWarningLevel>
301 </RMs>
302
303     <intervalLengthSeconds>7200</intervalLengthSeconds>
304 </scenario>
305
306     <forecasting>none</forecasting>
307 <!--     <forecasting>HoWi</forecasting> -->
308 <!--     <forecasting>ARIMA</forecasting> -->
309
310 <!--     <algorithm>BFF</algorithm> -->
311     <algorithm>GA</algorithm>
312
313 <run id="0">
314     <rmIdleTimeoutSeconds>900</rmIdleTimeoutSeconds>
315     <vmConsolidationInertiaSeconds>600</vmConsolidationInertiaSeconds>
316
317     <utilizationCostFunctionMu>10</utilizationCostFunctionMu>
318     <utilizationCostFunctionAllowedResponseTime>
319         1
320     </utilizationCostFunctionAllowedResponseTime>
321
322     <energyPenalty>600</energyPenalty>
323     <migrationPenalty>1</migrationPenalty>
324     <rmUtilizationPenalty>10</rmUtilizationPenalty>
325     <bootPenalty>1</bootPenalty>
326     <shutdownPenalty>5</shutdownPenalty>
327     <variancePenalty>0</variancePenalty>
328
329     <numberOfThreads>4</numberOfThreads>
330     <numberOfGenerations>200</numberOfGenerations>
331     <maxGenerationsOfFitnessNotIncreased>10</maxGenerationsOfFitnessNotIncreased>
332     <sizeOfPopulation>800</sizeOfPopulation>
333     <crossoverRate>0.1</crossoverRate>
334     <exchangeRate>0.1</exchangeRate>
335     <migrateVmRate>0.7</migrateVmRate>
336     <consolidateRmRate>0.1</consolidateRmRate>
337     <swapRmRate>0.1</swapRmRate>
338     <swapVmRate>0.1</swapVmRate>
339     <elitism>true</elitism>
340 </run>
341 </simulation>
```

Appendix C

Patch to Introduce Stop Conditions to the KVM Live Migration.

```
1  ---
2  arch_init.c | 10 ++++++---
3  1 files changed, 9 insertions(+), 1 deletions(-)
4
5  diff --git a/arch_init.c b/arch_init.c
6  index 4486925..57fcble 100644
7  --- a/arch_init.c
8  +++ b/arch_init.c
9  @@ -89,6 +89,9 @@ const uint32_t arch_type = QEMU_ARCH;
10 #define RAM_SAVE_FLAG_EOS 0x10
11 #define RAM_SAVE_FLAG_CONTINUE 0x20
12
13 +#define MAX_MEMORY_ITERATIONS 29
14 +#define MAX_TOTAL_MEMORY_TRANSFER_FACTOR 3
15 +
16 static int is_dup_page(uint8_t *page, uint8_t ch)
17 {
18     uint32_t val = ch << 24 | ch << 16 | ch << 8 | ch;
19 @@ -107,6 +110,8 @@ static int is_dup_page(uint8_t *page, uint8_t ch)
20 static RAMBlock *last_block;
21 static ram_addr_t last_offset;
22
23 +static int numberFullMemoryIterations = 0;
24 +
25 static int ram_save_block(QEMUFile *f)
26 {
27     RAMBlock *block = last_block;
28 @@ -158,7 +163,10 @@ static int ram_save_block(QEMUFile *f)
29     offset = 0;
30     block = QLIST_NEXT(block, next);
31     if (!block)
32 +    {
33 +        numberFullMemoryIterations++;
34 +        block = QLIST_FIRST(&ram_list.blocks);
35 +    }
```

Appendix C Patch to Introduce Stop Conditions to the KVM Live Migration.

```
36         }
37
38         current_addr = block->offset + offset;
39 @@ -295,7 +303,7 @@ int ram_save_live(Monitor *mon, QEMUFile *f, int
40     stage,
41     void *opaque)
42     {
43         expected_time = ram_save_remaining() * TARGET_PAGE_SIZE / bwidth;
44 -     return (stage == 2) && (expected_time <= migrate_max_downtime());
45 +     return (stage == 2) && ((expected_time <= migrate_max_downtime() ||
46 (numberFullMemoryIterations == MAX_MEMORY_ITERATIONS) || (
47     bytes_transferred >
48 (MAX_TOTAL_MEMORY_TRANSFER_FACTOR*ram_bytes_total()))));
49     }
50     static inline void *host_from_stream_offset(QEMUFile *f,
51     ---
52 1.7.0.4
```

List of Figures

2.1	Idle vs. peak power consumption of single server systems participating in the SPECpower_ssj2008 benchmark.	12
2.2	Google server utilization as in [BH07]. The typical operating region is between 15% and 55% utilization.	15
2.3	Efficiency profile of a current server, lacking energy proportionality as in [BH07].	16
2.4	A hypothetical, more energy-proportional server as in [BH07].	16
2.5	Shifting the operating region to 40-60% would provide software-managed energy efficiency.	17
2.6	System power consumption, relative to peak, depending on the CPU utilization for almost 300 systems participating in SPECpower_ssj2008.	19
3.1	a) Without virtualization; b) With virtualization.	24
3.2	Resource partitioning using virtualization.	25
3.3	Resource aggregation using virtualization.	25
3.4	Operating System Level Virtualization.	29
3.5	Type 1 Hypervisor based virtualization.	31
3.6	The KVM hypervisor architecture.	31
4.1	Static consolidation to increase the utilization of formerly idle servers. . . .	39
4.2	(a) Static over-provisioning wastes a significant amount of resources. (b) Provisioning for peak still wastes resources, and additionally risks resource shortage in case of unpredicted peaks. (c) Under-provisioning leads to loss of revenue, as QoS/QoE metrics will suffer. (d) Dynamic provisioning tries to minimize the gap between resource demand and online capacity. Plots (a), (b) and (c) as in [AFG ⁺ 09].	42
4.3	Time series of a VM's CPU utilization operated by the University of Vienna Central IT Department.	44
4.4	VM mapping, addition of vectors (VMs) until a capacity bound would be violated. Additionally, servers are heterogeneous regarding their CPU and memory capacities, and energy consumption.	49
5.1	Dynamic provisioning for actual load to increase the energy efficiency of a computing infrastructure.	62
5.2	MetaVisor's integration into a data center infrastructure.	66

5.3	Non-linear overload cost function.	72
5.4	Decomposition of a typical CPU utilization trace into a trend, seasonal and random component.	79
5.5	Diagnostic time series plot of the same trace as in Figure 5.4.	80
5.6	Autocorrelation plot of the same trace as in Figure 5.4.	80
5.7	Autocorrelation of all VMs at lag 0.5, meaning 12 hours.	81
5.8	Autocorrelation of all VMs at lag 1, meaning 24 hours.	81
5.11	The time series of total VM CPU demand, server capacity and quota capacity within the warning level used in the simulations.	85
5.12	Total weighted costs and their inner structure for all combinations of interpolation, mapping algorithm and forecasting method.	87
5.13	Provisioning efficiency for an interval length of 3600 s and the BFF heuristic without forecasting.	88
5.14	Provisioning efficiency for an interval length of 3600 s and the BFF heuristic with Holt-Winters forecasting.	88
5.15	Provisioning efficiency for an interval length of 900 s and the BFF heuristic with Holt-Winters forecasting.	90
5.16	Influence of changing the overload penalty on BFF/GA solution quality. . .	91
5.17	Influence of changing the migration penalty on BFF/GA solution quality. .	92
5.18	Influence of changing the energy penalty on BFF/GA solution quality. . .	93
5.19	Influence of the VM consolidation inertia parameter on the total weighted costs when Holt-Winters forecasting used.	95
5.20	Influence of the VM consolidation inertia parameter on the total weighted costs when no forecasting is done.	95
5.21	Influence of the RM idle timeout parameter on the total weighted costs. . .	96
5.22	Runtime per interval of BFF with/without Holt-Winters forecasting on a low power CPU and a high-end desktop CPU.	98
5.23	Runtime per interval of GA with/without Holt-Winters forecasting on a low power CPU and a high-end desktop CPU.	98
5.24	Speedup achieved by parallelizing the genetic algorithm.	99
7.2	Influence of Live Migration on HTTP Response Time and service level during rising load for unpatched KVM.	132
7.3	Influence of Live Migration on HTTP Response Time and service level during rising load for unpatched KVM with 1 Gbit/s migration speed.	133
7.4	Influence of Live Migration on HTTP response time and service level during rising load for unpatched KVM with 1 GBit/s migration speed and strict downtime target.	134
7.5	Influence of Live Migration on HTTP Response Time and service level during rising load for patched KVM implementing Xen-like stop conditions. .	135
7.6	Influence of the rate of page dirtying on the time required to complete a live migration. Note the logarithmic scaling on the y-axis.	136

7.7	Influence of Live Migration on HTTP Response Time and service level during rising load for service level managed KVM.	137
7.8	Comparison of the service downtime development with increasing CPU utilization.	138
7.9	Influence of the rate of page dirtying on the time required to complete a live migration.	139
7.10	Comparison of the service level's development with increasing CPU utilization.	140
8.1	Vienna: Probability density of power consumption, combinations not containing the Apache VM.	149
8.2	Vienna: Probability density of power consumption, combinations containing the Apache VM.	149
8.3	Vienna, Exact Match Condition: Recognition Rates are very high for combinations not containing wp.	151
8.4	Vienna, Subset Match Condition: Recognition Rates are generally increased.	151
8.5	Vienna, Set Match Condition: Average degree of combination recognition.	152
8.6	Vienna, Per VM Recognition Rates: wp is extremely well recognizable.	152
8.7	Vienna, False Positive/Negative Recognition Rates.	153
8.8	Vienna, Exact Match Condition using a Kolmogorov-Smirnov test: Recognition Rates are very high for combinations not containing wp.	155
8.9	Vienna, Subset Match Condition using a Kolmogorov-Smirnov test: Recognition Rates are generally increased.	155
8.10	Vienna, Set Match Condition using a Kolmogorov-Smirnov test: Average degree of combination recognition.	156
8.11	Vienna, Per VM Recognition Rates using a Kolmogorov-Smirnov test for validation purposes, results are of comparable quality to Figure 8.6.	156
8.12	Vienna, using a Kolmogorov-Smirnov test: False Positive/Negative Recognition Rates.	157
8.13	Lyon, ondemand governor: Probability density of power consumption, combinations not containing the Apache VM.	157
8.14	Lyon, ondemand governor: Probability density of power consumption, combinations containing the Apache VM.	158
8.15	Lyon, ondemand governor: False Positive/Negative Recognition Rates.	158
8.16	VM recognition rates for the ondemand governor scenario.	160
8.17	VM recognition rates for the powersave governor scenario.	160
8.18	VM recognition rates for the performance governor scenario.	161
A.1	MetaVisor GUI showing an empirical test run.	181
A.2	MetaSim GUI showing a simulation test run.	182

List of Tables

2.1	Estimated worldwide energy consumption of data centers , compiled from [Koo08] and [Koo11].	10
2.2	Idle and Peak Power Consumption of typical legacy servers, compiled from [DHKC09].	11
4.1	Compilation of research findings from [BKB07] regarding workload signatures and their benefit from dynamic placement.	41
5.1	Parameters used for yielding results shown in Figure 5.12.	89
5.2	Description of platforms used in the performance evaluations.	96
6.1	The most influencing and significant variables of the final model produced by stepwise AIC.	123

Curriculum Vitae

Personal	Mag.rer.nat. Thomas Treutner Born: 15.3.1984, Krems an der Donau, Austria Nationality: Austria Status: Unmarried, no children
Affiliation	Research Group Entertainment Computing Faculty of Computer Science, University of Vienna Währinger Straße 29/3.42A, 1090 Vienna, Austria E-Mail: thomas.treutner@univie.ac.at Phone: +43 1 4277 78720
Academic Education	Ongoing Ph.D. in Computer Science, University of Vienna Anticipated graduation November 2012 Secondary School Teacher Accreditation Computer Science and Physics, University of Vienna Diploma Thesis: <i>Virtual Home Environments</i> Graduated in 2008 with distinction
Professional Experience	July 2011: Guest Researcher INRIA RESO / LIP Laboratory Ecole Normale Supérieure de Lyon, France Since February 2009: Research and Teaching Associate Research Group Entertainment Computing (Former: Department for Distributed Systems) University of Vienna
Languages	German (native) English (fluent)

Publications

T. Treutner, Virtual Home Environments, Diploma thesis, 2008

H. Hlavacs, R. Weidlich, T. Treutner, Energy Saving in Future Home Environments, 2nd Home Networking Conference at IFIP Wireless Days, Dubai, Nov. 24-27, 2008.

A. Berl, H. de Meer, H. Hlavacs and T. Treutner, Virtualization in Future Home Environments, IEEE Communications Magazine 47-12 (2009), pp.62-67.

H. Hlavacs, R. Weidlich, T. Treutner, Energy Efficient Peer-to-Peer File Sharing, accepted at Journal of Supercomputing (Springer), special issue on Energy-efficient High-Performance Parallel and Distributed Computing.

H. Hlavacs, T. Treutner, Privacy Risks for Virtual Machines in a Cloud, to appear in COST Action IC0804 proceedings 2011, ISBN : 978-2-917490-18-1 - EAN : 9782917490181

M. Niedermayer, N. Ali, H. de Meer, H. Hlavacs, T. Treutner, L. Lefevre, J.-P. Gelas and I. Koutsopoulos, Energy monitoring and its impact on individual user privacy, 4th Future Internet Cluster Workshop, 2011, Budapest.

H. Hlavacs, T. Treutner, Migration of Virtual Machines: Performance and QoS, to appear in COST Action IC0804 proceedings 2011, ISBN : 978-2-917490-18-1 - EAN : 9782917490181.

H. Hlavacs, T. Treutner, Predicting Web Service Levels During VM Live Migrations, 5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud (SVM11), 24th October, 2011, Paris, France.

H. Hlavacs, T. Treutner, J.-P. Gelas, L. Lefèvre and A.-C. Orgerie, Energy consumption side-channel attack at Virtual Machines in a Cloud, accepted at International Conference on Cloud and Green Computing (CGC 2011), Sydney, Australia, Dec. 12-14 2011.

T. Treutner, H. Hlavacs, Service Level Management for Iterative Pre-Copy Live Migration, accepted at the 8th International Conference on Network and Service Management (CNSM 2012), 22-26 October, Las Vegas, USA.

H. Hlavacs, T. Treutner, Genetic Algorithms for Energy Efficient Virtualized Data Centers, accepted at 6th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud (SVM12), Las Vegas, Oct. 26, 2012.

H. Hlavacs, R. Weidlich, T. Treutner, Load-Dependent Energy Saving in Future Home Environments, to appear in Telecommunication Systems 51-4 (Dec. 2012), Springer, 2012.

