CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Resource Management in Virtualized Data Centers Regarding Performance and Energy Aspects

Dissertation zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften

vorgelegt von

Dipl.-Inform. Marko Hoyer

18. Februar, 2011

Contents

Acknowledgement

First of all I would like to thank my supervisor Prof. Dr. Wolfgang Nebel for his support and helpful advices concerning the principals of academic work. His contacts to industrial companies further helped me to gather some practical insights into the topic of this thesis. The theoretical concepts developed in this thesis could very closely address real practical issues due to this background. In addition, I would like to thank Prof. Dr. Michael Sonnenschein for taking the time to review this document.

Much of the work presented in this thesis was supported by two of my students: Pierre Petliczew and Daniel Schlitt. Thank you both for your good work. I would also like to thank my colleagues for many constructive discussions; especially Henrik Lipskoch for helping me with mathematical background, Kiril Schröder and Daniel Schlitt for discussions about the concepts, and Domenik Helms and Gunnar Schomaker for some tips concerning the formal representation of some optimization problems. Additionally, special thanks to the NOWIS company for providing me very good evaluation data to assess my concepts.

A large portion of my work was further supported by the OFFIS Institute for Information Technology. I did most of the research within the research project "Energy Efficiency in Data Centers". Finally, an internal scholarship helped me to finish my work and to write down this thesis.

And last but not least I want to thank my family for their support. Especially the last two month of my work were hard for me for several reasons. Thank you for your help.

***

Ein spezieller Dank soll an dieser Stelle an meine Familie für ihre mentale Unterstützung gehen. Die letzten zwei Monate dieser Arbeit waren aus verschiedenen Gründen nicht einfach für mich. Vielen Dank für Eure Hilfe.

Abstract

Todays data centers take an ever growing share of the energy consumption of Germany's Information and Communication Technology (ICT). The energy efficiency of data center components has been continuously increased over the last few years. But the energy consumption is still expected to grow further. The demand for IT services increases faster than energy consumption can be reduced by the improvements.

A completely different approach is followed in this thesis to counteract this development. It is tried to reduce the number of active IT components (servers) needed to deploy a set of IT services. Thus, the part of energy consumed by IT components as well as the energy consumed by data center infrastructure can be reduced. The respective approach first tries to minimize the number of servers required at all. In a second step, it is tried to take advantage of the varying resource demand of the services. Services are consolidated to a few servers in times of low overall demand. Unused servers are switched off to save energy.

This approach belongs to the class of resource management approaches. It improves known ones by taking the resource demand behavior of the IT services into account in a special way. Hence, an important part of this work is to properly model this behavior. In addition, the new approach supports trading off resources against service performance. Clients can define a wanted performance behavior using a new kind of specification. Finally, the algorithms that assign the services to servers form another challenge that is addressed within this thesis. One of them initially distributes the services to a minimal set of servers. A second one redistributes them at runtime according to the current resource demand. The performance goals must not be violated.

The developed approach was evaluated based on simulations using the demand behavior of services observed in a real data center. The initial distribution of services to servers already achieved between 25% and 42% of server savings compared to a widely used approach. These savings can reduce the energy consumption by 12.5% up to 42% depending on the data center. The consumption can be further reduced by additional 20%, if services are dynamically redistributed at runtime. Violations of the defined performance goals can basically occur, when the demand behavior is taken into account for resource management decisions. But they actually occurred during the simulations in less than 0.2% of the simulated time.

Zusammenfassung

Rechenzentren zählen mit einem Anteil von etwa 17% zu den Großverbrauchern der heutigen Informations- und Kommunikationstechnologie. Verschiedene Maßnahmen haben die Energieeffizienz der Rechenzentrumskomponenten bereits deutlich erhöht. Dennoch wird ein weiterer Anstieg des Energieverbrauchs prognostiziert. Der Bedarf an IT-Diensten steigt offensichtlich stärker, als die Weiterentwicklung der Komponenten deren Verbrauch reduzieren kann.

Um dieser Entwicklung entgegen zu wirken verfolgt diese Arbeit einen anderen Ansatz. Es wird versucht, die für eine bestimmte Menge an IT-Diensten benötigte Anzahl aktivier IT-Komponenten (Server) zu reduzieren. Auf diese Weise lässt sich der Verbrauchsanteil der IT-Komponenten als auch der zum Betrieb nötigen Infrastruktur senken. Ein entsprechender Ansatz versucht hierbei zunächst die überhaupt benötigte Anzahl an Servern zu minimieren. In einem zweiten Schritt wird variierender Ressourcenbedarf der Dienste genutzt, um in Zeiten geringer Anfragelast die Dienste auf weniger Server zu konsolidieren. Ungenutzte Server werden abgeschaltet um Energie zu sparen.

Dieser Ansatz fällt daher in den Bereich des Ressourcenmanagements. Das Einsparpotential ergibt sich gegenüber bisherigen Ansätzen durch die gezielte Nutzung des Bedarfsverhaltens der Dienste. Dessen Modellierung ist daher wichtiger Kernbestandteil der Arbeit. Auch die gezielte Abwägung zwischen angebotener Ressourcenkapazität und der Performanz der Dienste wurde verfolgt. Nutzer können hierzu das gewünschte Performanceverhaltens der Dienste geeignet spezifizieren. Schlussendlich bilden die Algorithmen, welche die Zuordnung der Dienste zu den Servern vornehmen, eine weitere Herausforderung, der sich diese Arbeit stellt. Ein erster Algorithmus bestimmt hierbei eine initiale Verteilung, während ein zweiter die Umsortierung zur Laufzeit vornimmt. Die Performanzziele dürfen hierdurch nicht verletzt werden.

Das entwickelte Konzept wurde in Simulationen mit dem von echten Diensten beobachteten Bedarfsverhalten evaluiert. Bereits die initiale Verteilung der Dienste konnte zwischen 25% und 42% an Servereinsparungen erzielen, die je nach Rechenzentrum zu Energieeinsparungen von zwischen 12.5% und 42% führen können. Die zusätzlich dynamische Umsortierung der Dienste zur Laufzeit konnte gegenüber der statischen Verteilung weitere 20% an Energieeinsparungen erreichen. Bei der Berücksichtigung des Bedarfsverhaltens kann eine Verletzung der vorgegeben Performanzziele grundsätzlich nicht ausgeschlossen werden. Diese traten aber nur in weniger als 0.2% der Simulationszeit tatsächlich auf.

1 Introduction

The energy consumption of the information and communication technology (ICT) continuously increases year by year. Around 55TWh of electric energy were consumed by ICT devices in Germany in 2007, which corresponds to about 10% of the country's overall power consumption[112]. Only 38 TWh electric power were generated by wind power plants in the same year[68]. Power generation only for ICT causes more carbon dioxide emission as the whole German aviation produces[99] regarding the overall energy mix in Germany. This part of energy consumption is expected to grow to more than 66 TWh in the year 2020[112].

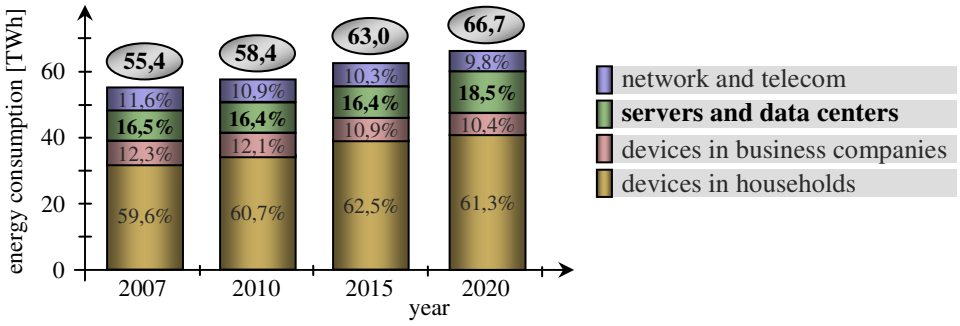


Figure 1.1: Estimated energy consumption caused by ICT consumers in Germany[112]

Consumer electronics - first and foremost the entertainment electronics - will take the biggest portion of the ICT's energy consumption in the next few years, as illustrated in Figure 1.1. But one can also note that especially data centers and the Internet will take an ever growing share due to an exponentially increasing demand.

Different reasons motivate to address the high power consumption in data centers already today. Carbon dioxide emitted while electric power is generated is thought to cause ecological problems. Reducing the energy consumption may help to solve this issue. Additionally, data center operators are motivated by economical aspects. Energy cost form a significant part of the overall costs in data centers[115]. In addition, simple technical reasons need to be addressed. Some modern data centers would have the capacity to host additional servers but they do not get the power from their local supplier to run them.

The energy efficiency in data centers has been continuously increased over the last few years.

The improvements mainly concerned infrastructure components needed to run the servers. As a consequence, the power usage efficiency value (PUE) that relates IT power to overall power consumption got closer and closer to its theoretical limit of 1.0. Earlier data centers typically had a PUE of far above 2.5. The best of today's data centers can achieve a PUE down to 1.25[40]. This means that 1.25 kW power are needed to run 1 kW server hardware.

The energy efficiency of server hardware continuously increases as well. Each new CPU generation has a higher ratio between computing power and energy consumption than the generation before. Power management techniques such as DVFS[1] or clock gating adapt the power consumption of CPUs to their utilization to save energy in times of low workload.

But the energy consumption of data centers is expected to increase further despite of these improvements[112]. The demand for IT services obviously increases faster than the improvements reduce the energy consumption of IT components and the data center infrastructure.

The work described in this thesis does not directly address the energy efficiency of individual IT or infrastructure components. Instead, a concept will be introduced for more efficiently using them. The aim is to reduce the number of active servers that are required to deploy a given set of IT service. It is expected that the IT power consumption in a data center as well as power consumed by the infrastructure can be further reduced this way.

Determining the hardware resources needed to run a set of IT services is known as resource or capacity management. Mainly the number and type of required servers and the assignments of the services to them need to be determined. It will be distinguished between static and dynamic resource management within this thesis. Static resource management determines an assignment of services on servers that is constant at runtime. Dynamic resource management, in contrast, can reassign services at runtime. It will be detailed some more in the following how both types of resource management can reduce the power consumption in data centers by more efficiently using active servers.

1.1 Static Resource Management

Static resource management tries to minimize the number of servers required at all to deploy a given set of IT services. Common modern server hardware typically provides far more computing power than actually required by many services deployed in typical data centers. Hence, deploying each service on an individual server would waste much resources and energy.

As a consequence, virtualization came back up again in the last view years. A set of different services can be deployed on one powerful server using this technique. The server's resources are shared between the services, which reduces the number of servers required. Modern virtualization environments such as *VMware ESX Server*, *Citrix Xen Server*, or *Microsoft Hyper-V*

[1] dynamic voltage and frequency scaling

allow assigning parts of the resource capacity to each service in a fine grained way. Resource capacity can be exactly adjusted to the demand of a service.

The main challenges of static resource management are determining the resources required by services and finding an appropriate assignment of them to servers. Providing less resource capacity than actually required can slow down or completely break down a service depending on the resource type (e.g. CPU time or RAM). If more resources are reserved for a service than actually required, resources and hence energy are wasted.

Static resource management must take care of maximally expected workload while determining resource capacity required by a service, since services cannot be reassigned at runtime. Such maximum is typically part of an contract between a Service Provider and the clients, which is called Service Level Agreement (SLA). A certain performance goal (such as response time or throughput) must be guaranteed by the Service Provider, if this maximum is not exceeded by the clients. Such performance goals are expressed by Service Level Objectives (SLOs). Common static resource management approaches perform benchmarks to find the amount of resources maximally required by a service. Services are then distributed to servers according to these estimates. A certain amount of reserve capacity is included to take care of unexpected peeks.

But applying this pessimistic approach often results in much unused resource capacity. Many services require their maximal resource capacity not very often. Different services hardly require their maximum all at the same time. Regarding these facts, resources can be used far more efficiently. Further resource savings can be achieved, if clients define not fixed but more soft performance goals in their SLA with the service provider. Resource management can apply resource performance trade-offs, if in definable seldom cases a slower service response is accepted.

In a first part of this thesis, an approach for static resource management is presented that supports such ideas. This new approach statistically evaluates the demand behavior of the services and hence belongs to the class of statistic static resource management approaches. In contrast to previous works, a new kind of SLO specification is supported that allows performing resource power trade-offs in a more flexible way, which increases the achievable resource savings. In addition, correlations between the workload behavior of different services are considered. All known approaches in contrast expect uncorrelated workload, which is not given in most data centers. Hence, such assumption can lead to violations of SLOs in worst case.

1.2 Dynamic Resource Management

The workload of most IT services is not constant but varying over time. Services have hardly something to do at night while they are heavily used by day in many cases. Sometimes, the

maximum occurs in the evening or at the weekend. In any case, such variations cause varying resource demand of the services. Hence, any static resource management approach will waste resources in times of low workload.

Power management techniques can reduce the power consumption of CPUs in these times as mentioned before. But different other components such as memory and the main board limit the overall savings that are achievable by these techniques. The reason is that their consumption is independent from the utilization[21, 13]. As a result, today's servers consume still between 65% and 80% of their maximal power even when they have nothing to do. Really energy efficient ones can reduce their consumption down to 50%[13].

Quite more energy savings can be achieved when servers are switched off or put into a suspend state such as STR (Safe to RAM). But no services can run on the servers in this state. Hence, all services must be moved away to other servers first before a low power state is entered. Common server virtualization environments support moving services between different servers without the need to stop them. The move is nearly transparent for the service. Using this so called live migration technique, services can be consolidated to only a view servers in times of low workload. Unused servers can be switched off to save energy.

Such an approach will be called dynamic resource management within this thesis. An important challenge of dynamic resource management is to reactivate servers right in time when they are needed. Servers need time for reactivation as well as it takes time to move a service between two servers. Such delays must be considered by dynamic resource management. It must be ensured that servers have been reactivated and services have been redistributed right before an upcoming resource shortage actually occurs. For this, the underlying algorithm needs to know the resource demand behavior of the services in the future.

Known approaches typically use trend based forecasting methods. Capacity buffers equal out the delay of a control decision. But this approach can lead to strong resource shortages depending on how fast the resource demand changes and on the size of the buffer. Furthermore, such resource shortages can last for a long time until the algorithm resolves them. Technical restrictions of the virtualization environment can lead to deadlocks in some cases. In those cases, the approach cannot resolve the resource shortage at all. SLO violations are the consequence.

A new approach for dynamic resource management will be presented in the second part of this thesis. This approach overestimates the resource demand of the services expected in the future. Such forecasts base on periodic workload behavior observed in the past. A scheduling algorithm ensures that at each time an upcoming resource shortage can be resolved right in time as long as the actual resource demand does not deviate from the expected one.

1.3 Contributions of this Work

The main outcome of this thesis is a holistic concept for static and dynamic resource management in virtualized data centers. This concept supports the ideas shortly motivated in the previous two sections. The main weaknesses of known approaches are considered by the new concept as well. The following tasks form the main contributions of this work:

- **A holistic concept for static and dynamic resource management was developed that is described in Chapter 3.**
  - A general approach was worked out.
  - Technical parameters, limits, and basic prerequisites determined by data center operation were analyzed.
  - Challenges and problems to be addressed were extracted and formalized based on this information.
- **The part of this concept that realizes static resource management was worked out. It is presented in Chapter 5.**
  - A new kind of SLO specification was developed that supports resource performance trade-offs in a flexible way.
  - A modeling approach was developed that allows estimating the resources maximally required by the services in the future based on demand behavior observed in the past.
  - A scheduling algorithm was developed that distributes services to a minimal set of required servers with respect to the estimated resource demand.
- **The part of this concept that realizes dynamic resource management was worked out. It is presented in Chapter 6.**
  - The SLO specification was extended in a way that negative impact of forecasting errors on the service performance can be limited.
  - The models were extended in a way that the time dependent resource demand expected in the future can be forecasted.
  - A scheduling algorithm was developed that realizes dynamic resource management based on the forecasts. This algorithm can guarantee that no resource shortages will occur, if the forecasts are correct.
  - An extension of the models and the scheduling algorithm was developed that adapts changed demand behavior at runtime.

- **The developed concepts, models, and optimization algorithms were evaluated in a simulation environment. The results are presented in Chapter 7.**
  - Energy savings and resource savings were compared to known approaches.
  - The amount of occurred resource shortages that could lead to SLO violations were determined.
  - The impact of different relevant parameters on the savings and on the amount of SLO violations was analyzed.

1.4 Document Overview

The content of this thesis is structured into eight chapters. Resource management has been motivated in this first one as a way to significantly reduce energy consumption in today's data centers. The aim is to more efficiently use active server hardware. The main contributions of this thesis have been presented as well.

Works related to energy efficiency in data centers is discussed in the following chapter. Resource management will be pointed out as one of the important research areas in this field. Known approaches that are related to the ones presented in this thesis are shortly presented. They will be discussed more detailed later on in the chapters, in which the new approaches are presented.

The holistic concept for static and dynamic resource management is presented in Chapter 3. Challenges to be addressed will be extracted and transfered into formal optimization problems. These challenges are addressed in the Chapters 4 to 6. The chapters 4 and 5 deal with static resource management. The dynamic resource management is addressed in Chapter 6. All of these chapters have nearly the same structure. The SLO specification, the modeling approach, and finally the scheduling approach that distributes services to servers are subsequently worked out.

Finally, the whole resource management concept is evaluated in Chapter 7. A short summary and conclusion closes this thesis in Chapter 8.

2 Context and Related Work

Energy efficiency is currently addressed in a data center mainly at three different layers. Different improvements try to reduce the energy consumption of the IT components (servers and network components) at different points. A second group of approaches addresses the additional infrastructure needed to run the IT components. And a third group aims to improve the energy efficiency at software level. The state of the art in each of these groups will be shortly presented in the following.

2.1 IT Components

Hardware manufacturers such as Intel, AMD, or IBM continuously increase the energy efficiency of electronic devices such as memory or the CPU. The transistor technologies and hardware design processes are improved with respect to performance and energy consumption. Power management techniques reduce the power consumption, when devices or parts of devices are not used. An overview of the state of the art in this field is presented in [52, 63].

Power consumed by disks has been addressed by several works. Mainly different speeds and low power modes of disks have been exploited to save energy[20, 46, 47]. Caches have been taken into account to increase the idle periods[133, 132]. Disks can remain switched off for a longer time this way. Intelligent power management strategies that could be also used to control the power state of a disk have been presented for instance in [57, 16].

Modern blade server systems from IBM, HP, or Dell for instance contain different additional power management techniques. They can control the speed of the fans in the enclosure depending on the number of active servers and on their workload. Power supply units can be switched on and off according to the current power demand, which increases their efficiency. Servers can be switched off, can be put into a suspend state, and can be reactivated using a software interface.

2.2 Data Center Infrastructure

Data center infrastructure is optimized mainly at two different points. The first one concerns all components that are involved in power supply. Critical devices are the uninterruptible power

supply (UPS) and the power distribution units. Manufacturers of UPS devices continuously try to increase the energy efficiency of their hardware. This is important, since nearly most of the power consumed in a data center passes the UPS. Hence, only a view percent of overhead power consumed by the UPS will already lead to significant energy wasting. Efficiency goals for new UPS devices have been defined by the European Commission in conjunction with the manufacturers in a Code of Conduct [39].

Besides the UPS, different additional ways to increase the energy efficiency of the power supply chain are focused as well. Replacing AC by DC power in a data center is a heavily discussed field [93]. A further trend seems to be strongly followed at the moment. It is tried to increase the scalability of the devices [92]. They are designed in a way that they can be continuously extended by new ones with increasing power demand. This prevents over provisioning, which increases the energy efficiency of the currently running ones.

Several improvements increased the energy efficiency of the cooling infrastructure as well. Classical room based cooling has been more and more replaced by rack or row based cooling strategies in large data centers[37]. Modern powerful hardware and the server virtualization concept concentrate a huge amount of energy in a comparable small area. Special cooling techniques such as water cooling for instance address this issue[84, 64]. It is further tried to reuse heat energy for different purposes. Finally, free cooling is a popular technique to reduce the energy consumption as well. Cool air from outside the data center is used to support the chillers. A good overview of such modern cooling concepts is presented in [48].

2.3 Operating Systems, IT Services, and Software

Some of the power reduction techniques implemented in hardware must be controlled by a so called power management controller from operating system level or even above. Some of them that are used in today's servers or data centers will be presented in the following. A second common way to optimize the energy consumption at software level is resource management. The aim is to minimize the hardware resources needed to deploy a given set of services, as already described in the introduction section. Known approaches for static as well as dynamic resource management will presented later on in this section as well.

2.3.1 Power Management

The controller for DVFS is typically implemented in the operating system. At this level, the controller knows the amount of computing power needed at a time. The power state of individual devices of servers are typically controlled by the operating system as well for the same reason. A controller, in contrast, that switches off a complete server needs a broader view. Most IT services must be available all the time. But a service that is deployed on a

suspended server cannot process requests. Hence, it needs to be moved to another server first, which requires to regard a set of servers to realize such a controller. The dynamic part of the resource management approach presented in this thesis can be regarded as such a server comprehensive controller.

2.3.2 Resource Management

Resource management approaches strongly differ depending on the kind of IT services managed. It can be distinguished between request based and job based IT services to cluster known approaches. The first category can be further subdivided into small-scale and large-scale request based IT service. The main characteristics will be shortly presented in the following. Known approaches will be presented for each of these categories as well. The approach presented in this thesis targets mainly small-scale request based IT services.

Job Based IT Services

A significant part of today's IT services is job based. One job typically consists of different tasks that are processed partly sequentially and partly in parallel. Billings, for instance, are often created using batch jobs. They process data base requests, select some data from a file server, and concatenate and print the results. Fundamental research in different areas often starts simulations in batch jobs that can occupy whole server clusters for a long time.

The challenge for resource management approaches that focuses on job based IT services is mainly to assign the jobs to servers when they arrive. Possible priorities and deadlines must be taken into account.

Such resource management approaches belong to the field of high performance computing (HPC) that has been widely addressed by past research. Most works focused on performance only. In the last view years, classical HPC frameworks such as MPI[78] have been combined with virtualization techniques [80, 41, 59]. Such trend enabled the way to additionally take energy aspects into account[119]. In general, all HPC approaches strongly differ from the work presented in thesis, since a completely different method is required to deal with request based IT services.

Large-Scale Request Based IT Services

Request based IT services are characterized through a dialog between the service and the client. The client sends a request to the service and expects a comparable fast answer. Normally, further requests are send based on this answer. Furthermore, many clients send requests in parallel to the service. And finally, more than one IT service can be involved, while a request is processed. Web services and data bases are typical examples.

Complex IT services that have to deal with a large number of requests at the same time often require far more resources than provided by one server. They are called large-scale request based IT services within this thesis. Several instances of the same service are typically deployed on different servers, which are called nodes in this context. Requests can be dynamically routed to these nodes to evenly distribute the workload between them. This concept is widely known as workload dispatching.

The major challenge for a resource management approach is to route the requests. Nodes can be switched off in times of low workload to save energy. Simple algorithms can realize such resource management, since workload and hence resource demand can be shifted between nodes without any significant delay. Such an algorithm was presented in [25] for instance.

These algorithms form the base for many additional works. Two market model based approaches were presented in [24, 73], in which users bid for resources. Granted resources are negotiated between all users according to their respective SLAs. An optimization algorithm aims at minimal overall costs in terms of energy and money for the data center. Another idea has been followed by [73, 96, 103]. They suggest adapting the varying utilization of a service by measuring the current Quality of Service (QoS) level. Requests are assigned to the nodes dynamically using a control loop. And finally, different approaches take thermal conditions in a data center into account to reduce the energy consumption of the whole data center[88, 77].

Most of these algorithms cannot be directly applied to small-scale request based IT services. Such services are typically only deployed once in a data center. Hence, the load dispatching approach does not work for them. The dynamic part of the concept presented in this thesis can be regarded as a basis to support such ideas for small-scale request based IT services. Only energy and performance will be regarded as optimization criteria within this thesis. But the underlying optimization algorithms can be easily extended to consider additional parameters as well.

Small-Scale Request Based IT Services

Only one or a few small-scale request based IT services were deployed on one server in the past due to maintenance reasons and to achieve performance and data isolation. Such services typically need hardly the whole resource capacity of one modern server. Server virtualization that allows different services to run on the same server was the answer to that issue.

The challenges of resource management approaches for small-scale request based IT services in virtualization based data centers have already been presented in the introduction section. Hence, only the known approaches will be shortly presented here.

Different trace based tools already exist for static resource management [55, 122, 60]. They measure the resource demand of the services over time. Services are then assigned to servers based on this information. Measured resource demand is multiplied by a certain factor to take

care of virtualization overhead and noise. This kind of modeling can either lead to significant over provisioning or to resource shortages in some cases depending on the behavior of the services [128]. Furthermore, neither SLOs nor resource performance trade-offs are supported by these tools. Many research groups followed the idea of trace based resource management as well [44, 43, 95]. The static part of the concept presented in this thesis is partly inspired by the idea of trace based modeling. But the underlying resource demand models support SLOs and resource performance trade-offs in contrast to known works.

Another class of approaches uses statistic models to describe the resource demand behavior of the services [102, 117, 58, 17, 45]. SLOs and resource performance trade-offs are supported. These approaches are similar to the one presented in this thesis. In contrast to them, non stationarity of the resource demand behavior as well as correlations between the resource demand of different services are considered by the new one.

Different dynamic resource management approaches have been presented for small-scale request based IT services as well. Some of them leave open how to change from an existing to a new distribution of services to servers [17, 90, 102]. Others address this issue by incrementally redistributing services [53, 12, 66, 129, 118, 15]. The dynamic part of the concept presented in this thesis focuses on an incremental redistribution as well. But in contrast to known approaches, the new one can ensure that any upcoming resource shortage can be resolved right in time, if the forecasted resource demand meets the actual one.

3 Problem Statement

The main goal of this thesis is to develop a concept for static and dynamic resource management that optimizes the energy consumption in virtualized data centers by utilizing active server hardware maximally.

First, a given set of services must be assigned to servers in a way that minimizes the overall number of required servers. Service Level Agreements between clients of the service and the Service Provider must be taken into account. This minimizes required hardware resources and hence energy consumption. In a second step, an extended algorithm tries to take advantage of varying workload that causes varying resource demand of the services. The services should be consolidated to less servers in times of low workload. Unused servers are switched off to save energy. This further reduces the energy consumption by adapting the active hardware resources to the actual workload anytime in the best possible way.

Such a resource management concept will be worked out within this chapter. Some technical background will be given in a first section that helps to understand the conceptual decisions made. The concept itself is presented in a next section. It divides into different phases. For each of them individual challenges and optimization problems will be worked out. Relevant components of the data center as well as additional components used for the resource management are presented and related to each other in a third section. This analysis provides parameters and constraints that additionally have to be taken into account by the resource management concept. Finally, the optimization problems, the system, and the derived parameters and constraints will be mathematically described to derive formal problem statements from them.

3.1 Technical Background

Different technical circumstances given in common data centers must be taken into account by the resource management concept. They will be presented one by one within this section.

3.1.1 Service Level Agreements

The deployment of IT services typically involves different parties such as Service Providers and clients. So called Service Level Agreements (SLAs) must be negotiated between them to

define rights and obligations. Different specifications, protocols, and specification languages [87, 62, 116, 106] exist for defining these SLAs.

A typical scenario taken from [106] is presented in Figure 3.1 to illustrate how Quality of Service (QoS) can be ensured by using SLAs.

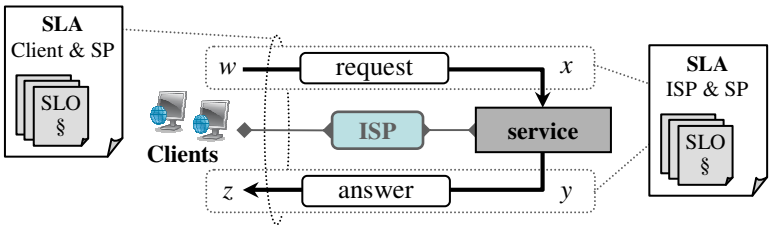


Figure 3.1: A typical scenario of service hosting in data centers and possible SLAs between different parties involved.

Three parties are involved in this scenario. Clients are using a service provided by a Service Provider (SP). Both are typically connected via a network operated by an Internet Service Provider (ISP).

The clients send requests (w) through the network to the service in a typical use case. The service processes these requests and sends the answers (y) back to the clients. A client has two expectations. First, the answer should be present within a predefined time period. Second, the service must behave as described by the SP before.

Clients and the SP can ensure these expectations by negotiating SLAs. Attributes such as the required response time, performance, or throughput are defined by throughput conditions. They are expressed in so called Service Level Objectives (SLOs). The service behavior is captured by reliability constraints also expressed in SLOs. Limited resources of the SP restrict its ability to guarantee SLOs. These limits concern the usage of the service mainly referred to a limit of requests in a time interval. The respective SLO can be violated, if the client exceeds the defined limit. These limits are part of many existing SLOs [106, 7] already today.

The reliable work of the service does not only depend on the SP as can be seen in Figure 3.1. Especially the response time, performance, and throughput are influenced by the network as well. Thus, a second SLA exists between the SP and the ISP ensuring that the time between w and x and between y and z does not exceed a predefined limit.

SLAs can not only be used as a contract between different parties but also as input for tools that autonomously manage the hardware resources of the Service Provider [87]. This objective is pursued within this thesis. It will be detailed in Section 3.3.4 how the resource management concept presented in this thesis takes care of SLOs concerning throughput and reliability conditions.

3.1.2 Server Virtualization

The server virtualization technique serves as a basis for the resource management presented in this thesis. This concept allows different services to share the same hardware server[107]. An abstraction layer separates the service from the hardware at a certain point. An interface is integrated to control the parallel access of different services to the shared hardware resources.

Each service is implemented in an individual isolated environment. The services cannot directly interact with each other this way, which is import for security reasons. They do not even know that they share hardware resources with other services in most cases. An external software component schedules the access of all services to the hardware resources.

The abstraction layer separates the services from the hardware resources at different layers depending on the virtualization technique. One can distinguish between operation system level and server level virtualization. Operation system virtualization provides an individual instance of a virtual operating system to each service in a so called container. This virtual operating system looks like a real one that directly runs on hardware from the service's perspective. Actually, one real operating system schedules the parallel access of different containers with different virtual operating systems.

Server virtualization, in contrast, does not only simulate operating systems but complete servers. Any kind of compatible operation system and the respective service can be installed on each virtual server. Different services with different individual operating systems can share the same hardware server this way. The virtual server looks like a real one from the perspective of the installed operating system[1]. An underlying scheduler regulates the parallel accesses of different virtual servers to the real one.

Server virtualization is focused within this thesis because many services in common data centers require individual operating systems for security and maintenance reasons. A virtual servers is called Virtual Machine (VM) within this thesis. The underlying scheduler is typically called Virtual Machine Monitor (VMM). The whole hardware and software system that realizes the server virtualization is called virtualization environment.

Common virtualization environments for server virtualization can be divided further into three different classes. An overview of each of the underlying concepts is illustrated in Figure 3.2 and will be shortly described in the following. Further information is provided in [107, 134] and other related literature.

Full Virtualization

Any access of the guest operating system (guest OS) installed in a VM to shared hardware resources must pass the VMM, if full virtualization is used (cf. Figure 3.2 a)). Direct access

[1] This is not completely true, if paravirtualization is used. At least the kernel of the guest operating system is aware of the virtual server in this case.

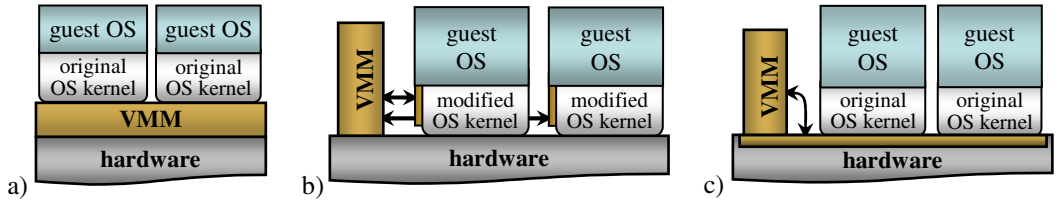


Figure 3.2: Three different concepts of server virtualization. a) full virtualization b) paravirtualization c) hardware assisted virtualization

is impossible because the VMM can not control when the kernel of a guest OS accesses the hardware resources. Hence, the VMM has to simulate individual hardware components such as processors, a system bios, the ram, and the network cards to gain control. The VMM can decide when the virtual components are actually accessing the physical ones this way. Special drivers installed in the guest OS enable the communication with the virtual hardware components.

The VMM itself can be realized in two different ways. First, a common operation system (Windows, Linux, or Sun OS) is installed on the real hardware server. The VMM is a program that runs in this operating system besides other programs. *VMware Server* [125] and *Microsoft Virtual PC* [76] are popular examples. The *VMware ESX Server* [124] forms a more efficient full virtualization environment. This VMM is an operating system itself. Hence, more direct access to hardware resources is possible. The kernel schedules parallel accesses of different simulated hardware components to the physical ones.

Paravirtualization

The guest OSs in paravirtualization environments can directly access real hardware components in contrast to full virtualization (cf. Figure 3.2 b)). Guest OSs must communicate with the VMM before they access any hardware. This allows the VMM[2] to schedule the access of different VMs to the same hardware component. Therefore, they typically must implement an interface of the VMM. Hence, only operating systems that support the underlying virtualization environment can be installed on paravirtualizated VMs.

The efficient way for the VMM to schedule parallel hardware requests is an advantage compared to full virtualization. It is not required to analyze the accesses of guest OSs to the virtual hardware components to find out whether or not they can be granted at the moment. The guest OS simply requests an access to a certain hardware component and waits until it gets a grant from the VMM.

[2]The VMM is often called hypervisor in paravirtualization environments.

Examples for server virtualization environments that are based on paravirtualization are *Citrix Xen Server* [29] and *IBM z/VM* [61].

Hardware Assisted Virtualization

The need of modified operating system kernels of the guest OSs is a big disadvantage of paravirtualization. Especially closed source operating systems can not be used in most of the paravirtualization environments. *Microsoft Windows XP*, for instance, can not run as an paravirtualized guest OS in a classical *XenServer* environment without any extensions.

Hardware assisted virtualization(cf. Figure 3.2 c)) overcomes this drawback by using special features of today's CPU families (VT-x/VT-i(in Intel cores), Pacifica (in AMD cores)). These features allow the VMM to schedule hardware accesses of the guest OS at hardware level. Modifications of the guest OS are no longer required.

An extension called HVM enables hardware assisted virtualization for the paravirtualization environment *Citrix Xen Server* [29]. Microsoft integrated virtualization support into the server operating system *Windows Server 2008*. The underlying virtualization environment *Hyper-V* [75] is also based on hardware assisted server virtualization.

3.1.3 Server Virtualization and Live Migration

The idea of live migration is to move VMs between two hardware servers without stopping them. The abstraction layer between virtual and real hardware provides a basis for this technique. Most common virtualization environments for server virtualization such as *VMware ESX Server* [124], *Citrix Xen Server* [29], and *Microsoft Hyper-V* [75] already support live migration of VMs.

It is required to move the context of the VM concerning memory (RAM), CPU state, and the state of additional hardware devices used by the VM (e.g. the network adapter) to migrate a VM between two servers at runtime. Common virtualization environments assume that hard disk content is stored in centralized storages that are accessed via network to support live migration. Hence, it is not needed to copy this content as well.

Three different strategies for live migration exist. A pure stop-and-copy strategy [98, 70] stops the VM, copies the content through the network to the destination server, and starts the VM at the new server. The problem of this strategy is that the service remains suspended for the whole time needed to copy the context.

A second strategy [131] addresses this issue. Only the CPU and hardware context are copied first. The VM is then started on the new server. The memory content is copied while the VM is already running on the new server. A major disadvantage of this approach is the unpredictable performance loss when requested memory pages are not yet present and must be loaded through the network.

A third strategy [30] copies the memory content of the VM to the new server while the VM remains on the current one. Memory pages that are changed by the VM after they have already been copied have to be copied again. The VM is stopped when almost the whole content of the memory has been transferred to the destination server. The changed pages and the hardware context are copied and the VM is started on its new server. The time a VM is actually suspended can be far below 100ms [30] this way.

This strategy is supported by the dynamic part of the resource management concept presented in this thesis since most common virtualization environments use it to realize live migration.

3.1.4 Dealing with Shared Resources in Virtualized Data Centers

This section provides some background on how virtualization environments shares the resource capacity of servers between different VMs. This knowledge is an essential prerequisite to understand how shared resources are modeled in the resource management concept.

Mainly three types of resources are shared between VMs. They will be discussed one by one in the following.

CPU Time

Modern servers contain several CPUs. Each modern CPU has typically more than one core. All cores of all CPUs form the CPU capacity of a server. Typically, one core of one CPU is reserved for the VMM itself so that the overall CPU capacity provided to VMs is reduced by one core.

VMs are consuming CPU time. CPU time is a measure that indicates how long a certain VM uses or requires one core of one CPU. In general, CPU time is not used as an absolute value but related to a fixed measuring interval and expressed in percent in most cases[3].

VMs can provide one or more virtual CPUs to their guest operating system. The number of virtual CPUs of all VMs that are running on the same server can exceed the number of real hardware cores present. In this case, different virtual cores are mapped onto the same hardware core. The CPU time is scheduled between them. One virtual core can provide the whole computing power of one real hardware core to its guest OS at a maximum.

Common virtualization environments allow specifying upper and lower limits to control the distribution of CPU time to virtual CPUs that are mapped on the same core. A lower limit guarantees minimal CPU time to the respective virtual CPU. The maximal CPU time a virtual CPU can provide to its guest OS can be defined by an upper limit. Setting such limits helps to isolate different VMs from each other. If a service exceeds the expected CPU time demand

[3]Sometimes MHz is used as unit for CPU time. This unit can be regarded as CPU cycles per second. Knowing the duration of one CPU cycle one can translate this unit into percent as well.

because of failures or an attack, the performance of none of the other services that are running on the same server will be influenced.

The first virtualization environments supported only an fixed assignment of virtual CPUs to real cores that must be manually set by the administrator. Modern ones such as *VMware ESX Server* and *Citrix XenServer* (when the *Credit-Based CPU-Scheduler*[108] is used) integrate load balancing schedulers that dynamically reassign the virtual CPUs according to current workload conditions. A big advantage of this dynamic assignment is that all cores of all CPUs can be regarded as one big pool of CPU time capacity. All virtual CPUs can individually take CPU time out of this pool as long as the sum of all does not exceed the overall capacity. If for instance three virtual CPUs require 40%, 80%, and 70% CPU time, this capacity can be provided by two real cores. The scheduler continuously reassigns the three virtual CPUs to the two real cores so that the CPU time provided to the virtual CPUs fits on average.

Of course, the accuracy the CPU time is scheduled to the virtual CPUs depends on the time interval that is regarded. Conventional schedulers have rescheduling periods of the virtual CPUs of below 50ms (*XenServer:* $30ms$). Hence, actually used CPU time should not significantly deviate from the specified values any more already after a few multiples of this period. Running applications on virtual servers that require provisioning in smaller times scales is a big challenge at all because any kind of resource scheduling must be able to deal with these small deadlines. This issue will be not addressed any deeper in this thesis. It must be mainly addressed by the virtualization environment and the underlying schedulers.

RAM

Memory capacity is allocated by the VM in different ways depending on the virtualization environment used. *Citrix Xen Server*, for instance, allocates the complete memory that is assigned to a VM directly when the VM is started [109]. The amount of assigned memory can be changed by the VMM at runtime using a technique called ballooning. But the VMM is not aware of the amount of memory that is actually used by the guest OS in a VM. Hence, automatically adjusting provided memory capacity to the demand of a VM is not possible without any additional communication to the operating system. The resource management must allocate a fixed amount of memory capacity that satisfies the maximal demand of the VM ever expected in the future as a consequence.

Full virtualization environments such as *VMware ESX Server* provide special hardware driver for the simulated hardware components as mentioned in Section 3.1.2. These drivers are installed in the guest OS and hence enable a kind of special communication between the guest OS and the VMM. The VMM knows about the amount of memory capacity actually required by the guest OS at any time. Hence, it does not have to allocate the whole amount of memory assigned to a VM directly when the VM is started. It can allocate memory when

it is needed. Especially memory that is not needed any more by the guest OS can be released at runtime and used by other VMs placed on the same server.

In principal, both kinds of virtualization techniques allow overbooking the memory. The virtual memory capacity provided to all VMs can exceed the capacity of the server. Swapping files are used to compensate the missing memory capacity. But possible performance losses caused when memory pages must be reloaded from the swap file are hardly predictable [109]. Hence, the resource management concept presented in this thesis will not draw on this technique.

Network

The concept of virtual network adapters is widely distributed in most common server virtualization environments. One or more virtual network adapters can be assigned to a VM. Each of them has its own MAC address. Hence, they look like physical ones from an outside view.

The direct assignment of physical network adapters to VMs is not supported in most cases. Instead, the VMM provides methods to connect virtual network adapters with physical ones. *Citrix Xen Server* introduced the bridging concept [110] already known from the Linux operating system for this purpose. Different virtual and physical network adapters can be freely connected to each other, which allows a very flexible configuration. A similar approach is followed by the *VMware ESX Server*. They introduced virtual switches (called *vNetwork Standard Switches* in *VMware vSphere 4.X*) that are a software implementation of a hardware switch [120]. Virtual as well as physical network adapter can be connected to these virtual switches. Different of them can be instantiated in a VMM.

The capacity (throughput) of a virtual network adapter typically is much higher compared to a physical one. It is mainly limited by the CPU capacity provided to the VM and to the underlying VMM. But the virtual adapter is not the limiting resource in most cases. Different physical components contained in the server and especially the network infrastructure behind can form the bottle neck depending on the destination of the network traffic. Hence, network capacity provided to the VM depends on all components of the network infrastructure (physical and virtual) but also on the destination of the traffic itself.

A further challenge in virtualization based data centers arises when live migration of VMs must be supported. Older server virtualization environments (e.g. VMware ESX 3.5 and XenServer) require that the same virtual bridges and virtual switches are instantiated on all possible destination server of a VM. The virtual network adapters can be simply connected to them on each of the servers this way. All of these duplicated virtual switches must be connected to the same underlying physical network. Moving a virtual adapter from one switch to another is not a problem for open connection due to the individual MAC and IP address. Respective network packets are simply routed to their new position by the switches.

VMware recently introduced a more holistic network concept into its newest server virtualization environment *VMware vSphere 4.X*. So called *vNetwork Distributed Switches* [121] have been introduced in addition to the *vNetwork Standard Switches*. They function as a single switch across different servers. They better support VM migration and load balancing in the whole network compared to the local switches.

3.1.5 Power States of Servers

Power management techniques that transfer the whole system into a low power state to save energy are widely distributed in the area of notebook and desktop computers. One can mainly distinguish between three low power states [54] that are supported by most common systems:

- **ACPI S3 - Save to RAM(STR)**: The states of all periphery components and the CPU are saved to RAM before they are switched off. Only the RAM and some components that are needed to reactivate the system remain powered on. All hardware components must be powered back on and their context must be restored from RAM to reactivate the system. The operating system resumes in the same state, as it was before the low power state was entered. The fast reactivation time (of typically a few seconds) is an advantage of STR compared to other low power states.

- **ACPI S4 - Save to Disk(STD)**: This low power state is similar to STR. The operating system resumes in the same state, as it was before the low power state was entered. But the context of the hardware components as well as the content of the RAM are stored to hard disk in contrast. This allows the power management controller to power down all hardware components. Any power supply could be disconnected in principal. The transition times to the low power state and back to the active state are quite longer compared to STR. The concrete times mainly depend on how fast the memory content can be stored to disk and restored back.

- **ACPI S5 - Complete Shutdown**: The operating system completely shuts down and switches off the whole hardware. Any power supply could be completely disconnected similar to the STD state. The operating system must reboot for reactivation. Transition times mainly depend on the operating system and the speed of the underlying hardware.

Common server virtualization environments such as *VMware ESX Server* and *Citrix Xen Server* support only STR or a complete server shutdown. STR works not with all servers. Only a complete server shutdown is feasible for them. In any case, all active VMs must be removed from the server before the low power state is entered.

Several mechanism exist to remotely reactivate the server. The classical way is called *Wake on LAN* (also known as *WoL*). The PCI subsystem and the network adapter in the server

remain powered on while the server is in low power mode. Special network packets sent to the network adapter of the server can remotely reactivate the whole system. Modern integrated servers solutions, such as *HP Blade Systems*, have own management hardware that among other things can controls the state of the servers included in the same enclosure. They provide a standardized API to remotely control the server state.

3.2 Conceptual View

The concept for static and dynamic resource management can be divided into three phases. The purpose of the first and second phase is to determine a static assignment of services to servers. In the final third phase, services are also dynamically reassigned to servers according to their current resource demand. Unused servers are powered down to save energy. An overview of the three phases and the tasks that are performed within each phase is given in Figure 3.3.

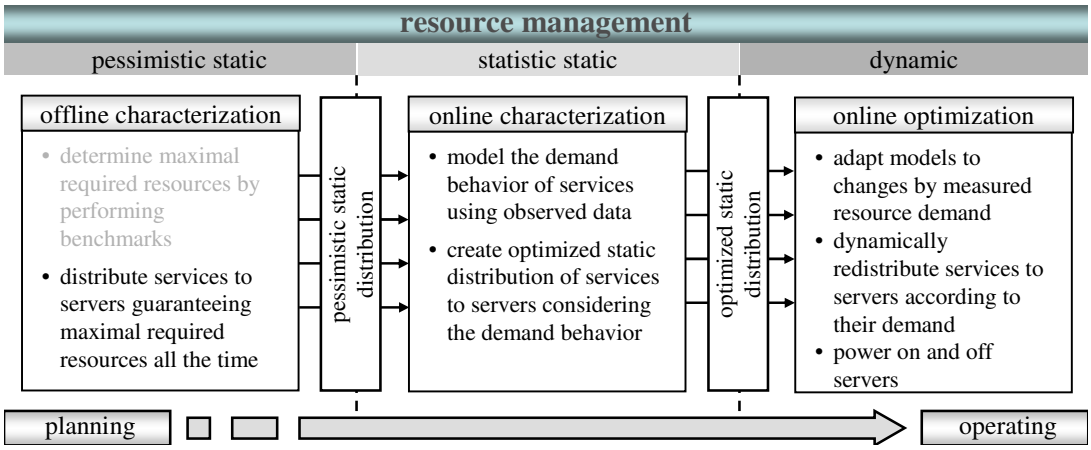


Figure 3.3: The three phases of the static and dynamic resource management concept and tasks that are performed within. The black colored tasks are addressed within this thesis. Tools and concepts already exist for performing the light gray task. Hence, it is only shortly discussed.

Benchmarks must be performed during the first phase comparable to classical capacity planning [8]. They are needed to determine the resources required by a service in case of the maximal expected workload defined in the SLOs. An algorithm then determines a distribution of the services to servers based on this information. The resulting distribution very pessimistically provides the maximally required resources to all services all the time. It will work without violating any performance goals assuming that the maximal workload is not exceeded.

For the following second phase, services are deployed to servers according to that pessimistic

distribution. Their resource demand behavior is observed while users are now doing their normal work with them. Models are characterized that describe the resource demand behavior. Finally, an optimized but also static distribution of services to servers is derived based on these models. This new distribution can require less servers compared to the pessimistic one depending on the workload behavior. This optimized static distribution will also work without any SLO violations under the assumption that the workload behavior does not change with respect to the observed one.

The third phase begins with the optimized static distribution. The data center now operates in its normal mode. The services are redistributed to servers according to their current resource demand using the models trained in phase two. Unused servers are switched off to save energy. Furthermore, the models are adapted by ongoing measures of the resource demand to take care of changed workload behavior.

The tasks to be performed within each phase will be worked out some more in the following to extract challenges that need to be addressed.

3.2.1 Pessimistic Static Resource Management

Hardly something is known about the maximally required resources when new services should be deployed in a data center. Nothing is known about the resource demand behavior at all. But this information is required for proper resource management especially when different service should share the same server as targeted in this thesis.

Hence, benchmarks are typically performed on the services in an isolated environment first. The benchmarks simulate the workload that is maximally expected according to the SLOs. The resource capacity provided to the service can then be adjusted so that the performance goals defined by SLOs as well are achieved. This approach is already well known from ceiling based capacity planning [8]. Hence, this task will not be detailed much deeper within this thesis. Different concepts and tools [55, 122, 85] already exist for performing this tasks. A good overview is presented in [135].

Once the maximal required resources are found for each service, there is still no information present about the demand behavior at runtime. But this information is indispensable to determine the optimized static distribution and especially to perform dynamic resource management. Hence, the services must be observed while users are doing their normal work with them to learn about the demand behavior. Therefore, an algorithm is needed that distributes services to servers in a way that none of the SLOs related to throughput constraints (cf. Section 3.1.1) is violated. Any violation will lead to invalid resource demand values observed. Such an algorithm can only draw on maximally required resources already determined. Hence, only a pessimistic distribution of services to servers is possible. This distribution ensures the maximal required resources to each service all over the time. An algorithm that finds such a

distribution with a minimal number of required servers is presented in Chapter 4.

3.2.2 Optimized Static Resource Management

This phase aims to use knowledge about the demand behavior of the services to find a new distribution of services to servers. This distribution should require less servers compared to the pessimistic one in best case.

It was shown in [102, 117, 58] that typical services require their maximal resources only in a very small fraction of time. Furthermore, different services rarely require their maximum all at the same time. Regarding these facts, services can be assigned to less servers compared to the pessimistic approach. So called statistical static resource management approaches are based on this idea.

It is aimed to use such an approach within this second phase to reduce the overall number of servers required for a given set of services. But it has to be noticed that such approaches are overbooking hardware resources. This can lead to performance reductions caused by resource shortages in some cases. It must be guaranteed that a certain probability as well as a certain strength of performance reduction is not exceeded to use these approaches in real data centers. Both parameters must be defined as SLOs in the SLA between the client and the Service Provider.

Appropriate modeling the resource demand of the services is essential to apply statistical approaches. Especially missing stationarity and possible correlations between the resource demand of different services must be considered. Furthermore, SLOs are needed that describe the performance goals in an appropriate way. Finally, an algorithm that based on the models distributes the services to servers with respect to the SLOs is needed. These challenges are addressed in this thesis in Chapter 5.

3.2.3 Dynamic Resource Management

The dynamic resource management phase starts with the optimized static distribution of services to servers obtained at the end of phase two. It is assumed that all SLOs that concern throughput constraints are satisfied at any time. Services do not have to be redistributed as discussed before.

Services can be now consolidated to fewer servers in times of less overall resource demand. Unused servers can be switched off to save energy. Servers must be reactivated and VMs must be redistributed, when the resource demand increases to prevent resource shortages. The whole static distribution must be restored in worst case.

The main challenge to be addressed is that redistributing services and powering up servers takes time. The dynamic scheduling approach must ensure that the resource demand of the

services remains low for a while, before the safe static distribution is left. There must be enough time to redistribute the services, to power down the servers, to save some energy, and in worst case to power up all servers and to restore the whole static distribution.

This requires properly modeling the resource demand of the services. One must be able to extrapolate the demand behavior of the services expected in the future based on these models. The time dependence of the resource demand needs to be explicitly modeled in contrast to the optimized static resources management. One can only take any advantage of varying resource demand, if it is known when and how long this demand will be low.

A second big challenge to be addressed is the scheduling algorithms itself. It must ensure that a way back to the safe static distribution exists at any time and that it can be performed right in time before the resource demand increases.

And finally, a third challenge concerns possible forecasting errors caused by changed demand behavior of the services. The models used to forecast the resource demand are characterized by demand behavior observed in the past. Unexpected behavior can lead to resource shortages resulting in a reduced service performance. The clients of a service must be able to define the maximally allowed duration of possible performance problems as well as their maximally allowed strength in the SLOs to accept these incidents. Furthermore, the models should adapt changed demand behavior to prevent SLO violation in the future.

An approach for dynamic resource management that addresses these challenges will be presented in Chapter 6.

3.3 System Description

The challenges and optimization problems of static and dynamic resource management are derived from the conceptual view in a very abstract way until now. Hardly some technical aspects are regarded. A closer look to the data center components that are mainly involved in the concept will be provided within this section. Parameters and constraints are derived from this analysis that must be additionally considered. Especially the terms *resource demand* and *SLA* will be detailed some more with respect to the targeted virtualization based data center.

3.3.1 Involved Components

An overview of the data center components involved in the resource management concept is presented in Figure 3.4. The main relevant hardware components are servers, the storage system, and the network. The services to be deployed in the data center and a load and power management (LPM) component are software components. And finally the behavior of clients of the services plays a significant role.

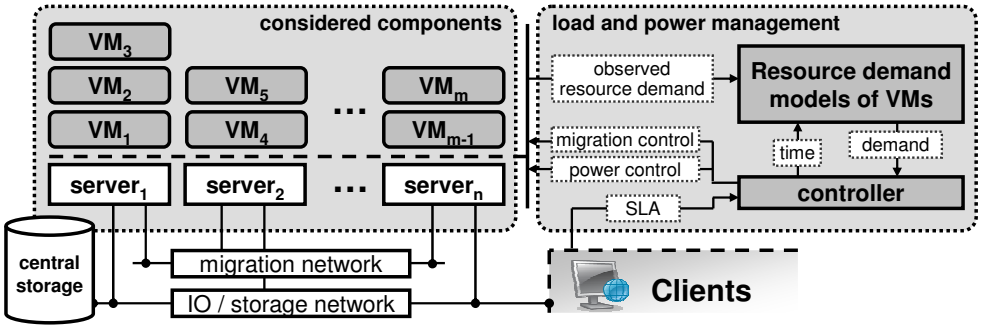


Figure 3.4: Overview of data center components mainly involved in the resource management concept and their dependencies on each other.

The relation between all three parties is quiet simple. Clients are sending requests through the network to the services. The server must answer in a predefined time period according to a SLA. A service allocates resource capacity of the server on which it runs to process a request. Required data is retrieved from the storage system. Finally, the result is sent back to the client through the network again. The load and power management component assigns services to servers and powers servers up and down according to the expected resource demand. Parts of this system will be regarded more detailed in the following.

Servers and Services

The resource management concept developed in this thesis targets small-scale request based IT services as already outlined in Chapter 2. These services require the whole resource capacity of one modern server at a maximum. Most of them require far less capacity most of the time so that one server provides enough resource capacity for several services.

Hence, it is assumed that services are not deployed directly on hardware but implemented in VMs to allow several of them to share the same server (cf. Section 3.1.2). This approach is widely distributed in today's data centers for better utilizing the server hardware.

It is further required that the underlying virtualization environment supports live migration (cf. Section 3.1.3) to realize dynamic resource management. The LPM component must be able to move VMs between servers at runtime without the need to stop the service.

And finally, the power management component must be able to remotely power up and down servers. Respective mechanism have been presented in the background section as well.

Storage System

Most common virtualization environments such as VMware ESX, Citrix Xen Server, or Microsoft Hyper-V require a centralized storage system to support the live migration technique.

The storage is typically connected to the servers via the network system. The services can access the file system from any server in the network. This approach prevents that the whole storage content needed by a service must be copied through the network during the live migration process. Data is already often stored centralized in common medium and large data centers for security and maintenance reasons.

Network

The servers, the storage, and the clients outside the data center are connected via a network. Large network bandwidth should be provided while VMs are migrated between two servers to minimize the duration of this process [30]. Hence, a dedicated network for migrating VMs is recommended [127]. Many modern data centers already use the live migration technique for administration, autonomous load management (e.g. supported by VMware DRS [123]), or recovery solutions (such as e.g. [74]) so that the additional infrastructure is already present very often. In data centers that require fast answers from the storage or that have high data throughput even the storage and the user IO network are implemented in dedicated nets.

Load and Power Management

The LPM component implements the resource management concept. A controller assigns VMs to servers based on the SLAs and the resource demand forecasted by the models. Additionally, the controller powers up and down the servers. The resource demand models are characterized using observed resource demand. Appropriate interfaces provided by most common virtualization environments connect the LPM component to the virtualization environment for this purpose.

3.3.2 Limited Resources

Resource demand of VMs was regarded in a very abstract way until now. A closer look to the different resources types will be provided within this section to understand, which ones need to be considered by the resource management concept. Generally, limited resources for services in a data center are CPU time, network bandwidth, memory (RAM), and disk space. In the following they will be discussed one by one.

CPU Time

CPU time provided to VMs is a resource that generally is limited by the capacity of the server on which the VMs are running. All VMs placed on the same server must share its capacity as discussed in Section 3.1.4. Hence, CPU time must be considered by the resource management concept.

It has been further detailed in Section 3.1.4 that the capacity of different CPUs contained in one server can be handled like one big resource pool of CPU time. This means that one single value can describe the CPU capacity of a whole server w.r.t. all contained CPUs and cores.

The CPU time consumed by a VM is varying over time and can be split between different virtual CPUs assigned to the VM. One virtual CPU can completely consume the time of one core of a hardware CPU at a maximum. The CPU time consumed by all virtual CPUs of one VM can be summed up to get the overall CPU time taken from the servers capacity due to the dynamic mapping of virtual CPUs to cores of real CPUs.

A VM takes exactly the part of CPU time it needs as long as enough CPU time capacity is provided. This amount of taken CPU time will be denoted by CPU time demand of the VM within this thesis. A service will typically not completely fail, when this demand exceeds the CPU time actually provided by the virtualization environment. It will only get slower. Hence, provided CPU time can be adjusted to the demand of a VM to trade off service performance against resource capacity.

The CPU time required to meet a performance goal of a service can vary depending on the CPU type and the main board architecture of the server. Hence, not only the resource demand but also the server on which the VM is currently placed decide about the CPU time needed to meet a certain SLO. Such heterogeneities are not addressed within this thesis. It is assumed that providing a fixed amount of CPU time to a VM will lead to the same performance independent from the server it is deployed on.

Network Bandwith

An overview of common network concepts that support server virtualization has been given in Section 3.1.4. It has been pointed out that distributing VMs to servers only considering their required bandwidth will fail in many cases. Additional information about the underlying network (virtual as well as physical) and the source or destination of the traffic itself must be taken into account.

In general, one would not place two VMs that require a bandwidth of 80MBit/s together on a server that is physically connected to the network via a 100MBit/s network adapter. But if the whole traffic is exclusively caused by communication between these two VMs, placing them together at a server might be the best solution at all. The traffic is routed directly through the local virtual net of the host, which is quiet faster and less expensive in terms of required resources. Hence, a more detailed view into the services implemented in the VMs and knowledge about the network concept behind are required.

It is expected that the administrator has completely worked out the network concept for the VMs that are controlled by the LPM system. This concept defines restrictions for the resource management concerning VMs that must, can, or must not be placed together on the

same server. Furthermore, a list of invalid hosts is defined for each VM on which they must not be placed at all. The administrator can at least prevent that two traffic intensive VMs are using the same physical network adapter this way. Furthermore, servers can be excluded that do not provide appropriate network connections to a VM. It is assumed that no SLA violation are caused by the network system, when these restrictions are considered by the resource management.

Additional restrictions concerning the actual network traffic caused by the VMs might be needed for resource management decisions as well. VMs that are connected to their clients via a physical network adapter must share its bandwidth. This should be considered as well. But a possible bottle-neck might not be the adapter itself but the network infrastructure behind. Hence, the whole infrastructure must be regarded as well, which requires modeling the behavior of all components involved and the traffic of the services implemented in the VMs.

This complex research area is not worked out any more within this thesis. It must be part of required future work. It will be shortly outlined in the outlook section of the conclusion chapter, how appropriate network models can be integrated into the resource management concept presented in this thesis. An interface can connect a holistic concepts for network management (such as provided by *VMware vSphere 4.X*) with the LPM component to also consider network aspects.

Memory (RAM)

The memory capacity is limited per server comparable to CPU time. Different servers can have different memory capacities. All VMs placed on the same server must share its memory capacity. Hence, memory must be considered by the resource management concept as well.

But in contrast to CPU time, the service implemented in a VM will completely fail, when the memory demand exceeds the provided capacity[4]. Furthermore, memory demand of a VM can either vary over time or can be a fixed value at runtime depending on the virtualization environment as pointed out in Section 3.1.4.

Disk Space and Disc IO Operations

The storage system is centralized as discussed in the previous section. Hence, the disk space provided to each VM is independent from the distribution of VMs. The number of active servers does not influence disk space as well. Hence, this resource type does not have to be considered by the resource management concept as limited resource.

[4] As discussed in 3.1.4, swapping techniques allow exceeding the capacity by swapping out parts of the memory to disk. Because of hardly predictable performance loss when memory content must be restored, the concept presented in this thesis will not draw on this technique.

The maximal throughput of storage I/O operations is typically limited by the storage system itself but not by the connection between the storage and the server. Hence, the distribution of VMs to servers does not influence the response time and throughput of storage I/O operations. Modern virtualization environments typically support storage I/O prioritization that based on shares and limits handles I/O requests of different VMs [126]. This storage access management works cluster-wide and hence is independent from the positions of VMs and the number of the VMs placed on the same server.

3.3.3 Overhead and Prerequisites of Control Mechanisms

Some technical properties and constraints concerning the control mechanism must be considered by the resource management concept in addition to the limited resources of servers.

Migrating VMs

The whole memory (RAM) content as well as some additional informations about the hardware context are copied through the network when a VM is moved from one to another server. This additional traffic does not conflict with normal IO traffic of the VMs because of the dedicated network for migrations.

Furthermore, no additional CPU time is required. The migration is processed by the VMM itself, that typically owns its individual CPU core. It is suggested to individually reserve CPU capacity for live migration on each server to prevent any interactions with the VMs, if none of the CPU cores can be individually reserved for the VMM [83]. The resource management concept does not have to take care of this kind of overhead. It can be simply regarded as a constantly reduced amount of CPU capacity.

Potential slowdowns caused by concurrent accesses to memory or other hardware resources contained in the server are not considered within this thesis. Such artifacts do not significantly influence the response time or throughput of normal services even if they are heavily loaded regarding analyses presented in [30, 83].

But the migration operation will take time that must be considered by the scheduling algorithm in the LPM component as discussed in the concept section. It was found in [83, 89] that this time mainly depends on the amount of memory to be copied and on the network bandwidth. But the activity of the service implemented in the VM also slightly influences the migration time[89, 30, 83]. High activity increases the probability for memory pages to get invalid after they have already been copied. Hence, they must be copied again, which increases the overall migration time. It is assumed within this thesis that the time for migrating a VM can be safely overestimated, if the VM is the only one that is migrated in the whole data center at a time. It is further assumed that this time is valid for the migration of the VM between all possible combinations of source and destination server.

A second constraint concerns the limited resources CPU time and RAM. The complete required memory (RAM) must be available for the VM at the destination server already at the beginning of the migration process of a VM. The migration process will not start, if not enough memory is present. Nearly the same is true for CPU time. Basically, the migration process would start and CPU time of the destination server is not used until the migration process is completely finished. But the scheduling algorithm does not know when exactly the migration is finished, since the migration time of the VM is only overestimated. Thus it is unclear when exactly the CPU time is needed. As a result, the required CPU time as well as the memory capacity must be available at the source and destination server during the whole migration process.

Powering Up and Down Servers

Powering up and down servers will take time comparable to the migration process of virtual machines. It is assumed within this thesis that safe shutdown and resume delays can be overestimated for each server in the data center.

In addition to the delays, the scheduling algorithm must also consider the fact that once the server starts the shutdown or resume process, it cannot be stopped until it is completely finished. Furthermore, no active VMs can be deployed on the server during the shutdown or resume process. Especially the last point requires that the servers remains switched off for a while to compensate the energy overhead caused by the shutdown and resume process.

Gathering Data and Managing the System

The concept that will be presented in this thesis requires monitoring the resource demand of the VMs. Measured data will be used to train the models in the first and second phase of the concept. The demand must be observed further in the third phase to detect changed demand behavior and to adapt the models if necessary.

This monitoring task will require some additional resource capacity that needs to be provided by the servers besides the normal work. But in most common data centers, such data (memory and CPU time demand) is already gathered for capacity planning and trouble shouting reasons[9]. Hence, a fixed amount of resource capacity is already reserved on each server for the virtualization environment to support this task.

Additionally, the LPM component itself needs some resource capacity in the data center that must be provided by one single server or a single PC. Especially in virtualized data centers, any kind of system management software is still running in most cases to observe and maintain the data center. The LPM component should be designed in a way that it could be deployed on such existing systems as well without increasing the overall resource demand for the management software to much.

3.3.4 Service Level Agreements

SLAs were introduced in Section 3.1.1 as a contract between the Service Provider and the clients. SLOs are defined to ensure Quality of Service (QoS). It was pointed out that SLAs can be additionally used as constraints for autonomous resource management tools. This thesis focuses on the second aim.

It is presented in Figure 3.5 how resource management and SLAs depend on each other.

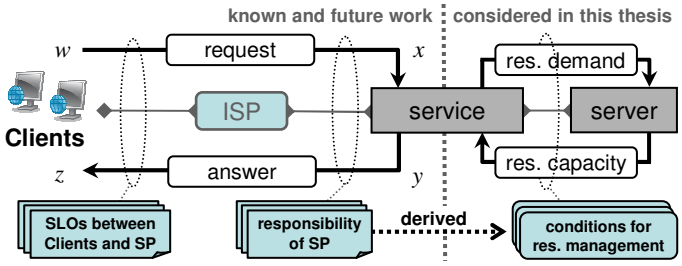


Figure 3.5: Dependency between resource management tools and SLOs containted in SLAs to ensure QoS

The SLOs between clients and the SP normally concern properties of requests (w) and answers (z). But the part of the system that the SP can control by own provisioning decisions is limited to requests that already passed the network (x) and to answers before they will pass it (y). Hence, only this part can be used by resource management tools for service provisioning.

In general, processing requests requires hardware resources. The number of requests in a time period is varying so that the amount of required resources is varying as well. The amount of resource capacity provided to this demand decides about how fast requests are processed or even if they can be processed at all as discussed before. Hence, constraints for the resource management concerning the amount of required resource capacity for a given resource demand must be derived from the SLOs of a SLA.

The SLOs the resource management concept is responsible for will be presented within the chapters in which the individual phases of the concept are worked out.

3.4 Formal Definition

Modeling the resource demand of VMs, distributing the VMs to servers, and specifying appropriate SLOs turned out to be the main challenges of static and dynamic resource management in the last two sections. These challenges as well as the extracted parameters and constraints to be considered will be described more mathematically in this section to formally specify the optimization problems behind. The main import terms are introduced first, before the

optimization problems themselves are presented.

3.4.1 Terminology and Declarations

The terms, functions, and variables shortly introduced in the following are consistently used throughout the whole thesis.

VMs and Servers

All VMs in a data center considered for resource management are indexed by an i. Variables and functions that describe properties of VMs (e.g. the resource demand) are indexed by an i as well to relate them to the respective VM. Servers, in contrast, to which these VMs can be assigned, are indexed by an k as well as functions and variables that belong to properties of the respective servers.

Mapping of VMs onto Servers

The mapping of VMs onto servers is expressed by a function $B(i) : i \mapsto k$ that associates one server k to each VM i. This function is time independent in case of static resource management. It turns into $B(i,t) : i \mapsto k(t)$ for dynamic resource management.

Restrictions of the Mapping

The administrator must be able to restrict the mapping of VMs onto servers to take care of the network concept as discussed in Section 3.3.2. First, the administrator must be able to define, if two VMs must, can, or must not be placed together at the same servers. Second, it can be required to forbid the placement of VMs on certain servers at all.

A function $VM\_VM\_allow(i_1, i_2)$ is introduced to realize the first of these restriction. This function returns *true*, if two VMs i_1 and i_2 are allowed being placed on the same server. The administrator can use this function to prevent that two VMs are placed together at the same servers. No additional function is needed to force that VMs are always placed together. These VMs can be simply treated like one single VM by the resource management. The resource management can decide where to place them based on the joint resource demand of both.

The second restriction can be realized by implementing a function $VM\_server\_allow(i, k)$. This function returns *true*, if a VM i is allowed being placed on server k.

Resource Demand of VMs

The limiting resources that directly must be considered by the resource management are CPU time and memory according to the outcomes of Section 3.3.2. The CPU time demand is time

dependent in any case. The memory demand can be either a fixed value or varying over time depending on the underlying virtualization environment.

Thus, the resource demand of a VM i can be expressed by a two dimensional vector $\vec{r}_i(t)$ that depends on time t. The elements of this vector are functions $R_i(t)$ that describe the CPU time and memory the VM takes at time t if available. Both resources types are handled nearly the same way within this thesis. Hence, it is not required denoting the functions by different names. Both resources types are meant, when it is talked about $R_i(t)$ without an additional note.

The function $R_i(t)$ is limited by a maximal resource demand R_i^{max}, which occurs in case of the maximally expected workload specified in the SLOs. It is assumed that $R_i(t)$ never exceeds R_i^{max}. A vector $\vec{r}_i^{max}$ describes the maximal resource demand of VM i with respect to both resource types.

The amount of resources demanded by a VM is mainly defined by interactions between the clients and the service implemented in the VM. Client interactions occur randomly from the system's perspective. Hence, the resource demand $R_i(t)$ is not deterministic as well. A discret-time stochastic process called $Y_i(t)$ is used within this thesis to model $R_i(t)$. This process is not necessarily stationary due to possible time dependent trends and a varying noise performance.

Different VMs placed on the same server cause joint resource demand, which is simply the sum of their individual resource demands. Hence, the joint resource demand is described by a two dimensional vector as well. This vector is called $\vec{jr}_k(t)$ and is formally defined by following equation:

$$\vec{jr}_k(t) = \sum_{\forall i: B(i)=k} \vec{r}_i(t). \tag{3.1}$$

Required Resource Capacity for VMs

$\vec{a}_i(t)$ is introduced in addition to the resource demand. This vector describes the amount of resources needed to be reserved for a VM at time t to meet the SLOs the resource management is responsible for. The elements $A_i(t)$ of $\vec{a}_i(t)$ are functions comparable to $\vec{r}_i(t)$. But they are deterministic and overestimate the trend and random noise of $R_i(t)$ with respect to the SLOs in contrast to $R_i(t)$. In other words: The respective SLO of the service implemented in the VM will be never violated, if the resource capacity $\vec{a}_i(t)$ is provided to VM i at each time t.

An upper limit of resources a VM can take from a server can be specified in common virtualization environments. No additional resources are provided even if the demand $R_i(t)$ exceeds this limit. Furthermore, a fixed amount of resources can be individually reserved for each VM. Such reserved resources can not be provided to other VMs no matter if they are

actually used or not. Both limits can be regarded as upper and lower bound of $A_i(t)$. They are denoted by A_i^{min} and A_i^{max} or $\vec{a}_i^{min}$ and $\vec{a}_i^{max}$ respectively within this thesis.

Furthermore, the server capacity required to satisfy all SLOs of the VMs placed on the same server k is introduced and will be denoted by $\vec{ja}_k(t)$. The relation between $\vec{a}_i(t)$ and $\vec{ja}_k(t)$ can be expressed by following equation:

$$\vec{ja}_k(t) \leq \sum_{\forall i:B(i)=k} \vec{a}_i(t). \tag{3.2}$$

It will be shown later in this thesis that the required resources for a couple of VMs can be lower than the sum of the resources individually required by them.

Resource Demand and Required Capacity Purged by a Long Term Trend

The model that will be presented in this thesis is split into two parts. One part models possible seasonal trends and the noise while the other one adds the influence of possible long term trends. The functions $R_i(t)$, $A_i(t)$, and $Y_i(t)$ refer to resource demand and required capacity including the long term trend. These functions but purged by the long term trend are needed from time to time to describe the concept. They will be marked by a star as follows: $R_i^*(t)$, $A_i^*(t)$, and $Y_i^*(t)$, to distinguish them from the initial ones.

Provided Resource Capacity of Servers

Servers provide resource capacity to the demand of the VMs. The resource capacity of a server is time independent but different servers can have different resource capacities. The capacity of server k is described by a two dimensional vector $\vec{c}_k$ in this thesis comparable to the resource demand. This vector contains individual resource capacities C_k each for one resource type.

Server and Migration Delays

The control mechanisms used by the LPM component consume time as discussed in Section 3.3.3. This time needs to be considered by the optimization algorithms. These delays will be denoted by Δt_i^{mig} (time for migrating VM i between two servers) , Δt_k^{up}, and Δt_k^{down} (time for powering up or down server k) within this thesis. In addition, servers need to stay switched off for a while after they have been powered down to at least save some energy at all. This so called break even time is denoted by Δt_k^{be}. And finally, $\Delta t^{B(i,t_1)\rightarrow B(i,t_2)}$ is introduced as the time needed to redistribute the mapping at time t_1 to a new mapping at time t_2.

Starting Time and Duration of the Three Phases of the Concept

All three phases presented in the concept description in Section 3.2 are performed subsequently. The starting times of the phases are denoted by t_{p_1}, t_{p_2}, and t_{p_3}. Their respective durations are described by Δt_{p_1}, Δt_{p_2}, and Δt_{p_3}. The duration of the first phase (pessimistic resource management) is only listed for completeness and has no relevance to the resource management concept. Δt_{p_2} defines the duration of the trainings phase. The data center performs normal work during the third phase (dynamic resource management). Hence, this phase should last for ever in best case. But in practice, possible long term trends prevent that the static distribution of VMs to servers determined in phase two remains valid for unlimited time. Hence, the duration of phase three is limited to Δt_{p_3} as well.

3.4.2 Problem Definition: Static Resource Management

The goal of static resource management is to find one fixed mapping $B(i) : i \mapsto k$ of VMs onto servers as pointed out in Section 3.2. This mapping should require a minimal number of servers. Hereby, the jointly required resources of all VMs placed on the same server must not exceed its capacity to not violate any SLO. This condition must only be valid in the predefined time interval $[t_{p_3}, t_{p_3}+\Delta t_{p_3}]$ (e.g. for one year in future), since possible long term trends can continuously increase the resource demand. Static resource management should be applied again after this period is expired to take care of changed demand behavior. The whole condition can be formally expressed by following equation:

$$\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}], k : \vec{ja}_k(t) \leq \vec{c}_k \tag{3.3}$$

whereby the operator $\leq$ is defined as follows:

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \leq \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \Leftrightarrow u_1 \leq v_1 \wedge u_2 \leq v_2. \tag{3.4}$$

A second and third constraint concern the mapping restrictions that must be regarded as well. The first one ensures that two VMs are not placed together at the same server if they are not allowed to according to function $VM\_VM\_allow(i_1, i_2)$. Following equation must hold for the mapping $B(i)$ to not violate this constraint:

$$\forall k, i_1, i_2 : B(i_1) = k \wedge B(i_2) = k \Rightarrow VM\_VM\_allow(i_1, i_2). \tag{3.5}$$

The second one ensures that VMs are only placed on servers on which they are allowed to be placed on according to $VM\_server\_allow(i, k)$, which can be formally expressed by following

equation:

$$\forall k, i : B(i) = k \Rightarrow VM\_server\_allow(i, k). \tag{3.6}$$

These constraints are valid for the pessimistic static resource management (cf. Section 3.2.1) as well as for the statistic static one (cf. Section 3.2.2). The only differences are the characterization of the stochastic processes $Y_i(t)$ that model $R_i(t)$ and the way the joint resource demand $JA_k(t)$ to be reserved is derived from them. The first approach pessimistically reserves the maximum required resources of all VMs all over the time. The optimized one uses information about the demand behavior of the VMs to more optimistically derive $JA_k(t)$ from the $Y_i(t)$s as presented in Chapter 4 and Chapter 5 respectively.

As an outcome, the two challenges for static resource management can be now expressed more detailed. First, the stochastic process $Y_i(t)$ that models the resource demand $R_i(t)$ of VM i must be characterized. Second, a way to determine resources required for a couple of VMs that should be placed together at one server must be derived from the processes with respect to appropriate SLOs. And finally, an optimal mapping $B(i)$ must be found that minimizes the overall number of required servers with respect to the Equations (3.3),(3.5), and (3.6).

3.4.3 Problem Definition: Dynamic Resource Management

The goal of dynamic resource management is to reduce the energy consumption in times of low utilization by redistributing VMs at runtime and switching off unused servers according to the concept described in Section 3.2. This leads to the need of a time dependent mapping $B(i, t)$ of VMs onto servers. Different conditions must be met that will be shortly presented in the following.

Guaranteeing SLA

Equation (3.3) that was presented for static resource management before must hold for the dynamic case as well to not violate any SLA. The only difference to the static case is that the mapping $B(i, t) : i \mapsto k(t)$ of VMs onto servers can now change over time.

The second and third constraint worked out for the static resource management must be extended a bit to consider the time dependent mapping. The first one can be expressed by following equation:

$$\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}], k, i_1, i_2 : B(i_1, t) = k \wedge B(i_2, t) = k \Rightarrow VM\_VM\_allow(i_1, i_2). \tag{3.7}$$

The second one respectively is extended to:

$$\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}], k, i : B(i, t) = k \Rightarrow VM\_server\_allow(i, k). \tag{3.8}$$

Limited Number of Servers

A static safe distribution of VMs to servers is determined during phase two according to the concept. This distribution defines the number of servers maximally required at all. The dynamic scheduling algorithm must not exceed this maximal number of servers at any time.

Required Resources While VMs Are Moved

A third constraint of the resulting mapping function $B(i,t)$ concerns the resources required while VMs are moved between servers. The required resource capacity $\vec{a}_i(t)$ (for CPU time and memory) must be provided by a source server k_1 and a destination server k_2 during the whole migration process of a VM i from k_1 to k_2 as discussed in Section 3.3.3. This results in the following additional condition for the destination server k_2:

$$\forall t \in [t_0, t_0 + \Delta t_i^{mig}[: \vec{ja}_{k_2}(t) + \vec{a}_i(t) \leq \vec{c}_{k_2}. \tag{3.9}$$

t_0 is the starting time and Δt_i^{mig} the duration of the migration phase.

Considering Delays

Powering up and down servers and moving VMs needs time as discussed earlier. The scheduling algorithm must consider these delays, when it determines a mapping $B(i,t)$ of VMs onto servers that dynamically redistributes VMs and switches on and off servers. It must ensure that the time period between two different subsequent distributions of VMs to servers is not lower than the time the operations will take to switch between them. This constraint can be formally expressed by following equation:

$$\forall t \in [t_{p_3}, t_{p_3} + \Delta t_{p_3}], s : \Delta t^{B(i,t) \to B(i,t+s)} \leq s. \tag{3.10}$$

Challenges

The challenges for dynamic resources management worked out in Section 3.2.3 can be detailed some more based on the formal descriptions provided in this section.

One big challenge is to model the resource demand $\vec{r}_i(t)$ of the VMs to derive the resource capacity $\vec{ja}_k(t)$ required by all VMs placed on server k comparable to static resource management. The time dependence of $\vec{ja}_k(t)$ must be modeled explicitly in contrast to the static case as discussed before. The model must be able to forecast $\vec{ja}_k(t)$ for a time period of up to a few hours to take care of the delays caused when VMs are moved or servers are powered up and down.

A second big challenge forms the scheduling algorithm that finds $B(i,t)$. It must ensure at any time that any kind of upcoming resource demand can be handled by redistributing VMs to servers based on the forecasts provided by the models. The constraints worked out in this section must be considered.

3.5 Summary

A concept for static and dynamic resource management has been worked out within this chapter. This concept consists of three different phases and aims to reduce the overall energy consumption in a data center. Service Level Agreements must be taken into account by the resource management decisions. First, it is planned to statically assigned services to servers in a way that the overall number of required servers is minimized. In a next step, services should be dynamically reassigned at runtime according to their current resource demand. Unused servers are switched off to save energy. In each of the three phases specifying appropriate Service Level Objectives, modeling the resource demand, and distributing the services to servers turned out to be the main challenges.

Server virtualization has been chosen as an appropriate technical platform to realize the resource management concept. This technique allows placing different services with their individual operating system at the same server. Furthermore, services can be moved between servers at runtime without any significant interruption.

The concept of server virtualization and additional relevant components have been analyzed within this chapter as well. CPU time, memory capacity, and network infrastructure turned out to be shared resources that must be mainly considered by the resource management. Different additional timing and resource constraints have been worked out as well. Finally, they all have been translated into a mathematical representation from which formal optimization problems have been derived.

The challenges and optimization problems worked out in this chapter will be individually addressed for each phase in the next three chapters.

4 Pessimistic Static Resource Management

The aim of pessimistic static resource management is to find a static distribution of VMs to a minimal number of servers according to the concept worked out in previous chapter. All SLOs of the services implemented in the VMs must be satisfied at any time. The algorithm that finds the distribution has no information about any real life workload behavior, since none of the services has been used by any client so far. Expected resource demand can be only estimated using artificial benchmarks.

It will be shortly presented within this chapter how such a distribution can be determined based on information about maximally expected workload. The method described is hardly new. It is strongly related to common ceiling based off-line capacity planning approaches [8]. Nonetheless, it will be shortly described in this chapter since parts of this method serve as a basis for the following phases of the concept. Especially the heuristic optimization algorithm that finds the distribution of VMs to servers will be required in Chapter 5 again.

4.1 Service Level Objectives

Satisfying SLOs that define performance constraints mainly depends on resource management as pointed out in Section 3.3.4. The amount of provided resource capacity decides if and how fast requests are processed by a service. Hence, these SLOs can be directly used as a kind of constraint for resource management considered while the resources required by the service are determined.

Only a simple kind of SLO specification will be supported in this first phase of the concept. One performance goal (throughput or response time) can be defined for each service. This goal must be achieved, if a certain amount of workload (e.g. number of requests in a time interval) also defined in the SLO is not exceeded. More complex SLO specifications will be introduced later on in the following chapters. They will allow trading off provided resource capacity against service performance based on observed workload behavior.

It is important to already consider possible long term changes, when the maximally expected workload is specified. The resource management concept assumes that this maximum will be

never exceeded until the end of phase three is reached. Consequences when this assumption is violated will be discussed in the sections about changed demand behavior in Chapter 5 and 6.

4.2 Modeling the Resource Demand

It was mentioned in the problem statement chapter that the resource demand of a VM will be modeled by stochastic processes. But these processes can be hardly characterized without knowing anything about the real life workload behavior. The only information the resource management concept can draw on so far is a maximal resource demand $\vec{r}_i^{max}$ of the VMs, which occurs in case of maximally expected workload. Such information can be determined using appropriate benchmarks as already discussed in Section 3.2.1.

Furthermore, benchmarks can be used to determine a maximal amount of resource capacity $\vec{a}_i^{max}$ that will be required by a VM to meet its performance goal. Starting with capacity that satisfies $\vec{r}_i^{max}$, the provided capacity $\vec{a}_i^{max}$ can be simply adjusted downwards until the performance goal is violated. Such an approach is known from classical ceiling based capacity planning [8].

Once $\vec{a}_i^{max}$ is determined for each VM i, a scheduling algorithm can now distribute the VMs to servers. Appropriate algorithms will be described in the following section.

4.3 Static Scheduling

Three different constraints have been worked out in Section 3.4.2 (cf. Equation (3.3), (3.5), and (3.6)). They must meet for a resulting distribution $B(i)$ of VMs to servers to not violate any SLO . The first one concerns the joint resources required by all VMs placed on the same server. The second and third one restrict the mapping of VMs onto servers.

The jointly required resources $\vec{ja}_k(t)$ of all VMs placed on the same server can be overestimated using the maximally required resources $\vec{a}_i^{max}$ of the VMs. This turns the first condition into:

$$\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}], k : \vec{ja}_k(t) \leq \sum_{i:B(i)=k} \vec{a}_i^{max} \leq \vec{c}_k. \tag{4.1}$$

The sum of the maximally required resources $\vec{a}_i^{max}$ of all VMs i placed on the same server k must be lower than the server's resource capacity $\vec{c}_k$.

Finding an optimal mapping $B(i)$ with respect to this constraint is a classical vector (or multidimensional) bin packing problem (off-line version), which is known to be NP-complete[69]. A set of given items each with an individual (multidimensional) size must be assigned to bins with individual (multidimensional) capacities. The second and third constraint additionally limit the solution space of feasible mappings $B(i)$ of VMs on servers.

4.3.1 Known Approaches

Many heuristic solvers for the one-dimensional case (known as classical bin packing problem) have already been presented. A good analysis of many of them can be found in [56]. The presented approaches range from classical *first-fit*, *next-fit*, and *best-fit* heuristics to more complex ones that allow rearrangements of already scheduled items.

The *first-* and *best-fit* heuristics have been extended to the multidimensional case in [69]. The authors of [26] introduced an own heuristic solver and compared its efficiency to classical ones. Different of these known heuristics have been applied for resource management in data centers in [102]. A set of services has been assigned to servers with respect to their multidimensional resource demand. The results have been compared and discussed as well.

The heuristics of all of these approaches are optimized for equally sized bins. In principal, VMs can be distributed to a heterogeneous set of servers. There might exist different classes of servers each with an individual capacity configuration from which a mixed set can be selected. This generalized bin packing problem is known as *variable size bin packing* and has been addressed for instance in[31, 65].

4.3.2 Vector Bin Packing and Resource Management

A multidimensional bin packing solver will be required for the pessimistic as well as for the statistical static resource management concept presented in this thesis. This solver must determine the mapping $B(i)$ of VMs onto servers as initially discussed. In principal, any of the known ones can be used by the concept since these algorithms typically need only little information about the underlying system.

The easiest solver(that applies a *first-fit* heuristic) must only be able to decide whether or not a certain combination of VMs will fits together on a certain servers. This can be done by simply evaluating the three constraints worked out at the beginning of this chapter. It can assign the VMs one by one to servers on which already VMs have been placed based on this information. A new one is taken, if not enough resource capacity is left on any of these servers to host the next VM. This method is repeated until all VMs are assigned to servers.

VMs can be sorted by their size first, before the *first-fit* strategy is applied to improve the results. The largest VMs are assigned to servers first. The smaller ones can then fill up remaining resource capacity. This strategy is known as *first-fit decreasing* and requires the definition of a quasi order of the VMs with respect to required resource capacity. This order is not unique, since two different resource types are involved (CPU time and memory). Different ways to deal with such multidimensional sizes have been presented in [69, 102].

A further improvement promises the *best-fit* strategy. A VM is not placed on the first server it would fit on but on the best one. This requires to rate the placement of a VM to a certain

server. Conventionally, the resource capacity is taken into account that would remain on a server, if a VM is placed on it. The more capacity remains at a server, the more balanced will be the resulting distribution of VMs to servers. All servers will have left more or less the same amount of unused resource capacity that can serve as a buffer. This buffer accommodates resource demand of VMs that slightly exceed their expected maximum and hence prevents SLO violations in some cases. Again, additional heuristics are needed in case of multidimensional resource demand. Different of them have been presented in [102] as well.

No further information about the VMs or services are required to support different capacity configurations of servers using for instance the approaches presented in [31, 65]. Only costs must be assigned to servers so that the approach can find the most efficient solution with respect to these costs.

This thesis does not focus very much on improving the efficiency of bin packing approaches. Much work has already been investigated in this area. The *best-fit* heuristic was applied during the evaluation of the concept. One of the heuristics presented in [102] was used to deal with multidimensional resource demand. The authors showed in an exhaustive analysis that this heuristic already works very well with resource demand of typical services in data centers. Furthermore, only servers with identical capacity configurations were used during the evaluation.

5 Statistical Static Resource Management

Phase two of the concepts starts with the pessimistic distribution of VMs to servers derived using the algorithm presented in Chapter 4. The runtime behavior of the resource demand of the VMs is observed. Models are characterized and used to determine an improved distribution that leads to less servers required compared to the pessimistic approach.

In principal, using observed demand behavior of the VMs can help to save required server hardware in two ways. First, there can exist VMs that will never demand their maximally required resources at the same time due to negatively correlated workload processed by them. Less than the sum of their maximally required resources needs to be reserved for both, when they are placed together at the same server. A second way is to trade off granted resources against service performance. Service performance can be adjusted by CPU time provided to a VM as pointed out in Section 3.3.2. Resources can be saved depending on the SLOs when this opportunity is exploited.

Once the resource demand models of the VMs are characterized, a scheduling approach heuristically assigns VMs to servers. The algorithm is similar to the one used to find the pessimistic distribution. The differences are the models and the SLOs that are used to decide whether a certain combination of VMs will fit together at the same server or not.

The main challenges addressed in this chapter are the SLOs for specifying the resource performance trade-offs and the models that describe the demand behavior. Furthermore, a way must be worked out to find an optimal distribution of VMs to servers based on the models and the SLOs.

5.1 Mathematical Background

Different mathematical background mainly concerning random variables and stochastic processes is required to understand the concepts that will be presented in this chapter. Such background is provided in this section. Those who are familiar with the concept of random variables, operations on random variables, and the concept of stochastic processes can easily skip this section.

5.1.1 Discrete Random Variables

The concept of random variables has been widely presented in literature (e.g. in [97, 94]). Typically, random variables are used to mathematically describe random experiments or random behavior. Such variables are used in this thesis for the same purpose. They describe the resource demand of a VM with respect to random noise.

Mathematically regarded, a random variable is a mapping from a probability space into a measurable space. All possible random events are mapped onto real numbers out of the domain of the random variable (e.g. out of $\mathbb{R}$). Concrete value x of a random variable X (the randomly generated real numbers) are called realizations.

It can be distinguished between discrete and continuous random variables. Discrete variables map events to values of a countable set. Continuous ones have an uncountable number of different events mapped onto an uncountable number of values. Discrete variables are used within this thesis, since the resource demand of VMs is discrete[1] as well.

A discrete random variable X is completely characterized by its probability distribution function $f_X(x)$. This function returns the probability that a certain realization $x \in X$ will occur, which can be formally expressed by $f_X(x) = P(X = x)$.

An import property concerning random variables is statistical independence between different of them. Events of different random variables that lead to concrete realizations can depend on each other. The variables are correlated or not statistically independent. Statistical dependence, in contrast, means that no dependency between all events of two different variables exists. Events of one random variable are occurring completely randomly according to the respective probability distribution not depending on concrete events of the other one and vise versa.

This definition can be expressed more formally as follows. Two random variables M and N are defined to be statistically independent, if following equation holds for the probabilities of all possible realizations of both of them[49]:

$$\forall m \in M, n \in N : P(M = m \wedge N = n) = P(M = m) \cdot P(N = n). \tag{5.1}$$

5.1.2 Operations on Discrete Random Variables

The concept presented in this chapter requires different operations to be performed on discrete random variables. They will be introduced and described in the following.

[1] This fact will be discussed later.

Sum of Random Variables

Let N and M be two (not necessarily discrete) random variables with realizations out of a subset of $\mathbb{R}$. The sum $Z = N + M$ of both is a random variable with realizations out of a different subset of $\mathbb{R}$ as well. All possible pairs $n \in N$ and $m \in M$ of realizations lead to a realization of z of Z as follows: $z \in Z : z = n + m$.

To describe how the probability distributions $f_Z(z)$ of Z can be calculated from the ones of N and M, first an event $E_X(x)$ is introduced. It is defined that $E_X(x)$ occurs, when a random variable X leads to a realization $x \in \mathbb{R}$. Using this definition, $f_Z(z)$ can be determined as follows:

$$f_Z(z) = \int_{-\infty}^{\infty} P(E_n(n) \cap E_m(z - n))dn \tag{5.2}$$

The probability of all pairs of possible realizations of N and M that added up result to a certain z are summed up to calculate the probability of getting z as a realization of Z.

$P(E_n(n) \cap E_m(z - n))$ can be replaced by $P(E_n(n)) \cdot P(E_m(z - n))$, if N and M are statistically independent, which is equivalent to $f_N(n) \cdot f_M(z - n)$. Therewith, Equation (5.2) turns into

$$f_Z(z) = \int_{-\infty}^{\infty} f_N(n) \cdot f_M(z - n)dn, \tag{5.3}$$

which corresponds to the definition of the convolution operation [50] applied on f_N and f_M.

Hence, the probability distribution of the sum of two statistical independent random variables can be determined by applying convolution on their individual distributions.

Applying Functions on Discrete Random Variables

Let M be a discrete random variable with realizations out of a subset of $\mathbb{R}$. A function $h : \mathbb{R} \to \mathbb{R}$ applied on M will lead to another random variable $N = h(M)$ that has realizations out of $\mathbb{R}$ as well. Mapping M onto N by a function h simply means mapping all realizations of M to the ones of N [22]. As a result, the dependence between the probability distributions $f_M(m)$ and $f_N(n)$ of M and N respectively can be described as follows:

$$f_N(n) = \sum_{m:h(m)=n} f_M(m) \tag{5.4}$$

The probability $f_N(n)$ of a certain realization $n \in N$ equals the sum of the probabilities $f_M(m)$ of all realizations $m \in M$ that will be mapped onto n by function h. Each n belongs to only one m, if h is invertible. This simplifies the equation to:

$$f_N(n) = f_M(h^{-1}(n)). \tag{5.5}$$

Maximum of a Discrete Random Variable

Let Z be a discrete random variable. The maximum $Z^{max} = max(Z)$ is defined as follows:

$$\forall z \in Z : Z^{max} = max(Z) \Rightarrow P(z > Z^{max}) = 0. \tag{5.6}$$

Maximum of a Discrete Random Variable Scaled by a Constant Factor

Let Z be a discrete random variable. Scaling Z by a constant factor β means scaling all realizations $z \in Z$ as learned before. Hence, one can easily derive that following dependence holds:

$$max(\beta Z) = \beta max(Z), \tag{5.7}$$

which will be needed later on.

5.1.3 Stochastic Processes

The idea of a stochastic process is to describe non deterministic behavior over time. In contrast to deterministic models (such as e.g. differential equations), different concrete realizations (concrete time series) of a stochastic process are possible, even if they would have been started under the same initial condition. Some of these concrete time series are more likely than other depending on the nature of the process.

One can distinguish between discrete-time and continuous stochastic processes. Continuous processes describe a behavior at each possible point in a continuous time. Discrete-time processes are limited to predefined time intervals. Time is discretized in most practical uses cases of stochastic processes. The reason is that observations of behavior are performed in discrete time steps as well in most cases. Outside temperature is measured in fixed time intervals for instance. One sample represents the temperature of a complete time interval. The same is true for prices of the stocks in a stock market.

The concept presented in this thesis describes the demand behavior of VMs using stochastic processes. These processes are characterized by observed data as well. Resource demand can be only sampled in discrete time steps comparable to temperature measures and stock prices. Hence, discrete-time stochastic processes will be used to describe this behavior as well.

A discrete-time stochastic process extends the idea of a random variable by an additional dimension: the time. One individual random variable describes the random behavior at each discrete time step. Hence, the discrete-time stochastic process simply is a sequence of individual random variables.

Stochastic processes are often characterized in conventional time series analysis using data observed. Hence, they initially can only describe the behavior within the period in which the characterization data was observed. Some assumptions concerning the modeled behavior are

made to use the processes for future extrapolation. One important of such assumptions is stationarity of the behavior and hence stationarity of the resulting stochastic process.

According to [32], the basic idea of stationarity is that

> *... the probability laws that govern the behavior of the process do not change over time. In a sense, the process is in statistical equilibrium. ...*

They formally defined a stochastic process $Y(t)$ (Y_t in their notation) as strictly stationary, if

> *... the joint distribution of* $Y_{t_1}, Y_{t_2}, ..., Y_{t_n}$ *is the same as the joint distribution of* $Y_{t_1-k}, Y_{t_2-k}, ..., Y_{t_n-k}$ *for all choices of time points* $t_1, t_2, ..., t_n$ *and all choices of time lag* k.

The authors of [32] derived from a special case of this definition (when $n = 1$) that all random variables Y_{t_x} have the same univariate probability distribution $f_{Y_t}(y)$. This property of stationary stochastic processes will be mainly required within this thesis.

5.1.4 Probabilities of Realizations of Stochastic Processes

The univariate probability distribution of a stochastic process $X(t)$ at each time t is defined by the respective random variable X. The probability distribution of concrete realizations (time series) $x(t)$ of the process in contrast can not be determined that easily.

Obviously, the probability distribution of $x(t)$ depends on the distributions of the single random variables involved. But even if all random variables have the same distribution, correlations between them influence the resulting distribution of a concrete time series. All concrete resulting time series would have a constant value, if all random variables are completely positively correlated[2]. Pairwise negatively correlated random variables would show a completely different behavior.

The random variables of all stochastic processes used within this thesis are assumed to be statistically independent. This assumption will be discussed later on individually for each respective case. But it can not be guaranteed that all random variables will have the same probability distribution because of missing stationarity.

The concept that will be presented in this chapter needs to calculate how many samples of any concrete realization $x(t)$ of such a stochastic process $X(t)$ are expected to exceed a certain threshold α within a certain time interval $[t_0, t_1]$. Formally expressed, the probability $P(x([t_0, t_1]) \geq \alpha)$ needs to be calculated.

All samples x in an interval $[t_0, t_1]$ of any concrete realization $x(t)$ of $X(t)$ can be regarded as a results of $|[t_0, t_1]|$ independent experiments due to the statistical independence of the random

[2] Of course, different repetitions of the experiment must lead to different time series to fulfill the probability distributions of the random variables. But all realizations would be constant time series because of the correlations.

variables of $X(t)$. Hence, they also could be a result of one single random variable $\hat{X}^{[t_0,t_1]}$ as well. The realizations of $\hat{X}^{[t_0,t_1]}$ are generated by subsequently generating realizations of all random variables X that belong to the process $X(t)$ in the interval $[t_0, t_1]$. This means that an infinite set of realizations of $\hat{X}^{[t_0,t_1]}$ contains equally sized shares of realizations out of each random variable X involved.

Hence, the probability $P(\hat{X}^{[t_0,t_1]} \geq \alpha)$ can be calculated from the random variables X of $X(t)$ in the interval $[t_0, t_1]$ as follows:

$$P(\hat{X}^{[t_0,t_1]} \geq \alpha) = \frac{1}{|[t_0,t_1]|} \sum_{\forall X \in X([t_0,t_1])} P(X \geq \alpha). \tag{5.8}$$

For sufficiently long intervals $[t_0, t_1]$, the probability that each possible realization $x([t_0, t_1])$ of $\hat{X}^{[t_0,t_1]}$ (and hence of the interval $[t_0, t_1]$ of $X(t)$ as well) exceeds α can be calculated the same way:

$$P(x([t_0,t_1]) \geq \alpha) = \frac{1}{|[t_0,t_1]|} \sum_{\forall X \in X([t_0,t_1])} P(X \geq \alpha) \tag{5.9}$$

For short intervals $[t_0, t_1]$, a correction term can be added to at least overestimate the probability $P(x([t_0, t_1])$. This fact will be picked up again in the section in which the content worked out here will be used.

5.2 Service Level Objectives

It has been shown in previous chapter how SLOs that define performance constraints can be used as constraint for resource management. The SLOs supported so far are very simple. A more complex SLO specification will be worked out within this section that additionally allows trading off resource capacity against service performance.

Therefore, first a deeper insight into SLOs is given that specify performance constraints. Known of these SLOs are presented as well. But it will be pointed out that the conventional way of specifying SLOs is quite inflexible. It does not allow trading off between resources and performance very well. Hence, a more fine grained way for ensuring QoS will be presented. It will be further shown how constraints can be derived from the SLOs that are directly usable for resource management.

5.2.1 Known Approaches

A common way for specifying SLOs is to define one performance goal in terms of e.g. throughput or response time by a threshold. Exceeding (or deceeding) the limit is treated as violation.

In general, such violations are accepted by users, if they are not happening very often, or if some how any kind of compensation is payed in case of violations[117]. Hence, different known resource management approaches try to take advantage of this fact by trading off required resource capacity provided to services against service performance mainly following two different ways.

For instance, it is suggested in [117, 102, 17, 66] that users can limit the amount of violations allowed in a predefined time period by SLOs. Resource capacity is assigned to the service by the resource management in a way that these limits are not exceeded using an appropriate model that describes the demand behavior of the respective service. Less restrictive SLOs can be provided cheaper by the SP this way because of a reduced amount of hardware resources needed.

The second group of approaches [130, 106, 28] assigns (financial) penalties to possible violations in SLOs. The more violations occur the more penalties must be payed by the Service Provider. Resource management concepts based on this kind of SLOs are trading off costs caused by providing resource capacity to costs caused when violating SLOs to achieve an economical optimum.

It is further suggested in [102] to limit the impact of violations by a threshold defined in a SLO. Performance slowdowns below this threshold must not occur in any case. Their resource management concept guarantees this SLO by exclusively reserving a fixed amount of resources for each service.

One big disadvantage of all of these specifications is the fixed performance goal. Regardless of how far away from the specified limit a performance slowdown is, it is treated as a violation with all respective consequences. It is not possible to allow weak performance slowdowns to occur more often while the occurrence of stronger ones is limited more restrictively. In principal, common SLA languages such as WSLA[62] or SLAng[106] allow defining different performance goals. But none of the known resources management approaches are taking any advantage of this opportunity.

The work presented in this thesis focuses on limiting the occurrences of violations similar to the first group of approaches. But a more flexible way for specifying performance goals will be suggested in contrast to them. Furthermore, the resource management concept presented in this chapter as well will be able to exploit this new specification for more efficient resource usage.

5.2.2 Fine Grained SLO Specification

The fine grained SLO specification associates performance goals with probabilities comparable to conventional SLOs. The probability states how often the goal must not be violated in a predefined time period. But the fine grained specification provides the opportunity to define

not only one but different performance goals for a performance metric (e.g. response time) in contrast to the conventional way. Each performance goal has an individual probability assigned. Weak performance slowdowns can be accepted more often this way, while stronger ones can be limited more restrictively.

In general, such SLOs can be expressed by a function $P_{\eta_i}^{min}(\eta)$ that returns a probability for each defined value η of the targeted performance metric. Hereby, each η represents an upper (or lower[3]) limit which must not be exceeded (or deceeded) more often than specified by the probability returned by the function.

An exemplary fine grained SLO is presented in Figure 5.1. The targeted performance metric for the service is the response time in this case. The performance goals and the respective probabilities are listed in Table a). Response times up to $1s$ must be achieved in 50% of time. The response time must not exceed $2s$ in 80% of time. And in 90% the response time must not pass the $3s$ limit. Finally, response times above $6s$ are not allowed at all.

a)

η	$\leq 1.0s$	$\leq 2.0s$	$\leq 3.0s$	$\leq 6.0s$
$P_{\eta_i}^{min}$	0.5	0.8	0.9	1.0

c) condition for meeting the SLO

$$\forall \eta^{max} \in dom(P_{\eta_i}^{min}), t_0, t \in [t_0, t_0 + \Delta t^{SLO}] : P(\eta_i(t) \leq \eta^{max}) \geq P_{\eta_i}^{min}(\eta^{max})$$

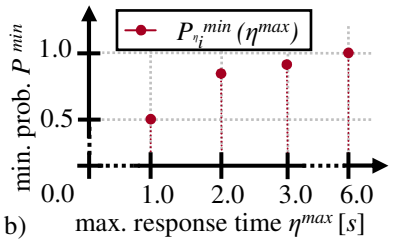


Figure 5.1: Exemplary fine grained SLO for the response time η of a service implemented in VM i. a) Probabilities are given that state how often different maximal response times must not be exceeded. b) This kind of SLO can be expressed by a function $P_{\eta_i}^{min}(\eta)$ that assigns one probability to each defined response time. Condition c) must hold to not violate the SLO. $\eta_i(t)$ is the actual response time of the service over time.

The respective function that belongs to this example is presented in Figure 5.1 b). The condition that must be met to not violate the SLO can be described by an equation using this function as presented in Figure 5.1 c). It must be ensured for each defined response time limit η^{max} that the actual response time $\eta_i(t)$ in each time period Δt_i^{SLO} does not exceed the limit with a probability equal or greater than specified by $P_{\eta_i}^{min}(\eta^{max})$.

5.2.3 Mapping Performance Metrics on Required Resource Capacity

Conventionally, SLOs do not directly refer to required resource capacity as the clients normally do not know how much resources processing their requests will take. They especially cannot

[3] depends on the performance metric

estimate, how much slower the service will get with less resource capacity provided. The question about the required resources has already been answered after having passed the benchmarks in phase one of the concept. But any kind of mapping is need to apply resource performance trade-offs that describes the dependency between the difference of provided and required resources and the resulting performance.

CPU time and memory are limited resources that are under direct control of the resource management concept as pointed out in Section 3.3.2. It was mentioned that missing memory capacity will directly lead to complete service failures. CPU time shortages, in contrast, will only slow down the service. Hence, only CPU time can be used to trade off resource capacity against service performance.

CPU time is traded off against service performance within this thesis by adjusting the ratio between provided and required CPU time like the authors of [111, 18, 91] already suggested. A new variable α is introduced that describe ratios that will lead to a performance slowdown. Possible values of α are limited by the interval $]0, 1]$. A value of 1 means that there is no resource shortage at all, which occurs when the provided CPU time A is higher as or equal to the demand R. If R exceeds A, α is the respective ratio of A and R. This dependence can be formally expressed by a function $g : \mathbb{R} \times \mathbb{R} \rightarrow]0, 1]$ as follows:

$$\alpha = g(R, A) = \begin{cases} \frac{A}{R} & \text{for } A < R \\ 1 & \text{else} \end{cases} \tag{5.10}$$

Finally, a way is needed that maps a resource shortage α onto a resulting value η of the targeted performance metric. It is assumed that a function $f(\alpha) : \alpha \mapsto \eta$ can be characterized for services implemented in a VM that returns a resulting η of the respective performance metric for a certain α. Other known resource management approaches such as described in [102, 117, 58] are based on the same assumption.

The modeling of this mapping itself is not part of this thesis since the modeling approach strongly depends on the service implemented in a VM. Different known works in the field of performance modeling already addressed this topic. This work can be split into different categories. Black box models, for instance, describe the demand behavior of services without knowing their internal behavior. They are typically characterized by observing the targeted performance metrics and the respective resource demand while varying the kind and amount of workload processed by them. Such a black box model based on fuzzy methods was presented in [130] that is trained at runtime. In principal, this approach can deal with any kind of service deployed in a VM. Different modeling approaches specialized to web service applications were presented in [11, 1]. Models based on request queues were proposed in [79, 23]. Some of these approaches are specialized to applications that directly run on hardware servers. The additional overhead caused by the virtualization environment can be estimated using the

approach presented in [128].

5.2.4 Deriving Constraints for Autonomous Resource Management

Once having found $f(\alpha)$, a function $P_{\alpha_i}^{min}(\alpha)$ can be derived from $P_{\eta_i}^{min}(\eta)$ as follows:

$$P_{\alpha_i}^{min}(\alpha^{min}) = P_{\eta_i}^{min}(f(\alpha^{min})) \tag{5.11}$$

This function states how often the ratio of provided and required CPU time must not fall below a lower limit α^{min} to not violate the SLO defined by $P_{\eta_i}^{min}(\eta)$.

A condition can be described by following equation comparable to $P_{\eta_i}^{min}(\eta)$ that decides whether or not providing resources $A_i(t)$ to the resource demand $R_i(t)$ of VM i will satisfy the respective SLO.

$$\forall \alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min}), t_0 : P(g(R_i(IV_{t_0}^{SLO}), A_i(IV_{t_0}^{SLO})) \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min})$$

with

$$IV_{t_0}^{SLO} = [t_0, t_0 + \Delta t_i^{SLO}] \tag{5.12}$$

The ratio of provided and required CPU time[4] must be equal or greater than the respective α^{min} for each defined resource shortage α^{min} with a probability equal or greater than specified by $P_{\alpha_i}^{min}(\alpha^{min})$ in each time interval $IV_{t_0}^{SLO} = [t_0, t_0 + \Delta t_i^{SLO}]$ to not violate the SLO. The SLO specification $P_{\eta_i}^{min}(\eta)$ will not be defined for all possible values of η in most cases due to practical reasons. Hence, $P_{\alpha_i}^{min}(\alpha)$ is not necessarily defined for all α as well. As a consequence, Equation 5.12 must be only evaluated for the subset of values of α, for which $P_{\alpha_i}^{min}(\alpha)$ is actually defined.

This equation will be used in conjunction with resource demand models to decide whether or not a certain combination of VMs will fit together at the same server with respect to their SLOs. The way this is done will be described in Section 5.4 in this chapter. The resource demand models will be described in the following section first.

5.2.5 Discussion

In principal, resource management can be regarded as a feedback control system. The varying amount of user requests to a service forms the disturbance variable. The resource management tries to adjust CPU time capacity provided to the service with respect to this disturbance. The goal is to keep a certain performance (e.g. the response time) constant or to at least ensure a minimal performance.

[4] described by function $g(R_i(IV_{t_0}^{SLO}), A_i(IV_{t_0}^{SLO}))$ that has been defined in Equation (5.10)

Only small interfaces connect the service with the resource management to be independent from the actual service implemented in a VM. The disturbance variable is not directly observed this way. Required CPU time demand R is measured as an indicator for varying user interaction with the service. Function $f(\alpha) : \alpha \mapsto \eta$ forms a second interface. This function must be characterized individually for each service to describes its performance depending on the ratio of CPU time demand R and provided capacity A.

These small interfaces can lead to problems for some services. First, function $f(\alpha)$ assumes that a constant α leads to a constant performance η independent from values of R. This might not be true for all services. Second, observing R can be a challenge in some cases especially when performance goals must be guaranteed. Measured CPU time demand can be strongly averaged depending on the sample rate. As a result, the measured value is often quite lower than the actual one. Providing resource capacity A to this measured R can result in an unexpected performance loss in this case.

A way out of this problem is a stronger connection between resource management and the service. User interactions are directly observed to overcome the problems caused by indirect measures. A more complex function must map the user interaction and provided CPU time capacity onto a performance measure. This function can than be directly evaluated in the condition presented in Figure 5.1 c) to decide whether or not a certain amount of CPU time capacity will satisfy the SLO.

This way will not be followed any deeper within this thesis since especially the mapping function typically strongly depends on the service implemented in the VM. Much work in the area of performance modeling already addresses this topic as already mentioned before. Future research can try to improve the work presented in this thesis for special classes of services.

5.3 Modeling the Resource Demand

A closer look into the characteristics of the resource demand behavior caused by a service implemented in a VM is provided at the beginning of this section. Requirements on the modeling approach are derived as well before the modeling approach itself will be presented. Finally, assumptions and limitations of the models are discussed. The way, these models are used for static resource management is not part of this section. Only the modeling approach itself and the characterization process are described.

5.3.1 Requirements on the Model

Models must describe the demand behavior of the VMs in a way that resource demand required in the future can be extrapolated from them as described in the problem statement chapter. They are trained using data $\vec{r_i}(t)$ observed in a characterization phase (between time t_{p_2} and

t_{p_3}). VMs are then distributed to servers based on these models. The resulting distribution will be fixed so that the resource capacity provided must meet the demand in the predefined time interval $[t_{p_3}, t_{p_3}+\Delta t_{p_3}]$ in future. Hence, there is no need for explicitly modeling the time dependence of the resource demand. It is only important to know how much resource capacity is required at a maximum in future but not exactly when this maximum will be actually used.

One must carefully consider the characteristics of the underlying reality for an appropriate modeling especially when extrapolation is required. Hence, the characteristics of a VM's typical resource demand will be shortly discussed in the following.

The resource demand of a VM can be split into a long term trend, a seasonal trend, and random noise as suggested in classical time series analysis. The three components of an exemplary resource demand time series are illustrated in Figure 5.2.

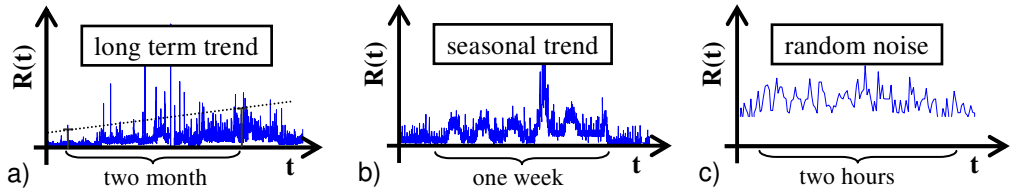


Figure 5.2: A resource demand time series of an exemplary VM. a) A long term trend is noticeable when a time periods of several month is regarded. b) A periodic seasonal trend is caused by daytime dependent resource demand which c) is overlaid by random noise.

The resource demand of many services shows a continuously increasing trend over time in most cases due to an increasing number of clients using the service [10]. Daytime varying workload causes a seasonal trend on top of the long term trend, which shows periodic behavior very often [44, 104]. And finally, the trends are overlaid by random noise. It will be shown later that this noise is not necessarily stationary.

An additional requirement on the models addresses the SLOs. The SLOs define different performance goals annotated with probabilities that define how often they must be met as described in previous section. Hence, one must be able to determine from the models how often which amount of resources is demanded by the VM to use these SLOs for resource performance trade-offs. This information is needed to determine the amount of resource capacity that must be provided to the VMs to fulfill their SLOs.

5.3.2 Known Approaches

Known statistical resource management approaches have been presented for instance in [102, 117, 58, 17, 45]. They all suggest to model the demand behavior by discrete-time stochastic processes. But different ways are suggested how to characterize them and how to use them for

resource demand prediction in the future. The authors of [117, 58, 45] simply model the demand behavior by one single random variable assuming stationarity all the time. Furthermore, statistical independence of the resource demand values along the time is assumed. Onces the model is characterized, it is expected to be valid in the future as well under the assumption of stationarity and statistical independence.

But resource demand behavior is not stationary because of time dependent workload variations. Hence, [102, 17] suggested dividing the time series into small slices (e.g. one hour) that are assumed to be stationary. One discrete random variable is characterized for each slice. Statistical independence of the demand behavior in each slice is assumed as well comparable to the first two approaches. The modeling approach followed in this thesis is based on the same assumptions. But in contrast to them one random variable is characterized for each time step. The reasons are detailed some more later on in this chapter.

A second class of approaches performs trace based resource management as presented for instance in [44, 43, 95]. Seasonal trends caused by daytime varying workload are used to optimize the distribution of VMs to servers by taking advantage of negative correlations. VMs that have contrary demand behavior over time are placed together at the same server. The seasonal trends are found by applying moving averaging. But the remaining noise is neglected, which can lead to unpredictable performance losses. The modeling approach presented in this thesis takes care of both: the noise as well as the seasonal trends.

Finally, some general approaches for modeling time series [33] known from classical time series analyses will be shortly discussed. A typical way is to split up the time series into three parts: a long term trend, a seasonal trend, and the noise. Both trends are modeled simply by deterministic functions. A challenge forms the random noise.

The classical way is to apply ARMA models that are a weighted combination of auto regression and moving average time series. ARMA models require stationarity as discussed in [34]. The authors of [104] found that this condition is not met by the usage behavior of typical services because of local indeterministic trends. An extension called ARIMA that tries to achieve stationarity through differencing can be applied to remove such trends.

But the resulting time series will still not be stationary because the noise performance is varying over time as well as the trend. Classical time series analyses suggest performing a logarithmic transformation in addition to differencing before the ARMA approach is applied. The assumption behind this transformation is an exponential dependence between trend and noise.

Indeed, in times of low utilization (resulting in a low mean value of the time series) the noise is low as well because resource demand is bounded below to zero. Stronger noise must lead to a higher mean value because the demand can not get negative. But the same is true for very high resource demand because of limited resource capacity. A trend (mean value)

of the resource demand close to the capacity boundary of the server leads to low noise as well. Otherwise the noise must exceed the capacity border, which is not possible. As a result, the logarithmic transformation as well as any other transformation by a strictly monotonic function will not work at all.

A completely different approach is followed by a further extension of the ARIMA model that is called SARIMA (seasonal ARIMA) [35]. In contrast to ARIMA, stationarity is not assumed along the time but along time points of instances of a period that is continuously repeating. If for instance the period is a week, similar demand behavior is expected every Monday morning at the same time only scattering around a fixed mean. Hence, each time point within the period is modeled by an individual discrete random variable. Stationarity and statistical independence is assumed over different instances of the period.

A disadvantage of the SARIMA model concerns the characterization process. The random variables are characterized using different instances of the period observed in the past. But the workload behavior of IT services is mainly dominated by weekly periods [44, 104]. Hence, the workload behavior must be observed for many weeks to get appropriately characterized models, which is quite impractical in data centers. The workload behavior can completely change already after one year [8].

The model presented in this thesis is inspired by the SARIMA approach. Finding a non linear transformation function to apply ARIMA seamed not to be a promising way regarding the results of some experiments with resource demand time series of real services. But in contrast to classical SARIMA, the random variables are characterized using data close to the respective time point assuming stationarity and statistical independence in closest proximity like also the authors of [102, 17] did.

5.3.3 Modeling Approach

The resource demand of a VM is modeled by a discrete-time stochastic process $Y_i(t)$ as mentioned in the problem statement chapter. This process is characterized using the resource demand $R_i(t)$ observed. The stochastic process consists of individual discrete random variables Y_i to take care of missing stationarity. Each random variable is described by its own probability distribution comparable to the SARIMA approach. Furthermore, the random variables are discrete, since the observed resource demand that is used for characterization is discrete as well. The resolution of memory demand is technically limited by one byte. Relative CPU time could be a continuous measure but is discretized as well in all common virtualization environments.

A probability distribution function completely describes a discrete random variable that models the resource demand at a certain time t. Probabilities state how often a certain demand is expected to occur. This information is exactly what is needed to support the SLO

specification worked out in the previous section. Resource capacity that must be provided to a VM can be estimated based on this probability distribution and the SLO. It will be described later on in Section 5.4 how this is done. This section is limited to the description of the model and the respective characterization process.

The model is split into two parts comparable to the SARIMA approach. The first part captures possible long term changes of the demand behavior. They are assumed to be strictly monotonic increasing or decreasing over time. The second part models seasonal trends and the noise behavior without any monotonic long term changes. Both parts of the model are presented separately in the following.

Modeling the Long Term Trend

Conventionally, a long term trend of a time series is assumed to be an additive component of the time series. This component is modeled by a polynomial that is fitted using linear regression. This modeling does not capture the resource demand behavior of a VM very well as can be seen in Figure 5.3.

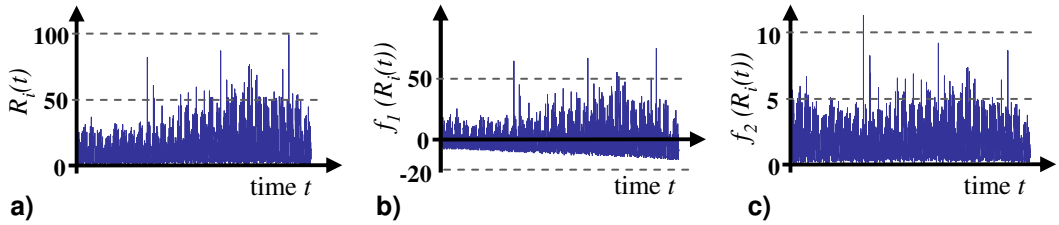


Figure 5.3: a) An exemplary time series of seven month resource demand that has a long term trend. b) The time series detrended the conventional way by subtracting a linear trend function from it. c) The time series detrended by dividing it by a linear trend function.

The seasonal trend as well as the noise performance seems not to be increased by an additive component with increasing time but scaled by a factor. Hence, it was tried to divide the time series by a linear function for detrending it as follows:

$$R_i^*(t) = R_i(t) \cdot \frac{1}{at + b}. \tag{5.13}$$

The resulting time series is presented in Figure 5.3 c). The parameters a and b were found conventionally using linear regression [51] as well. The idea behind this modeling is the assumption that the number of clients that use the service increases over time. This means that in times of high utilization an increased number of clients has to be served. The number of clients increases also in times of low utilization but not by the same number but the same

factor.

Based on this idea, the resource demand model can now be detailed some more by following equation:

$$Y_i(t) = Y_i^*(t) \cdot LT_i(t). \tag{5.14}$$

The function $LT_i(t)$ models the long term trend by a linear equation that is characterized using linear regression. The stochastic process $Y_i^*(t)$ captures the seasonal trend and the noise.

Modeling the Seasonal Trend and the Noise

Classical time series analysis suggests now splitting up the seasonal trend and the noise as discussed before. This is done for an exemplary time series in Figure 5.4. One can clearly see that the remaining residual $\epsilon_i(t)$ is not stationary which was also found in [104].

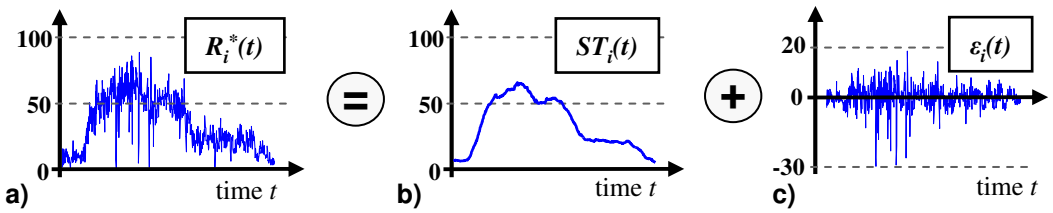


Figure 5.4: a) One day resource demand of an exemplary service. b) The seasonal trend derived using moving averaging. c) The residual noise that remains when the seasonal trend is removed from the time series.

As a result, splitting up the seasonal trend and the noise will not lead to any advantage. Instead, it is required that each random variable Y_i^* of the process $Y_i^*(t)$ is characterized individually comparable to the SARIMA approach. This means that one probability distribution for each variable must be derived from the data observed in history. Therefore, it is assumed that within a small time interval (e.g. half an hour) denoted by Δt^{avg} the trend as well as the noise performance is not significantly changing. Hence, small intervals $[t - \frac{1}{2}\Delta t^{avg}, t + \frac{1}{2}\Delta t^{avg}]$ of the time series $R_i^*(t)$ can be treated as if the demand behavior within these intervals is stationary. Furthermore, it is assumed that the resource demand values are statistically independent in these intervals.

Based on these assumptions, the probability distribution of the random variable Y_i^* that describes the resource demand at time t_0 can be derived from the data $R_i^*([t_0 - \frac{1}{2}\Delta t^{avg}, t_0 + \frac{1}{2}\Delta t^{avg}])$ observed in this interval. This approach is illustrated in Figure 5.5 for clarification. The validity of both assumptions will be discussed later on as well as the consequences for resource management when they are violated.

Until now, the model only describes the demand behavior of a VM during the characterization phase. In the following, it will be shown how the demand behavior expected in the future

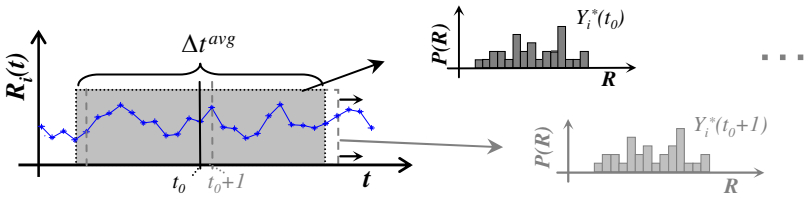


Figure 5.5: Each random variable Y_i^* of $Y_i^*(t)$ must be individually characterized using the data $R_i^*(t)$ observed in the past due to missing stationarity. Therefore, it is assumed that within a small interval around a time t_0 the demand behavior is nearly stationary and statistically independent. Hence, the probability distribution that describes the random variable Y_i^* at t_0 can be derived from the data within this interval.

can be extrapolated from these models.

Extrapolating into the Future

The part of the model that captures the long term trend is described by a linear equation. This model can be simply extrapolated into the future under the assumption that these long term changes will go on in the future the same way like observed during the characterization phase.

The part of the model that describes the seasonal trend and the noise should not show any monotonic behavior, since a possible long term trend has been removed from the data used for characterization. It should only describe changes in the demand behavior that are caused by day time dependent workload variations. It is further assumed that during the characterization phase of the models a time period with peak resource demand (e.g. at a Monday morning) was observed that never will be exceeded by resource demand in future[5], which can be formally expressed as follows:

$$\underset{\forall t \in [t_{p_2}, t_{p_2}+\Delta t_{p_2}]}{max}(Y_i^*(t)) \geq \underset{\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]}{max}(Y_i^*(t)). \tag{5.15}$$

As a result, a discrete random variable Y_i^{max} that describes the maximal resource demand of VM i within the time interval $[t_{p_3}, t_{p_3}+\Delta t_{p_3}]$ in future w.r.t. trends and the noise can be extracted from the model using following equation:

$$Y_i^{max} = \underset{\forall t \in [t_{p_2}, t_{p_2}+\Delta t_{p_2}]}{max}\left(Y_i^*(t) \cdot \underset{\forall t_{LT} \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]}{max}(LT_i(t_{LT}))\right). \tag{5.16}$$

The part of the model that describes the seasonal trend and the noise is scaled by the influence of the long term trend maximally expected in phase three. The random variable that describes

[5] after having purged a possible long term trend from it

the maximal resource demand is selected from the resulting stochastic process.

The definition of the outer function *max* depends on how the random variables are used for resource management. This will be described in Section 5.4. Hence, some more details will be presented to this function in this section as well.

5.3.4 Discussion

The modeling concept worked out in previous section is based on different assumptions. All of them will be discussed in the following subsequently with respect to their validity. Possible consequences for the resource management concept when they are violate will be discussed as well.

Modeling of the Long Term Trend

The described method to model the long term trend fits very well for the exemplary time series presented. This must not be true for any other time series of a VM's resource demand as well. It helps in any case to consider information about the reason of the trend to find the right method for modeling the long term trend as also stated by the authors of [36].

A wide range of modeling approaches can be applied depending on this reason. The model presented in this thesis is limited to long term trends that correspond to the assumption that the number of clients continuously increases over time as discussed before. It is further assumed that the resource demand of the service linearly depends on the number of clients that use the service. The whole modeling approach will be assessed in Section 7.2 using the resource demand traces of different services observed in real data centers. It will be shown how much the accuracy of the models can be increased, when the long term trend is explicitly modeled this way.

Stationarity and Statistical Independence of the Demand Behavior in Small Intervals

A critical point of the model is stationarity and statistical independence of the resource demand assumed in small time intervals Δt^{avg}. Samples around a certain time t_0 are treated as if they were different concrete realizations of a random variable that describes the behavior at t_0. This requires statistical independence of the samples observed, since realizations of a random variable are random as well and hence do not dependent on each other. Furthermore, stationarity is required in this interval. The resource demand values measured around t_0 must be realizations of random variables that all have the same probability distribution. This is required, since all realization of one and the same random variable are based on one fixed distribution as well.

One must say that resource demand samples in a small time interval are strongly correlated, if the resource demand is caused by one single request only. The same request repeated different times will more or less lead to the same sequence of resource demand values. But however, normally not only one but a plurality of requests is processed by a service. Hereby, the requests do not necessarily occur all at the same time but are distributed more or less randomly over time. Hence, the samples of the resulting overlay of the single resource demand sequences can be regarded as statistically independent.

Time dependent resource demand variations are the reason for missing stationarity in most cases. This concerns the seasonal trend but the noise performance as well. The stronger the changes are in a certain time interval, the higher are the characterization errors when stationarity is expected. Again, the plurality of clients of a service as well as their random behavior will prevent any abrupt changes in the demand behavior in most cases[6]. Hence, stationarity can be assumed in small intervals without leading to significant characterization errors.

Both assumptions are important for the resource management. The resulting models are used to find out how much resources must be provided to a VM to fulfill the respective SLO. Inaccurately characterized random variables can lead to SLO violations caused by underestimated resource demand. It will be shown in the evaluation chapter, if and how often these violations will actually cause any SLO violations for typical services.

5.4 Static Scheduling

The purpose of a static scheduling algorithm is now to distribute the VMs to servers based on the models worked out in the previous section. The aim is to minimize the number of required servers without violating any SLOs. Respective constraints to be met have been formally expressed by the Equations (3.3),(3.5), and (3.6) in Section 3.4.2.

In principal, the algorithm works the same way like the pessimistic one presented in the previous chapter. The VMs are assigned to servers one by one. First, it is tried to place them on servers to which other VMs have already been assigned before. If none of these servers has enough resource capacity left, a new one is taken. Heuristics decide about the order the VMs are assigned to servers. If more than one server can host a VM, again heuristics select the one to take.

The difference to the pessimistic approach is the way the jointly required resources $\vec{ja}_k(t)$ are derived from the resource demand models of the VMs with respect to their SLOs. Hence, mainly this topic will be addressed within this section. Furthermore, a way for sorting VMs

[6] Except when the service processes single batch jobs subsequently. Such job based services are not in the scope of this work. Completely other resource management strategies should be applied for them.

with respect to their resource demand is needed as well to support first-fit or best-fit bin packing heuristics. Such a way will be also presented.

5.4.1 Known Approaches

Statistical static management approaches that are similar to the one presented in this thesis have been suggested in [117, 58, 45]. They are all based on stochastic processes and perform resource management nearly the same way. The main idea of them will be shortly presented in the following as well as the main weaknesses.

These approaches model the resource demand of each VM by one single random variable $\overline{Y}_i$ for each resource type as mentioned in the modeling section. A random variable $\overline{JY}_k$ that describes the joint resource demand of the VMs is derived for each resource type as follows:

$$\overline{JY}_k = \sum_{i:B(i)=k} \overline{Y}_i. \tag{5.17}$$

This variable is used to decide whether or not a certain combination of VMs will fit together on the same server k.

It has been shown in Section 5.1.2 that the probability distribution of the sum of statistically independent random variables can be determined by applying convolution on their individual distributions. The authors of [117, 58, 45] suggests to determine the probability distribution that describes $\overline{JY}_k$ exactly this way under the assumption of statistical independence of these variables. The validity of this assumption will be discussed later in this section.

Once $\overline{JY}_k$ is found, the jointly required resources $JA_k(t)$ to be provided to the VMs to fulfill their SLOs must be derived. $JA_k(t)$ is a constant value $\overline{JA}_k$, since $\overline{JY}_k$ is time independent.

A common way for specifying SLOs has been introduced in Section 5.2. A minimal required probability P_i^{min} defines how often a defined performance goal must be satisfied. Such SLOs can be used to trade off resources against performance. The authors of [117, 58, 45] focused on such a SLO specification as well. They suggest to determine the resource capacity $\overline{JA}_k$ provided to the joint resource demand $\overline{JY}_k$ as follows:

$$P(\overline{JY}_k \leq \overline{JA}_k) \geq \underset{\forall i:B(i)=k}{max}(P_i^{min}). \tag{5.18}$$

The resource capacity $\overline{JA}_k$ provided to the VMs must satisfy the joint resource demand described by $\overline{JY}_k$ at least with the maximum of the required probabilities P_i^{min}. The maximum ensures that the most restrictive SLO of the VMs is satisfied. In principal, $\overline{JA}_k$ is selected as the P_i^{min}th percentile of $\overline{JY}_k$ using the most restrictive of the P_i^{min}s.

According to the first constraint for static resource management expressed by Equation(3.3) in the problem statement chapter, $\overline{JA}_k = JA_k(t)$ must be lower or equal to the server's resource

capacity C_k to fulfill all SLOs.

Based on this condition, classical bin packing algorithms can now decide whether or not a certain combination of VMs will fit together on the same server. But mainly two different assumptions prevent this approach from directly being used in real data centers. They will be discussed in the following.

Correlations

A major problem of this approach concerns the way the random variables $\overline{JY}_k$ are derived from the $\overline{Y}_i$s of the VMs. The convolution operation performed requires statistical independence which is not given in real data centers.

It can be mainly distinguished between two types of correlations. Structural correlations occur, if services are distributed across different VMs. Handling one request involves different VMs nearly at the same time in this case. Creating a dynamic web page, for instance, typically requires the work of a web server, a data base, and in some cases a storage system. Hence, the resource demands of the respective VMs are positively correlated. The second class of correlations, temporal correlations, typically occurs in data centers that support business activities at daytime. At night, most of the services lowly utilize hardware while they require maximal resources by day. Hence, their demand behavior is positively correlated as well.

The joint resource demand of different VMs will be either under- or overestimated when correlations are ignored depending on whether the random variables are positively or negatively correlated. An example for each case is illustrated in Figure 5.6. The probability of the highest resource demand is higher than the one that is calculated using convolution in case of positively correlated workload. Negatively correlated workload, in contrast, reduces the probability of high resource demand of different applications at the same time so that the calculated distribution overestimates the real one.

As a result, either resource are wasted or even SLO violations can occur when correlations are neglected while statistic resource management is performed. Two approaches will be presented within this thesis that deal with correlations in different ways. The first one pessimistically assumes completely positively correlated workload, which only leads to wasted resources but not to any SLO violations. The second one uses negative correlations for a more optimistic resource management, while it guarantees not to violate any SLO as well.

Interdependencies between Required and Provided Resource Capacity

Memory demand must be met by provided capacity at any time as discussed in Section 3.3.2. Provided CPU time, in contrast, can be traded off against service performance by defining appropriate SLOs. Resource shortages are tolerated as long as performance goals are not violated more often than specified. The approaches presented in [117, 58, 45] try to ensure

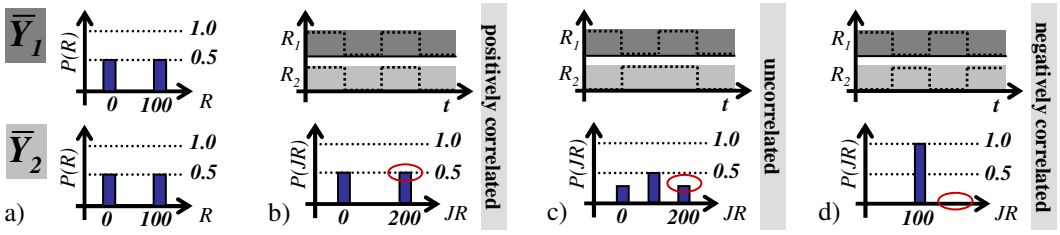


Figure 5.6: Shows the influence of correlations when two exemplary random variables $\overline{Y}_1$ and $\overline{Y}_2$ are added. Their respective probability distributions are presented in a). Three cases are regarded in which the variables are either positively correlated b), uncorrelated c), or negatively correlated d). Exemplary time series were selected one for each case. They fit to the variables and to the respective correlation as well. One can see that summing up the time series will lead to different probability distributions. Hence, the sum of different random variables depends on possible correlations between them.

meeting this condition. The probability distribution of the joint CPU time demand $\overline{JY}_k$ is used to decide whether or not a certain combination of VMs will fit together on the same server.

This method can only work, if $\overline{JY}_k$ is not influenced by actually occurring resource shortages. But this assumption is not true in most cases. An exemplary time series $JR_k(t)$ of the joint CPU time demand of different VMs will be analyzed in the following to point out the reason behind.

A certain amount of CPU time is needed by the VMs at each time step t. This demand can exceed the servers capacity C_k as intended by the statistical resource management approach. When the capacity is actually exceeded at t, a residual of the CPU time demand will remain. Hence it is demanded at time $t+1$ further on in addition to the initial demand $JR_k(t+1)$.

It is now assumed that the initial CPU time demand stated by $JR_k(t)$ will not change due to resources shortages as well. A second time series $JR_k^+(t, C_k)$ can be calculated from the initial one based on this assumption as follows:

$$\begin{gathered} JR_k^+(t, C_k) = JR_k(t) + \epsilon_k(t-1, C_k) \\ \text{with} \\ \epsilon_k(t, C_k) = max(\epsilon_k(t-1, C_k) + JR_k(t) - C_k, 0) \\ \text{and} \\ \epsilon_k(0, C_k) = 0. \end{gathered} \tag{5.19}$$

This time series additionally considers the remaining residuals. The residual $\epsilon_k(t, C_k)$ at time t is the sum of the residual remaining from time $t-1$ and the difference between the resource demand at t and the server's capacity C_k. The max function ensures that residuals do not fall

below 0 because unused CPU time cannot be used later for compensation. Furthermore, no residual can exist at time $t = 0$. The real CPU time demand $JR_k^+(t, C_k)$ at time t is now the initial demand $JR_k(t)$ plus the residual $\epsilon_k(t-1, C_k)$ that remains from time $t-1$.

An exemplary initial time series $JR_k(t)$ of joint CPU time demand as well as the resulting one $JR_k^+(t, C_k)$ are presented in Figure 5.7 a). One can clearly see that $JR_k^+(t, C_k)$ exceeds $JR_k(t)$ when resource shortages occur. Hence, the real demand $JR_k^+(t, C_k)$ will exceed C_k more often compared to the initial demand $JR_k(t)$, which results in an increased amount of resource shortages.

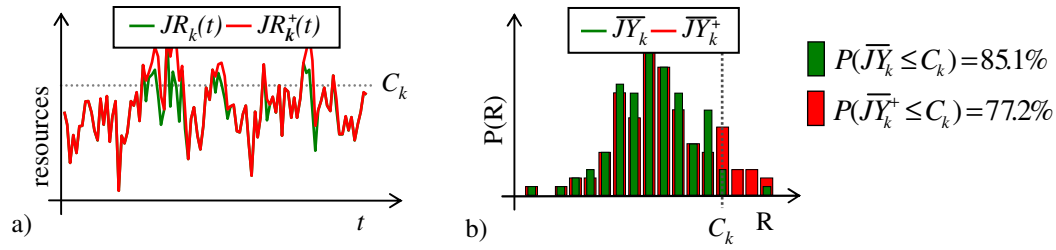


Figure 5.7: a) Resource demand $JR_k(t)$ of an exemplary time series and the resulting real resource demand $JR_k^+(t, C_k)$ when $JR_k(t)$ exceeds the server's capacity at some time. b) The respective probability distributions show that percentile based SLOs can be violated when interdependencies between resource demand and provided resources are neglected.

It is shown in Figure 5.7 b) that this changed demand behavior can lead to SLO violations. The initial time series $JR_k(t)$ does not exceed the server's capacity in 85.1% of time. The resulting real one will only meet the demand in 77.2%. A SLO is violated, if the respective P_i^{min} is higher than 77.2%.

5.4.2 Pessimistic Statistical Scheduling

Statistical independence required between the resource demand of different VMs was pointed out to be one of the major weaknesses of known approaches for statistical resource management. An approach for statistical resource management will be presented within this section that pessimistically deals with correlations. Later on, this approach will be extended to use negative correlations for a more optimistic assignment of VMs to servers.

No resources can be shared between different VMs without any further information about possible correlations. It is not clear, how often high resource demand of them will occur at the same time. Resources must be individually reserved for each VM the same way the pessimistic approach that has been described in Chapter 4 does. This way, even the worst case is considered, in which all VMs will demand their maximally required resources at the same

time.

Hence, the resources A_i^{max} maximally required by VM i in the interval $[t_{p_3}, t_{p_3}+\Delta t_{p_3}]$ in future must be determined for each resource type. It has been shown in the modeling section how a random variable Y_i^{max} can be derived from the resource demand model that describes the maximally expected resource demand in the interval $[t_{p_3}, t_{p_3}+\Delta t_{p_3}]$. A_i^{max} must be selected in a way that it provides enough resources capacity to meet the demand behavior described by exactly this random variable.

Y_i^{max} is selected as the maximum of the model $Y_i^*([t_{p_2}, t_{p_2}+\Delta t_{p_2}])$ scaled by the influence of a possible long term trend according to Equation (5.16). A size of the random variables Y_i that describe the resource demand of VMs must be defined to find this maximum. For the resource management concept, this size is simply defined by the minimal resources A_i that must be reserved to fulfill the SLO of the respective VM. A higher amount of required resources means a larger Y_i, which can be formally expressed as follows:

$$Y_i(t_0) > Y_i(t_1) \iff A_i(t_0) > A_i(t_1). \tag{5.20}$$

Hence, a function $q : Y(t) \mapsto A(t)$ must be found that determines the resources $A(t)$ required to fulfill the SLO with respect to the demand described by $Y(t)$. This function in conjunction with the method for finding Y_i^{max} (cf. Equation (5.16)) will result in following equation:

$$A_i^{max} = \max_{\forall t \in [t_{p_2}, t_{p_2}+\Delta t_{p_2}]} \left(q \left(Y_i^*(t) \cdot \max_{\forall t_{LT} \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]} (LT_i(t_{LT})) \right) \right). \tag{5.21}$$

The resources A_i^{max} maximally required by VM i in the interval $[t_{p_3}, t_{p_3}+\Delta t_{p_3}]$ in the future to fulfill their respective SLO can be directly calculated using this equation.

Different functions q must be applied to derive $A_i(t)$ from $Y_i(t)$ depending on the resource type. One function will be presented for memory and one for CPU time in the following.

Deriving Required Memory Capacity

The provided memory capacity $A_i(t)$ must meet the demand $R_i(t)$ all the time, since memory shortages can lead to complete service failures as discussed in Section 3.3.2.

Given a random variable Y_i that describes the memory demand at a certain time t, memory capacity A_i must be provided at t in a way that $P(Y_i > A_i) = 0$ holds. This leads to $A_i = max(Y_i)$ (cf. Section 5.1.2). Applying this to a whole stochastic process $Y_i(t)$ results in $A_i(t) = max(Y_i(t)) = q(Y_i(t))$. Hence, function q for memory determines simply the maximum of each random variable Y_i of the stochastic process $Y_i(t)$.

In principal, the resulting A_i^{max} should be the same like the maximal memory demand R_i^{max} determined using benchmarks in phase one of the concept. Hence, R_i^{max} could be directly used

for resource management. Nevertheless, the way required memory capacity is determined from the random variables that describe the memory demand behavior has been presented here. This method will be needed later on, when correlations are used for more optimistic statistical static resource management and for the dynamic resource management concept as well.

Deriving Required CPU Time

CPU time can be used to trade off provided capacity against service performance in contrast to memory. Provided CPU time A_i can be lower than the maximum of the demand Y_i, which can lead to performance losses as intended. These performance losses must be limited by the conditions derived from the SLOs.

A classical percentile based as well as a new more fine grained SLO specification have been presented in Section 5.2. Both are suited to support resource performance trade-offs. The resource management concept presented in this thesis can deal with both types of specifications. But only the way how fine grained SLOs are supported will be presented, since any percentile based SLO can be expressed by a fine grained one as well.

Fine grained SLOs can be defined by a function $P^{min}_{\eta_i}(\eta)$ as presented in Section 5.2. This function assigns a probability to each defined performance goal η_i stating how often this goal must be achieved. Additionally, it has been shown how a function $P^{min}_{\alpha_i}(\alpha)$ can be derived from $P^{min}_{\eta_i}(\eta)$ that defines how often a certain ratio α between provided resources A_i and resource demand R_i must not be deceeded. Finally, following condition for resource management has been derived in Section 5.2.4 that must be met to not violate the SLO:

$$\forall \alpha^{min} \in]0,1[\cap dom(P^{min}_{\alpha_i}), t_0 : P(g(R_i(IV^{SLO}_{t_0}), A_i(IV^{SLO}_{t_0})) \geq \alpha^{min}) \geq P^{min}_{\alpha_i}(\alpha^{min})$$

with

$$IV^{SLO}_{t_0} = [t_0, t_0 + \Delta t^{SLO}_i]. \tag{5.22}$$

This equation must hold for all possible realizations $R_i(t)$ of the stochastic process $Y_i(t)$ that describes the demand behavior of a VM. The major challenge is to calculate the probability $P(g(R_i(IV^{SLO}_{t_0}), A_i(IV^{SLO}_{t_0})) \geq \alpha^{min})$, which is needed to determine appropriate values for $A_i(t)$ with respect to the underlying process $Y_i(t)$.

First, a method will be presented that allows determining minimal resource capacity A_i required for demand described by Y_i to address this challenge. Resulting resource shortages must not exceed all possible values $\alpha^{min} \in]0,1[\cap dom(P^{min}_{\alpha_i})$ more often than specified by a certain function $P^{min}_{\alpha_i}(\alpha^{min})$. This condition can be formally expressed as follows:

$$\forall \alpha^{min} \in]0,1[\cap dom(P^{min}_{\alpha_i}) : P(g(Y_i, A_i) \geq \alpha^{min}) \geq P^{min}_{\alpha_i}(\alpha^{min}). \tag{5.23}$$

The transformation of a discrete random variable by a function leads to another discrete random variable [22]. Hence, the result X_i of the function $X_i = g(Y_i, A_i)$ is a discrete random variable as well because of Y_i. Y_i describes the resource demand R_i with respect to the noise. X_i describes the resulting resource shortages α_i, when a certain resource capacity A_i is provided to the resource demand Y_i.

The probability $P(g(Y_i, A_i) \geq \alpha^{min})$ can be calculated based on this dependence using the probability distribution $f_{X_i}(\alpha)$ of the random variable X_i as follows:

$$\begin{aligned} P(g(Y_i, A_i) \geq \alpha^{min}) &= P(X_i \geq \alpha^{min}) \\ &= 1 - P(X_i < \alpha^{min}) \\ &= 1 - \int_{\alpha \in]0, \alpha^{min}[} P(X_i = \alpha) d\alpha \\ &= 1 - \int_{\alpha \in]0, \alpha^{min}[} f_{X_i}(\alpha) d\alpha. \end{aligned} \tag{5.24}$$

The probability distribution $f_{X_i}(\alpha)$ is not known so far. It will be shown in a next step how this probability distribution can be derived from the known one $f_{Y_i}(R)$ of Y_i.

It has been shown in Section 5.1.2 how the probability distribution $f_N(n)$ of a discrete random variable N that is a mapping of another discrete random variable M can be calculated from Ms distribution $f_M(m)$. Applying the respective equation (Equation (5.4)) to the random variables Y_i and X_i that are mapped on each other by function g leads to:

$$f_{X_i}(\alpha) = \sum_{R: g(R, A_i) = \alpha} f_{Y_i}(R). \tag{5.25}$$

One can derive from the definition of function g in Section 5.10 that g reduces to the invertible function $g(R, A_i) = \frac{A_i}{R}$ for $\alpha < 1$. As a result, each value $\alpha \in]0, 1[$ will lead to exactly one value R. Hence, R can be calculated from α using $R = \frac{A_i}{\alpha}$, which will reduce Equation (5.25) to:

$$f_{X_i}(\alpha) = f_{Y_i}(\frac{A_i}{\alpha}) \text{ for } \alpha \in]0, 1[. \tag{5.26}$$

Inserting now Equation (5.26) into Equation (5.24) leads to following condition:

$$\forall \alpha^{min} \in]0, 1[\cap dom(P_{\alpha_i}^{min}) : \int_{\alpha \in]0, \alpha^{min}[} f_{Y_i}(\frac{A_i}{\alpha}) d\alpha < 1 - P_{\alpha_i}^{min}(\alpha^{min}) \tag{5.27}$$

that is equivalent to the one initially expressed by Equation (5.23). It can be now simply tested if a certain value of A_i will violate the SLO or not based on this equation and the probability distribution $f_{Y_i}(R)$ of the resource demand described by Y_i. Starting with R_i^{max}, possible values of A_i are iterated downwards to find the minimal A_i that barely will satisfy

the SLO.

In principal, this method can be applied individually for each time t to determine $A_i(t)$ from $Y_i(t)$. But it is not obvious that $A_i(t)$ determined this way will satisfy the condition this sections starts with (Equation (5.22)) as well. Determining $A_i(t)$ individually for each time t only ensures that the univariate probability distribution of the resource demand at each fixed time t will satisfy the SLOs. But this does not necessarily implicate that the same is true for all possible explicit time series $R_i(t)$ in each interval $IV_{t_0}^{SLO}$ as well (cf. Section 5.1.4). It will be shown in the following that under the assumption of statistical independence of the random variables of $Y_i(t)$ this implication is actually valid.

Proof. Formally expressed, it needs to be shown that if condition (5.23) meets for all random variables Y_i of the stochastic process $Y_i(t)$, the same will be true for any realization $R_i(t)$ in any interval $IV_{t_0}^{SLO}$ as well, which can be expressed by following equation:

$$\begin{aligned}
&\forall \alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min}), t_0, t \in IV_{t_0}^{SLO}, Y_i = Y_i(t), A_i = A_i(t) : \\
&\quad P(g(Y_i, A_i) \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min}) \Rightarrow \\
&\quad P(g(R_i(IV_{t_0}^{SLO}), A_i(IV_{t_0}^{SLO})) \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min}) \\
&\text{with} \\
&\quad IV_{t_0}^{SLO} = [t_0, t_0 + \Delta t_i^{SLO}].
\end{aligned} \tag{5.28}$$

First, each Y_i is mapped on a random variable X_i by function $g(Y_i, A_i)$, which turns the left part of the implication into $P(X_i \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min})$. All X_i together form the stochastic process $X_i(t)$. Now, a lower bound of all $X_i \in X_i(t)$ is defined by a new discrete random variable $\widetilde{X_i}$ as follows:

$$\forall \alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min}) : P(\widetilde{X_i} \geq \alpha^{min}) = \min_{\forall X_i \in X_i(t)} (P(X_i \geq \alpha^{min})). \tag{5.29}$$

For this $\widetilde{X_i}$ it holds that:

$$\begin{aligned}
&\forall X_i \in X_i(t), \alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min}) : \\
&\quad P(X_i \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min}) \Leftrightarrow P(\widetilde{X_i} \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min}).
\end{aligned} \tag{5.30}$$

Let now be $x_i(t)$ any possible realization of $X_i(t)$. As shown in Section 5.1.4, the probability that any $x_i(t)$ exceeds a certain threshold α^{min} in an interval $IV_{t_0}^{SLO}$ can be calculated from the individual statistically independent random variables involved as follows:

$$P(x_i(IV_{t_0}^{SLO}) \geq \alpha^{min}) = \frac{1}{|IV_{t_0}^{SLO}|} \sum_{X_i \in X_i(IV_{t_0}^{SLO})} P(X_i \geq \alpha^{min}). \tag{5.31}$$

With $\widetilde{X_i}$ being a lower bound of all X_i (as defined by Equation (5.29)) it further holds that:

$$\forall \alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min}), X_i \in X_i(t) : \\ \frac{1}{|IV_{t_0}^{SLO}|} \sum_{X_i \in X_i(IV_{t_0}^{SLO})} P(X_i \geq \alpha^{min}) \geq P(\widetilde{X_i} \geq \alpha^{min}). \tag{5.32}$$

If now $P(X_i \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min})$ holds for all $X_i \in X_i(t)$ and all $\alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min})$, $P(\widetilde{X_i} \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min})$ will hold as well according to Equation (5.30). In this case, the inequation $P(x_i(IV_{t_0}^{SLO}) \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min})$ will also hold for any possible $x_i(t)$ and all $\alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min})$ because of Equation (5.31) and (5.32).

Let now be $R_i(t)$ any concrete realization of the stochastic process $Y_i(t)$. Applying function g on $R_i(t)$ leads to a time series $x_i(t)$ that is a realization of $X_i(t)$[7]. In a final step, it needs to be shown that

$$\forall \alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min}) : P(x_i(IV_{t_0}^{SLO}) \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min}) \\ \Leftrightarrow P(g(R_i(IV_{t_0}^{SLO}), A_i(IV_{t_0}^{SLO})) \geq \alpha^{min}) \geq P_{\alpha_i}^{min}(\alpha^{min}) \tag{5.33}$$

holds, which is not obvious since function g is not invertible.

Therefore, one can first transform the left side as follows:

$$P(x_i(IV_{t_0}^{SLO}) \geq \alpha^{min}) = 1 - P(x_i(IV_{t_0}^{SLO}) < \alpha^{min}). \tag{5.34}$$

Regarding the range of α^{min} in Equation (5.33) and the definition of g (cf. Equation (5.10) in Section 5.2.3) one can derive that indeed

$$\begin{aligned} &1 - P(g(R_i(IV_{t_0}^{SLO}), A_i(IV_{t_0}^{SLO})) < \alpha^{min}) \\ &= 1 - P(\frac{A_i(IV_{t_0}^{SLO})}{R_i(IV_{t_0}^{SLO})} < \alpha^{min}) \\ &= 1 - P(x_i(IV_{t_0}^{SLO}) < \alpha^{min}) \\ &= P(x_i(IV_{t_0}^{SLO}) \geq \alpha^{min}) \end{aligned} \tag{5.35}$$

holds. The reason is that function g either returns the ratio between A_i and R_i or simply 1 depending on the parameters. A result of 1 will not influence the probability in Equation (5.35) for all $\alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min})$. Hence, only the invertible part of g needs to be regarded. ■

[7] $X_i(t)$ is a mapping of $Y_i(t)$ which means that all random variables X_i of $X_i(t)$ are mappings of the respective Y_i of $Y_i(t)$. Mapping a random variable Y_i on a variable X_i means mapping all possible realizations on each other by the respective mapping function. Hence, if $X_i(t)$ is a mapping of $Y_i(t)$, a mapping of a concrete realization of $Y_i(t)$ is a concrete realization of $X_i(t)$.

Please note that Equation (5.31) will be only valid for sufficiently long intervals $IV_{t_0}^{SLO}$ as discussed in Section 5.1.4. A correction term must be added to the probability $P(g(Y_i, A_i) \geq \alpha^{min})$ in Equation (5.23) for shorter intervals to determine $A_i(t)$. Such correction terms depend on the number of concrete realizations (samples in the interval $IV_{t_0}^{SLO}$ in this case) of a random variable. They can be characterized using known approaches out of the field of the theory of samples as presented for instance in [38]. This topic will not be detailed any deeper within this thesis.

Finally, function q that determines CPU time capacity $A_i(t)$ required to support the CPU time demand $Y_i(t)$ with respect to a SLO can individually determine A_i at each time t as described before.

Comparable to memory, Equation (5.21) can be used with function q to determine the CPU time capacity A_i^{max} maximally required in the future. But A_i^{max} can be lower than the maximally expected demand R_i^{max} in contrast to memory depending on the demand behavior of the VM and on how restrictive the SLO is.

Distributing VMs to Servers

Once the values A_i^{max} are found for each resource type of all VMs, the distribution of VMs to servers can be determined the same way the pessimistic static approach presented in Chapter 4 does. But the maximal CPU time A_i^{max} provided to VM i can be lower compared to the value found by the initial approach because of the resource performance trade-off.

Providing less CPU time than required can lead to resources shortages as intended, if the overall CPU time demand of all VMs placed on the same server exceeds its capacity. But it is important that required resource capacity A_i^{max} is individually reserved for each VM to prevent SLO violations caused by these resource shortages. This way, the amount of provided capacity never gets below a value that can lead to SLO violations. Common virtualization environments allow reserving such fixed amount of resource capacity individually for each VM as described in Section 3.4.1.

Finally, VMs need to be sorted to apply heuristics such as first-fit or best-fit, while the distribution of VMs to servers is planned. This can be done by their A_i^{max} values the same way the pessimistic approach does.

5.4.3 Interdependence between Required and Provided Resource Capacity

A method has been presented in previous section that allows trading off resources capacity against performance by providing less CPU time $A_i(t)$ than demanded. Resulting resources shortages will lead to demand behavior that differs from the one initially observed as discussed in Section 5.4.1.

Periods of the resource demand time series were modeled in previous work using one random variable. Hence, the interdependence between required and provided resource capacity changes the random variable itself. The concept introduced in this thesis models the demand behavior by individual random variables each for one discrete time step t. Hence, resource shortages at time t will not influence the random variable that describes the behavior at t but the one at $t+1$.

To consider remaining CPU time demand, a function $res : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ is defined first as follows:

$$\epsilon_i = res(R_i, A_i) = \begin{cases} R_i - A_i & \text{for } R_i > A_i \\ 0 & \text{else} \end{cases} \tag{5.36}$$

This function determines the remaining CPU time demand ϵ_i, when a certain amount of resource capacity A_i is provided to a demand R_i. The residual is simply the difference between R_i and A_i, when the demand is higher than the provided capacity. A demand lower than the capacity leads to a residual of 0 because unused CPU time can not be used later for compensation.

Function res can now be applied on the random variable Y_i that describes the demand behavior of VM i at a certain time t. A residual $E_i = res(Y_i, A_i)$ that is a random variable as well will be the result. This resulting random variable describes the remaining CPU time demand with respect to the random noise. The following equation can be used to determine the respective probability distribution $f_{E_i}(\epsilon)$ from the distribution $f_{Y_i}(R)$ of Y_i according to the outcomes of Section 5.1.2:

$$f_{E_i}(\epsilon) = \sum_{R:res(R,A_i)=\epsilon} f_{Y_i}(R). \tag{5.37}$$

Applying now function res on the whole stochastic process $Y_i(t)$ leads to the stochastic process $E_i(t) = res(Y_i(t), A_i(t))$. $E_i(t)$ describes the CPU time demand that could not yet be satisfied at each time t.

The remaining CPU time demand at time t increases the demand at time $t+1$. Hence, a stochastic process $Y_i^+(t)$ that describes the CPU time demand with respect to the initial as well as to the additional demand can be calculated as follows:

$$Y_i^+(t) = Y_i(t) + E_i(t-1). \tag{5.38}$$

The probability distribution of each Y_i^+ of $Y_i^+(t)$ can be calculated using convolution (cf. Section 5.1.2) assuming statistical independence of the random variables Y_i of $Y_i(t)$.

Finally, the function q worked out for CPU time in Section 5.4.2 must not be applied directly on $Y_i(t)$ to derive $A_i(t)$ due to the interdependencies between them. Instead, it must be applied

on the process $Y_i^+(t)$. For the rest of this thesis, deriving $A_i(t)$ from $Y_i(t)$ means that the step over $Y_i^+(t)$ is included as well.

5.4.4 Separating Seasonal Trend and Noise from Long Term Trend

Until now, the model $Y_i^*(t)$ was scaled by the influence of a possible long term trend first before the required resources $A_i(t)$ have been determined (cf. Equation (5.21) in Section 5.4.2). Later on, it will be required that first the resources $A_i^*(t)$ required for the resource demand described by $Y_i^*(t)$ are determined. After that, the resulting time series $A_i^*(t)$ is scaled by the influence of the long term trend, which finally leads to $A_i(t)$. It is not obvious that both ways lead to the same result especially with respect to the way, required CPU time is determined from the models. Hence, it will be shown in the following that actually both ways will result in the same.

Formally expressed, it needs to be shown that if a certain A_i^* satisfies the demand described by Y_i^* without any SLO violation, the same is true for $LT_i(t_x) \cdot A_i^*$ with respect to the resource demand $LT_i(t_x) \cdot Y_i^*$ and vise versa. The implication from A_i^* to $LT_i(t_x) \cdot A_i^*$ ensures that no SLO are violated, when the influence of the long term trend is scaled after the required resources have been determined. The implication back ensures that if A_i^* is the minimal resource capacity required to support Y_i^*, the resulting capacity $LT_i(t_x) \cdot A_i^*$ is the minimum for $LT_i(t_x) \cdot Y_i^*$ as well.

It was pointed out in Section 5.4.2 that the condition $P(Y_i > A_i) = 0$ must meet for memory demand Y_i and provided memory capacity A_i to not violate the SLO. Hence, it needs to be shown that:

$$P(Y_i^* > A_i^*) = 0 \Leftrightarrow P(LT_i(t_x) \cdot Y_i^* > LT_i(t_x) \cdot A_i^*) = 0, \tag{5.39}$$

which is obviously true regarding the fact that $max(LT_i(t_x) \cdot Y_i^*) = LT_i(t_x) \cdot max(Y_i^*)$ (cf. Section 5.1.2).

According to the condition to be met for CPU time to not violate any SLO(Equation (5.23) in Section 5.4.2), it must be shown that the following holds:

$$\begin{aligned} \forall \alpha^{min} \in]0,1[\cap dom(P_{\alpha_i}^{min}) : P(g(Y_i^*, A_i^*) < \alpha^{min}) \leq 1 - P_{\alpha_i}^{min}(\alpha^{min}) \Leftrightarrow \\ P(g(LT_i(t_x) \cdot Y_i^*, LT_i(t_x) \cdot A_i^*) < \alpha^{min}) \leq 1 - P_{\alpha_i}^{min}(\alpha^{min}). \end{aligned} \tag{5.40}$$

Function $g(Y_i^*, A_i^*)$ can be replace by $\frac{A_i^*}{Y_i^*}$, if only results of g are required that are lower than 1 as already discussed in Section 5.4.2. This replacement leads to the following:

$$g(Y_i^*, A_i^*) = \frac{A_i^*}{Y_i^*} = \frac{LT_i(t_x) \cdot A_i^*}{LT_i(t_x) \cdot Y_i^*} = g(LT_i(t_x) \cdot Y_i^*, LT_i(t_x) \cdot A_i^*), \tag{5.41}$$

which proofs that the equivalence stated in Equation (5.40) is actually given.

5.4.5 Using Correlations for Improved Statistical Scheduling

Until now, the statistical static resource management approach deals very pessimistically with possible correlations. It reserves enough resource capacity to support the worst case (positively correlated maximal resource demand of all VMs), even if this case will never occur. An example of such a case is presented in Figure 5.8 a). Both VMs are never demanding their maximal resources at the same time. Hence, individually reserving the maximum for each VM leads to wasted resources in this case.

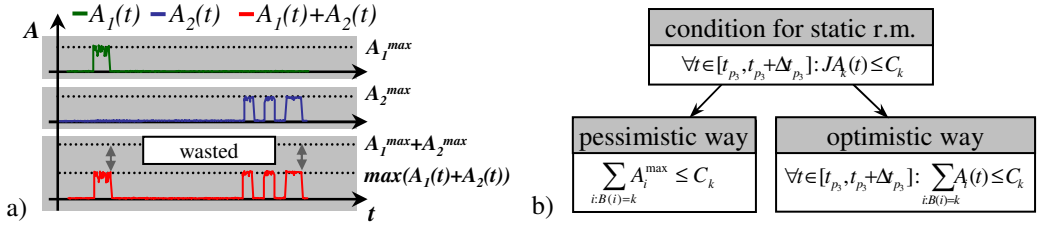


Figure 5.8: a) Individually reserving maximally required resources for each VM can waste resources. The overall required resource capacity can be reduced, if correlations are taken into account. b) Two conditions are derived from the initial resource constraint (top) worked out in the problem statement chapter. They can be used for resource management. The first one (left) pessimistically overestimates correlations. The second one (right) allows more optimistically using them for an improved assignment of VMs to servers.

Correlations can be taken into account, if the pessimistic condition (Figure 5.8 b), left side) applied until now is relaxed a bit inspired by the idea of trace based resource management [43, 95]. An assignment of VMs to a server will be still valid, if the sum of the individual required resource capacities $A_i(t)$ of the VMs does not exceed the servers capacity C_k at any time t. This leads to the condition presented at the right side of Figure 5.8 b). Negative correlations are used for a more optimistic planning this way. This more optimistic condition does not conflict with the initial one (Figure 5.8 b), top) worked out for static resource management in the problem statement chapter. The sum of the individually required resource capacities $A_i(t)$ is higher as or equal to the jointly required resources $JA_k(t)$ in any case.

First, the influence of the long term trend must be separated from the time series $A_i(t)$ to use this relaxed condition for resource management. This leads to following condition:

$$\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}] : \sum_{i:B_i(t)=k} LT_i(t) \cdot A_i^*(t) \leq C_k. \tag{5.42}$$

It has been shown in previous section that this condition is equivalent to the initial one.

But resource management can still not directly use this condition. The resource demand model $Y_i^*(t)$ required to determine $A_i^*(t)$ describes the resource demand during the characterization phase (time interval $[t_{p_2}, t_{p_2}+\Delta t_{p_2}]$) only. Hence, only the resources $A_i^*(t)$ that would have been required during this phase can be determined but not the resources required in the future.

An assumption already made in the modeling section is taken up again to solve this issue. It has been assumed that the maximal resource demand detected during the observation phase will not be exceeded by any demand behavior in the future (cf. Equation (5.15) in Section 5.3.3). Of course, the influence of the long term trend is excluded. This assumption is now extended to the joint resource demand of all possible sets of VMs as follows:

$$\max_{\forall t \in [t_{p_2}, t_{p_2}+\Delta t_{p_2}]} \left(\sum_{i:B(i)=k} Y_i^*(t) \right) = \max_{\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]} \left(\sum_{i:B(i)=k} Y_i^*(t) \right). \tag{5.43}$$

All trace based resource management approaches (e.g. [43, 95]) depend on this prerequisite as well. It will be discussed later on under which conditions this is assumption is actually given.

The same dependence exists between the resources $A_i^*(t)$ required in phase two and three under this assumption, which leads to following equation:

$$\max_{\forall t \in [t_{p_2}, t_{p_2}+\Delta t_{p_2}]} \left(\sum_{i:B(i)=k} A_i^*(t) \right) = \max_{\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]} \left(\sum_{i:B(i)=k} A_i^*(t) \right). \tag{5.44}$$

The influence of the long term trend expected in phase three can be overestimated as follows:

$$\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}] : LT_i(t) \leq \max_{\forall t_{LT} \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]} (LT_i(t_{LT})). \tag{5.45}$$

Both equations together can then be used to calculate the maximally required resources expected in phase three from the demand behavior observed in phase two as follows:

$$\begin{aligned} & \max_{\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]} \left(\sum_{i:B(i)=k} A_i^*(t) \cdot LT_i(t) \right) \\ \leq & \max_{\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]} \left(\sum_{i:B(i)=k} A_i^*(t) \cdot \max_{\forall t_{LT} \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]} (LT_i(t_{LT})) \right) \\ = & \max_{\forall t \in [t_{p_2}, t_{p_2}+\Delta t_{p_2}]} \left(\sum_{i:B(i)=k} A_i^*(t) \cdot \max_{\forall t_{LT} \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]} (LT_i(t_{LT})) \right). \end{aligned} \tag{5.46}$$

The resources $A_i^*(t)$ that would have been required in phase two are derived from the model

$Y_i^*(t)$ and scaled by the maximal influence of the long term trend expected in phase three.

Taken all together, the resource management concept must ensure that following equation holds to not violate the relaxed condition presented in Figure 5.8 b):

$$\max_{\forall t \in [t_{p_2}, t_{p_2}+\Delta t_{p_2}]} \left(\sum_{i:B(i)=k} A_i^*(t) \cdot \max_{\forall t_{LT} \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]} (LT_i(t_{LT})) \right) \leq C_k. \tag{5.47}$$

In principal, bin packing can now directly be performed to find an optimal distribution of VMs to servers. Equation (5.47) is evaluated for a certain set of VMs to decide whether or not they will fit together on server k.

But performing bin packing this way can require much computational effort depending on the sample rate and the duration of the observation phase. The algorithms must evaluate the equation for different sets of VMs, which requires summing up the time series $A_i^*(t)$ of all VMs in each set. The number of samples can be decreased before bin packing is performed to reduce this effort. Equidistant intervals of samples of $A_i^*(t)$ can be replaced by their maximum. But resource savings can get lost when required resources of different VMs are negatively correlated within this intervals. As a result, the reduction of the sample count to reduce computational effort required must be traded-off against achievable hardware savings.

5.4.6 Discussion

The approach for statistical static resource management presented in this section is mainly based on three assumption. It will be discussed in the following, in which cases they meet and how violations of them will affect the results.

Statistical Independent Random Variables of the Stochastic Process Y_i

The assumption of statistical independence of the random variables Y_i of $Y_i(t)$ has already been discussed in Section 5.3.4. The same is assumed again two times within this section to trade off CPU time against performance. The cases in which this assumption is actually valid have been discussed before. Hence, it will be only pointed out in this section how missing independence will impact the results.

First, statistical independence is required to ensure not to exceed the allowed probability of performances losses specified in SLOs in a given time frame (cf. Section 5.4.2). The worst case occurs, when all random variables that describe the demand behavior within the time frame are completely positively correlated. If in this case performance losses occur, they will occur during the whole time frame because of the positive correlations. The SLO will be violated.

Correlations must be explicitly modeled to deal with them in this case. These models must be characterized using external knowledge. Simply observing resource demand will not provide

any information about correlations because at each time t only one sample can be observed. This means that only one realization of each random variable can be observed, which is quite not enough to derive any information about correlations between them.

In addition, statistical independence of two successive random variables of the stochastic process $Y_i(t)$ is required to deal with interdependencies between resource demand and provided resource capacity (cf. Section 5.4.3). It was suggested to use convolution to derive the probability distribution of the sum of the residual demand ϵ_i and the initial demand Y_i. Incorrect probability distributions are calculated, if this assumption is violated. The real demand is underestimated in case of positive correlations. Less resource capacity than actually required is provided, which can lead to SLO violations.

A second option to deal with correlations is to pessimistically overestimate the resulting probability distribution of two added random variables the way presented in [58]. This method helps to prevent any underestimates, when the demand behavior model $Y_i^+(t)$ is derived from the initial one $Y_i(t)$ as described in Section 5.4.3.

Interdependence between Performance Loss and User Interaction

It has been shown in Section 5.4.1 that the demand behavior of a VM is influenced by provided resource capacity. Mainly residual resource demand that could not yet be satisfied by the capacity at a certain time t remains and hence increases the demand at time $t + 1$. This interdependence was addressed in Section 5.4.3.

The presented approach works only under the assumption that the initial resource demand behavior described by $Y_i(t)$ does not depend on the provided resource capacity. This is actually not true in any case because resource shortages will slow down the response time of requests.

A sequence of requests and answers is typically sent between a client and the service. An increased response time of the service can delay the following requests of the client. This shifts the respective resource demand caused by the sequence. Such shifts can lead to reduced but also to increased resource demand in the future depending on the overall workload behavior.

Detailed information about the service deployed in the VM are required to analyzing this kind of user influence. This interdependence is not pursued any deeper within this thesis, since the scope of the thesis is limited to a black box view of the service only. Future work must address this issue.

Using Negative Correlations Requires Periodic Demand Behavior

The idea of trace based resource management has been picked up in previous section to take advantage of negatively correlated resource demand of the VMs. Resource savings are achieved by putting VMs together at the same server that show complementary resource demand behavior caused by negative structural or temporal correlations.

Known approaches, such as presented in [43, 95], try to derive correlations from the demand behavior of the VMs observed in the past. It has been simply implied that negatively correlated resource demand observed in the past leads to negatively correlated resource demand in the future as well. Real negative correlations (temporal or structural) are assumed behind the observed ones. The concept presented in Section 5.4.5 is based on the same assumption as well. Hence, in the following it will be discussed under which condition this assumption is actually valid.

It has been distinguished between structural and temporal correlation earlier in this thesis. Structural correlations are caused by dependencies between VMs. Temporal correlations are caused by time depended workload variations.

Structural correlations mainly depend on the kind of requests processed by the VMs. Some requests involve different servers others do not. Conventionally, this behavior does not change over time. Requests that require the work of two different VMs will require the work of both VMs in the future as well. Hence, possible structural correlations observed will be still valid in the future, if during the characterization phase all possible requests and sequences of requests ever expected in the future have been observed.

The situation is more difficult for temporal correlations since they depend on time. It has been assumed that all kinds of requests that could possibly occur in the future have already been observed in the past to deal with structural correlations. This is definitely not given for the time and temporal correlations. One point in time observed in the past will never occur in the future again. The same is true for intervals of time. Hence, one can not derive correlations in the future from the ones observed in the past without any restriction.

An example can be simply constructed. Two VMs show strongly daytime dependent workload behavior. Over night both have nothing to do. At about noon both have their peak resource demand. One of the VMs shows this behavior every day in a week. The demand behavior of the second one differs during the weekend. After both VMs have been observed for a few days, one would say that their workload is strongly positively correlated. But the demand behavior turns out not to be that positively correlated, when the weekend is reached.

This fact has been completely neglected in [95]. The authors of [43] instead suggest to perform trace based resource management only on VMs that show periodic resource demand behavior. A period of the resource demand time series repeats over and over again in this case (e.g. the resource demand during a week is repeating every week). Hence, different instances of the period observed in the past can be used to predict the demand behavior expected in the future. The same is true for possible correlations of the demand behavior of different VMs. All VMs without any periodic behavior need to be treated pessimistically by the resource management by reserving A_i^{max} all the time. It is unknown when in the future they will demand their maximal required resources.

The model worked out for static resource management within this chapter will be extended for dynamic resource management later on. It will be needed to determine whether the demand behavior is periodic or not. Hence, a method for finding respective periods will be presented that can also be used to decide whether or not the VMs can be used for the optimistic static resource management approach.

One aspect not yet mentioned in [43] concerns the case when VMs with periodic demand behavior have different periods. The instances of the periods of different VMs are shifted against each other over time, which affects the observations. The characterization phase must have at least a duration of the lowest common multiple of the periods of all VMs with periodic behavior. This ensures that all possible cases that can ever occur have been observed. The resource demand of a typical service shows periods of one day, two days, or one week. Hence, a duration of the characterization phase of a view weeks is already enough in most cases. VMs with absolute inappropriate periods should be treaded as non periodic ones. This allows taking advantage of negatively correlated workload of the rest of the periodic VMs.

5.5 Changes in Demand Behavior

The resource management concept presented in this chapter imposes mainly two conditions on the demand behavior of the VMs in the future. First, the long term trend observed during the characterization phase will go on in the future as well. Second, the resource demand in future purged by the long term trend will not exceed the maximal demand observed during the characterization phase especially with respect to the noise performance. It will be shown in the following how violations of these condition will affect the VMs, how they can be detected, and what can be done to prevent negative impact of such violations.

5.5.1 Impact of Changed Demand Behavior

Changes of the seasonal trend or the noise performance can lead to increased resource capacity required to not violate SLOs. Some capacity reserves are present most of the time when VMs have an increasing long term trend. These reserves are needed to provide enough resources capacity for some time in the future. Furthermore, unused capacity remains at a server in most cases because no further VMs could be supported by this residual capacity. Hence, no SLO violations will occur in most cases, if the resource demand only slightly exceeds the expected one.

But an increased long term trend, in contrast, can lead to continuously occurring resource shortages. Capacity reserves could be depleted before the end of phase three is reached depending on how strong the actual trend exceeds the expected one. The data center must be reconfigured earlier than expected.

5.5.2 Detecting Changed Demand Behavior

Both kinds of changes of the VMs' demand behavior can be detected at runtime in most cases before any resource shortage will occur. Respective procedures are illustrated in Figure 5.9.

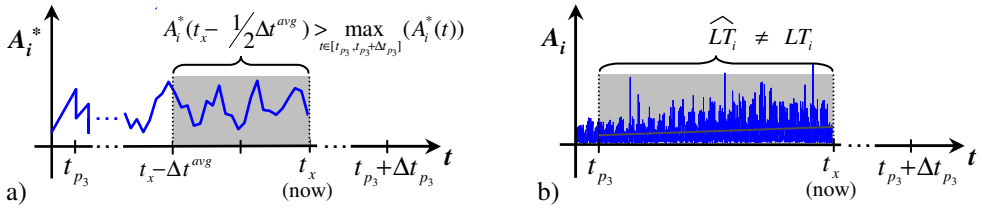


Figure 5.9: Procedures for detecting changes of a) the seasonal trend or the noise performance and b) the long term trend of a VM's demand behavior.

In principal, a new instance of the resource demand model is characterized using data observed at runtime. This new model is compared to the one characterized during the characterization phase.

Changes of the seasonal trend and the noise that occurred at a time $t_x - \frac{1}{2}\Delta t^{avg}$ can be detected at time t_x. The resource capacity A_i^* that would have been required at $t_x - \frac{1}{2}\Delta t^{avg}$ is determined from data observed in the interval $[t_x - \Delta t^{avg}, t_x]$ the same way like used to build the initial model (cf. Section 5.4.2). Changed demand behavior has been detected, if any of these A_i^*s exceeds the maximal required resources determined during the characterization phase.

Nearly the same approach can be performed for the long term trend. A second function $\widehat{LT_i}$ is continuously characterized using data observed at runtime. The long term trend has increased, if the slope of this function exceeds the one of the initial function LT_i.

5.5.3 Preventing SLO Violations Caused by Changed Demand Behavior

Once changes have been detected that lead to increased resource demand, the operator of the data center must react. He can determine how much earlier the reconfiguration of the data center must be performed based on the models. Such an approach is a well known task of classical capacity planning in data centers[10].

5.5.4 Discussion

SLOs have been worked out in Section 5.2 that mainly define a minimal quality of the service (in terms of response time or throughput). The resource management decides about the resource capacity assigned to the VMs based on these SLOs and on the user behavior observed.

The validity of the resulting distribution of VMs to severs can only be guaranteed, if the demand behavior does not change with respect to the long term trend, the seasonal trend, and the noise performance. But such a condition is conventionally not part of the SLO. Hence, the clients can provoke SLO violations to obtain financial penalties from the Service Provider.

As a result, varying demand behavior can be only taken into account for resource management, if additional conditions are integrated into the SLOs. These conditions must define, how demand behavior (or service usage) will develop in the future. Such extended SLOs are not worked out within this thesis any more. Ongoing future work must address this topic.

5.6 Summary

A new concept for statistical static resource management in virtualization based data center has been presented within this chapter. This concept allows assigning a set of VMs to servers in a way that the number of required servers is minimized. SLOs of the services are considered. This approach additionally trades off resource capacity against service performance in contrast to the pessimistic one presented Chapter 4. Furthermore, negatively correlated resource demand of different VMs is used for a more optimized assignments of VMs to servers. It is expected that these extensions can significantly reduce the number of servers required for a set of services compared to the pessimistic approach. Less energy is required in a data center this way to deploy the services. Investment costs can be saved as well.

This concept supports a new kind of SLO specification that allows trading off resource capacity against service performance in a very fine grained way. Individual probabilities can be specified for different performance goals. They state how often the respective goal must be achieved to not violate the SLO.

Furthermore, a new kind of resource demand modeling has been developed for the concept. The models are needed to determine resources required by a VM in a predefined time interval in the future and are characterized using observed demand behavior. They consider long term and seasonal trends and in contrast to previous work especially the non stationary random noise as well.

Finally, a method has been introduced that based on the models and the SLO specification can derive resource capacity required to meet the demand of a VM. This method considers interdependencies between the resource demand and the provided capacity in contrast to previous work. Such interdependencies can occur, when resource capacity is traded off against performance. Finally, it has been shown how conventional algorithms for heuristically solving vector bin backing problems can be applied to find an optimal assignments of VMs to servers based on the determined resources capacity for each VM. Correlated demand behavior of different VMs has been taken into account.

The presented approach can guarantee not to violate any SLO in a predefined time interval in future based on different assumptions. These assumptions have been exhaustively discussed within the respective sections in this chapter.

6 Dynamic Resource Management

The statistical static resource management worked out in previous chapter finds an optimal static distribution of VMs to servers at the end of the characterization phase. The concept tries to minimize the number of required servers to save hardware resources and energy.

Dynamic resource management can additionally draw on two further degrees of freedom that can be used to further reduce the energy consumption. VMs can be moved at runtime between different servers. Servers can be switched on and off. This way, active hardware resources can be dynamically adjusted to the varying resource demand to save energy in times of low overall resource demand.

The purpose of the dynamic resource management concept presented in this chapter is to realize such a dynamic system. Since both control mechanism (VM migration and server activation) take time, a major challenge of the concept is to trigger them right in time. Upcoming resource shortages must have been resolved right before they would actually occur to not violate any SLOs.

For this, respective models must be developed that can be used to forecast the time dependent demand behavior of VMs in the future. Furthermore, a scheduling algorithm is required that dynamically redistributes VMs and switches servers on and off. Finally, the concept must be able to deal with unexpected changes of the VMs' resource demand behavior. These challenges are addressed within this chapter.

6.1 Theoretical Background

It will be referred to some theoretical background while the dynamic resource management approach is presented in this chapter. This background will be laid in this section.

The modeling part requires methods for finding periods in time series. They will be presented in this section first. The dynamic scheduling approach ensures some constraints using a graphs. Required algorithms known from classical graph theory are presented in a second part of this section.

6.1.1 Autocorrelation Analysis

Performing autocorrelation is a common way in classical time series analysis to find dependencies between the individual samples of a time series [36]. A so called sample autocorrelation function $p(s)$ is derived from a time series $R(t)$ with length Δt_R as follows:

$$p(s) = \frac{\sum_{t=1}^{\Delta t_R - s}(R(t) - \bar{R})(R(t+s) - \bar{R})}{\sum_{t=1}^{\Delta t_R}(R(t) - \bar{R})^2}. \tag{6.1}$$

The covariance of $R(t)$ and $R(t+s)$ (R shifted by a lag of s) is calculated and normalized by the variance of $R(t)$.

In principal, this function is a measure for correlations between a time series and the time series shifted by a certain lag. Values between -1 and 1 can be returned by $p(s)$. 1 means a complete positive correlation (e.g. in case of a lag of 0). A values of -1 indicates a negative correlation (e.g. $sin(t)$ shifted by a lag of π). A plot of $p(s)$ versus lag s is called a correlogram [36].

It will be suggested within this chapter to apply an autocorrelation analysis to find the predominant period of a VM's resource demand. More details about this purpose will be presented in the respective section.

6.1.2 Testing Whether a Graph is Acyclic

Given a directed graph, two known approaches for finding out whether or not it remains acyclic after an edge is added are presented in the following. The first one (taken from [4]) represents the graph by a common adjacency matrix. An algorithm tries to topologically sort the nodes to find cycles after an edge is added. The second one [14] in contrast represents the graph in a way so that one can directly find out whether adding a certain edge will lead to any cycles or not. Some more computational effort must be spent, when the edge is actually added to adapt the representation.

Topology Sorting Based Test

The topology sorting based test searches for so called backward edges in a directed graph (or directed multigraph) that indicate cycles. The two steps of this algorithm are illustrated in Figure 6.1.

The algorithm starts with a classical depth first search to enumerate the nodes of the graph in postorder [5]. It has been shown in [6] that a backward edge has been detected between two nodes S_x and S_y, if S_y is an successor of S_x[1] but the post order of S_x is lower than the

[1] A successor of a node S_x is a node S_y that is the destination node of an directed edge from S_x to S_y.

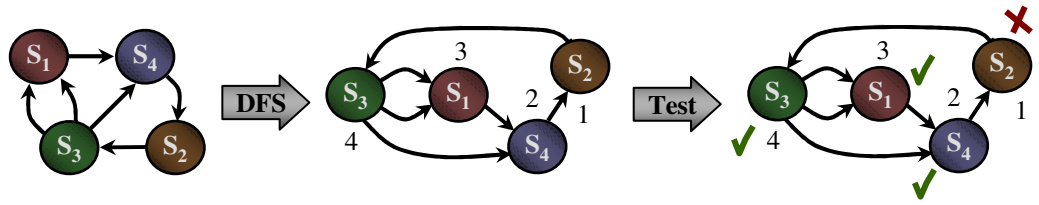


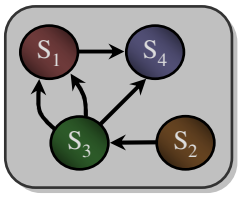
Figure 6.1: Overview of the algorithm that finds cycles in a directed (multi-)graph.

one of S_y. Hence, the algorithm tests in a second step, if all successors of all nodes have lower postorder numbers than the nodes itself. A cycle has been detected, if this test fails.

The runtime complexity of the depth first search depends linearly on the number of edges or nodes depending on which of the values is the higher one [5]. The complexity of the test itself depends linearly on the number of nodes.

Matrix Based Test

The matrix based test represent the directed graph by a quadratic matrix M, from which one can directly derive whether or not a certain operation will lead to cycles in the graph. The rows of the matrix represent nodes, from which paths to other nodes start. The columns are the destination nodes of possible paths. Each entry $m_{i,j}$ in the matrix contains the number of paths that start at node i and end in node j. An example is presented in Figure 6.2.



	S_1	S_2	S_3	S_4
S_1	0	0	0	1
S_2	2	0	1	3
S_3	2	0	0	3
S_4	0	0	0	0

Figure 6.2: Exemplary graph and its matrix representation. Each element $m_{i,j}$ in the matrix contains the number of paths from node i to node j in the graph.

Adding now an edge to the graph from node S_i to node S_j will only lead to a cycle, if already a path from S_j to S_i exists. This information can be directly figured out using the matrix.

The matrix needs to be adapted, when actually an edge is added. An respective algorithm is presented in [14]. The worst case runtime complexity of this algorithm quadratically depends on the number of nodes[2].

[2] In [14], the complexity has been bounded some more to the number of nodes ending at incoming and outgoing paths

6.2 Service Level Objectives

First of all, the dynamic resource management concept should support the same SLOs like the statistic static resource management does (cf. Section 5.2). A fine grained SLO specification associates performance goals with probabilities that state how often the goal must be achieved. These SLOs must be considered while the resource capacity $A_i(t)$ required for a VM is derived from the resource demand $Y_i(t)$ comparable to the static approach. The respective constraints derived in Section 5.2.4 must meet for the dynamic resource management as well.

In addition, two further limits are needed as already discussed in Section 3.2.3. Resource shortages can occur in case of unexpected demand behavior. But the client of a service must be able to limit the duration as well as the strength of resulting performance slowdowns. For this, two further parameters are introduced.

First, a client or Service Provider can define a maximal time period $\Delta t_i^{resolve}$. Performance problems of the service deployed in VM i must be resolved within this time period. Second, a minimal amount of resource capacity A_i^{min} can be defined for each VM that is individually allocated. It must be guaranteed by the resource management that even in case of a resource shortage the respective minimum is available for each VM. This guarantees a minimal performance of the service in any case.

The fine grained SLOs are addressed by the concept already within the modeling section in this chapter. The way, the concepts deals with the additional two parameters is described later on in Section 5.5.3. First, it will be worked out how the dynamic resource management deals with changed demand behavior at all.

6.3 Modeling the Resource Demand

The model required for dynamic resource management has nearly the same purpose like the one for static resource management. The resource demand of VMs observed in a trainings phase must be modelled in a way that the required resource capacity expected in the future can be extrapolated from the models. In contrast to static case, the dynamic resource management obtains its energy saving potential from time dependent workload variations. Hence, the models for dynamic resource management must be able to explicitly consider the time dependence of the resource demand. The model must be able to forecast exactly when a certain amount of resource capacity will be required in the future.

This section starts with a more detailed description of the requirements on the models. A discussion of different known forecasting approaches follows. Finally, the modeling concept developed for static resource management is extended in a way that it supports the requirements of dynamic resource management. Therefore, some ideas of known approaches will be picked up again.

6.3.1 Requirements on the Model

In principal, the model for dynamic resource management must meet nearly the same requirements like the one used for statistical static resource management. It must describe the resource demand of VMs observed during the trainings phase in a way that the resource capacity required in future to fulfill the SLOs of the VMs can be derived.

In contrast to the static case, not only the maximally required resource capacity expected in the interval $[t_{p_3}, t_{p_3}+\Delta t_{p_3}]$ in the future needs to be forecasted. The goal of dynamic resource management is to save energy by consolidating VMs to only a few servers in times of low overall resource demand and switching off the unused servers. Hence, the algorithm that dynamically redistributes VMs at runtime must be able to determined maximally required resource capacity expected in different sub intervals of $[t_{p_3}, t_{p_3}+\Delta t_{p_3}]$ from the models. It can decide whether or not a certain reassignment of VMs can be performed or not based on this information.

A basic approach to support such requirement is to use periodic behavior of the seasonal trend and the noise. Similar sequences of demand behavior (for instance the ones of different days) are assumed to repeat over and over again. A model characterized by some of them can be used to forecast the following ones in the future as well. This idea is the basis of the model presented in this thesis as well. Hence, the behavior of the resource demand of typical services in a data center will be discussed some more concerning periodicity in the following. Further requirements on the modeling approach are derived from this analysis.

Most services in a data center show typically periods of multiples of days or weeks [44, 104]. But other periods are possible as well [17]. Furthermore, different periods can overlay each other. Typical daytime dependent demand behavior can be overlaid by a batch job that is processed every 36 hours for instance. Finally, periods can be interrupted by single discontinues such as bank holidays. The modeling approach must take care of these characteristics.

Finally, the dynamic resource management concept should support trade-offs between provided resource capacity and service performance like the statistical static one does. Hence, the models should support the fine grained SLO specification introduced in the previous chapter as well.

6.3.2 Known Approaches

Different models for extrapolating time series into the future have been presented for resource management approaches and for several other application areas (e.g. weather forecast, forecasting stock prices, ...) as well. They are mainly based on the same principals known from classical time series theory. These main principals will be presented and discussed in the following.

The classical way separates the time series into a deterministic long term trend, a determin-

istic or stochastic seasonal trend, and a random noise. Individual modeling approaches can be applied for the different parts.

The long term trend is classically regarded as an additive strictly monotonic increasing or decreasing component. It is typically modeled by a deterministic function (e.g. an polynomial) that is fitted using regression methods. The influence of the long term trend can be extrapolated for nearly an arbitrary time into the future due to the analytical form of the models. The modeling approach developed within this thesis works nearly the same way. But the long term trend is not regarded as an additive but as an scaling component in contrast to the classical way. The reasons have already been discussed in Section 5.3.3.

The seasonal trend and the noise can be extrapolated by one of the following two principal ideas. The first kind of approaches extrapolates the trends observed in closest proximity to the current time to forecasts into the closest future. The random noise around this trend is assumed to be stationary. Resource management approaches that follow this idea were presented for instance in [23, 114, 72, 71, 15]. But such auto regression based methods can only safely forecast the time series for a few sample into the future. Dynamic resource management approaches require forecasts of at least half an hour. Hence, these kind of forecasting approaches are not suited very well.

Most other approaches focus on periodic behavior similar to the approach developed for this thesis. A model describes one (or more) periodic components that is(are) assumed to repeat over and over again in the future. Different known approaches require a manually specified predominant period (e.g. one week or one day) [27, 104, 102]. Others use an autocorrelation analysis to automatically determine appropriate periods [44, 43]. The authors of [17] extends such autocorrelation based approaches in a way that they can also deal with overlaid periodic components in the time series by decomposing it. But it remains unclear how they deal with periods that are not multiples of each other. The approach presented in this thesis also uses an autocorrelation analysis to find one appropriate predominant period. It will be discussed later on in this chapter how this approach can deal with overlaid periodic components of the time series as well.

One can further distinguish between deterministic and stochastic seasonal models [36]. Deterministic ones further split up the time series[3] into a deterministic seasonal trend and a residual random noise mostly performing moving averaging. Stochastic ones capture the trend and the noise by one model.

Deterministic seasonal models were suggested in [17, 104, 43, 44]. The resulting model is derived as an average of all instances of the period observed. Two kinds of random residuals remain. The first one is formed by random noise around the seasonal trend. The second kind of residual is caused by deviations of the single instances of the period. Both types of

[3] after a possible long term trend has been purged from them

residuals are treated as stationary statistically independent random noise in [43, 44]. But this assumption is not valid because of local indeterministic trends and varying noise performance as already discussed in Section 5.3.2. Especially the differences caused by deviations of the instances of the period are strongly correlated. The authors of [17, 104] suggest performing autoregression to remove such local trends. This approach is similar to the one used by trend based approaches to forecast a seasonal trend. Hence, they share the same problem. Forecasts into the future are only accurate in closest proximity to the current time. Furthermore, the noise is still not stationary.

Stochastic seasonal models capture the seasonal trend and the noise in one model. A familiar one is the SARIMA (seasonal ARIMA) approach[35] that has already been shortly discussed in Section 5.3.2. A further extension has been presented in [27] that combines the local ARMA approach with the idea of SARIMA. But both share the same problem that they can not be characterized properly.

The approach presented in this thesis combines the ideas of deterministic and stochastic seasonal models. A stochastic model has already been presented for static resource management that captures the seasonal trend and the noise. A deterministic time series is derived from this model that describes the required resource capacity. The idea now is to derive a forecasting model from this deterministic time series using known methods for forecasting deterministic seasonal trends. Deviations between instances of a predominant period are handled pessimistically.

6.3.3 Modeling Approach

The modeling approach for dynamic resource management is mainly an extension of the one developed for static resource management (cf. Section 5.3). It is also based on the stochastic process $Y_i(t) = Y_i^*(t) \cdot LT_i(t)$ that has been introduced before. A possible long term trend is separately modeled by a linear function $LT_i(t)$, while a stochastic process $Y_i^*(t)$ models the seasonal trend and the random noise. The process and the function are characterized using demand behavior observed during the trainings phase the same way like for static resource management as well (cf. Section 5.3.3).

Only the maximal resource demand in the time interval $[t_{p_3}, t_{p_3}+\Delta t_{p_3}]$ needs to be determined for static resource management. It does not matter when this maximum will be actually required. Dynamic resource management in contrast needs to know the required resources at any time t in this interval. The idea is to use periodic behavior of the seasonal trend and the noise to support this requirement comparable to known approaches. The demand behavior (w.r.t. the seasonal trend and the noise) is assumed to repeat over and over again after a certain period. Hence, a model can be extracted from the resource demand observed during the trainings phase that appropriately describes all instances of the period. This model can

be used to extrapolate future resource demand as well under the assumption that upcoming instances will behave like the ones observed.

Initially, the seasonal trend and the noise of the resource demand is described by the stochastic process $Y_i^*(t)$. But dynamic resource management does not need to know the resource demand $R_i^*(t)$ (described by $Y_i^*(t)$) but the resource capacity $A_i^*(t)$ required to fulfill the SLOs. Hence, resources capacity $A_i^*(t)$ required by a VM is derived from $Y_i^*(t)$ first. For this, the same method like presented in Section 5.4.2 is applied for both resource types: memory and CPU time. Each of the resulting functions $A_i^*(t)$ describes now the resource capacity that would have been required in the time interval $[t_{p_2}, t_{p_2}+\Delta t_{p_2}]$. A model must be extracted in a next step to extrapolate this function into the future. This model must represent all instance of a period according to the idea presented just before.

In a first step, a predominant period $\Delta t_{\hat{A}_i^*}$ must be found in each time series $A_i^*(t)$. Once this period is found, $A_i^*(t)$ is cut into pieces of this period. In a final step a representative $\hat{A}_i^*(t)$ of all pieces is extracted as model. This model describes all instances of the period during the characterization phase as well as all upcoming ones in the future. An overview of this procedure is presented in Figure 6.3. All single steps contained will be detailed some more

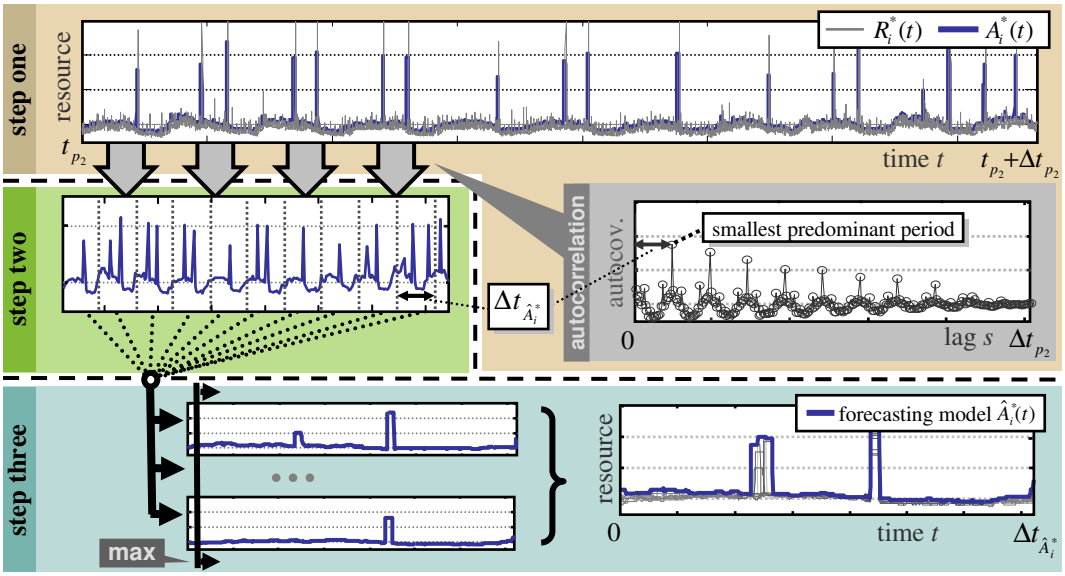


Figure 6.3: Overview of the method performed to derive a forecasting model from the resource demand behavior of a VM observed during the trainings phase. First, the time series $A_i^*(t)$ that describes the required resource capacity for a VM is derived from the demand series $R_i^*(t)$ the same way like for static resource management. Next, an autocorrelation analysis is performed on $A_i^*(t)$ to find a predominant. Once this period is found, $A_i^*(t)$ is cut into single pieces of the length of this period. Finally, the forecasting model is derived from all of these pieces.

in the following. Finally, the way how this model can be used for future extrapolation will be presented in this section as well.

Finding a Predominant Period

A good way to find appropriate periods is performing an autocorrelation analysis the way described in Section 6.1.1. The resulting correlogram of an exemplary time series and the time series itself are presented in Figure 6.4 a). One can clearly see peaks repeating with a fixed distance. This means that the time series shifted by each multiple of a certain lag strongly correlates with itself. As a consequence, the time series must contain periodic behavior with a dominating period that equals to the distance between two peaks or simply to the lag of the first peak[4].

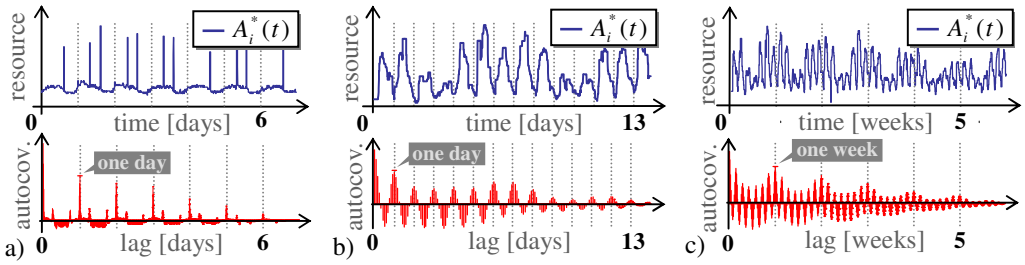


Figure 6.4: Exemplary time series $A_i^*(t)$ and respective correlograms. The lag with the highest local maximum (despite the first one) is selected as predominant period. In case of a weekly period overlaid by an intermittent daily period (like in b) and c)), the length of the time series decides about whether the daily or the weekly period is selected as the predominant one.

One can further see that the height of the peaks is continuously decreasing with increasing lag. Increasing the lag decreases the overlapping region of the original time series and the time series shifted by the lag. As a result, differences of the instances of the period are more and more dominating the analysis. The autocovariance is decreasing.

It is a bit more complicated to determine an appropriate period for time series with different overlaid periods. The selected one strongly decides about the quality of the model. One can take an exemplary service that has very similar demand behavior every day for five days of a week for instance. Both days of the weekend show strongly different demand behavior. One could now decide to take one day as a period. Five of seven days would be described very well. But the forecasts of the model might be very bad for the remaining two days. A better choice would be a period of one week. The model would describes the sequence of five similar days followed by the two days of the weekend this way.

[4]that has of course a lag different from zero

An autocorrelation analysis was applied to such a case once with a 14 days time series and once with a time series of 6 weeks. The time series and the respective correlograms are presented in Figure 6.4 b) and c) respectively. Again, peaks (or local maxima) with a fixed distance (one day in this case) could be observed. But the height is not continuously decreasing with increasing lag in contrast to a). A lag of 1 still led to the strongest autocovariance, when the 14 days time series was used. But the time series obviously autocorrelates stronger with a lag of 7 days than with a lag of 5. This behavior is caused by the weekly period that overlays the daily one. The peak at 7 days even exceeds the one at 1 day in case of the 6 weeks time series.

As a result, not the lag of the first but of the highest local maximum (despite the one with lag 0) should be used as predominant period. This approach deals very well with overlaid periods, when the trainings phase is sufficiently long.

Extracting the Forecasting Model

Once a suitable period is found, the time series $A_i^*(t)$ is cut into subsequent parts each with the length $\Delta t_{\hat{A}_i^*}$ of this period. Each part represents one instance of this period (e.g. one day). A representative $\hat{A}_i^*(t)$ must then be extracted in a final step that appropriately describes all of them and the ones expected in the future as well.

Different known approaches simply derive the model as an average of all instances [44, 17]. This implicates that variations around this mean are random noise that must be overestimated. The problem of this approach is that naturally time intervals of this noise are strongly correlated. A bank holiday in a normal working week for instance would lead to strong forecasting errors. Whole intervals of the day would differ from the model, which could lead to bursts of performance losses. Such correlations likely lead to SLO violations in many cases regarding SLO specifications that define a minimal required percentage of satisfied performance goals in a time interval.

Hence, the model must pessimistically describe the maximum of all instances taking care of even the worst instance ever observed. An upper envelope of all instances is derived as a model for this the following way:

$$\forall t \in [0, \Delta t_{\hat{A}_i^*}[: \hat{A}_i^*(t) = \max_{l=[0,\ldots,\left\lfloor \Delta t_{p_2} / \Delta t_{\hat{A}_i^*} \right\rfloor - 1]} (A_i^*(l \cdot \Delta t_{\hat{A}_i^*} + t + \Delta t_{p_2})) \tag{6.2}$$

This approach was applied to an exemplary time series, which is presented at the lower side in Figure 6.3. One can see that the model captures the worst case of all instances at any relative time t.

The time series that represents the model can contain many samples depending on the period and on the sampling rate of the resource demand. Slices of samples of $A_i^*(t)$ have nearly or

even exactly the same value. Discretization of the values of the time series in conjunction with run length decoding can help to strongly reduce the memory complexity of the model. The computational effort required to use the model to forecast the demand behavior can be reduced this way as well.

Extrapolating into the Future

To extrapolate resource capacity required at a certain time $t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}]$ into the future, the relative time in the period that belongs to the absolute time t must be found. Therefore, a reference point is required that defines the start time of one of the instances of the period. According to the characterization process of the model $\hat{A}_i^*(t)$, the first observed instance of the period starts at the beginning of phase two. Hence, time t_{p_2} can be used as a reference. A fixed number of complete instances of the period fits in the time interval between any time t and the reference point t_{p_2}. The remaining part (an incomplete instance of the period) defines the relative time in the period that belongs to the absolute time t.

Once this relative time is found, resource capacity required by a VM with respect to the seasonal trend and the noise can be derived from $\hat{A}_i^*(t)$. The result can be finally scaled by the influence of the long term trend to get the resource capacity required for the VM at t. This method can be expressed more formally by following equation:

$$A_i(t) = A_i^*((t - t_{p_2}) \bmod \Delta t_{\hat{A}_i^*}) \cdot LT_i(t). \tag{6.3}$$

6.3.4 Discussion

Different assumptions and requirements on the demand behavior are decisive for the quality of the forecasts. The three main important ones are discussed in the following.

Dependence between Observed and Future Resource Demand

The most important assumption concerns the dependence between observed and future demand behavior very similar to the static case. In contrast to the static case, not only the maximal demand but the demand behavior of whole intervals observed is assumed not to be exceeded in the future.

This assumption can be violated in different cases. An adaption method to take care of those cases was developed that will be presented in Section 6.5. Once changes are detected, the model is adapted at runtime to consider the changed demand behavior in the following instances of the period. The main reasons for changed demand behavior and the consequences for the SLOs of the VMs will be discussed in this section as well. It will be further shown in

the evaluation chapter how often such forecasting errors occur and how often they actually lead to SLO violations.

Finding Exact Periods

The modeling approach presented in this section can lead to problems, when predominant periods are determined that slightly miss the real existing one. Continuously repeating forecasting errors will occur because the forecasts of the model are slightly shifted against the real behavior. This lag gets greater with each following instance of the period, which results in a continuously increasing amount of forecasting errors. Hence, it is very import to determine the exact period.

Mainly very strong noise can lead to wrong periods. The local maximum is not clearly defined by one peak in the correlogram. Different lags in close proximity to the exact period lead to similar autocovariance values. Noise can cause that the maximum of them belongs to a lag that slightly misses the real period.

The modeling concept developed in this thesis needs to find periods in the time series $A_i^*(t)$ that describe the resource capacity required for a VM. Different of such time series have been derived from a wide variety of services. The resulting time series $A_i^*(t)$ did not contain much noise any more mainly due to the overestimated random noise (cf. Section 5.4.2) of the real demand behavior. Hence, real periods (days or weeks in most cases) could be determined already very exactly. Different additional methods (presented for instance in [44]) can be applied to better find the exact period. This further increases the robustness of the approach.

Lowest Common Multiple of Different Periods

An example of resource demand that consists of different overlaid periods was introduced, while the modeling approach was presented. It was shown that the approach could find appropriate periods even for those cases. A sequence of daily periods was overlaid by a weekly period in the example. This exemplary case worked quite well because a period of one week is an exact multiple of a period of one day. Selecting one week as a period will capture all daily periods as well even if some days show differing behavior.

But this will not necessarily work that easy for periods that are not multiples of each other. None of both periods would be a good choice for a time series with a daily period overlaid by a period of 25 hours. All resulting instances would strongly differ from each other because the individual periodic components of the time series are shifted against each other with increasing time. These shifts will go on until both periods start again at the same time. As a result, the most appropriate period for the model would be the lowest common multiple of all of the overlaid periods.

The resulting optimal period for the model can be a very long one depending on the individual periods involved. Several instances of the period must be observed to automatically find it the way described in this section. Hence, a very long trainings phase is required in bad cases to find the optimal period for the model.

Trainings phases that are to short will lead to correlograms without any significant local maxima. The extracted instances can strongly differ, when nevertheless one of them is selected. The derived model will overestimate the real behavior in the future very often, since it is an upper envelope of all instances. But strong underestimates are possible as well depending on the number of instances the model is derived from.

As a result, a model should not be derived from the observed demand behavior, if the respective correlogram does not indicate a significant predominant period. Only a static model should be applied for the VM in those cases.

6.4 Dynamic Scheduling

The purpose of the dynamic scheduling approach is now to redistribute VMs at runtime and to power up and down servers based on the forecasted resource demand. The goal is to minimize the number of active servers at any time.

This section starts with an analysis of known approaches. Scheduling algorithms for dynamic resource management but also for related application areas will be presented and discussed. It will be pointed out that all share the same problem. They do not know how long resolving an upcoming resource shortage will take. Hence, they can not guarantee resolving it right in time. Some of them can not even guarantee resolving the problem at all because they require operations that can not be performed in some cases without temporary resource shortages.

A new scheduling algorithm for dynamic resource management will be presented in the following sections. Several restrictions have been worked out for this algorithm in the problem statement chapter. It will be worked out one by one how they are addressed. The delays of the operations will be taken into account in a next step. Finally, the results are put together to get to the complete scheduling algorithm. A discussion of assumptions and some properties of the new approach (such as stability) closes this section.

6.4.1 Known Approaches

A first class of dynamic resource management approaches [17, 90, 102] follows a simple idea. They determine an appropriate distribution of VMs to servers based on the forecasted resource demand. Classical vector bin packing algorithms are used. A new distribution is determined the same way when the resource demand is expected to change.

These approaches share mainly two different problems. First, it remains unclear how a sequence of operations is found that redistributes a current distribution to the new one without temporary resource shortages. Second, it is unclear whether or not a sequence exists at all that redistributes VMs right in time to prevent upcoming resource shortages.

An extended approach [53] addresses the first of these issues. The authors try to find a sequence with a minimal number of operations that redistributes the VMs without resource shortages. But this approach requires additional servers in some cases to temporary host VMs (e.g. for swapping operations). Furthermore, the time constraint has not been taken into account yet.

Another class of approaches [12, 66, 129, 118, 15] follows a different idea. They do not target to redistribute a current distribution to a predefined new one. Instead, it is tried to find a sequence of feasible operations that resolves upcoming resource shortages or consolidate VMs and shutdown servers respectively. Abstractly spoken: it is tried to sequentially improve the situation as good as possible by applying feasible operations.

A problem of these approaches is that in some situations upcoming resource shortages cannot be resolved at all without an additional server. Resource constraints prevent operations (such as swapping) that are essential to resolve the problem. Such a case is exemplarily presented in Figure 6.5.

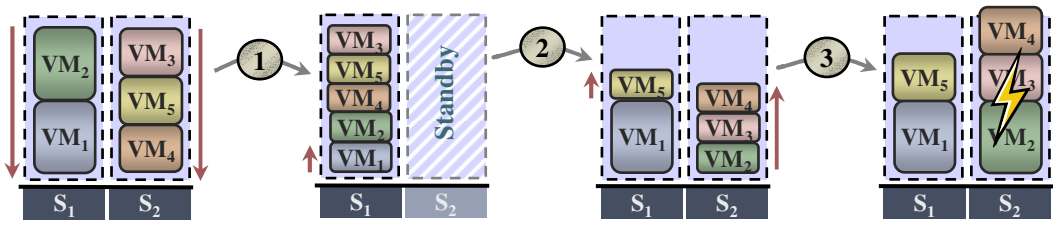


Figure 6.5: Exemplary redistributions that lead to a deadlock situation. This deadlock cannot be resolved without an additional server.

Five VMs that currently require their maximal resources are distributed to two servers. The resource demand of all decreases so that they can be consolidated to only one server. The second one is switched off. Now, the resource demand of VM_1 increases back to its maximum. Server S_2 is reactivated and any heuristic decides to remove VM_2, VM_3, and VM_4 from server S_1 to solve the problem. The resource demand of all other VMs increases now back to their respective maxima as well. A resource shortage develops at S_2 that can not be resolved any more. VM_2 and VM_5 must be swapped, which is not possible because of missing resources to temporally move away one of both first.

The underlying problem that these algorithms try to solve can be generalized to the known load rebalancing problem [2]. This NP hart problem belongs to the class of online optimization

problems [19]. Different algorithms have been developed that aim to heuristically solve it. A good overview is given in [2, 42].

Different problems in the area of dynamic task scheduling and rescheduling can be mapped onto the load rebalancing problem. An overview of some of them is presented in [86]. Several load management approaches that dynamically dispatch workload on different computation nodes[101, 27, 67, 81] heuristically solve the problem as well.

But all of them are not directly applicable to the dynamic resource management concept presented in this thesis. No resource constraints (like the ones worked out in Section 3.4.3) are regarded in most cases. It is assumed that elements can be easily swapped. Other approaches that consider such constraints can cause deadlock scenarios such as presented in Figure 6.5. These deadlocks cannot be resolved without an additional server that temporarily hosts VMs.

The approach that will be presented in this chapter combines a solver for the load rebalancing problem with a deadlock prevention strategy mainly inspired by ideas taken from [113, 14]. In addition, this solver will be extended by mechanisms that also consider timing constraints (cf. Section 3.4.3), which does none of the known approaches.

6.4.2 Basic Idea

The major challenge for the dynamic scheduling algorithm is to ensure that any upcoming resource shortages can be resolved in time. There must exist a sequence of feasible operations (moving VMs and powering up servers) at any time that resolves the problem. Additionally, this sequence must resolve the problem right in time.

The basic idea is to distinguish between so called safe and unsafe distributions of VMs to servers. By definition, a safe distribution provides enough resource capacity to each VM at any time to support its demand without any redistribution. The static resource management approaches presented in Chapter 4 and 5 determine such a safe distribution. A distribution that can lead to resource shortages is called unsafe in contrast.

It is further distinguished between safe and unsafe VMs. VMs that are placed on servers according to the initial safe distribution are safe or in a safe position. VMs become unsafe or get into an unsafe position, when they are moved away from their initial server. They remain unsafe until they are moved back.

Finally, the initial server on which a VM is placed according to the safe distribution will be called home server of the VM. Moving home a VM means moving back the VM from any unsafe position to its respective home server.

The scheduling algorithm starts with a safe distribution. VMs are only moved to an unsafe position, if two conditions are met. First, there must exist a way back to the safe distribution, on which all operations are feasible no matter how the demand behavior of the VMs develops. Second, there exist enough time to in worst case completely restore the safe distribution before

any possible upcoming resource shortage actually occurs.

A scheduling algorithm that is based on this idea was developed. It will be presented in the following sections step by step.

Only the resource constraints worked out in the problem statement chapter (Section 3.4.3) are considered first. Further conditions concerning the time constraint will be extracted in a next step. Finally, the whole scheduling algorithm will be worked out.

6.4.3 Ensuring Resource Constraints

Several resource constraints have been worked out in the problem statement chapter in Section 3.4.2 and in Section 3.4.3. They will be shortly summarized in the following for clarity reasons.

Constraint One

The first constraint (Equation (3.3) in Section 3.4.2) ensures that enough resource capacity is provided to the demand of the VMs to not violate any of their SLOs at any time. Therefore, the jointly required resources of all VMs placed on the same server must not exceed its capacity. The jointly required resources are overestimated by the sum of the resources individually required by the VMs comparable to the static approach, which leads to following equation:

$$\forall t \in [t_{p_3}, t_{p_3}+\Delta t_{p_3}], k : \sum_{i:B(i,t)=k} \vec{a}_i(t) \leq \vec{c}_k \tag{6.4}$$

that describes constraint one. The sum of the resources $\vec{a}_i(t)$ individually required by all VMs placed on the same server must not exceed its capacity $\vec{c}_k$ at any time[5].

Constraint Two and Three

The second and third constraint ensure that the mapping restrictions concerning the distribution of VMs to servers are met. They are formally expressed by the Equations (3.7) and (3.8) in Section 3.4.3.

Constraint Four

A further constraint concerns the resources required by a VM i while it is migrated from a server k_1 to a server k_2. Both servers must provide the resource capacity (memory and CPU time) required by the VM during the whole migration phase (cf. Equation (3.9) in Section 3.4.3). Overestimating again the jointly required resource $\vec{j}\vec{a}_k(t)$ by the sum of the individually

[5]The operation $\leq$ in this equation has been defined by Equation (3.4) in Section 3.4.2.

required resources $\vec{a}_i(t)$ leads to following equation:

$$\forall t \in [t_0, t_0 + \Delta t_i^{mig}[: \vec{a}_i(t) + \sum_{j:B(j,t)=k} \vec{a}_j(t) \leq \vec{c}_{k_2} \tag{6.5}$$

that describes constraint four. The migration starts at t_0 and has a duration of Δt_i^{mig}.

Constraint Five

A last resource constraint worked out in Section 3.4.3 states that the number of servers required by the initial safe distribution must not be exceeded by the dynamic scheduling algorithm at any time.

A formal way will be presented in the following to decide, if possible operations prevent a way back to the safe distribution or not with respect to all of these constraints.

For this, a formal description of the "state of safety" of a distribution is introduced. This state is represented by a directed multigraph $G = (N, E)$. The nodes N are the servers. An edge E represent a way for an unsafe VM to become a safe one. Such a graph is presented for an exemplary unsafe distribution in Figure 6.6 b). The edges in the graph point to servers to which the respective VMs must be moved to become safe. Moving back all VMs to their safe postions leads to a graph without any edges (cf. Figure 6.6 a)). Hence, a graph without any edges describes a safe distribution while graphs that contain any edges represent unsafe ones.

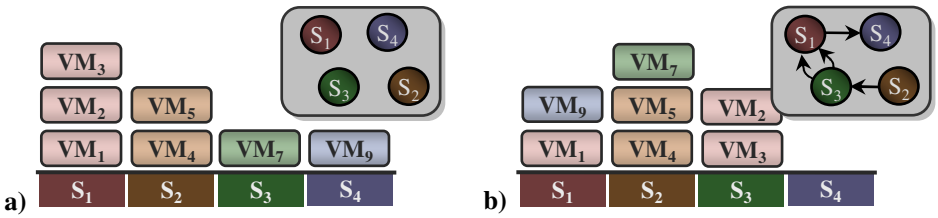


Figure 6.6: a) An exemplary safe distribution of VMs to servers and the respective graph. b) An exemplary unsafe distribution that belongs to the safe one of a) and the respective graph. Edges point the way for unsafe VMs to become safe.

In principal, any redistribution can be performed as long as the first, second, and third constraint are met. VMs can be consolidated to less servers in times of reduced resource demand as long as Equation (6.4) holds and the mapping restrictions are not violated. The way back for unsafe VMs is annotated in the graph by inserting respective edges. Unused servers can be switched off. All inactive servers must be reactivated and all unsafe VMs must be simply moved back to their initial server according to the edges in the graph to restore the

safe distribution in worst case.

But the last two resource constraints prevent moving back the VMs in some cases because of missing resource capacity on the destination server. Other VMs must be moved away from this server first to free some resources for the VM. Moving away the other VMs requires free resource capacity at their initial server as well. This can lead to situations in which cyclic dependencies prevent that any of the VMs can be moved at all without an additional server. An upcoming resource shortage can not be resolved.

This situation is strongly related to classical deadlock problems known from operating system and data base theory [113, 3]. There exist mainly four different strategies to deal with deadlocks [113]:

- Ignoring the problem (ostrich strategy),
- detecting and resolving the deadlocks,
- dynamically preventing deadlocks at runtime, or
- conceptually preventing deadlocks.

The first two options are not applicable in the given context. Applying the first one can not guarantee to meet any SLOs. Resolving deadlocks is not possible without an additional server, which is forbidden by one of the resource constraints. Hence, deadlocks must be prevented.

The scheduling algorithm developed within this thesis prevents deadlocks by avoiding cyclic dependencies inspired by the idea presented in [14]. Such cyclic dependencies can be directly detected in the graph introduced before. Each distribution of VMs to servers whose graph contains any cycles can in principal cause such deadlocks. All servers with nodes in the graph that belong to a cycle can contain unsafe VMs that could not be moved back home because of other unsafe VMs allocating the resource capacity they require. Hence, no migration operation must be performed that leads to a cycle in the graph to prevent such deadlock scenarios.

It will be shown in the following that indeed preventing cycles in the graph ensures a way back to the safe distribution with respect to all resource constraints no matter how the resource demand of the VMs develops. An algorithm will be presented that can resolve any upcoming resource shortage assuming that the underlying graph of a current distribution is acyclic.

First, nodes without any outgoing edges are regarded (e.g. S_4 in Figure 6.6 b)). These nodes represent servers that only contain safe or even no VMs. Hence, unsafe VMs that are safe on these servers can be moved back in any case with respect to the resource constraints. The initial safe distribution ensures that enough resources capacity is left for them to support their maximal demand at any time. None of the mapping restrictions will be violated as well in this case. The respective incoming edges are removed in the graph after the respective VMs have been moved back.

Now, an arbitrary unsafe distribution with an acyclic graph is regarded. All outgoing paths from any node in the graph will end in nodes that have no more outgoing edges. The outgoing path from S_2 ends in S_4 in the graph in Figure 6.6 b) for instance. All incoming edges of nodes at the end of the paths can be removed by migrating back the respective VMs as described before. This will lead to other nodes along the path that do not have any outgoing edges any more so that their incoming ones can be removed as well. Finally, all edges to all nodes can be removed by recursively removing edges from all paths starting at their respective ends. In the example, first the edge from S_1 to S_4 is removed by migrating VM 9 back home before the edges between S_3 and S_1 can be removed by moving VM 2 and 3. Finally, the edge between S_2 and S_3 can be removed by moving VM 7, which will end up in the safe initial distribution.

Please note that VMs must be recursively migrated back only in worst case. In most case, only a few VMs must be moved to solve an upcoming resource shortage. But before VMs are moved, it has to be ensured that the graph remains acyclic. This ensures a way back to the complete safe distribution at any time.

It will be presented in the following how a set of feasible operation can be extracted from a current distribution that prevent cycles in the graph.

6.4.4 Extracting a Set of Feasible Operations

In principal, two different ways exist for finding operations that will not lead to any cycles in the graph. First, one can define constraints on migration operations that must be met to prevent cycles after they have been performed. The scheduling algorithm only selects migration operations that meet these constraints. A second approach tests how a possible migration operation changes the graph. The operation can be applied on the real VMs, if the graph is still acyclic after the operation has been performed. Otherwise, the operation is not feasible.

Both ways form a base for the scheduling algorithm developed within this thesis to find feasible migration operations. Hence, they are worked out some more in the following.

Constraints for Feasible Operations

Three sorts of operations can be safely performed without getting cycles in the underlying graph. They are illustrated in Figure 6.7.

An unsafe VM is represented by an edge that points to the server on which it is safe as introduced before. A move of such a VM from one unsafe to another unsafe position only changes the source node of the respective edge. Hence, such an operation can only lead to cycles in the graph, if the node that represents the VM's initial server has an outgoing path. An inactive server cannot host any VM. Hence, the respective node has no outgoing edges. As a consequence, VMs whose initial server is currently switched off can be moved anywhere.

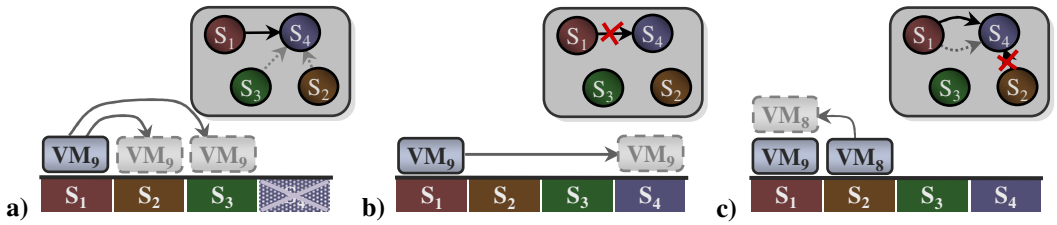


Figure 6.7: Three sorts of migration operations that can be performed without leading to any cycles in the graph. a) VMs whose initial server is currently switched off can be moved to any server. b) Moving VMs back to their initial servers will also not lead to any cycles. c.) And finally, a VM can be moved to another server, if already other VMs are placed on it that are safe on the same initial server like the one to be moved.

Of course, VMs can be safely moved back to their initial servers. Moving back a VM means bringing the VM in its safe position. The respective edge is removed in the graph which will not lead to any cycles.

A third feasible operation is moving a VM to a server on which already another VM is placed that belongs to the same initial server. Already an edge exists between the planned destination server and the initial server of the VM in this case. This existing edge points into the same direction like the one to be added. Hence, the new edge will not cause any additional cycles.

Several migration operations will not satisfy any of these constraints in most cases. Nonetheless, not all of them will lead to cycles in the graph. An additional way will be presented in the following to find out whether a certain operation will cause a cycle or not.

Examining Additional Operations

The scheduling algorithm is limited to a subset of operations until know. A second possible way is to simply apply a certain operation to the graph and check, if it remains acyclic. Tests for cycles in graphs are known from classical graph theory. Two of them have been presented in Section 6.1.2.

A first approach topologically sorts the nodes in the graph to detect cycles. It has a linear runtime complexity. This kind of approaches can cause much computational effort. All VMs must be removed from a server to switch it off. Different other servers can be suitable destinations for the VMs. Hence, many possible operations must be tested depending on the heuristics the scheduling algorithm uses to decide which server to choose.

A more efficient way for finding operations that will leave a graph acyclic has been suggest in [14]. The main idea is to represent the graph by a matrix, from which one can directly

derive whether or not a certain operation will lead to cycles in the graph. Feasible operations can be found in constant runtime. Some more computational effort is only required when the operation is actually performed to adapt the matrix. Some more details are provided to this approach in the background section (Section 6.1.2).

6.4.5 Ensuring Time Constraints

VMs can now be redistributed without loosing a possible way back to the safe distribution using operations out of a set of extracted feasible operations as described in previous section. A sequence of feasible operations exists for any unsafe distribution of VMs to servers that can restore the safe one. It must be now ensured in a next step that the unsafe distribution remains valid[6] at least for the time the sequence of operations needs to restore the safe one.

For this, the algorithm must first determine how long the redistribution to a targeted unsafe distribution and back to the safe one will take. It can then evaluate whether or not this unsafe distribution will remain valid long enough using the resource demand models. Finally, the algorithm can redistribute the VMs and switch off servers, if the new distribution will be valid for the required time period.

This time period is obviously not fixed but depends on the targeted unsafe distribution. It will be shown in the following how this time period (formally called planning period Δt^{ftr} from now on) is calculated depending on a certain distribution and on sequences of operations that are planned to be performed. In addition, constraints are derived that must be considered by the scheduling algorithm to ensure not to violate the time constraint worked out in Section 3.10 at any time.

Planning Period Δt^{ftr} in Steady State

First, Δt^{ftr} is calculated assuming that currently an arbitrary (safe or unsafe) distribution of VMs to servers is present. Δt^{ftr} describes in this case, how long (if necessary) restoring the safe distribution will take based on the current one.

A way has been presented at the end of Section 6.4.3 to get back from any possible unsafe distribution to the safe one. Each unsafe VM is directly moved back to its safe position according to the edges in the graph. Hence, each unsafe VM needs to be moved only once. The overall migration time of all is thus simply the sum of their individual migration times Δt_i^{mig}, if all VMs are migrated strictly one by one.

In principal, different VMs could be migrated in parallel as long as the respective edges in the graph belong to paths that have disjunct node sets. No dependencies concerning the allocated or required resource capacity exist between them in this case. But a fixed maximal

[6] w.r.t. the first resource constraint

migration time Δt_i^{mig} can only be guaranteed, if VM i is the only one that is migrated in the whole data center at a time as discussed in the problem statement chapter. Migrating different VMs in parallel can increase the migration time of them depending on the source and destination server of the VMs and on the network. Hence, the scheduling algorithm will only subsequently schedule the migration operations. Some ideas will be presented later on in the discussion section to relax this pessimistic approach a bit.

The time that servers need to wake up from standby or to completely reboot is independent from the number of servers that start up at the same time. Hence, in principal different server could be reactivated at the same time. Nevertheless, each reactivation time must be individually considered by Δt^{ftr}. A case can be regarded in which the scheduling algorithm needs to reactivate only one of two inactive servers to resolve an upcoming resource shortage. It remains unclear so far, when the next one needs to be reactivated as well. There must still exist enough time for powering up the second one as well to resolve an additional upcoming resource problem. Some ideas will be presented in the discussion section that can be followed by ongoing research to more parallelize server startups.

As a result, the planning period Δt^{ftr} for an arbitrary distribution of VMs to servers can be calculated as follows:

$$\Delta t^{ftr} = \sum_{i:\text{VM } i \text{ is unsafe}} \Delta t_i^{mig} + \sum_{k:\text{server } k \text{ is off}} \Delta t_k^{up} \tag{6.6}$$

The scheduling algorithm must ensure that after a redistribution to a certain unsafe distribution is finished, this new one remains valid for the time period Δt^{ftr} calculated using this equation to not violate the time constraint.

Onces the unsafe distribution is reached, the algorithm must further take care of possible upcoming resources shortages in the future. It must evaluate at any time t, if the current distribution will be still valid at time $t+\Delta t^{ftr}+1$ using the forecasting models. Redistributions must be planned and initiated, if resource shortages are detected.

The whole safe distribution must be restored in worst case. All inactive servers are switched on and VMs are migrated one by one according to the algorithm presented in Section 6.4.3. The duration of the whole procedure will not exceed Δt^{ftr} so that any possible resource shortage can be resolved right in time.

Only a steady state has been regarded until now. It will be shown in the following how the planning period Δt^{ftr} is calculated that must be considered before operation are performed. This planning period is a simple extension of the one that belongs to a current steady state in many cases. The notation $\Delta t^{ftr}_{B(i,t)}$ will be used in the following to describe Δt^{ftr} of a certain distribution $B(i,t)$ in steady state at a time t for clarity reasons.

Planning Period Δt^{ftr} When Moving VMs from Safe to Unsafe Positions

The planning period Δt^{ftr} is 0 for the initial safe distribution. To shut down any server, VMs must be removed from it first. Hence, they are migrated into an unsafe position.

Time is needed to migrate VMs one by one from a safe into an unsafe position. Furthermore, the same time is needed to move them back again when the safe distribution must be restored. Hence, the planning period is simply two times the sum of the respective migration times.

Nearly the same is true, when already an unsafe distribution is present. Each additional unsafe VM increases Δt^{ftr} by its migration time. Furthermore, the time for migrating the VM to its unsafe position must be considered as well. Hence, the planning period which needs to be considered by the algorithm before any VM i is migrated at time t from a safe to an unsafe position can be generally calculated as follows:

$$\Delta t^{ftr} = \Delta t^{ftr}_{B(i,t)} + 2 \cdot \Delta t^{mig}_{i}. \tag{6.7}$$

The migration operations can be initiated, if the targeted distribution fits for the calculated planning period. The demand behavior of the VMs in future needs not to be evaluated any more, while the sequence of planned operations is performed. The planning period already contains the time for the migrations. A steady state is reached after all migration operations have been finished. This state must be handled like already described.

Planning Period Δt^{ftr} When Moving VMs from Unsafe to Unsafe Positions

The situation only slightly differs, when VMs are already located in an unsafe position. In this case, the planning period extends only by once the migration time of all VMs that are planned to be moved. The time for migrating them back to their safe position is already contained in the planning period of the steady state.

Hence, the scheduling algorithm must only ensure that the targeted distribution stays valid for the planning period of the steady state plus the additional time needed to move the VMs.

The steady state is reached again after the migration process has been finished.

Planning Period Δt^{ftr} When Moving VMs to Safe Position

Moving back VMs to their safe position does not extend the planing period any more. The VM can be safely migrated, if enough resource capacity is left at its initial server for the whole current planning period Δt^{ftr}. No additional evaluations of future resource demands are needed. The planning period in steady state is decreased by the duration of the migration phase of the VM after the move. This VM is now in its safe position and hence must not be moved again to restore the whole safe distribution.

Planning Period Δt^{ftr} When Powering Server Up or Down

The shutdown time Δt_k^{down} and the reactivation time Δt_k^{up} must be considered, when a server shutdown is planned. Once a shutdown process has been initiated, it can not be canceled any more as discussed in Section 3.3.3. The scheduling algorithm must wait until the shutdown process has been completely finished before the server can be reactivated again. Additionally, the algorithm must take care of the break even time Δt_k^{be} as well. The server needs to remain switched off for a while to compensate additional energy overhead caused by the shutdown and reactivation process.

Hence, the planning period must be calculated as follows before a single server k can be switched off at a certain time t:

$$\Delta t^{ftr} = \Delta t_{B(i,t)}^{ftr} + \Delta t_k^{down} + \Delta t_k^{be} + \Delta t_k^{up}. \tag{6.8}$$

All three delays of the server must be added to the planning period Δt^{ftr} that belongs to the steady state of the current distribution.

The server can be safely switched off, if the current distribution of VMs to servers remains valid for this increased planning period. The steady state is reached after Δt_k^{down} and Δt_k^{be} have been expired. The planning period takes now care of the additional inactive server as well.

An inactive server is switched back on nearly the same way like the VMs are moved to its safe position. The startup process can be initiated at any time without extending the planning period. The steady state is entered after the startup process is finished.

6.4.6 Scheduling Algorithm - Overview

The results of the previous sections can be put together now to get to a whole scheduling algorithm. An overview of this algorithm is presented in Figure 6.8.

The system starts in a steady state at a certain time t. All VMs are placed on servers. This placement is valid for the respective planning period $\Delta t_{B(i,t)}^{ftr}$. The algorithm must first evaluate, if the current distribution will be still valid at time $t + \Delta t_{B(i,t)}^{ftr} + 1$ as worked out in Section 6.4.5. It must find and perform a sequence S_{res}^{op} of operations that will resolve the upcoming problem if resource shortages are detected. The system goes back to a steady state after the required operations have been performed.

The algorithm can try to consolidate VMs and switch off unused servers, if no resource problems need to be resolved. A respective sequence S_{con}^{op} of operations is extracted and performed.

The system remains in steady state without any changes, if neither resource problems must be resolved nor any consolidation operation can be performed. The algorithm needs to restart

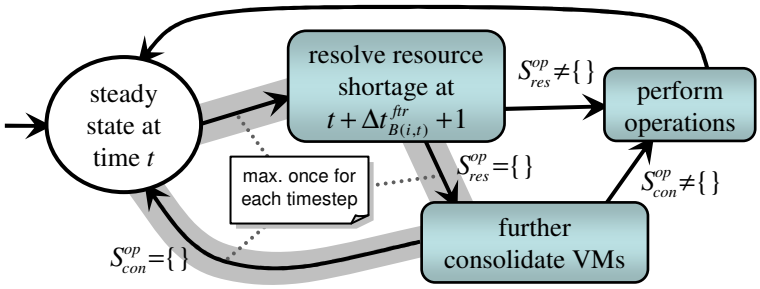


Figure 6.8: Overview of the dynamic scheduling algorithm. The algorithm starts in a steady state at a certain time t. It first looks for upcoming resource shortages at $t + \Delta t^{ftr}_{B(i,t)} + 1$. If resource problems will occur, it performs a sequence S^{op}_{res} of operations that will resolve these problems. Otherwise it tries to consolidate VMs and to switch off unused servers. The system remains in steady state for the rest of the discrete time step, if no changes need to be performed at all. The evaluation starts again in the next time step $(t + 1)$.

the evaluation again not before the next time step begins, since the demand behavior of the VMs is described by the models in discrete time steps.

The ways for finding the sequences S^{op}_{con} and S^{op}_{res} respectively will be detailed some more in the following two sections.

6.4.7 Scheduling Algorithm - Consolidating VMs

VMs are only moved, if at least one server can be emptied and switched off to prevent unnecessary migrations. Hence, it must be checked for each server, if all VMs currently placed on it can be moved away to other servers.

The algorithm must ensure that a targeted distribution remains valid long enough before any VM is moved. It has been further discussed that the respective planning period Δt^{ftr} strongly depends on the targeted distribution. But one already needs to known the planning period to find a new distribution in which the server is switched off. This period decides about whether a VM will fit on a certain server or not.

To solve this cyclic dependency, the planning period is overestimated as follows:

- All VMs that are safe on the server to be switched off are moved to an unsafe position. Hence, the current planning period must be extended by twice the sum of the VMs' migration times (cf. Section 6.4.5).
- It is pessimistically assumed for all unsafe VMs placed on the server that they all will be moved to a server on which they are unsafe as well. Hence, the planning period further extends by the sum of the migration times of these VMs (cf. Section 6.4.5).

- Finally, the three delays of the server to be switched off must be added to Δt^{ftr} as well (cf. Section 6.4.5).

Hence, to start a sequence S^{op}_{con} of operations at a certain time t that will lead to an inactive server k, the targeted distribution of VMs to servers must at least fit for a planning period Δt^{ftr} that can be calculated as follows:

$$\Delta t^{ftr} = \Delta t^{ftr}_{B(i,t)} + 2 \cdot \sum_{\substack{i:B(i)=k \ \&\& \\ i:B(i,t)=k}} \Delta t^{mig}_i + \sum_{\substack{i:B(i)\neq k \ \&\& \\ i:B(i,t)=k}} \Delta t^{mig}_i + \Delta t^{down}_k + \Delta t^{be}_k + \Delta t^{up}_k. \tag{6.9}$$

Function $B(i)$ represents the initial safe distribution of VMs to servers. VMs i to which function $B(i)$ assigns a certain server k are safe on k. Others are not.

One can now find appropriate destination servers for all VMs currently placed on server k based on this planning period and on the models. Hereby, the resource constraint expressed by Equation (6.4) in Section 6.4.3 must not be violated in the time interval $[t, t + \Delta t^{ftr}[$. Additionally, the migration operation must not lead to any cycles in the underlying graph of the distribution (cf. Section 6.4.4). And finally, the mapping constraints expressed by the Equations (3.7) and (3.8) must be considered as well. The sequence of migration operations can be performed and the server can be switched off, if all VMs can be removed from the server without violating these constraints.

In many cases, different target positions are possible for one VM. Preferably, VMs should be moved back to their initial servers. Moving VMs home will remove edges in the graph, which increases the flexibility for following operations. Furthermore, Δt^{ftr} decreases with any additional safe VM so that shorter phases of reduced overall resource demand can be used to save energy. Conventional bin packing heuristics can select appropriate positions for a VM that cannot be moved home. The heuristics should aim to evenly distribute the resource demand to all active servers to delay possible upcoming resource shortage as long as possible[7].

6.4.8 Scheduling Algorithm - Resolving Resource Shortages

The scheduling algorithm (in steady state) evaluates at any time t, whether or not the distribution of VMs to servers will be still valid at $t + \Delta t^{ftr}_{B(i,t)} + 1$. The distribution is invalid, if the first resource constraint expressed by Equation (6.4) in Section 6.4.3 is violated for one or more servers. VMs must be removed from the affected servers to resolve this problem.

The way to find sequences S^{op}_{res} that resolve upcoming resource shortages is a bit more complicated compared to the consolidation part. Other servers must provide resource capacity for VMs that must be removed from a server. Sometimes, other VMs must be moved away

[7] Selecting the host with most remaining resource capacity for each VM will evenly distribute the VMs to servers. This heuristic is known as worst-fit strategy and has been extensively discussed in [56].

first for this. Whole sequences of VMs must be moved in some cases to solve the problem. Inactive servers must be reactivated, if active servers do not provide enough resource capacity.

An algorithm for resolving upcoming resource shortages was developed. An overview is presented in Figure 6.9.

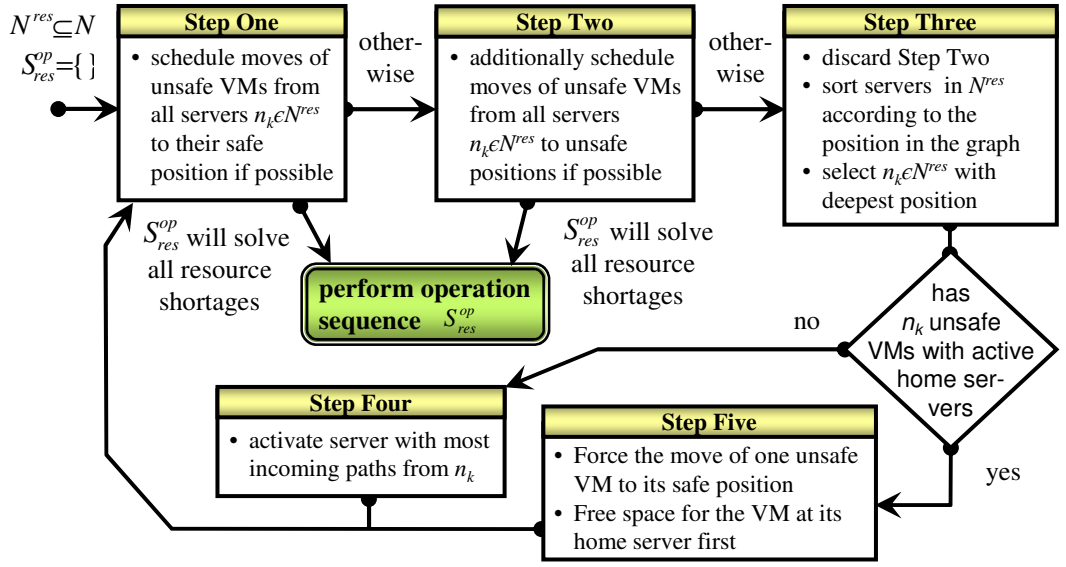


Figure 6.9: Overview of the algorithms that resolves upcoming resource shortages on servers. A subset $N^{res} \subseteq N$ of servers that will run into resource problems is given. A sequence S^{op}_{res} is extracted in five steps that resolves the problems.

This algorithm starts with a subset $N^{res} \subseteq N$ of servers. Servers in this subset will get resource problems at $t + \Delta t^{ftr}_{B(i,t)} + 1$. The algorithm tries to find a sequence S^{op}_{res} of operations in five different steps that resolves the problems of all of theses servers. Once such a sequence is found, it can be performed.

These five steps will be described more detailed in the following.

Step One - Move unsafe VMs home

It is tried first to resolve resources shortages by moving unsafe VMs from servers $n_k \in N^{res}$ directly back into their safe position. The respective servers must run and must have enough resource capacity left. It must be further ensured that the new position of each VM remains valid for the whole planning period $\Delta t^{ftr}_{B(i,t)}$ as pointed out in Section 6.4.5. And finally, none of the mapping restrictions must be violated by the planned moves.

These operations should be preferred to resolve resource problems. They cause no additional costs. Neither servers needs to be reactivated nor additional other VMs must be moved before.

Additionally, moving VMs back home increases the flexibility for further operations.

Step Two - Additionally Move Unsafe VMs to Unsafe Positions

Additional VMs must be removed from servers $n_k \in N^{res}$, if a sequence S^{op}_{res} determined in *Step One* does not completely resolve the resource problems of all servers. For this, it can be tried to move unsafe VMs to other unsafe positions. These operations are useful, when servers on which these VMs are safe, are currently switched off and other active servers have enough resource capacity left to host them.

The planning period Δt^{ftr} extends, when moves of VMs from an unsafe to another unsafe position are planned as pointed out in Section 6.4.5. But it is unclear so far, how much the planning period must be extended to resolve all upcoming resource shortages by moving VMs this way.

In principal, the planning period can be extended continuously with each additionally scheduled VM move. But this way, previously scheduled VM moves can get invalid with an increased Δt^{ftr}. These invalid moves must be removed from the schedule and the resources shortage of the respective server must be resolved again using other VM moves.

It can cause much computational effort to find suitable operations this way because of the invalid operations. It might be better to significantly extended Δt^{ftr} directly at the beginning of *Step Two* to reduces the probability of getting invalid moves. No operations will get invalid, if the actual resulting period Δt^{ftr} will not exceed the one overestimated at the beginning.

All unsafe VMs from all servers with resource problems are moved to other unsafe positions in worst case. Hence, a maximal planning period that will be never exceeded in any case can be overestimated as follows:

$$\Delta t^{ftr} = \Delta t^{ftr}_{B(i,t)} + \sum_{n_k \in N^{res}} \sum_{\substack{i:B(i)\neq k \\ i:B(i,t)=k}} \text{\&\&} \Delta t^{mig}_i. \tag{6.10}$$

This pessimistic planning period will be quite too long in most cases because only a subset of VMs must be actually moved. But nevertheless, this pessimistic planning period should be selected. It additionally ensures that no resource shortages will come up for a longer time on a possible destination server of the VMs.

Unsafe VMs can be now moved from servers $n_k \in N^{res}$ to other unsafe positions, if no resource constraints are violated during the new planning period Δt^{ftr}.

In a final step, it must be ensured that the untouched servers in the data center will not run into any resource problems for the extended planning period as well. The extension of Δt^{ftr} can be determined exactly in this case (cf. Section 6.4.5), since the sequence of additional VM moves is now known.

But none of the operations determined in this step must be scheduled, if not all performance problems can be resolved this way. The planning period of the steady state only ensures that the safe distribution can be completely restored, when all unsafe VMs are moved directly into their safe position. Hence, no moves of unsafe VMs to other unsafe positions must be scheduled, until it is clear how all resource problems can be solved.

Step Three - Select Server for Forced VM Move

VMs must be recursively moved into safe positions, if no sequence S^{op}_{res} can be found in *Step One* and *Step Two* that completely resolves all upcoming resource shortages at $t + \Delta t^{ftr}_{B(i,t)} + 1$. Servers must be reactivated in worst case as well.

The idea is now to free only space for one single unsafe VM of one server $n_k \in N^{res}$ first, so that it can be moved back to its safe position. For this, either other VMs must be moved away first (*Step Four*) or an inactive server must be reactivated (*Step Five*). The purpose of *Step Three* is to select server n_k and to decide whether a new server must be reactivated or VMs are moved away (recursively, if necessary).

Each time, *Step Three* is entered, only space to move at least one VM home is freed. This will not necessarily resolve all resource problems. But especially the reactivation of a server can increase the flexibility for the scheduling algorithm to resolve other resource problems as well. Hence, the algorithm starts again with *Step One* after space has been freed for one VM. *Step One* and *Step Two* might now find a sequence of operations that resolves all resource problems. If this is still not possible, space for other unsafe VMs is additionally freed the same way, until all problems are completely resolved.

The selection of server $n_k \in N^{res}$ should consider the impact of the following steps. As much potential as possible should be provided after *Step Four* or *Step Five* have been executed to also resolve resource shortages of other servers out of N^{res}. This especially concerns a server reactivation.

For this, a server should be reactivated that has as many incoming path as possible in the underlying graph that start at servers out of N^{res}. This way, VMs can be moved either directly or at least recursively from servers with resource problems to the reactivated one. Following heuristic can be applied to find a suitable server n_k.

All servers in N^{res} are sorted according to their topological order in the underlying graph G of the current distribution. The server with the deepest position in the graph[8] is selected as n_k. Servers that must be reactivated to remove one VM from n_k are located even deeper in the graph compared to n_k itself. The deeper a server is located in the graph, the higher is the chance that paths from other servers will lead to it. Hence, there is a good chance that other

[8] The server with the deepest position in the graph has the highest distance to nodes without any incoming edges.

resource problems can be solved using this reactivated server as well.

Two different ways exist to free space for an unsafe VM once a suitable server n_k is found. Either the home server of the unsafe VM is reactivated or space must be freed on it to host the VM. It should be preferred to move VMs home, whose home server is already running for two different reasons. First, recursively moving VMs home (see the description of Step Five) can resolve resource problems without reactivating any additional server in some cases. Second, moving home VMs removes edges in the graph, which increases the flexibility for following operations.

Hence, as long as VMs with running home server are placed on n_k, they should be moved home first (*Step Five* is entered). Servers must be reactivated (*Step Four* is entered), if none of these VMs exists any more on n_k.

Step Four - Activate Server

Server n_k only contains unsafe VMs with inactive home servers in *Step Four* due to the heuristic just presented. One of these home servers must be selected for reactivation. The one with most incoming edges from n_k should be selected to maximize the chance that the resource problem at n_k can be completely resolved. The highest number of VMs can be moved home this way. The one with the most overall incoming paths should be selected, if different servers with the same number of incoming edges exist. This increases the chance that problems of other servers can be resolved too.

Step Five - Forcing the Move of an Unsafe VM Home

The purpose of *Step Five* is to schedule operations that will force the move of a VM e_i from its current unsafe position at server $n_{k_{src}}$ to its safe position at server $n_{k_{dst}}$. Other VMs must be removed first, if not enough resources are present at $n_{k_{dst}}$ to host e_i.

An overview of an algorithm that finds such a sequence of operations is presented in Figure 6.10. This algorithm works nearly the same way like the global one that resolves resource shortages (cf. Figure 6.9). First, operations are scheduled that move unsafe VMs from $n_{k_{dst}}$ home to free resources for e_i (*Step 5.1*). The move of e_i itself from $n_{k_{src}}$ to $n_{k_{dst}}$ is scheduled (*Step 5.4*) as well, if enough resources can be freed this way. The algorithm ends. Otherwise, additional operations are required.

No VMs can be moved from $n_{k_{dst}}$ to other unsafe positions in contrast to the global algorithm. Such operations can be only scheduled, if it is clear that the resources shortages of all servers in N^{res} are completely resolved after they have been performed as discussed before. But this guarantee can not be given at this point.

Hence, it is directly tried to force the move of an unsafe VM home similar to *Step Three*, *Step Four*, and *Step Five* of the global algorithm. The inactive server with most incoming

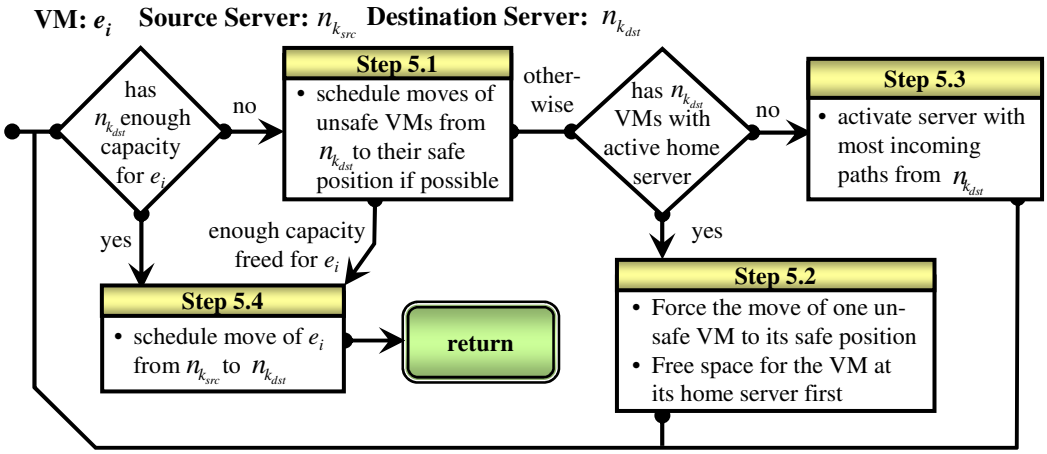


Figure 6.10: Overview of an algorithm that schedules the move of a VM e_i from a server $n_{k_{src}}$ to a server $n_{k_{dst}}$. Operations are additionally scheduled that move (recursively if necessary) other VMs away first to free resources at $n_{k_{dst}}$ for e_i. Additional servers are reactivated as well if necessary.

edges from $n_{k_{dst}}$ is reactivated, if no VMs with an running home server are present at $n_{k_{dst}}$. Otherwise, the move of one unsafe VM currently placed on $n_{k_{dst}}$ is forced by applying this algorithm again to the unsafe VM to be moved home.

The algorithm starts again at the beginning after one VM move has been forced or one server has been reactivated to find out if now e_i can be moved. *Step 5.2* or *Step 5.3* must be executed again, if this is still not possible. Otherwise, the move of e_i is scheduled and the algorithm ends.

Summary

It will be shortly shown in the following that the algorithm just presented can actually resolve any possible upcoming resource shortage right in time. It will be shown first that the algorithm can resolve any upcoming resource shortages at all. In a second step, it will be shown that the algorithm resolves the problems right in time.

An arbitrary distribution $B(i,t)$ of VMs to servers is assumed to have any kind of resource problems at $t + \Delta t^{ftr}_{B(i,t)} + 1$. All unsafe VMs must be removed from all servers in worst case to resolve the resource problem. It is tried first to directly move away unsafe VMs either to safe or to unsafe positions. The respective operations are performed, if they resolve all problems.

Otherwise, only the moves of VMs to safe positions are scheduled. In addition, moves of VMs from servers with problems to their safe servers are forced until all problems are solved. Forcing a VM move means that capacity for the VM is freed first by recursively migrating

back home other VMs along outgoing paths from the respective servers. This is possible in any case as long as the underlying graph is acyclic as shown in Section 6.4.3. Servers at the end of these paths are reactivated as well if necessary.

The algorithm schedules the move of VMs only from unsafe to safe positions despite in *Step Two*. Hence, each unsafe VM in a current distribution is only moved once. Additionally, all inactive servers are reactivated as well in worst case. Hence, *Step One*, *Step Three*, *Step Four*, and *Step Five* will resolve any upcoming resource shortage with a sequence of operations that will not take longer than $\Delta t^{ftr}_{B(i,t)}$.

Step Two schedules only operations (of unsafe VMs to other unsafe positions), if it is clear that all resource problems can be resolved with the resulting schedule. It is further ensured that the whole targeted distribution will be valid for the extended planning period Δt^{ftr} as well.

6.4.9 Discussion

Different properties such as stability, run time behavior, and scalability of the scheduling algorithm will be discussed within this section. Furthermore, some ideas to improve the efficiency of the algorithm will be shortly introduced as well. They can serve as a basis for future work.

Stability of the Algorithm

The scheduling algorithm developed within this thesis redistributes VMs and switches on and off servers based on forecasted resource demand. Operations are initiated, if certain lower or upper thresholds are deceeded or exceeded[9]. Such threshold based algorithm can tend to swing, especially when the varying value has a strong noisy part. It will be discussed in the following how the scheduling algorithm will behave concerning this stability problem.

VMs are only moved, if either a resource problem will come up or at least one server can be switched off as pointed out in Section 6.4.6 and 6.4.7. Before a server is switched off, the scheduling algorithm ensures that a certain targeted distribution of VMs on servers remains valid for a certain planning period Δt^{ftr}. This ensures that no resource shortages come up until the new distribution is reached and in worst case the complete safe distribution is restored. Hence, normal noise that swings around a threshold within a time period shorter than Δt^{ftr} will not lead to any consolidation operation at all.

The shortest planning period Δt^{ftr} ever possible occurs, when starting with the safe distribution of VMs on servers one server k should be switched off. The respective planning period

[9]Thresholds are defined by the server capacities contained in the resource constraint

can be calculated as follows as worked out in Section 6.4.7:

$$\Delta t^{ftr} = 2 \cdot \sum_{i:B(i)=k} \Delta t_i^{mig} + \Delta t_k^{down} + \Delta t_k^{be} + \Delta t_k^{up}. \tag{6.11}$$

The migration time of the VMs that are safe on this server must be considered twice, since they are moved into an unsafe position. Furthermore, the planning period must take care of the server's deactivation, break even, and reactivation time.

As a result, the time interval between exceeding and deceeding the threshold must be larger than the time needed to move the VMs of at least one server twice, switch off the server, save some energy, and finally switch it back on. Otherwise, no operations will be performed at all. Hence, the algorithm will not lead to any unwanted behavior. VMs are only moved, if any server remains switched off for at least its break even time and thus saves energy.

The algorithm can be nonetheless modified in two different ways to artificially slow down the reaction speed. First, the break even time of the servers can be increased. Servers can be only powered down this way, if they remain switched off for a longer time. Of course, potential energy savings are decreased as well this way. A second option is to implement hysteresis [100]. The threshold is no longer a fixed value but an interval. Upper and lower thresholds are different values, which can prevent resolve operations initiated by slightly increased resource demand.

Runtime Behavior and Scalability

The algorithm must be able to test between two discrete time steps, if resource problems are coming up. It must find a sequence of operations that will resolve the problems if necessary. Otherwise, it tries to find a sequence of operations that consolidates VMs and switches off servers. Hence, the runtime complexity of all three tasks is important and will be shortly discussed in the following.

The resource demand of each VM must be determined at $t+\Delta t_{B(i,t)}^{ftr}+1$ using the forecasting models to test, whether a certain distribution of VMs to servers will be still valid at this time. Hence, the runtime complexity depends linearly on the number of VMs. This step should not be a problem for the number of VMs in a conventional data center since the evaluation of the forecasting model is only a simple table lookup.

Each VM is maximally touched once in each time step to find a sequence of operations that consolidates VMs to fewer servers. The algorithm tests, whether each VM fits on another server or not. Hence, the resulting complexity is $\#servers \cdot \#VMs$. Furthermore, the sum of the resource demands at the respective destination server must be evaluated for each time t in the respective planning period Δt^{ftr} for each VM. This further increases the complexity by an additional dimension. But this complexity should not be a problem for normal numbers of

servers and VMs and a time step larger than one $1min$. Appropriately caching result can help to nonetheless reduce the computational effort required.

The resolve part of the algorithm is more critical concerning the runtime complexity. VMs are moved from an unsafe to their safe position in *Step One* and *Step Five*. Hence, the worst case complexity in each of these steps depends linearly on the number of unsafe VMs, which is not critical so far. Additionally, the whole planning period $\Delta t^{ftr}_{B(i,t)}$ must be evaluated for each VM, which spans again a new complexity dimension. The computational effort required for sorting the set of servers with resource problems in *Step Three* can be neglected for normal numbers of servers. Finally, all unsafe VMs are tried to be moved to other unsafe position in *Step Two* . Hence, the complexity is $\#servers \cdot \#VMs$ in this step. Again, not only one point in time must be evaluated but the whole interval Δt^{ftr}.

As a result, the overall runtime complexity of one loop of the resolve part is $\#servers \cdot \#VM \cdot |\Delta t^{ftr}|$ comparable to the consolidation part[10] . But the complexity further increases due to the outer loop of the algorithm. This loop must be performed several times in some cases to resolve all upcoming resource shortages. The number of iterations depends on the number of performance problems and on the number of VMs that must be removed to resolve the problems. Hence, the number of VMs will quadratically influence the runtime complexity, which can lead to runtime problems.

Again, appropriately caching result can help to prevent unnecessary reevaluations. The move of VMs in *Step One* and *Step Two* must be only reevaluated for servers, whose assigned VMs have changed in the previous iteration.

The algorithm can fall back on a simple but suboptimal solution, if some resource problems can not be resolved between two discrete time steps. Unsafe VMs can be simply forced to be moved home by applying only *Step Five* to solve resource problems on the servers. Finding such a solution has a runtime complexity that linearly depends on the number of VMs and on the current planning period $\Delta t^{ftr}_{B(i,t)}$ as discussed before. This should be feasible with manageable computational effort.

Finally, one can conclude that the overall runtime complexity of the whole algorithm is $\#servers \cdot \#VM^2 \cdot |\Delta t^{ftr}|$. Resource shortages can be suboptimally resolved with a complexity of $\#VM \cdot |\Delta t^{ftr}|$. Some analyses concerning the runtime behavior will be discussed in the evaluation chapter. The results will show whether or not runtime problems must be expected in normal scenarios.

Some ideas to improve the efficiency of the algorithm are presented in the following. They are not worked out any more within this thesis but can serve as a basis for future work.

[10] $|\Delta t^{ftr}|$ represents the number of samples in the planning period Δt^{ftr}

Improving Heuristics

The first point concerns the heuristics applied to find sequences of operations that will either consolidate VMs or resolve upcoming resource shortages.

The consolidation part of the algorithm tries to remove all VMs from a server to switch it off. But this step only succeeds, if the VMs can be directly moved to other servers. Especially VMs that require a significant part of a server's capacity can prevent that a server can be switched off. Sometimes it is hard to find servers that have enough free capacity to host them.

It can be tried in an additional step to free additional space by redistributing other VMs first to overcome this issue. VMs with less required resource capacity can fill up servers that have only little resource capacity left. Space is freed for bigger ones this way. This finally leads to a denser distribution of VMs to servers. An additional empty servers that can be switched off could be the result.

The heuristic applied in the resolve part of the scheduling algorithm can be improved the same way. The algorithm presented until now performs suboptimal solutions to resolve resource problems in some cases. More servers than actually required are reactivated in worst case. The evaluation of different ways while the sequence of operations is searched could improve the results. Especially the selection of a VM for a forced move can strongly influence the results.

The runtime complexity for both optimization is exponential. The optimum cannot be found in most cases because of the time limit between two discrete time steps. Hence, a valid solution should be determined first. The remaining time can then be used to improve the result.

Parallelize Migration Operations and Server Startups

One pessimistic restriction concerns the assumption that migrations can be only performed strictly sequentially. The support of parallel migration operations can strongly decrease the planning period Δt^{ftr}. Therewith, the dynamic resource management can use shorter periods of reduced resource demand to save energy.

Technically, different migrations could be performed at the same time, even if the the same source or destination servers are involved. But the migration process can slow down depending on the network and on the servers involved. This fact must be considered by the dynamic resource management concept. Hence, a more complex modeling of the migration time of a VM is required to exploit this opportunity. The scheduling algorithm needs to know how long a migration will take depending on the source and destination server. Other migrations performed at the same time must be taken into account as well.

A second way to decrease the planning period is to startup servers in parallel. Different servers could be started at the same time in principal. Hence, only the start up time of the slowest server needs to be regarded for the planning period. But parallel startups of servers can cause problems as already discussed. It remains unclear during the startup process of

one server, when an additional server must be reactivated. The algorithm can address this issue by observing the demand behavior of the VMs in future further on, even if it is not in steady state. The startup of an additional server must be directly initiated, if further resource shortages come up.

Delaying Operations that Resolve Resource Shortages

Once a sequence of operations that resolves upcoming resource problems at $t + \Delta t^{ftr}_{B(i,t)} + 1$ is found, it is immediately executed by the scheduling algorithm. But the execution of the operations is already finished long before the planning period $\Delta t^{ftr}_{B(i,t)}$ is expired in most cases. Hence, they could have been delayed for a while to leave servers switched off for a longer time.

But delaying these operations will delay other possible operations that consolidate VMs and switch off servers as well. Hence, in some cases delaying resolve operations can increase the time, a server is still running and hence can waste energy as well.

One should try to discard the strict separation of consolidation and resolve sequences to address this problem. Resolve operations are scheduled at their latest possible time, consolidation operations can fill empty slots. Mainly the interdependencies between the individually scheduled operations must be considered to realize such an improved algorithm. Consolidation operations must not invalidate the schedule that resolves the upcoming resource problems. Second, one must consider the development of the resource demand in the future. The algorithm must be able at any time to restore the safe distribution right in time.

Independent Clusters in Large Data Centers

A third possible improvement concerns the planning period Δt^{ftr}. Δt^{ftr} can quickly increase with an increasing number of servers and VMs in times of reduced overall resource demand. Strongly volatile resource demand behavior of VMs can not be used to save energy any more, when the planning period gets too long.

It can be tried to divide large data centers into smaller clusters to overcome this drawback. Each cluster is controlled individually by its own management system. VMs are not moved beyond cluster borders. One advantage of this approach is a shorter planning period Δt^{ftr}, which increases the chance to use shorter periods of reduced resource demand to save energy. Furthermore, VMs in different clusters can be migrated at the same time[11]. The disadvantage of this approach is the reduced flexibility for operations. The more servers and VMs are present in a data center, the higher is the chance to find appropriate destination servers for VMs that must be removed from a server.

This trade-off will be analyzed in the evaluation section. It will be shown how large data centers can get until the increasing Δt^{ftr} significantly prevents additional energy savings.

[11] Each cluster is assumed to have its individual net for migrating VMs

6.5 Changes in Demand Behavior

The dynamic resource management approach assumes unchanged demand behavior of the VMs concerning the long term trend, the seasonal trend, and the noise comparable to the static one. Again, it will be discussed first how changed demand will impact the VMs and how such changes can be detected. It can be useful to adapt the forecasting models at runtime to prevent possible resource shortages in the future, since dynamic resource management can dynamically change provisioning decision at runtime. Furthermore, an actually occurred resource shortage should be resolved as fast as possible by redistributing VMs. Respective mechanisms will be discussed in this section as well. Finally, a method will be introduced that allows limiting the impact of possible resource shortages and the time required to resolve them.

6.5.1 Impact of Changed Demand Behavior

Changed demand behavior will lead to unexpected resource shortages much more likely compared to the static approach, because dynamic resource management directly adjusts resource capacity to the expected demand.

Reserve capacity required to support an increasing long term trend can prevent that temporarily increased resource demand leads to actual resource shortages in the static case. These reserves are not present in the dynamic case. The time dependent influence of the long term trend is directly considered, when the resource capacity required by a VM at a certain time is calculated. Only capacity reserves that are caused by discretization artifacts can buffer resource demand of a VM that slightly exceeds the expect one. Remaining capacity of a server cannot support the demand of any other VM in the system in many cases and hence is left unused.

6.5.2 Detecting Changed Demand Behavior

Changes of the long term trend can be detected the same way like the static resource management approach does (cf. Section 5.5.2). A second model is built up at runtime from observed data and compared to the one created during the characterization phase.

Changes of the seasonal trend and the noise performance can not be detected that easy. Actually required resource capacity can be calculated from the observed demand as long as the overall resource demand of all VMs placed on the same server does not exceed the servers capacity. The same method like presented for static resource management can be applied. But if the demand exceeds the server's capacity, one can only measure the actually provided resource capacity but no longer the resource demand. Hence, the actual demand can not be directly measured in all cases.

A way out of this problem is to directly measure the targeted QoS attribute (e.g. response time or throughput), for which a performance goal is defined in the SLO. In case of violations, obviously a resource shortage has been detected that must be caused by incorrectly forecasted resource demand. The respective service must support to measure such QoS attribute and to send this information to the resource management to apply this technique.

An artificial resource buffer can be used, if a direct feedback from the service is not applicable. Slightly less resource capacity than actually present can be defined for each server to generate this buffer. Exceeding demand can be directly measured as long as this buffer is not depleted.

6.5.3 Adapting the Model

Adapting the models to changes at runtime can help to prevent unexpected resource shortages in the future. Once the models are changed, the scheduling algorithms directly adapts these changes, when it looks for upcoming resource shortages or when it tries to consolidate VMs.

Adapting the long term trend is quite simple. The parameters of the forecasting model $LT_i(t)$ are continuously updated by new ones, which are determined using the demand behavior observed at runtime.

The resources $R_i^*(t)$ that have actually been demanded by each VM i at runtime are required to update the models that describe the seasonal trend and the noise. The resource capacity $A_i^*(t)$ that should have been provided to the VM to not violate any SLO can be determined from this resource demand the same way like used before to characterize the model $\hat{A}_i^*(t)$ itself. The respective values in $\hat{A}_i^*(t)$ must be updated upwards, if the result exceeds the forecasts of the model.

The actual resource demand can be determined either indirectly by getting feedback from the service or directly using an artificial resource buffer as discussed in previous section.

A model similar to the one introduced in Section 5.2.3 is needed in the first case. This model must map the difference between the required and the actual value of the targeted performance metric onto the additionally required resource capacity. The actual resource demand can then be determined from the measured value of the performance metric using this model.

In the second case, the actual resource demand can be directly measured from the virtualization environment. The measures equal the demand as long as the demand does not exceed the provided capacity including the additional buffer. Increased demand behavior has been detected but cannot be quantified exactly, when the demand exceeds the capacity reserve. The models are only adjusted upwards by the size of the buffer in this case. This can require to update the model later again to capture the whole increase of the demand.

6.5.4 Resolving Resource Shortages

Slightly increasing the forecasted resource demand must not necessarily lead to resource shortages. Unused capacity caused by discretization effects can be used to compensate such exceedances as already discussed. This remaining capacity can be provided to all VMs currently placed on a server as shared resources. Resource shortages will not occur this way before the additional capacity is depleted by one or more VMs.

The dynamic resource management must react, if nonetheless resource shortages occur. The models that describe the demand behavior of one or more VMs seem not to be valid any longer. Hence, they should not be used for dynamic resource management any more. Instead, the scheduling algorithm allocates the maximally required resource capacity A_i^{max} for the affected VMs to not violate the respective SLOs any more. The resolve part of the dynamic scheduling algorithm can be used to immediately free the additional resource capacity required. Setting the required resources for a VM to the respective A_i^{max} informs the algorithm about the resource shortage, which is then resolved by an appropriate sequence of redistribution operations.

Time dependent resource demand variations of the VM are not considered for dynamic resource management any more once the forecasting model of a VM has been disabled. The achievable energy savings are reduced. The administrator must analyze the reason of the deviation between the forecasts and the actual demand of a VM. The original model can be reactivated again without any changes in case of an onetime event. Otherwise, the new demand behavior must be observed and the whole characterization process (cf. Section 6.3.3) must be repeated again.

6.5.5 Limiting the Impact of Changed Demand Behavior

Two additional parameters have been worked out in Section 6.2 that should be part of the SLA, when dynamic resource management is applied. The purpose of both is to limit the impact of resource shortages caused by unexpected demand behavior of VMs.

The first one, $\Delta t_i^{resolve}$, defines the maximal duration of possible resource shortages of a VM i. It must be ensured that any possible resource shortage can be resolved in a time period shorter than $\Delta t_i^{resolve}$ to meet this constraint. In worst case, a resource shortages can be only resolved by completely restoring the safe distribution of VMs to servers. This will not take longer than specified by the actual planning period Δt^{ftr} according to the dynamic scheduling algorithm. Hence, the scheduling algorithm must ensure that the resulting planning period Δt^{ftr} does never exceed the limit $\Delta t_i^{resolve}$.

The second parameter A_i^{min} ensures a minimal QoS by reserving a fixed amount of resource capacity at any time. The scheduling algorithm must consider following additional constraint

to support this parameter:

$$\forall k, t : \sum_{i:B(i,t)=k} A_i^{min} \leq C_k. \tag{6.12}$$

The sum of resources A_i^{min} minimally required by all VMs i placed on the same server must not exceed its capacity. Furthermore, the virtualization environment must ensure that independent from $A_i(t)$ the respective minimal amount of resource capacity A_i^{min} is reserved for each VM at any time.

6.5.6 Discussion

The efficiency the dynamic resource management can deal with changed demand concerning energy savings and occurring SLO violations strongly depends on the kind of changes. Hence, different kinds of possible changes will be presented and discussed in the following. Furthermore, some questions are left open for ongoing research. They will be discussed in the following as well.

Different Kinds of Changes

It can be distinguished mainly between three types of changed demand behavior.

First, unexpected demand behavior in an instance of the predominant period can occur that has not yet been observed in the past (e.g. on a bank holiday during normal weekly work). The model for the seasonal trend and the noise will be adapted as described in Section 6.5.3, if the observed demand is higher than the expected one. The changes will take effect in the next instance of the period.

The adapted models lead to wasted resources from now on, if the unexpected demand behavior was an onetime event. More resources than actually required are allocated in some cases. Additional occurrences of such onetime events at different times will more and more adapt the model upwards. Finally, the forecasts will not vary any more. The resource management can not use them to save energy. Hence, models should be completely recharacterized from time to time using data observed in the closest past to prevent this case.

Second, the predominant period or the trend behavior within the period can significantly change. Underestimates are corrected when they are detected, while overestimates are ignored as described before. The models are adjusted upwards finally leading to a static model again, which can not be used any more to save energy. Again, a complete recharacterization will be required to create a new model that appropriately describes the changed behavior.

Third, the long term trend can change over time. Such changes are adapted by the part of the model that described the long term trend. The scheduling algorithm directly considers such changes for ongoing redistribution decisions. Hence, neither an increasing nor an decreasing

long term trend will negatively influence the work of the dynamic resource management.

Changed Demand Behavior and SLOs

The demand behavior can be intentionally changed by clients to provoke SLO violations as already discussed in the static statistical resource management chapter. Such changes will very likely lead to resource shortages because provided resource capacity is directly adjusted to the expected demand by the dynamic resource management.

Time dependent usage behavior that causes time dependent resource variations must be integrated into the SLA to prevent such infiltration. Such extensions are not part of this thesis any more. Future research must address this question not only to improve the resource management concept presented in this thesis. Any kind of dynamic provisioning in conjunction with SLOs requires taking care of this issue.

Parallel Migrations While Resolving Resource Shortages

Finally, ongoing research can consider parallel migrations to resolve unexpected resource shortages. Depending on the situation, resource shortages of single VMs can be resolved significantly faster than specified by the respective planning period Δt^{ftr}. This will strongly loose the constraint concerning the SLO parameter $\Delta t_i^{resolve}$. Much more potential can be used by the resource management to save energy, if Δt^{ftr} is allowed to exceed $\Delta t_i^{resolve}$.

6.6 Summary

A new concept for dynamic resource management in virtualization based data centers has been presented within this chapter. This concept extends the static approach by the use of two additional degrees of freedom. VMs are moved between servers at runtime and servers can be switched on and off. The goal is to save energy in times of low overall resource demand by consolidating VMs to only a few servers and switching off unused ones.

This idea requires models that describe the resource demand of VMs expected in future. They must overestimate this demand with respect to long term trends, seasonal trends, and especially the random noise to not violate any SLOs. A weakness of known approaches is very often the way, they deal with the noise. The approach presented in this chapter combined the ideas of deterministic and stochastic seasonal models to overcome this drawback. The fine grained SLOs developed for the statistic static resource management are supported as well.

A second challenge that has been addressed in this chapter is the scheduling approach that dynamically redistributes VMs and switches on and off servers. None of the known ones can guarantee to resolve upcoming resource shortages right in time before they actually occur. Some of them can not even resolve them at all. The underlying problem has been decomposed

into a classical load rebalancing and a deadlock problem. An algorithm has been presented that resolves both at the same time also taking care of the delays of VM moves and server reactivations.

The dynamic resource management concept can guarantee not to violate any SLOs at any time based on some assumptions concerning the expected resource demand behavior. These assumptions are not met in any case especially when the demand behavior changes. Resource shortages can occur in some cases that can lead to SLO violations as a consequence. A method has been further presented for detecting and adapting such changes and for resolving possible resulting resource shortages. Furthermore, additional SLO parameters allow the clients to limit the impact of resource shortages caused by changed demand behavior.

7 Experimental Assessment

A concept for static and dynamic resource management has been developed within this thesis. Mainly three different challenges have been addressed. A new kind of SLO specification has been worked out that allows trading off service performance against provided resource capacity. In addition, the resource demand of VMs has been modeled in a way that resource capacity required in future can be determined from the demand observed in a characterization phase. Trade-offs between service performance and provided capacity defined in SLOs are supported. Finally, scheduling algorithms have been developed that based on the models and the SLOs can distribute VMs to a minimal number of active servers.

These main outcomes of this thesis will be evaluated and discussed within this chapter. First, the novel SLO specification will be compared to known ones concerning additional resource savings. The accuracy of the modeling approach will be evaluated in Section 7.2. Possible underestimates that can lead to SLO violations as well as overestimates that are wasting resources will be presented and discussed. Finally, the whole concept will be evaluated separately for static and dynamic resource management in Section 7.3 and Section 7.4 respectively. Section 7.5 will shortly conclude the results at the end of this chapter.

7.1 Fine Grained QoS Specification

A new kind of SLO specification has been introduced in Section 5.2. It allows trading off granted CPU time against service performance by defining performance goals and probabilities. The probabilities state how often the respective performance goals must be achieved. Different performance goals can be defined each with an individual probability in contrast to known percentile based specifications. This way, strong performance slowdowns can be limited very restrictively while weaker ones can be allowed to occur more often, which increases the flexibility for specifying SLOs. The advantages of such fine grained SLO specifications compared to classical percentile bases ones will be evaluated within this section.

7.1.1 Methodology

The CPU time savings that are achievable by a fine grained SLOs were determined in different analyses. They will be compared to the savings that can be achieved when classical percentile

based SLOs are applied. Furthermore, it will be shown how the number of defined performance goals will influences the savings. In both cases, the baseline forms the resource capacity that would be required, if no resource performance trade-offs are applied at all.

These analyses were limited to only one single VM first to not influence the results by unwanted side effects. It was determined how much CPU time must be provided to this VM to not violate its SLO. In further analyses, resource and energy savings that are obtainable in whole data centers when fine grained SLOs are applied were estimated as well. The results will be presented later on in Section 7.3 and 7.4 respectively.

The evaluation criteria used to assess the new SLO specification will be defined more formally in the following. The concrete evaluation scenario will be introduced as well.

Evaluation Criteria

Resource capacity $A_i(t)$ required by VM i to fulfill its SLO is derived from observed resource demand $R_i(t)$ according to the concept presented in this thesis. One resource demand time series $R_i(t)$ can lead to different resulting time series $A_i(t)$ depending on the SLO specification. Hence, these resulting $A_i(t)$s can be compared to assess different SLO specifications concerning resource savings.

Required resources capacity derived when fine grained SLOs were applied will be denoted by $A_{FG}(t)$ in the following. The approach suggested in Section 5.4.2 to derive required CPU time from the demand was used to determine $A_{FG}(t)$ from $R_i(t)$. A set of classical percentile based SLOs can be derived from each fine grained SLO. All of them can guarantee the same service performance like the fine grained specification itself. Required resource capacity determined with them will be called $A_{P_x}(t)$ in the following. The index x enumerates the different percentile based SLOs that belong to one fine grained specification. The common way to derive $A_{P_x}(t)$ from $R_i(t)$ has been presented in Section 5.4.1. Finally, $A_{base}(t)$ is the resource capacity that is required when no performance slowdowns must occur at all. The way to derive memory capacity from the demand (presented in Section 5.4.2 as well) was applied to derive $A_{base}(t)$ from $R_i(t)$.

The required resources $A_i(t)$ determined for a VM are used in different ways depending on whether static or dynamic resource management is performed. Maximally required resource capacity $A_i^{max} = max(A_i(t))$ is reserved for the VM in the static case. Hence, the respective maxima A_{FG}^{max}, $A_{P_x}^{max}$, and A_{base}^{max} must be compared to assess the SLO specifications with respect to static resource management. $A_{base}(t)$ serves as a baseline as mentioned before so that resource savings are the differences $A_{FG}^{max} - A_{base}^{max}$ and $A_{P_x}^{max} - A_{base}^{max}$ respectively. These savings are additionally normalized by R_i^{max} to describe them independent from the maximal resource demand of a VM. As a result, resource savings that are obtainable for a VM by static resource management when fine grained or percentile based SLOs are used can be expressed

as follows:

$$\widetilde{A}_{FG}^{max} = \frac{A_{FG}^{max} - A_{base}^{max}}{R_i^{max}} \qquad \text{and} \qquad \widetilde{A}_{P_x}^{max} = \frac{A_{P_x}^{max} - A_{base}^{max}}{R_i^{max}}. \tag{7.1}$$

A model $\hat{A}_i^*(t)$ is derived from $A_i(t)$ for dynamic resource management that describes resource capacity required at a certain time t. Hence, the complete time series $A_{FG}(t)$ or $A_{P_x}(t)$ decide about achievable resource savings. Again, savings are the differences $A_{FG}(t) - A_{base}(t)$ and $A_{P_x}(t) - A_{base}(t)$ respectively. Normalizing these differences by R_i^{max} leads to the following two functions:

$$\widetilde{A}_{FG}(t) = \frac{A_{FG}(t) - A_{base}(t)}{R_i^{max}} \qquad \text{and} \qquad \widetilde{A}_{P_x}(t) = \frac{A_{P_x}(t) - A_{base}(t)}{R_i^{max}} \tag{7.2}$$

that describe the obtainable resource savings when fine grained or percentile based SLOs are used. The maximum, the mean, and the standard deviation of the respective functions $\widetilde{A}_{FG}(t)$ and $\widetilde{A}_{P_x}(t)$ will be compared to assess the two different SLO specifications.

Evaluation Scenario and Data Selection

The obtainable savings when service performance is traded off against resource capacity mainly depend on two different parameters. First, the SLO specification itself is relevant for the savings. Second, the demand behavior of the VM also strongly influences the results. A simple web server scenario with one single page was created to find realistic values for them.

A realistic fine grained SLO specification with response time as metric for performance goals was defined for this web page. The goals η^{max} and respective probabilities $P_{\eta i}^{min}$ of this specification are presented in Figure 7.1 a). Each probability states how often the respective performance goal must be achieved.

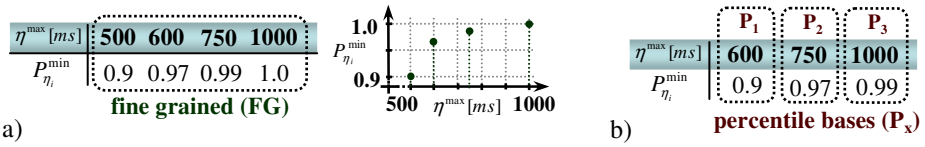


Figure 7.1: a) Fine grained SLO specification for an exemplary web server scenario. b) A set of classical percentile based SLO specifications that guarantee the same service performance like the fine grained one.

A set of three classical percentile based SLOs that guarantee the same service performance was derived from this specification. The respective parameters are presented in Figure 7.1 b). By definition, percentile based SLOs define one performance goal η^{max} and one probability $P_{\eta i}^{min}$ (cf. Section 5.2.1). But the performance goal defines the maximal allowed performance slowdown in contrast to the fine grained specification. The probability states how often any

performance slowdown down to this maximum is allowed. If for instance the targeted response time is $450ms$, the first SLO (P_1 in Figure 7.1 b)) allows exceeding this response time goal up to $600ms$ in 90% of time.

The fine grained SLO specification presented in Figure 7.1 served as a basis for all analyses. Several more restrictive ones were derived and analyzed as well. The results will be presented in the following section. The respective modification of the SLO will be explained in each case and the consequences will be discussed.

A real web server was installed in a VM to get realistic resource demand values of a VM and to characterize the mapping function $f(\alpha) : \alpha \mapsto \eta$ (cf. Section 5.2.3). Debian Linux with paravirtualization support was selected as underlying operating system. The virtualization environment was *XenServer 3.0.3*[1] installed on a desktop PC. The PC contained a single dual core CPU. One of the cores was assigned to the VM. The other one was exclusively reserved for the virtualization environment. A web page was created that consumes a fixed amount of CPU time for each request (28ms) to keep the analyses simple.

A lookup table based function $f(\alpha) : \alpha \mapsto \eta$ was characterized for this environment using synthetic load tests. This function returns the maximal response time η of the web server expected in case of a certain ratio α between resource demand R and provided capacity A. A wide range of values R and A has been evaluated for this characterization. All performance goals ever used in any of the SLOs during the evaluation are listed in Figure 7.2 with their respective α values. They were determined using this function $f(\alpha)$.

$\eta^{\max}[ms]$	**500**	**550**	**600**	**700**	**750**	**1000**
$\alpha^{\min}$	0.9	0.82	0.75	0.64	0.6	0.45

Figure 7.2: A set of response times and respective minimal values of α required to achieve these response time goals in the web server scenario.

A realistic resource demand time series $R_i(t)$ was determined in a next step using the web server environment. Therefore, a time series of requests to a real web server(the one from NASA Kennedy Space Center)[105] was selected. The corresponding log file contains time stamps of all accesses to the server in Juli 1995 (31 days). $R_i(t)$ was not directly determined by observing the VM while the requests are simulated. A complete replay of the whole recorded trace would have been required in real time. Instead, the resource demand of the VM was only measured for different numbers of concurrent accesses to the web page. $R_i(t)$ could then be easily calculated from the log trace of the real web server.

The CPU time required to fulfill a certain SLO specification ($A_{FG}(t)$, $A_{P_x}(t)$, and $A_{base}(t)$) could then be determined using the time series $R_i(t)$ and the function $f(\alpha)$. Different analyses

[1] This virtualization environment is contained as binary package in the current Debian Linux (Stable) operating system.

with different SLOs were performed. The respective SLO as well as the achieved savings will be presented and discussed in the following.

7.1.2 Comparison to Known Approaches

First, the new fine grained SLO specification will be compared to the known percentile based one. It will be worked out, how much and under which conditions the resource management can benefit from a fine grained specifications.

Three different fine grained SLOs were used. The first one has been defined in the previous section. The second and third SLO are modifications of the first one. More restrictive probabilities were assigned to the performance goals as a first modification. The second modification has a tighter range of the performance goals. The probabilities remained from the initial SLO specification in this case.

$\widetilde{A}_{FG}^{max}$ and $\widetilde{A}_{FG}(t)$ were determined for all of them. Furthermore, the percentile based SLOs were derived from the fine grained ones like described in the previous section. The resource savings $\widetilde{A}_{P_x}^{max}$ and $\widetilde{A}_{P_x}(t)$ achieved when they were used were determined as well. All results and the respective SLO specifications are presented in Figure 7.3.

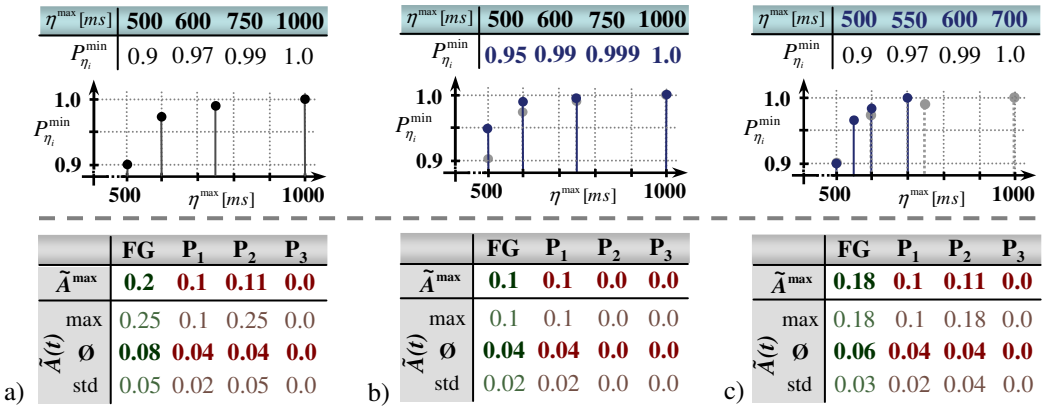


Figure 7.3: Normalized CPU time savings (bottom), when a fine grained SLO specification(top) is applied to a web server compared to percentile based specifications. a) the initial specification b) a modification with more restrictive probabilities c) a modification with a tighter range of performance goals

One can clearly see in Figure 7.3 a) that the fine grained SLO specification outperforms the percentile based one. None of the three classical SLOs was able to achieve the same savings like the fine grained one. Static resource management could save 20% of CPU time capacity when the fine grained SLO was applied to trade off resources capacity against service performance. The best of the classic SLOs was able to save only 11%. Dynamic resource management could

reserve 8% percent less CPU time capacity in average in the fine grained case. Only 4% could be saved when the percentile based specification was used.

The SLO specification P_3 was not able to provide any saving potential at all to the resource management. Any amount of reserved resource capacity A that is lower than the demand R will lead to performance slowdowns that occur more often than in one 1% of time. Due to this fact, the second fine grained SLO presented in Figure 7.3 b) was not able to achieve any additional savings compared to the percentile based one. The resource management could only take an advantage of the first out of four performance goals. The probabilities of all others are to restrictive.

The third case shows a different behavior (Figure 7.3 c)). The resource management can benefit from fine grained SLOs even when performance goals are more restrictive. The savings are slightly reduced compared to the initial specification but none of the percentile based SLOs could gain the same savings like the whole fine grained one.

7.1.3 Influence of the Number of Defined Performance Goals

Six different fine grained SLO specifications have been derived from the one that has been defined in Section 7.1.1 for a next analysis. All of them guarantee at least the same service performance like the initial one but define less performance goals. The missing performance goals are considered by restricting the actually defined ones some more. They are presented in Table 7.1 a).

a)

$P_{\eta_i}^{\min}$ \ $\eta^{\max}[ms]$	500	600	750	1000
FG	0.90	0.97	0.99	1.00
FG_1	0.97		0.99	1.00
FG_2	0.90	0.99		1.00
FG_3	0.90	0.97	1.00	
FG_4	0.99			1.00
FG_5	0.97		1.00	
FG_6	0.90	1.00		

b)

	$\tilde{A}_{FG}^{\max}$	$\tilde{A}_{FG}(t)$		
		max	**Ø**	std
FG	**0.20**	0.25	**0.08**	0.05
FG_1	**0.10**	0.10	**0.03**	0.03
FG_2	**0.10**	0.10	**0.04**	0.02
FG_3	**0.20**	0.25	**0.08**	0.05
FG_4	**0.00**	0.00	**0.00**	0.00
FG_5	**0.10**	0.10	**0.03**	0.03
FG_6	**0.10**	0.10	**0.04**	0.02

Table 7.1: Normalized CPU time savings with different numbers of defined performance goals in a fine grained SLO specification. a) six different specifications that guarantee all the same performance but with different numbers of defined performance goals b) the resulting CPU time savings when they are applied in contrast to the initial complete specification

One performance goal was removed in FG_1, FG_2, and FG_3. The probability $P_{\eta_i}^{min}$ of the next lower performance goal η^{max} was increased in each case to take care of the missing one. The same was done in FG_4, FG_5, and FG_5 but with two instead of one removed performance

goals. The respective savings provided to the resource management are listed in Table 7.1 b).

It can be mainly recognized that removing any performance goal leads to strongly reduced savings. Only FG_3 could achieve the same results like the complete SLO specification, which is not a big surprise in this case. Already earlier analyses showed that the last performance goal has no influence to the savings at all. Hence, removing it like done in FG_3 will not change the results.

Furthermore, removing two out of four performance goals vanishes the whole saving potential in this example, if none of both is the performance goal without any effect. Finally, the position from which single goals are removed seems not to strongly influence the results.

7.1.4 Conclusion and Limits of the Analyses

Finally, it can be concluded that the flexibility provided by a fine grained SLO specification can provide significant savings of CPU time that must be reserved for a VM. Static resource management could save 20% in an exemplary scenario. Dynamic resource management could benefit from 8% savings in average. None of the percentile based SLOs was able to exceeds these savings. The same savings could be achieved in some cases. In most cases, they were quite lower. Some of the percentile based SLOs were not able to achieve any savings at all. Different further analyses showed that removing performance goals from the fine grained SLO leads to strongly reduced savings. Defining only two out of four performance goals completely vanished the saving potential in most cases in the analyzed example.

The analyses presented and discussed so far are limited at some points concerning the assessment of the SLO specification. First, only CPU time savings obtainable for a single VM have been determined. These savings do not state anything about actual resource and energy savings in data centers. Furthermore, only one resource demand time series has been analyzed so far. Results of additional analyzes will be presented later on in Section 7.3 and Section 7.4 that took care of these weaknesses.

Furthermore, none of the analyses has evaluated so far, whether providing the determined resource capacity $A_i(t)$ to the demand $R_i(t)$ would actually satisfy the response time goals. Mainly an incorrectly determined resource demand time series $R_i(t)$, a wrongly characterized function $f(\alpha)$, or additional side effects could lead to violated performance goals. The scope of this thesis is limited to guarantees concerning the ratio between resource demand R and provided resource capacity A as already discussed in Section 5.2. Hence, experiments concerning the actual response time are out of scope. The web server scenario with response time as performance metric has been only introduced to get realistic values for $R_i(t)$ and the SLOs.

7.2 Resource Demand Model

The static and dynamic resource management presented in this thesis mainly depends on accurately forecasted resource demand of the VMs in the future. An appropriate modeling method was developed for this purpose. It has been presented in Section 5.3 and 6.3 respectively.

This modeling method will be evaluated within this section. One can distinguish between over- and underestimates concerning the forecasting accuracy. Underestimates can lead to SLO violations. Overestimates waste resources that otherwise could have been used by the resource management to save energy. Hence, the model will be mainly evaluated by forecasting accuracy in terms of underestimates and saving potential provided to the resource management.

7.2.1 Methodology

Different analyses were performed to evaluate the modeling and forecasting method. Resource demand time series of different real servers were used. Models were characterized using one part of the time series (duration: Δt_{p_2}) in all cases. The second part was compared to a time series that was forecasted using the models (duration: Δt_{p_3}). The assessment criteria are accuracy of the forecasts and provided saving potential as mentioned before. Both will be detailed some more in the following. The time series used for the evaluation will be shortly presented as well.

Accuracy of the Forecasts

The accuracy simply was evaluated by comparing the difference of the forecasted and the actually measured resources in relation to the maximal demand of the VM. One must differentiate between static and dynamic resource management. Maximally required resources A^{max} are extrapolated from the models in the first case. The time dependent function $A(t)$ is estimated for dynamic resource management based on the models. The accuracy can be formally described by an equation in each of both cases as follows:

$$\delta^{stat}(t) = \frac{A^{max} - A^{mes}(t)}{R^{max}} \qquad \delta^{dyn}(t) = \frac{A^{for}(t) - A^{mes}(t)}{R^{max}}. \tag{7.3}$$

Both functions return values between -1.0 and 1.0. A value below 0.0 indicates underestimates. The model overestimated the real demand in case of δ above 0.0. Mainly the probabilities $P(\delta < 0)$ and $P(\delta > 0)$ will be presented in the following to assess the modeling method concerning accuracy. They point out how often the model underestimated or overestimated the actually required resources. Additionally, the probability function will be presented in some cases as well.

Provided Saving Potential

To assess the saving potential provided by a forecasting method, resource savings were determined that an exact forecasting method could provide to the scheduling approach in case of a certain demand time series $A(t)$. These savings were compared to the ones that the analyzed method actually provided.

Resource savings can be expressed as the ratio between required resources of a VM at time t and its maximal demand A^{max}, which can be formally described by a function $\pi(t)$ as follows:

$$\pi(t) = 1 - \frac{A(t)}{A^{max}}. \tag{7.4}$$

Time intervals are needed in which a certain π^{min} is not deceeded to use these savings for dynamic resource management. Furthermore, such intervals must have a minimal length Δt^{min} because VMs must be migrated and servers must be switched on and off. Such saving intervals will be denoted by SI^l in the following and are formally defined as follows:

$$\forall t \in SI^l : \pi(t) \geq \pi^{min} \wedge |SI^l| \geq \Delta t^{min}, \tag{7.5}$$

whereas l enumerates different of such intervals along time t and $|SI^l|$ is the length of interval l. Additionally, SI^{all} is defined as a set of all disjoint saving intervals that all do not fall below a minimal length of Δt^{min} and have minimal savings of π^{min} during the whole interval. A function $\Delta\tilde{t}^{sav}(\pi^{min}, \Delta t^{min})$ can be derived from such definitions as follows:

$$\Delta\tilde{t}^{sav}(\pi^{min}, \Delta t^{min}) = \frac{\sum\limits_{\forall SI^l \in SI^{all}} |SI^l|}{\Delta t_{p_3}}. \tag{7.6}$$

This function describes the relative overall time, in which a forecasting method provides minimal saving potential of π^{min} to the scheduling approach. All considered saving intervals have a duration longer than Δt^{min}.

The relative maximal overall resource savings Π^{max} that are achievable for a given time series of length Δt_{p_3} can be determined in addition to $\Delta\tilde{t}^{sav}$ as follows:

$$\Pi^{max} = \frac{1}{\Delta t_{p_3}} \int_{t_{p_3}}^{t_{p_3}+\Delta t_{p_3}} \pi(t) dt. \tag{7.7}$$

Resource savings of Π^{max} can be achieved by a scheduling approach, if it would be able to use the whole amount of resource savings $\pi(t)$ at any time t.

Evaluation results concerning provided saving potential will be mainly represented by Π^{max} in the following. Additionally, function $\Delta\tilde{t}^{sav}(\pi^{min}, \Delta t^{min})$ will be visualized in some cases to

show how often which amount of savings could be provided in the analyzed case.

Data Selection

The resource demand time series used for evaluation were provided by a medium-sized Service Provider located in Oldenburg(Germany)[2]. The company hosts different IT services for different SMEs. The services range from simple e-mail and web server to individual test systems and whole ERP solutions. Different batch job based services such as backups or virus scans are performed in the data center as well.

Time series of CPU time and memory demand of a subset of services have already been recorded for capacity planning and observation reasons. These time series last up to one year in the past and have a resolution of one sample per five minutes.

All analyses discussed within this section were performed with many of these time series. But not all results will be presented for clarity reason. Only one appropriate representative was selected for each analysis to discuss the respective results. Rough estimates of average accuracy and saving potential will be presented at the end of this section for the whole set.

7.2.2 Comparison to Known Approaches

The modeling approach presented in this thesis was compared to known ones in a first analysis. Two month of the resource demand time series were used to characterize the model ($\Delta t_{p_2} =$ 61 days). The remaining ten month were used to compare the forecasts to the actually required resources ($\Delta t_{p_3} = 192$ days).

Selected Approaches

A common approach known from classical time series analysis was applied for comparison. The seasonal trend was modeled by a deterministic function derived using moving averaging. The residuals were treated as random noise that was modeled using the ARIMA approach.

The noise was pessimistically overestimated in a first version of this approach to derive required resources from the models the same way like suggested in [17, 129, 104]. This approach will be called MA^+ in the following. The saving potential the approach provides is expected to be very low, since MA^+ pessimistically overestimates the noise. Hence, a modified version of this approach, called MA^-, was additionally analyzed that (very optimistically) completely ignores the noise. Only the trend function is used for forecasting, which is expected to result in many and strong underestimates but also should strongly increase the saving potential.

Finally, the result achieved by MA^- and MA^+ can be regarded as upper and lower limits for the approach presented in this thesis. The new approach should provide significantly higher

[2]NOWIS Nordwest-Informationssysteme GmbH & Co. KG

saving potential compared to MA^+, while the underestimates of MA^- are not exceeded.

Discussion of Results

Accuracy and saving potential achieved by the different modeling approaches when they were used for dynamic resource management are presented in Figure 7.4 and 7.5 once for CPU time and once for memory. In both cases, the new model was analyzed once with and once without performing error correction at runtime the way presented in Section 6.5.3. The respective cases are annotated by *EC* and *no EC* respectively.

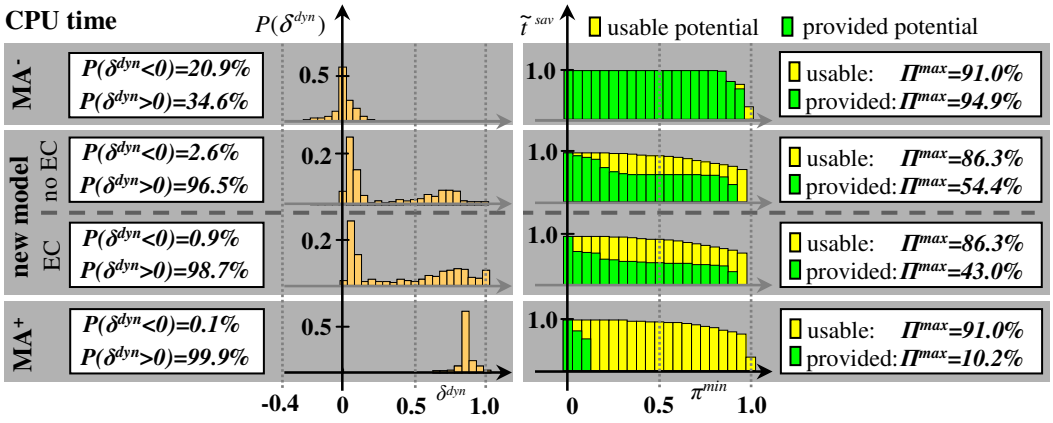


Figure 7.4: Accuracy and provided saving potential of different modeling approaches, when they are applied to forecast required CPU time for dynamic resource management.

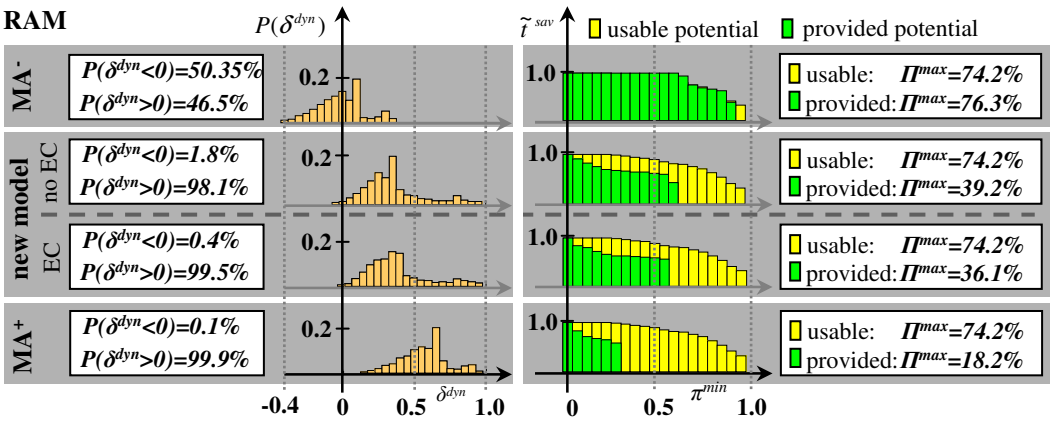


Figure 7.5: Accuracy and provided saving potential of different modeling approaches when they are applied to forecast required memory for dynamic resource management.

A probability distribution of $\delta^{dyn}(t)$ is presented on the left side of both figures for each approach. The cumulated probabilities of over- and underestimates are given as well. The saving potential determined the way introduced in previous section is presented for each approach on the right side of the figures. The bar plots show the overall time period, in which savings higher than a certain π^{min} could be provided to the scheduling approach related to the overall simulation time. The yellow bars represent the savings that theoretically could be provided in case of an exact forecasting method. The green ones are the savings that the analyzed methods actually provided. Furthermore, the overall resource saving potentials Π^{max} provided by the exact forecasting approach and by the real ones are presented as well.

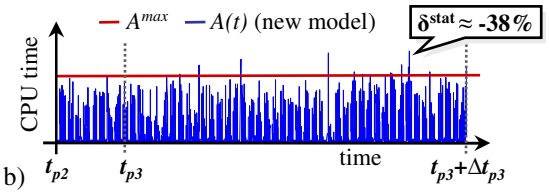
One can clearly see that the forecasting accuracy is already very close to the one of MA^+ even without error correction at runtime. The estimated resources were lower than the actual ones only in up to 2.6% of time. Error correction performed at runtime can further reduce underestimates to below 1.0%.

The new approach could significantly increase the savings potential of MA^+. Only very low savings π could be provided to the scheduling approach by MA^+. A reason is that not the trend of the resource demand time series but the noise performance is varying very strong over time. MA^+ assumes a constant noise performance and hence could not take any advantage of varying noise performance. This fact becomes further apart with respect to the results of MA^-. Neglecting the noise leads to even more provided saving potential than actually present. Underestimates in up to 50% of time are the consequence.

The forecasting accuracy of the models for static resource management was analyzed as well. The results are presented in the table in Figure 7.6 a). A^{max} needs not to be estimated for memory using the models, since $A^{max} = R^{max}$ is valid for memory capacity (cf. Section 5.4.2). R^{max} is determined in phase one of the concept using benchmarks and thus can be directly used as A^{max} in phase two and three as well. Hence, only the accuracy of the forecasted A^{max} for CPU time was analyzed.

CPU time	MA⁻	new model	MA⁺
$P(\delta^{stat} < 0)$	0.1%	0.2%	0.1%
$\min(\delta^{stat})$	-44.3%	-37.9%	-44.3%

a)



b)

Figure 7.6: a) Accuracy of the modeling approaches when they are applied for static resource management b) An exemplary time series and the forecasted A^{max}

All three analyzed approaches underestimated the actually required CPU time in only up to 0.2% of time. One can see in Figure 7.6 b) that only single spikes of the actually required resources $A(t)$ exceed the provided resource capacity A^{max}. But these spikes can be signifi-

cantly higher than A^{max}, which can lead to strong performance slowdowns in these cases. The reason for such violations is mainly a changed long term trend in phase three. The slope of the trend slightly increases so that A^{max} was estimated to low.

7.2.3 Finding the Predominant Period

Finding a good predominant period to extract the forecasting model for dynamic resource management from the observed demand behavior is very important. Provided saving potential as well as the probability of underestimates are strongly influenced by the period found. The length of the time period Δt_{p_2} used to train the models as well as the averaging interval Δt^{avg} used to characterize the stationary processes are influencing the periods determined by the modeling approach. The periods found for an exemplary VM are presented in Table 7.2 for different combinations of Δt_{p_2} and Δt^{avg}. Each combination leads to two periods one for CPU time and one for memory.

Δt_{p_2} / Δt^{avg}	3 days		7 days		14 days		28 days		56 days	
	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM
30 min.	0.0	0.0	1.0	0.92	1.0	1.0	1.0	7.0	7.0	7.0
60 min.	0.0	0.0	1.0	0.92	1.0	1.0	1.0	7.0	7.0	7.0
120 min.	1.25	0.0	1.0	0.92	1.0	1.0	7.0	7.0	7.0	7.0
360 min.	1.08	0.0	1.0	0.92	1.0	1.0	7.0	7.0	7.0	7.0

Table 7.2: Predominant periods (in days) found by the modeling approach depending on the duration Δt_{p_2} of the characterization phase and on the length of the averaging interval Δt^{avg}.

The approach could find periods of exact one day for CPU time as well as for memory after a characterization phase longer than 14 days. Periods of exact 7.0 days were found in some cases with data observed longer then 4 weeks. It seems to help to increase the averaging interval with increasing Δt_{p_2}. This smooths out differences of the different instances of the period, which helps to find appropriate periods.

Saving potential and forecasting accuracy of the new modeling approach were analyzed for three of the periods found. The results are presented in Table 7.3.

A period of 0.92 days obviously does not fit very well. The required CPU time has been underestimated in 13.1% of time. Adapting the model to forecasting errors at runtime can help to reduce the underestimates to 1.1%. But the saving potential is strongly reduced at the same time as well from 45.4% down to only 5.8%.

A period of 1.0 day fits quite better. The approach underestimates in only 5.8% of time without runtime error correction. Enabling error correction at runtime reduces the underes-

		period of 0.92		period of 1.0		period of 7.0	
		$P(\delta^{dyn}<0)$	Π^{max}	$P(\delta^{dyn}<0)$	Π^{max}	$P(\delta^{dyn}<0)$	Π^{max}
CPU	EC	1.1%	**5.8%**	1.0%	**39.8%**	3.2%	**62.6%**
	no EC	**13.7%**	45.4%	5.8%	56.8%	10.4%	74.6%
RAM	EC	0.4%	**4.3%**	0.2%	**34.0%**	1.7%	**55.8%**
	no EC	**10.3%**	33.7%	1.3%	36.7%	4.98%	59.8%

Table 7.3: Accuracy and saving potential in case of different predominant periods.

timates to 1.0%. But the saving potential additionally reduces as well by about 17%. The reason is that there are a few days in the time series that have significant higher resource demand compared to all others, which has been adapted by the model. The model strongly overestimates parts of the following days after these discontinuities occurred.

Finally, a period of 7 days further increases the saving potential compared to an one day period. But the underestimates are increased as well because not so many instances of the period were used for characterization, which resulted in a worse model. Runtime error correction performs worse as well because adaptations of the model affect the forecasting not before the following instance of the period (the following week). The instance repeats far less often compared to a daily period.

7.2.4 Influence of Minimal Duration of Saving Intervals

Minimal time periods Δt^{min} are required, in which the resources to be reserved for a VM must remain below a certain threshold to allow the dynamic resource management to save any energy at all. The theoretically usable saving potential, the actually provided potential, and the ratio between both of them was calculated for different values of Δt^{min} ones for CPU time and ones for memory. The analyzed values of Δt^{min} ranged from $5min$ up to $6h$. A value below $5min$ can hardly be used to save energy since moving one VM between two servers already takes between $30s$ and $60s$. $6h$ is quiet a long time for redistributing VMs and switching of some servers.

a)

CPU time	5min	30min	120min	360min
usable Π^{max}	87.8%	86.3%	82.1%	73.4%
provided Π^{max}	45.0%	43.0%	36.6%	32.3%
$\frac{\text{usable } \Pi^{max}}{\text{provided } \Pi^{max}}$	**51.3%**	**49.8%**	**44.6%**	**44.0%**

b)

RAM	5min	30min	120min	360min
usable Π^{max}	75.1%	74.1%	71.5%	65.4%
provided Π^{max}	37.2%	36.1%	32.3%	22.8%
$\frac{\text{usable } \Pi^{max}}{\text{provided } \Pi^{max}}$	**49.5%**	**48.7%**	**45.2%**	**34.9%**

Table 7.4: Theoretically usable and actually provided saving potential depending on different values of Δt^{min}. a) for CPU time b) for memory capacity

The results are presented in Table 7.4. One can see that the theoretically usable saving potential as well as the potential actually provided by our approach only slightly decreases with increasing Δt^{min}. As a result, the modeling approach can provide significant saving potential to the scheduling approach, even when longer saving intervals are required because of many unsafe VMs or many inactive servers.

7.2.5 Influence of Long Term Trends

The influence of a long term trend to the saving potential and the accuracy was analyzed in a next experiment. The CPU time and memory behavior of a VM with a significant long term trend was modeled once with and once without explicitly modeling the trend. The results are presented in Table 7.5.

a)

CPU time	LT modeled		LT not modeled	
	no EC	EC	no EC	EC
$P(\delta^{stat} < 0)$	0.0%		0.3%	
$P(\delta^{dyn} < 0)$	4.3%	1.4%	5.0%	0.6%
provided Π^{max}	36.2%	31.5%	29.7%	15.7%

b)

RAM	LT modeled		LT not modeled	
	no EC	EC	no EC	EC
$P(\delta^{stat} < 0)$	0.0%		0.0%	
$P(\delta^{dyn} < 0)$	**0.1%**	**0.1%**	**13.8%**	**3.2%**
provided Π^{max}	**14.4%**	**14.2%**	**12.9%**	**7.4%**

Table 7.5: The influence of long term trends to accuracy and saving potential, when the long term trend is either modeled explicitly or not. a) for CPU time b) for memory.

One can see that not explicitly modeling the long term trend can lead to many underestimates (13.2% in case of memory). Performing error correction could reduce them to 3.2% but also strongly reduced the provided saving potential.

7.2.6 Different VMs

A set of 23 different VMs was analyzed in a last analysis to get a more general impression of accuracy and saving potential provided by the new modeling approach. Observed data of two month ($\Delta t_{p_2} = 61$ days) was selected for the characterization phase.

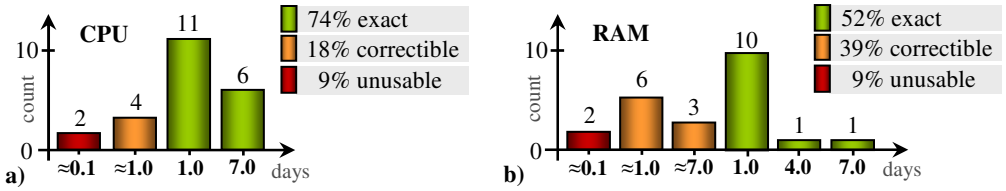


Figure 7.7: Predominant periods determined for a set of 23 different VMs. a) for CPU time b) for memory

First, the predominant periods were determined. The results are presented in Figure 7.7. An appropriate period could be found for memory as well as for CPU time for a significant number of VMs. The determined period slightly missed the appropriate one for some VMs. The method suggested in [44] was applied to fit them to the exact ones. Unusable periods were found in four cases (two for CPU time and two for memory). The demand behavior can be only statically modeled (cf. Section 5.3) in those cases because of missing periodic behavior.

Furthermore, the accuracy of the forecasts was determined for all VMs with respect to static and dynamic resource management. The provided saving potential was calculated as well. The results of all VMs were summarized and are represented in Table 7.6 by their mean, their standard deviation, and their maximum.

	RAM			**CPU (forecasted A^{max})**			**CPU (fixed A^{max})**		
	Ø	std	**max**	**Ø**	std	**max**	**Ø**	std	**max**
$P(\delta^{stat} < 0)$	0.0%	0.0%	0.0%	**1.3%**	3.0%	**13.0%**	0.0%	0.0%	0.0%
$P(\delta^{dyn} < 0)$	**2.7%**	2.9%	**10.0%**	**3.9%**	4.4%	**20.4%**	**2.3%**	2.6%	**9.0%**
provided Π^{max}	**35.1%**	20.0%	**78.2%**	25.6%	27.4%	**82.4%**	49.6%	21.1%	**84.6%**

Table 7.6: Accuracy and saving potential of a set of 23 VMs. The CPU time was analyzed twice once with forecasted A^{max} and once with $A^{max} = R^{max}$.

The left three columns contain the results for the memory demand of the VMs. Underestimates of $A(t)$ occurred in less than 3% of time in most cases. Saving potential of up 78% could be provided to the scheduling approach.

Only some VMs led to a significant larger amount of underestimates. Less resources than actually required were forecasted in up to 10% of time in worst case. The respective time series (forecasted and measured) is presented in Figure 7.8 b).

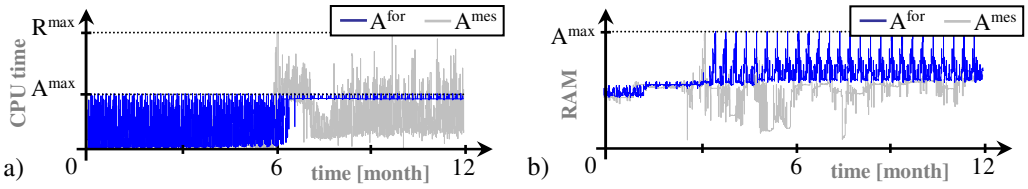


Figure 7.8: Exemplary time series for CPU time and memory that led to a significant amount of underestimates.

One can see that memory demand is hardly varying during the first two month(the ones used for characterization). The variation is suddenly getting significantly stronger after the third month. The forecasts are exceeded by the real demand very often and the model is continuously adapted upwards. The resource management or the administrator must change

to the static model and recharacterize the dynamic one again in such a case.

Results that belong to CPU time modeled the way presented in this thesis are presented in the middle of Table 7.6. Again, in most cases the actual demand was underestimated in less than 3% of time. Savings of up to 82.4% could be provided.

A few of the time series led to significant underestimates comparable to memory. The time series (forecasted and measured) that belong to the worst case is presented in Figure 7.8 a). Again, the resource demand behavior suddenly increased after about 6 month. Such changes influence the resource management even worse in contrast to memory demand. The reason is that the maximally required CPU time A^{max} is forecasted as well as the time dependent function $A(t)$ using the models. The value of A^{max} can not be changed at runtime in contrast to $A(t)$. There is no way to adapt to any changes. A complete recharacterization of the models must be performed. Furthermore, a new static safe distribution of VMs to servers must be found based on the new models.

Additional results of analyses with CPU time are presented at the right side of Table 7.6. But A^{max} was not forecasted but set to R^{max} in this case comparable to memory. One can see that the underestimates of this worst case could be strongly reduced this way.

7.2.7 Conclusion and Limits of the Analyses

Most of the resource demand time series of VMs can be modeled very well with the modeling approach that has been presented in this thesis. Underestimates typically occurred in less than 3% of time regarding the simulations with demand time series of real VMs that last over one year. Such an amount of underestimates is hardly larger than the one of a pessimistic known approach. But the saving potential that the new modeling approach provides to the resource management to save energy is significantly higher. Most of the saving intervals are long enough to be used by dynamic resource management to save energy.

The found predominant period significantly influences the accuracy and saving potential. An appropriate one can be found exactly for the most VMs. Some of them slightly missed the exact one, which can be corrected using known methods. These methods mainly try to adapt the incorrect period to time periods that are typical in data centers (multiples of hours, days, or weeks).

Finally, changed demand behavior can lead to significant underestimates. They can be adapted very well, if these changes do not exceed the respective A^{max}. Otherwise, phase two of the resource management concept must be repeated because A^{max} can not be increased without determining a complete new safe distribution of VMs to servers.

This requirement is a weakness of the resource management concept. Future research can try to extend the concept by a method that allows rebuilding a new safe distribution of VMs to servers at runtime to overcome this drawback. A^{max} could be automatically adapted at

runtime with this extension in case of unexpectedly increased resource demand.

The evaluation of the modeling concept is limited at some points. First, only a limited set of resource demand time series was available. This way, statements about the models are limited to only the respective VMs. Furthermore, the resolution of $5min$ per sample is quite too low to guarantee any performance goals at all. Providing CPU time that has been measured as an average over $5min$ will probably violate the performance goal, if a response times of a few seconds must be guaranteed. Therefore, measuring intervals of far below one second are required. The gathered data can be directly summarized at runtime in a histogram that describes the averaging interval Δt^{avg} to reduce the runtime complexity of the model characterization. Such fine grained resource demand time series were not available for the evaluation. Hence, respective analysis must be performed by future research as well.

7.3 Statistical Static Resource Management

The fine grained SLO specification and the modeling approach have been analyzed only for single VMs concerning resource saving potential and accuracy until now. The whole resource management concept will be assessed in the next two sections. First, statistic static resource management (cf. Chapter 5) will be analyzed within this section. Dynamic resource management (cf. Chapter 6) will be regarded later on in Section 7.4.

7.3.1 Methodology

Two different approaches for statistic static resource management have been developed within this thesis. The first one denoted by *NewStatPess* in the following pessimistically assumes positive correlations between the resource demand behavior of different VMs. The second one called *NewStatOpt* uses actual correlations for a more optimistic resource management.

Both were implemented in a simulation environment. VMs were simulated using demand time series observed from real services. Resource demand models were characterized using the first part of each time series comparable to the model evaluation. The analyzed resource management approach determined a distribution of VMs to servers based on these models. The second part of the time series simulated the actual demand of the VM. The simulation environment could determine the resource utilization of each server at each time step based on this demand to detect possible resource shortages.

The number of required servers as well as the amount of resource shortages that occurred was a result of each simulation run. The number of servers was related to the servers that pessimistic static resource management (cf. Chapter 4) required to quantify resource saving. This pessimistic approach served as the baseline and will be called *KnownPess* from now on.

KnownPess very pessimistically assigns VMs to servers without taking any information about the actual resource demand behavior into account to prevent any resource shortages.

Both evaluation criteria will be detailed some more in the following comparable to the previous sections. The resource demand time series used for the simulation will be shortly described as well before the results of the analyses will be presented and discussed.

Evaluation Criteria

The resource management approaches are evaluated with respect to server savings and the amount of possibly occurring resource shortages. Energy savings are not directly used as measure. The relation between saved servers and saved energy will be shortly discussed in Section 7.3.4 to give an impression about expected energy savings as well.

Server saving can be formally described by following equation:

$$\Delta\widetilde{\#}^{srv} = \frac{\#^{srv}_{pess} - \#^{srv}_{stat}}{\#^{srv}_{pess}}. \tag{7.8}$$

The number of servers required by *KnownPess* is denoted by $\#^{srv}_{pess}$. The resulting number when one of the statistic approaches was used is called $\#^{srv}_{stat}$. Savings $\Delta\widetilde{\#}^{srv}$ are simply the difference between both in relation to the number of servers required by *KnownPess*.

Resource shortages that can lead to SLO violations of a VM will only occur, if two conditions are met. First, the resource capacity required by the VM at a time t exceeds the expected maximum A^{max}_i for at least one resource type. Second, the overall resource demand at t of all VMs placed on the respective server exceeds the servers capacity of the same resource type. Only exceeding A^{max}_i will not lead to any problems as long as enough resource capacity is left at the server for the additional demand.

A relative frequency P^{rs}_i of resource shortages that occurred for a VM i during a simulation run can be defined as follows based on these conditions:

$$P^{rs}_i = P(A^{mes}_i(t) > A^{max}_i \ \wedge \sum_{j:B(j)=k} R^{mes}_j(t) > C_k) \quad \text{with} \quad k = B(i). \tag{7.9}$$

Actually required resources $A^{mes}_i(t)$ are determined from the simulated resource demand of VM i. This resource capacity should have been allocated to prevent any SLO violations. Resources A^{max}_i were determined by the resource management approach to find a suitable distribution of VMs to servers.

P^{rs}_i was determined for all VMs in each simulation run. The results will be represented by their maximum, their mean, and their standard deviation in the following sections.

Evaluation Scenario and Data Selection

The same data set like already used to evaluate the resource demand models (cf. Section 7.2.1) served as the basis for this part of the evaluation as well. This initial set consisted of resource demand traces of 23 services observed in a real data center. A set of 50 VMs was created out of the original data set to simulate at least a small data center with a realistic size.

The first two month of each time series were used to characterize the models (Δt_{p_2} = 61 days). The rest simulated the actual resource demand ($\Delta t_{p_3} = 192$ days).

All servers had the same configuration in each simulation. Four different server configurations were used in different analyses. They are presented in Table 7.7.

	normal	**low RAM**	**low CPU**	**low RAM & CPU**
memory cap.	32 GB	8 GB	32 GB	8 GB
CPU cores	4x	4x	2x	2x
$\#^{srv}_{pess}$	**12**	**17**	**24**	**25**

Table 7.7: Four different server configurations were used to evaluate the concept for static resource management presented in this thesis. The number of servers required by *KnownPess* is contained in the table for each configuration as well.

KnownPess was performed using each of the four configurations to distribute the VMs to servers. The resulting number of servers served as the base line for hardware savings as mentioned before. The results are presented in the table as well.

7.3.2 Comparison to Known Approaches

The following four different static resource management approaches were compared against each other concerning the required server count and the frequency of resource shortages in a first analysis:

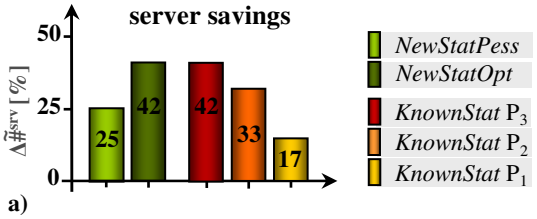
- *KnownPess*: Pessimistic not statistic resource management as described in Chapter 4 (baseline for savings),
- *KnownStat*: Classical statistic resource management as suggested in [117, 58, 45],
- *NewStatPess*: The new statistic resource management that pessimistically overestimates correlations as described in Section 5.4.2, and
- *NewStatOpt*: The new statistic resource management that additionally uses correlations for resource management as described in Chapter 5.4.5.

The first one distributes VMs to servers according to required resources that are determined using benchmarks. Enough resources are reserved for each VM to meet its demand even in case of the maximally expected workload. No SLO violations caused by resource shortages will occur. The second one has been shortly described in Section 5.4.1. Resource demand is modeled using one single random variable. Required resource capacity is determined as percentile of this variable. The second and third one are the approaches that were developed within this thesis. They have been presented in Chapter 5.

All four approaches were used to distribute the set of 50 VMs to servers. The same SLO was assigned to all of them in each simulation run. The fine grained specification that has already been introduced in Section 7.1.1 (Figure 7.1 a)) was selected for both of the new statistic approaches. Three different percentile based SLO specifications (Figure 7.1 b)) were used with *KnownStat*. All of them guarantee the same performance like the fine grained specification as described in Section 7.1.1. The *normal* server configuration (4 cores, 32 GB) was used in all simulation runs.

Discussion of Results

The results are presented in Figure 7.9. First of all, one can say that statistical resource management can in general significantly reduce the number of required servers compared to the pessimistic not statistic approach. Savings between 17% and 42% could be achieved.



b)

resource shortages

P_i^{rs}	*NewStat*		*KnownStat*		
	Pess	Opt	P_1	P_2	P_3
max	**0.15%**	**0.15%**	**0.00%**	**0.12%**	**0.90%**
Ø	0.02%	0.02%	0.00%	0.06%	0.40%
std	0.05%	0.05%	0.00%	0.05%	0.27%

Figure 7.9: a) Server savings of different statistical static resource management approaches. b) The respective relative frequencies of resource shortages.

Not all percentile based SLOs led to the same savings comparable to the evaluation result of the fine grained SLO specification. Mainly the minimal performance goals η^{max} that must be achieved in any case limited the savings in this evaluation scenario. The less restrictive this performance goal was selected, the more savings could be achieved by *KnownStat* independent from the assigned probabilities.

NewStatPess could not achieve the same savings like *KnownStat* in most cases. This result was expected, since positively correlated resource demand behavior is pessimistically assumed by *NewStatPess*. The same savings could be only achieved, when correlations are used for a

more optimistic planning (*NewStatOpt*).

An interesting fact concerns the additional savings of *NewStatOpt*. Although most VMs have more or less positively correlated workload (correlation coefficient: between 0.3 and 0.6), using correlations for resource management could increase the savings from 25% up to 42%. The reason is that mainly the lower resource demand values that are dominating the time series are positively correlated. The high spikes of different VMs are often uncorrelated. They are even negatively correlated in some cases. Hence, using correlations can in any case significantly increase the savings, since only the higher resource demand values are relevant for static resource management.

The resulting distribution of VMs to servers hardly led to actual resource shortages despite the comparable large amount of underestimates of the models (cf. Section 7.2). The reason mainly was unused resource capacity that remained at the servers. Most of the VM required far more than 50% of CPU time ($A_i^{max} > 0.5$). Hence, a significant amount of CPU time capacity remained unused, since only 400% of CPU capacity (4 cores) was available. The remaining capacity could not satisfy the demand of one further VM in many cases. Furthermore, VMs require their maximal resource demand rarely at the same time as discussed before. Hence, such negative correlations can prevent resource shortages, even if one VM exceeds the expected demand.

The disadvantages of *KnownStat* have been discussed in Section 5.4.1. The assumption of uncorrelated resource demand behavior was pointed out as the major weakness. Such weakness did not lead to significant resource shortages in this evaluation scenario. The reason is that mainly the minimal performance goal η^{max} that must be achieved in any case. This performance goal determined the required resource capacity A_i^{max}. Providing such minimal resource capacity A_i^{max} to a VM results in far less resource shortages than specified by the limits $P_{\eta_i}^{min}$. Hence, the incorrectly calculated probability distribution of the overall resource demand had no effect at all.

One could now believe that performing any statistics to ensure meeting $P_{\eta_i}^{min}$ does not make any sense at all for realistic values of $P_{\eta_i}^{min}$ and η^{max} and for realistic resource demand behavior. But these result are again an artifact of the low sample interval of the resource demand. Shorter resource demand time series (with a duration of 21 days) that had a sample rate of one sample per minute were analyzed as well. These time series already showed quite a stronger deviation of the noise performance. As a consequence, the resulting percentile is much higher compared to the one obtained when the strongly averaged time series were used. Analyses with such more fine sampled time series were presented in [58]. The authors showed that indeed neglecting correlations can lead to significant SLO violations.

It has been already pointed out in the previous section that sample intervals of $5min$ are far to long to guarantee any performance goal at all. A sample rate of several samples per

second would be more appropriate. Neglecting correlations between such fine sampled resource demand time series are expected to lead to significant larger errors according to the results presented in [58]. Such analysis could not be performed within this evaluation due to missing data.

7.3.3 Influence of Server Configuration

Server savings and the frequency of resource shortages were determined in a second analysis for different server configurations. Especially the differences between *NewStatPess* and *NewStatOpt* were focused.

Both of the new statistic approaches were simulated with each of the four server configurations presented in Section 7.3.1. The achieved server savings are presented in Figure 7.10.

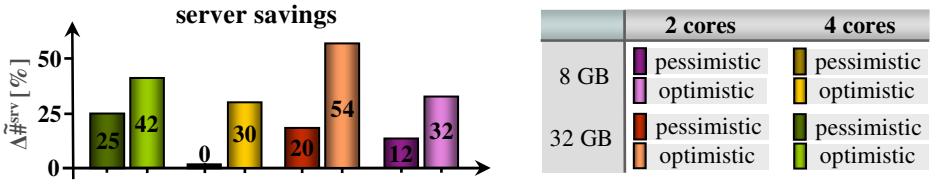


Figure 7.10: Servers savings when *NewStatPess* and *NewStatOpt* are applied for resource management with different server configurations.

In case of $32GB$ memory, CPU time was the limiting resource type that mainly decides about the number of required servers. *NewStatPess* could achieve slightly less savings with 2 cores compared to a 4 cores machine. Mainly the different ratios between resource capacity required by a VM and the overall capacity provided by a server are responsible for this result. Applying resource performance trade-offs typically slightly reduces the resource capacity required by each VM. The lower the required capacity is in contrast to the servers capacity, the higher is the probability that slightly reducing the capacity can lead to actual server savings.

Applying *NewStatOpt* could strongly reduce the number of required servers in both cases in contrast. Resource management could significantly benefit from additionally considering correlations especially in case of 2 CPU cores.

Memory was the limiting resource type, when servers with only $8GB$ memory capacity were used. Resource management could hardly take some advantage of applying *NewStatPess*, since resource performance trade-offs can be only applied to CPU time. *NewStatOpt* in contrast could achieve around 30% server saving. This result is not surprising since memory demand is varying over time as well as CPU time. Hence, correlations between the memory demand time series can be also used to more efficiently distribute VMs to servers.

The relative frequencies of resource shortages determined in each analysis are presented

Table 7.8. The maximal amount of resource shortages measured for a VM was below 1% in

	4 cores, 32 GB		4 cores, 8 GB		2 cores, 32 GB		2 cores, 8 GB	
	pess.	opt.	pess.	opt.	pess.	opt.	pess.	opt.
max	**0.15%**	**0.15%**	**0.00%**	**0.86%**	**0.26%**	**0.56%**	**0.15%**	**0.15%**
Ø	0.02%	0.02%	0.00%	0.03%	0.03%	0.08%	0.01%	0.02%
std	0.05%	0.05%	0.00%	0.12%	0.07%	0.14%	0.04%	0.04%

Table 7.8: Measured relative frequencies of resource shortages, when *NewStatPess* or *NewStatOpt* are applied. Four different server configurations were used.

all cases. The worst case of 0.86% occurred when memory capacity was the limiting resource type and correlations were used while performing resource management (*NewStatOpt*).

7.3.4 Expected Power Savings in Data Centers

Only server savings achievable by statistical static resource management have been presented until now. Actually resulting energy savings depend on the infrastructure required to run servers in a data center, since power consumed by servers forms only one part of the overall power consumption.

10% server savings for instance reduce server power consumption by 10% as well[3]. Power consumed by the infrastructure components scales down by the same factor in best case so that 10% of infrastructure power consumption are saved as well. The 10% server savings would thus lead to 10% savings of the overall power consumption. The ratio between overall power consumption and IT power consumption (known as PUE - power usage efficiency) remains constant.

Such assumption is too optimistic in most cases. In general, larger data centers have a better PUE than smaller ones. Hence, less than 10% energy saving must be expected if proper resource management leads to 10% less required servers even if the data center can be dimensioned smaller.

Only the power consumed by the saved IT components (server + network) can be saved in worst case. The infrastructure power consumption remains constant. Modern data centers can easily achieve a PUE of 2.0 or bellow. Half of the overall power consumption is caused by IT components and half by additional consumers in this case. If from such a data center 10% of IT components can be removed, the overall power consumption will be reduced by 5%.

As a result, the 42% of hardware saving achieved by *NewStatOpt* in the evaluation scenario can lead to overall power savings of between 21% and 42%.

[3]But only, if no dynamic resource management is performed that switches servers off from time to time

7.3.5 Conclusion and Limits of the Analyses

The evaluation results show that performing statistical static resource management can lead to significant server savings compared to a pessimistic approach. The approaches presented in this thesis achieved between 25% and 42% of server savings in a data center simulated with resource demand time series observed from real services. Such server savings can reduce the overall power consumption in a data center by between 12.5% and 42%.

The models used to perform resource management underestimated the actually required resources several times due to changed demand behavior over time. Nevertheless, such underestimates hardly led to actual resource shortages mainly because of unused capacity that remains on the servers.

As expected, the results further showed that resource management can only benefit from resource performance trade-offs, if CPU time is the limiting resource type. Using correlations improves the results in any case even if the workload behavior of the VMs is slightly positively correlated.

Limits of the analyses concern the sample rate of the resource demand time series used for the simulation again. Especially the actual server savings might deviate from the presented ones, when sample rates are used that are more appropriate to guarantee QoS goals such as response time.

7.4 Dynamic Resource Management

The dynamic part of the resource management concept (presented in Chapter 6) will be assessed in this last section of the evaluation chapter. Evaluation criteria are achievable energy savings compared to static resource management and the amount of possible resource shortages caused by forecasting errors of the models.

7.4.1 Methodology

The same simulation environment that has already been shortly presented in Section 7.3.1 was used to assess dynamic resource management as well. VMs were simulated using demand time series observed in a real data center. An implementation of the dynamic resource management concept redistributed the VMs dynamically during the simulation with respect to the expected resource demand. The utilization of all required servers at each simulated time point was measured to detect resource shortages and to determine the energy consumption of the data center.

A static distribution of VMs to servers was determined first according to the resource management concept. The dynamic part of this concept assumes that no resource shortages will

occur at any time, when VMs are distributed to servers according to this safe static distribution. Resources A_i^{max} maximally required by the VMs were determined based on the whole resource demand time series used for the simulation to not influence the analyses by violations of this assumption. Statistic static resource management as presented in Chapter 5 was applied to determined an optimal static distribution. Correlations between the demand behavior of the VM were taken into account as well (*NewStatOpt*) to find this distribution.

The evaluation criteria will be described more concrete in the following comparable to the previous sections. The used hardware configurations, assumed migration times, and different other parameters will be introduced as well.

Evaluation Criteria

The advantage of dynamic resource management compared to a static approach is that energy is saved in times of low workload by switching off unused servers. Hence, these energy savings are a main measure to assess the dynamic approach.

Time series describe the utilization and power state (active or standby) of each server at each simulated time step as an outcome of each simulation run. A model (taken from [82]) that calculates the power consumption in a data center depending on the number of servers, their current utilizations, and their current power states was used to get corresponding power consumption values. Besides the servers itself, this model takes care of additional components required to run them such as cooling infrastructure and network components for instance. Interactions between them are taken into account as well.

The resulting power time series could have been integrated over time to get the overall energy consumption. But to be independent of the simulation time, not energy consumption but the mean power consumption was used as measure.

Each simulation run was performed twice, since not the power consumption itself but power savings compared to static resource management should be used as evaluation criteria. Once dynamic resource management was enabled and once not. The resulting mean power consumptions are denoted by PW_{dyn}^{DC} and PW_{stat}^{DC} respectively in the following. Savings $\Delta\widetilde{PW}^{DC}$ are simply the difference between both normalized by the power value of the static case, which can be formally expressed by:

$$\Delta\widetilde{PW}^{DC} = \frac{PW_{stat}^{DC} - PW_{dyn}^{DC}}{PW_{stat}^{DC}}. \tag{7.10}$$

The dynamic resource management approach presented in this thesis can only cause resource shortage, if the resource capacity required by the VMs at a certain time t deviates from the forecasts. It has already been presented in section 7.2, how often such wrong forecasts actually occur using the demand behavior of real services. It will be determined within this section, how

often they will actually lead to resource problems that could cause SLO violations. Therefore, the frequency of resource shortages was determined during the simulations as follows:

$$P_i^{rs} = P(A_i^{mes}(t) > A_i^{for}(t) \ \wedge \sum_{j:B(j,t)=k} R_j^{mes}(t) > C_k) \quad \text{with} \quad k = B(i,t). \tag{7.11}$$

This approach is similar to the one used to assess the static resource management approach in previous section. The difference is that violations can already occur in case of dynamic resource management, if a VM requires more resource capacity than expected at a time t ($A_i^{mes}(t) > A_i^{for}(t)$). In the static case, they only occur when the maximum A_i^{max} is exceeded.

The maximal duration of all occurred resource shortages was determined in addition to the frequency as well. This value indicates, how fast a dynamic resource management approach can resolve the unexpected problem.

Evaluation Scenario and Data Selection

The same set of 50 VMs that has already been used to assess the static resource management concept (cf. Section 7.3.1) was used again to evaluate the dynamic one. The same four server configurations (concerning memory and CPU time capacity) were used as well.

A data center with a PUE of 2.0 was simulated to determine the mean overall power consumption. The underlying power models can be configured using three different parameter sets. The one that represents today's modern data center components concerning energy efficiency was selected. Server power consumption slightly depends on CPU utilization[4] in this configuration. The cooling infrastructure also slightly adapts to the heat dissipation of the IT components that are running in the data center.

In addition, dynamic resource management requires information about the duration of a server startup and shutdown (Δt_k^{up} and Δt_k^{down}), the servers break even time (Δt_k^{be}), and the migration time of a VM (Δt_i^{mig}). The startup and shutdown time of a server was set to $1min$ for the analyses, which is quite realistic when STR (Save to Ram) is supported. Already a break even time of below $3min$ would be long enough to compensate the energy overhead caused while shutting down and starting up a server. Despite this fact, a break even time of $15min$ was selected to limit the number of server startups and shutdowns. A dedicated $100Mbit/s$ network infrastructure was assumed to support live migration of the VMs between the servers to determine the maximal migration times of the VMs. This network infrastructure should achieve a throughput of around $10MBytes/s$. A maximal migration time was determined for all VMs based on this estimate and on the maximal memory demand A_i^{max} of the respective VM.

[4]DVFS or other power management techniques are supported and enabled.

7.4.2 Comparison to Known Approaches

The dynamic resource management concept presented in this thesis was compared to a known approach in a first analysis. The known approach used a trend based forecasting method. VMs are iteratively redistributed to servers based on the trend measured in the closest past. No safe distributions of VMs to servers are considered. The scheduling algorithm left a certain amount of reserve capacity on each server while planning the redistributions to take care of the noise. Furthermore, hysteresis was implemented to prevent unnecessary VM moves.

The free parameters of this approach (capacity borders and hysteresis levels) were characterized in two version. A pessimistic one, called *known pess.*, removed VMs from a server, when the trend exceeded 40% of the servers capacity. The optimistic version, *known opt.*, reacted, when the trend exceeded 70%. All VMs must be moved to other servers to shutdown a server. But VMs are only moved, if at none of the destination servers a second border (lower than the first one to realize hysteresis) is exceeded due to these moves. 20% was selected as lower border for the pessimistic approach. 40% was the lower border for the optimistic version.

Discussion of Results

The determined power savings as well as the measured frequency and maximal duration of resource shortages are listed for all three analyzed approaches in Table 7.9.

	power savings	resource shortages			
		max	Ø	std	max. duration
new	**23.3 %**	**0.05%**	**0.02%**	**0.02%**	**0:15h**
known pess.	19.9 %	0.06%	0.02%	0.02%	2:05h
known opt.	34.9 %	0.79%	0.57%	0.09%	2:00h

Table 7.9: Power savings, frequency of resource shortages, and maximal duration of resource shortages determined in simulations with three different dynamic resource management approaches.

The approach presented in this thesis could reduce the power consumption in a data center by about 23% in average in the simulated scenario. This would lead to overall energy savings of about 23% as well. The frequency of measured resource shortages is far below the one determined for the static resource management approach. The reason mainly is the ability of the dynamic approach to adapt to changed demand behavior. Once underestimates were detected, they never occurred again in following instances of the predominant period (typically 1 or 7 days in this scenario).

Resource demand suddenly strongly increased the expected one in seldom cases. The resulting resource shortages were detected at runtime and resolved by the approach. The maximal

duration until a resource shortage was resolved was between $10min$ and $15min$.

One can further see in the table that *known pess.* could achieve nearly the same power savings like the new approach with nearly the same amount of occurred resource shortages. But it took a significant longer time to resolve them. More than two hours were needed in worst cases.

The approach *known opt.* could strongly increase the power savings by more than 10%. But the frequency of measured resource shortages strongly increased at the same time as well. Resource shortages occurred in more than 0.5% of time in average, which sounds not very much so far. But the problem is that due to the reactive character of the approach, the resource shortages occurred more or less regularly not only when the demand behavior unexpectedly changed. A resource shortage occurred at some servers every two or three days in average that typically last for $5min$ up to $40min$. Such behavior is absolutely unacceptable for clients of an IT service.

Approaches that work nearly the same way like the one analyzed in this section can be currently bought as products (such as VMware DRS / DPM [123]). Different additional ones known from research might exceed the achievable power savings. But none of them takes care of migration and server reactivation time as exhaustively discussed in Section 6.4.1. Hence, they are expected to even exceed the frequency of resource shortages as well as the duration until resource shortages are resolved.

7.4.3 Influence of Server Configuration and Virtualization Environment

The dynamic resource management approach was analyzed with four different server configurations comparable to the static one. In addition, two different underlying virtualization concept were used as well. Memory demand can be either fixed (e.g. when Xen Server is used) or varying (e. g. in case of VMware ESX) depending on the virtualization environment as described in Section 3.1.4. Such difference was expected to strongly influence the achievable power savings and hence was analyzed additionally.

Discussion of Results

The results are presented in Table 7.10. One can clearly see that increasing the resource capacity provided by a server increases the power savings achievable by dynamic resource management as well. This result is mainly caused by additional flexibly the resource management can draw on, when the ratio between resource demand of VMs and server capacity decreases. Unused capacity can be more easily filled up with smaller VMs than with larger ones.

One can further see that limited memory capacity stronger reduces the savings than limited CPU time capacity. The reason simply is an offset of memory demand that typically is not

	varying memory demand (VMware)			fixed memory demand (Xen Server)		
	power savings	resource shortages		power savings	resource shortages	
		Ø	max. duration		Ø	max. duration
2 cores, 8GB	**15.8 %**	0.03%	**0:10h**	**6.9 %**	0.01%	**0:10h**
2 cores, 32GB	**14.9 %**	0.07%	**0:15h**	**14.9 %**	0.02%	**0:20h**
4 cores, 8GB	**9.7 %**	0.06%	**0:10h**	**0.0 %**	0.00%	**0:00h**
4 cores, 32GB	**23.3 %**	0.02%	**0:15h**	**21.8 %**	0.02%	**0:15h**

Table 7.10: Power savings and resource shortage for four different server configurations and two different virtualization concepts

deceeded by the VMs. CPU time demand very often decreases down to 0%. Memory demand typically does not.

One must ensure in any case that not memory capacity is the limiting resource type, if virtualization environments are used that only support fixed memory allocation. Otherwise, the benefit one can take from dynamic resource management strongly reduces or completely vanishes. The reason is simple again. Memory demand is not varying any longer, when such virtualization environmentss are used. No redistributions that lead to unused servers are possible at all without any variations.

Hardly some significant dependencies can be extracted from the results with respect to the resource shortages. One can only say that using fixed allocated memory seems to slightly reduce the amount of resource shortages in general. The reason might be that only the demand of one resource type is forecasted in this case so that the amount of overall forecasting errors is reduced this way.

7.4.4 Limiting the Impact of Forecasting Errors

A method has been presented in Section 6.5.5 to limit the impact of forecasting errors. Therefore, two different parameters can be set by an administrator. First, the strength of possible resource shortages can be limited per VM by allocating a fixed amount of resource capacity A_i^{min}. This minimal amount of resources is guaranteed to the VM in any case. Second, a maximal duration $\Delta t_i^{resolve}$ can be defined until which a possible resource shortage must be resolved.

The selection of both parameters can influence possible resource savings. Hence, analyses were performed for different values of them.

Discussion of Results

The results are presented in Figure 7.11. The left bar plot shows power savings achieved for different values of $\Delta t_i^{resolve}$. The right one presents the savings for different values of A_i^{min}.

The same $\Delta t_i^{resolve}$ was selected for all of the 50 VMs. The value of A_i^{min} was selected as relative share of the respective maximal resource demand A_i^{max} of the VMs.

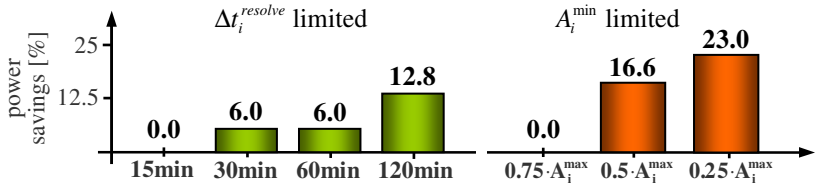


Figure 7.11: Achieved power savings, when the impact of forecasting errors is limited. Different values of the limiting parameters A_i^{min} and $\Delta t_i^{resolve}$ were selected.

Reserving fixed capacity A_i^{min} for a VM to prevent performance slowdowns below a certain threshold does not influence the achievable savings that much. Reserving 25% of the maximally required resources for a VM leads to nearly the same savings like without any limits.

Limiting Δt^{ftr} by setting $\Delta t_i^{resolve}$ strongly influences the savings in contrast. Setting $\Delta t_i^{resolve}$ to $15min$ can not lead to any savings at all as expected, since the break even time of a server is already $15min$. Limiting Δt^{ftr} to values up to $2h$ significantly reduced the achieved savings in any case. Δt^{ftr} grew up to $6h$ in the simulations without any limits. Hence, reduced savings when limits are set are not unexpected.

The strategy that ensures to meet $\Delta t_i^{resolve}$ is very pessimistic. Resolving any of the resource shortage did actually not exceed $20min$ in all of the analyses performed. Basically it could in worst case. But the question is, if in such a case restoring the safe distribution will resolve the problems at all. One must finally conclude that limiting the duration of possible performance slowdowns this pessimistic way does not make any sense for realistic values of $\Delta t_i^{resolve}$.

May be individually treating the respective $\Delta t_i^{resolve}$ of the VMs can help to improve the efficiency. The algorithm can try to find a resolve schedule that first solves problems of VMs with a more restricted value of $\Delta t_i^{resolve}$. Problems of VMs with weaker restrictions or without any restrictions at all can be resolved later.

Another option is to allow parallel migrations. Δt^{ftr} itself will not grow that much in this case as already discussed in Section 6.4.9.

7.4.5 Scalability

The dynamic resource management approach will be evaluated concerning scalability within this last section. It will be mainly answered, how the approach behaves with an increasing number of managed VMs. Mainly two aspects are important. First, it will be analyzed how the achievable power savings develop with an increasing number of VMs. And second, the computing power the algorithm needs to manage the VMs will be regarded.

Energy Savings - Large Cluster vs. Smaller Independent Clusters

It has been already discussed in Section 6.4.9 that the planning period Δt^{ftr} increases with an increasing data center size (in terms of the number of managed VMs). It was shown in the model evaluation section (Section 7.2.4) that the usable saving potential provided by the models gets more and more lost with an increasing planning period. On the other hand, the flexibility to manage VMs (statically as well as dynamically) increases with an increasing number of VMs. Hence, it is expected that a certain number of VMs can be managed optimally by the resource management. The VM set could be better split up into individually managed pools, when this number is exceeded.

A set of 200 VMs was created out of the initial set to find this threshold. This set was partitioned into equally sized clusters that were management individually by the dynamic resource management approach. Four different cluster sizes with between 25 and 200 VMs each were analyzed. The number of required servers and the average power consumption was determined by simulations in each case. Once only static and once additionally dynamic resource management was performed. The results are presented in Figure 7.12.

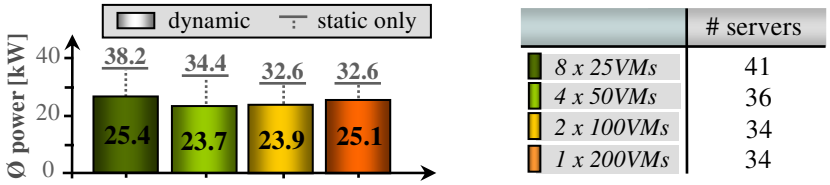


Figure 7.12: Overall power consumptions and the number of required servers in a data center with 200 VMs. Equal sized clusters of VMs were individually managed by the resource management approach. Four different cluster sizes were analyzed.

First, one can see that increasing the cluster size also increases the efficiency of static resource management as expected. With an increasing number of managed VM the number of required servers gets closer to the theoretical minimum. Hence, the overall power consumption of the data center decreases as well.

The same can be observed for dynamically managed VMs until a cluster size of 50 VMs is exceeded. The power savings that are achievable by dynamic resource management decrease to strong with more than 50 VMs managed together. The overall power consumption increased despite the fact that with a cluster size of 100 VMs two complete servers could be saved.

As a result, one can conclude that around 50 VMs can be optimally managed as a whole. Sets with significantly more than 50 VMs should be better split up and managed individually.

But it has to be noticed that this result is only valid for the analyzed scenario. The optimal cluster size will be much higher, if for instance a $1GBit$ migration network is used that

significantly increases the migration speed. The same is true for lower break even or startup and shutdown times of servers. Mainly the development of Δt^{ftr} and the saving potential that workload variations provide to the dynamic resource management decide about the optimal cluster size.

Computational Effort Required to Manage the VMs

The worst case runtime complexity has already been discussed in Section 6.4.9. Some information about the actually required computational effort will be provided within this section.

One can say that for a normal number of VMs no runtime problems are expected to occur with respect to the duration of the simulation runs. A simulation run of one month with 200 VMs performed on a normal desktop PC lasted only about $15min$. It is not expected that far more than 200 VMs should be ever managed as a whole regarding the results concerning the optimal cluster size.

A simulation step of $5min$ was used because of the sample rate of the used simulation data. This means that the algorithm reevaluates the current distribution every $5min$. Reducing this interval size would increase the computational effort the algorithm requires because of the increased numbers of samples to be evaluated.

But it is not expected that strongly reducing this interval size would lead to significant more benefit. If for instance the algorithm would reevaluate the current distribution every minute, a server is deactivated $4min$ earlier and can remain switched off for further $4min$ in best case. The amount of possible resource shortages is not influenced by this reevaluation interval as long as the forecasts do not deviate from the actual resource demand.

7.4.6 Conclusion and Limits of Analyses

One can conclude that dynamic resource management can strongly reduce the power consumption in a data center compared to only performing static resource management. The approach presented in this thesis could reduce the overall power consumption of a whole data center by about 20% in a simulated scenario. Unexpectedly changed demand behavior led to resource shortages in only a few cases that could be automatically resolved within maximally $20min$. A known trend based approach in contrast caused resource shortages that lasted significantly longer. Furthermore, these resource shortages occurred very regularly every two or three days.

One should ensure that not memory is the limiting resource type, when dynamic resource is applied. Otherwise, achievable energy savings are strongly limited especially if the virtualization environment supports only fixed memory allocation. Limiting the impact of resource shortages will lead to reduced energy savings as well. Especially a limit of the maximal duration of a possible resource shortage can vanish any benefit at all. Finally, analyses showed

that there exist an optimal number of VMs that can be managed by the resource management approach as a whole. If this number is exceeded, more energy can be saved when VMs are clustered into individually managed pools.

The analyses performed to evaluate the dynamic resource management are limited at some points again. The strongly averaged sample interval of the used simulation data limits the validity of the results comparable to previous analyses. It is expected that the actually required resource demand is varying stronger. The achieved energy savings might be lower than determined as a consequence. The frequency of actual resource shortages might increase at the same time.

A second limitation concerns the simulation data as well. Only resource demand time series of 23 different services could be used for the evaluation. The same time series were used more than once in one simulation run to get data centers with a realistic size. As a result, some of the VMs had perfectly positively correlated resource demand behavior, which typically does not occur in real data centers. It is expected that without this artifact actually more power can be saved than determined. Furthermore, the frequency of resource shortages should be lower without this artifact because the probability that uncorrelated resource demand exceeds the capacity border is far lower than with correlated demand.

7.5 Summary

The advantages of statistic resource management compared to a pessimistic approach were evaluated at the beginning of this chapter. CPU time savings of up to 20% could be achieved by static resource management for an individual service. Dynamic resource management could reduce the provided CPU time capacity by about 8% in average. It was further shown that the new fine grained SLO specification is much more flexible compared to known percentile based ones. At least the same or even more resource savings could be achieved, while the same performance goals were guaranteed.

Statistic static and especially dynamic resource management relies on exact forecasts of the demand behavior of the VMs in the future. Models that have been developed for this purpose have been assessed in the second part of this chapter. The models typically underestimated the actually required resources in less then 3% of time in the analyzed scenarios. Pessimistic known modeling approaches could achieve nearly the same results. But they hardly provided some potential for energy savings to the scheduling algorithm at the same time. A more optimistic known approach provided far more potential but underestimated significantly more often the actual demand.

Statistic static resource management as presented in this thesis was analyzed in the third section of this chapter. The analyzed approach bases on the new models and supports the

new fine grained SLO specification to trade off resources against performance. The number of required servers could be reduced by between 25% and 40% in a simulation of realistic services. Respective energy savings in data center of between 12.5% and 42% could be achieved by these server savings. Forecasting errors of the models hardly led to actual resource shortages of VMs (in below 0.2% of the simulated time) mainly because of resource capacity that remained unused on the servers. Considering correlations while performing static resource management can strongly reduce the amount of resource shortages. A known approach that expected uncorrelated demand behavior of the VMs achieved the same server savings like the new approach but led to far more resource shortages.

The dynamic resource management approach presented in this thesis was assessed in the last part of the evaluation. Energy saving of about 20% could be achieved compared to using only statistic static resource management in simulations of a realistic data center scenario. Known trend based approaches could achieve the same savings. But they led to far more resource shortages that occurred every two or three days and lasted for up to two hours, which is unacceptable for clients of the services. The new approach caused only resource shortages, when the demand behavior of the VMs unexpectedly changed. It automatically resolved the problem in less than 20 minutes in these cases.

The validity of the results is limited at mainly two points. First, only a limited set of real resource time series could be used for the analyses. Hence, the results are limited to the underlying services of these time series. A second limit concerns the sample rate of the time series (5 min. per sample). More fine grained resource demand time series are needed to train the models, when QoS attributes such as response time or throughput have to be guaranteed. Such time series can currently not be captured due to missing technical prerequisites in all available virtualization environments.

8 Summary and Conclusion

The continuously increasing energy consumption in data centers slowly becomes a serious problems. Not only from an ecological but also from an economical and in the meantime a technical view such increase in power demand is no longer feasible. Resource management can help to reduce the power consumption by more efficiently using active server hardware.

A novel holistic concept for static and dynamic resource management that follows this idea has been presented within this thesis. This concept consists of three phases and supports the whole process needed to deploy services into an automatically managed data center.

The concept extends different known approaches by several ideas that improve the efficiency of resource management and hence can save required hardware resources and energy. Different weaknesses of the known approaches have been addressed as well. The following characteristics distinguish the new concept from the state of the art:

- The introduced SLO specification allows trading off resource capacity against performance in a flexible way.
- The modeling approach takes care of the special characteristics of the services' demand behavior (e.g. missing stationarity due to varying noise performance).
- Correlations between the resource demand of different services are regarded, which is required to prevent SLO violations.
- Interdependencies between required and provided resource capacity are considered.
- The scheduling approach for dynamic resource management can resolve all upcoming resource shortages right in time assuming that the forecasted resource demand meets the actual one.
- The dynamic part of the resource management concept can adapt changed demand behavior at runtime.

8.1 Conclusion

Reserving maximally required resources for a service all the time is a widely used approach for static resource management at the moment. Applying statistic static resource management

as presented in this thesis can significantly reduce the number of required hardware servers compared to the pessimistic approach. Simulations with resource demand time series of services observed in a real data center could achieve server saving of between 25% and 42%. These savings can result in energy savings of between 12.5% and 42% depending on the infrastructure contained in the data center. Applying resource performance trade-offs and using correlations while performing resource management mainly led to these savings. Resource shortages that could lead to SLO violations occurred in less than 0.2% of the simulated time (ca. 1 year).

Additionally applying the dynamic part of the resource management concept can further reduce the energy consumption. Savings of about 20% could be achieved in simulations of the same data center scenario like used to assess the static part. The underlying server hardware and data center infrastructure represents the state of the art. Hence, such savings could be actually achieved in today's data centers. The concept led to resource shortages in less than 0.06% of the simulated time mainly due to the ability of the models to adapt to changed demand behavior.

Despite the comparable low frequency of resource shortages, one must conclude that they basically can occur. Observing resource demand in the past to extrapolate demand behavior expected in the future can result in forecasting errors in any case. Such errors can lead to resource shortages and finally to SLO violations. Furthermore, the clients can provoke SLO violations by changing the usage behavior to for instance get a financial penalty from the Service Provider. Only the pessimistic approach that does not take any user behavior into account for resource management decisions can completely exclude SLO violations caused by resource shortages.

Finally, the validity of the evaluation results is limited at some points due to the limited set of resource demand time series that was available. Especially the sample rate of 5 min. per sample is far too low to guarantee most realistic performance goals (such as a response time of a few seconds for instance). Using the models with a more realistic sample rate might influence the achievable energy savings as well as the amount of occurring resource shortages.

8.2 Outlook

Ongoing research must further work out the concept at different points to be applicable in a real data center. Some of them will be shortly summarized in the following to motivate some future research.

- **Include network resource management**
  Network resource management is only considered in a very static way not taking actual network traffic into account. Models that describe the network system in the data center and the network traffic of the services are needed to address this issue. Using these

models, the resource management concept can decide where to place VMs taking network as shared resource into account similar to memory and CPU time.

- **Directly measure user interaction with services for resource management**
  It has been discussed in Section 5.2.5 that determining required resource capacity from observed resource demand can be difficult. Especially if response time is the targeted QoS attribute, a very high sample rate for CPU time might be necessary. It can be easier to directly measure and model the user interactions with the service. In principal, the same models like those presented in this thesis should work for the user behavior as well, since resource demand is somehow only a mapping of the user behavior. Required resource capacity can then be derived using special performance models of the service.

- **Extend static resource management to adapt changed demand behavior**
  The concept until now assumes one static distribution of VMs to servers that never changes at runtime. Changed demand behavior can not be adapted this way. Furthermore, increasing long term trends must be taken into account, which limits the saving potential. Ongoing research can try to determine new static distributions at runtime that consider changed demand behavior. A challenge is to change between them at runtime while dynamic resource management is performed in parallel.

- **Allow parallel scheduling of operations**
  The dynamic scheduling algorithm plans redistribution operations strongly sequentially. In principal, the underlying virtualization environments supports parallel migrations and server startups. Using this ability could further increase the energy savings. For this, the scheduling algorithm must take care of the increased migration delay.

- **Integrate infrastructure components into resource management decisions**
  Dynamic resource management leaves several degrees of freedom unused. Typically, different servers can be reactivated to resolve a resource shortage. The same can be true when the resource demand decreases. A subset of different servers can be emptied to be shut down. These options can be used to take the state of the cooling infrastructure into account. The heat dissipation in a data center can be equalized out this way, which can help to save additional energy due to a more efficient cooling.

Finally, one can conclude that this thesis solved several open questions in the field of resource management that can help to reduce the energy consumption in data centers. The concept itself opens many new questions that provide much potential for ongoing research.

Glossary

home server A home server of a VM is the server on which the VM is placed according to a safe distribution of VMs on servers.

Internet Service Provider A company that provides its customers access to the Internet.

ISP Abbreviation for *Internet Service Provider*, see: Internet Service Provider.

live migration The live migration technique enables the move of a virtual server between two hardware servers without interrupting the service deployed on the virtual server.

LPM Abbreviation for *Load and Power Management*, see Section 3.3.1.

predominant period The dominating period in a time series with different overlaid periods.

QoS Abbreviation for *Quality of Service*, see: Quality of Service.

Quality of Service The quality of a service is typically defined by a set of service dependent metrics such as response time or throughput that assess how good a client can use the service at a time.

safe distribution A safe distribution of VMs to servers provides enough resource capacity to all VMs all the time. No redistributions are necessary to prevent SLO violations.

server virtualization A techniques that allows different virtual servers to share one real hardware server.

Service Level Agreement A contract between a client and a Service Provider that defines a wanted Quality of Service.

Service Level Objective Service Level Objective are parts of a Service Level Agreement that define an expected Quality of Service level.

Service Provider A provider of an IT service typically deployed in a data center.

SLA Abbreviation for *Service Level Agreement*, see: Service Level Agreement.

SLO Abbreviation for *Service Level Objective*, see: Service Level Objective.

SP Abbreviation for *Service Provider*, see: Service Provider.

unsafe distribution An unsafe distribution of VMs to servers does not provide enough resource capacity to all VMs all the time. Redistributions might be necessary to prevent SLO violations.

Virtual Machine A virtual server on which a service and its operating system can be installed. Different of these virtual servers can share the same real hardware server.

Virtual Machine Monitor The part of a virtualization environment that schedules the accesses of different Virtual Machines to the real hardware server.

virtualization environment A hardware and software system that realizes server virtualization. This technique is focused within this thesis to allow different services to share the same hardware server.

VM Abbreviation for *Virtual Machine*, see: Virtual Machine.

VMM Abbreviation for *Virtual Machine Monitor*, see: Virtual Machine Monitor.

Bibliography

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13:80–96, 2002.

[2] G. Aggarwal, R. Motwani, and A. Zhu. The load rebalancing problem. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 258–265, New York, NY, USA, 2003. ACM.

[3] R. Agrawal, M. J. Carey, and L. W. Mcvoy. The performance of alternative strategies for dealing with deadlocks in database management systems. *IEEE Transactions on Software Engineering*, SE-13(12):1348–1363, 1987.

[4] A. V. Aho and J. D. Ullman. *Foundations of Computer Science: C Edition.* W. H. Freeman, 1994.

[5] A. V. Aho and J. D. Ullman. *Foundations of Computer Science: C Edition*, chapter 9 The Graph Data Model, pages 484–495. W. H. Freeman, 1994.

[6] A. V. Aho and J. D. Ullman. *Foundations of Computer Science: C Edition*, chapter 9 The Graph Data Model, pages 495–497. W. H. Freeman, 1994.

[7] J. Allspaw. *The Art of Capacity Planning*, chapter 2 Setting Goals for Capacity. O'Reilly Media, Inc., 2008.

[8] J. Allspaw. *The Art of Capacity Planning*, chapter 1 Goals, Issues, and Processes in Capacity Planning. O'Reilly Media, Inc., 2008.

[9] J. Allspaw. *The Art of Capacity Planning*, chapter 3 Measurement: Units of Capacity. O'Reilly Media, Inc., 2008.

[10] J. Allspaw. *The Art of Capacity Planning*, chapter 4 Predicting Trends. O'Reilly Media, Inc., 2008.

[11] D. Ardagna, M. Tanelli, M. Lovera, and L. Zhang. Black-box performance models for virtualized web service applications. In *WOSP/SIPEW '10: Proceedings of the first*

joint WOSP/SIPEW international conference on Performance engineering, pages 153–164, New York, NY, USA, 2010. ACM.

[12] E. Arzuaga and D.R. Kaeli. Quantifying load imbalance on virtualized enterprise servers. In *WOSP/SIPEW '10: Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 235–242, New York, NY, USA, 2010. ACM.

[13] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[14] F. Belik. An efficient deadlock avoidance technique. *IEEE Transactions on Computers*, 39(7):882–888, 1990.

[15] A. Beloglazov and R. Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '10, pages 4:1–4:6, New York, NY, USA, 2010. ACM.

[16] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli. Policy optimization for dynamic power management. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(6):813–833, 1999.

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

[18] D. Borgetto, G. Da Costa, J.-M. Pierson, and A. Sayah. Energy-aware resource allocation. In *Proc. 10th IEEE/ACM Int Grid Computing Conf*, pages 183–188, 2009.

[19] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*, chapter 1 Introduction to Competitive Analysis: The List Accessing Problem. Cambridge University Press, 1998.

[20] E.V. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *Proceedings of the 17th annual international conference on Supercomputing*, ICS '03, pages 86–97, New York, NY, USA, 2003. ACM.

[21] J. Carter and K. Rajamani. Designing energy-efficient servers and data centers. *Computer*, 43(7):76–78, 2010.

[22] G. Casella and R. L. Berger. *Statistical Interference*, chapter 2, pages 47–48. Duxbury Press, 2nd edition, 2001.

[23] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 300–301, New York, NY, USA, 2003. ACM.

[24] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 103–116, New York, NY, USA, 2001. ACM.

[25] J. S. Chase and R. P. Doyle. Balance of power: Energy management for server clusters. In *In Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS*, 2001.

[26] C. Chekuri and S. Khanna. On multi-dimensional packing problems. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 185–194, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[27] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 337–350, Berkeley, CA, USA, 2008. USENIX Association.

[28] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 303–314, New York, NY, USA, 2005. ACM.

[29] Citrix Systems, Inc. XenServer - overview. `http://www.citrix.com/English/ps2/products/product.asp?contentID=683148`.

[30] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. 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.

[31] I. Correia, L. Gouveia, and F. Saldanha-da Gama. Solving the variable size bin packing problem with discretized formulations. *Comput. Oper. Res.*, 35(6):2103–2113, 2008.

[32] J. D. Cryer and K. Chan. *Time Series Analysis With Applications in R*, chapter 2 Fundamental Concepts. Statistics Texts in Statistics. Springer Science+Business Media, LLC, 2nd edition, 2008.

[33] J. D. Cryer and K. Chan. *Time Series Analysis With Applications in R*. Statistics Texts in Statistics. Springer Science+Business Media, LLC, 2nd edition, 2008.

[34] J. D. Cryer and K. Chan. *Time Series Analysis With Applications in R*, chapter 4 Models For Stationary Time Series. Statistics Texts in Statistics. Springer Science+Business Media, LLC, 2nd edition, 2008.

[35] J. D. Cryer and K. Chan. *Time Series Analysis With Applications in R*, chapter 10 Seasonal Models. Statistics Texts in Statistics. Springer Science+Business Media, LLC, 2nd edition, 2008.

[36] J. D. Cryer and K. Chan. *Time Series Analysis With Applications in R*, chapter 3 Trends. Statistics Texts in Statistics. Springer Science+Business Media, LLC, 2nd edition, 2008.

[37] K. Dunlap and N. Rasmussen. The advantages of row and rack-oriented cooling architectures for data centers. Technical report, American Power Conversion Corp. (APC), 2006.

[38] H. Eckey, R. Kosfeld, and M. Türck. *Wahrscheinlichkeitsrechnung und Induktive Statistik: Grundlagen - Methoden - Beispiele (Probability theorie and inductive statistics: basics - methods - examples)*, chapter Stichproben (Samples). Gabler, 2005.

[39] European Commission - DG JRC. Code of conduct on energy efficiency and quality of ac uninterruptible power systems. `http://www.bfe.admin.ch/php/modules/publikationen/stream.php?extlang=de&name=de_290024973.pdf`, Dec 2006.

[40] K. Fichter, C. Clausen, and M. Eimertenbrink. Energieeffiziente Rechenzentren - Best-Practice Beispiele aus Europa, USA und Asien (energy efficient data centers - best practice examples in europe, usa, and asia). `http://www.borderstep.de/details.php?menue=33&subid=101&le=de`, 2009.

[41] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCS '03, pages 550–, Washington, DC, USA, 2003. IEEE Computer Society.

[42] A. S. Fukunaga. Repairing bin packing constraints (extended abstract). In *First Workshop on Bin Packing and Placement Constraints BPPC'08*, 2008.

[43] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper. Adaptive quality of service management for enterprise services. *ACM Trans. Web*, 2(1):1–46, 2008.

[44] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proc. IEEE 10th Int. Symp. Workload Characterization IISWC 2007*, pages 171–180, 2007.

[45] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *Proc. 40th Annual Symp. Foundations of Computer Science*, pages 579–586, 1999.

[46] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Drpm: dynamic speed control for power management in server class disks. *SIGARCH Comput. Archit. News*, 31:169–181, May 2003.

[47] S. Gurumurthi, J. Zhang, A. Sivasubramaniam, M. Kandemir, H. Franke, N. Vijaykrishnan, and M. J. Irwin. Interplay of energy and performance for disk arrays running transaction processing workloads. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 123–132, Washington, DC, USA, 2003. IEEE Computer Society.

[48] P. Hannaford. Ten cooling solutions to support high-density server deployment. Technical report, American Power Conversion Corp. (APC), 2010.

[49] M. Hazewinkel. *Encyclopaedia of Mathematics*, chapter Independence. Springer, 2007.

[50] M. Hazewinkel. *Encyclopaedia of Mathematics*, chapter Convolution of functions. Springer, 2007.

[51] M. Hazewinkel. *Encyclopaedia of Mathematics*, chapter Linear regression. Springer, 2007.

[52] S. Henzler. *Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies*. Springer, 2007.

[53] F. Hermenier, X. Lorca, J. Menaud, G. Muller, and J. 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.

[54] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. Advanced configuration and power interface specification. `http://www.acpi.info/DOWNLOADS/ACPIspec30.pdf`, 2004. section 2.4 sleeping state definitions.

[55] Hewlett-Packard Development. HP Insight Capacity Advisor 6.2 user guide. `http://h20000.www2.hp.com/bc/docs/support/SupportManual/c02536173/c02536173.pdf`.

[56] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*, chapter 2 Approximation Algorithms for Bin Packing: A Survey. PWS Publishing Company, 1997.

[57] M. Hoyer, A. Baumgart, and W. Nebel. Adaptives Powermanagement für Desktop- und Notebooksysteme (adaptive power management for desktop and notebook systems). *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 32-2:96–104, 2009.

[58] M. Hoyer, K. Schröder, and W. Nebel. Statistical static capacity management in virtualized data centers supporting fine grained qos specification. In *e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 51–60, New York, NY, USA, 2010. ACM.

[59] W. Huang, J. Liu, B. Abali, and D. K. Panda. A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 125–134, New York, NY, USA, 2006. ACM.

[60] IBM Tivoli Performance Analyzer. `http://www.ibm.com/software/tivoli/products/performance-analyzer/`.

[61] IBM. z/VM - the newest VM hypervisor based on 64-bit z/Architecture. `http://www.vm.ibm.com/`.

[62] IBM. Web Service Level Agreement (WSLA) language specification, version 1.0, revision: wsla-2003/01/28. `http://www.research.ibm.com/wsla/`, January 2003.

[63] Intel Corporation. Power management in intel architecture servers. `http://download.intel.com/support/motherboards/server/sb/power_management_of_intel_architecture_servers.pdf`, April 2009.

[64] J.Niemann. Improved chilled water piping distribution methodology for data centers. Technical report, American Power Conversion Corp. (APC), 2007.

[65] J. Kang and S. Park. Algorithms for the variable sized bin packing problem. *European Journal of Operational Research*, volume 147, issue 2:365–372, June 2003.

[66] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. In *Proc. 10th IEEE/IFIP Network Operations and Management Symp. NOMS 2006*, pages 373–381, 2006.

[67] T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic application placement under service and memory constraints. In *WS on Experimental and Efficient Algorithms*, 2005.

[68] M. Klobasa, F. Sensfuß, and M. Ragwitz. CO_2-Minderung im Stromsektor durch den Einsatz erneuerbarer Energien im Jahr 2006 und 2007 (CO_2 reduction due to using renewable energy sources in the years 2006 and 2007). `http://www.bmu.de/files/pdfs/allgemein/application/pdf/gutachten_isi_co2_bf.pdf`, Feb. 2009.

[69] L. T. Kou and G. Markowsky. Multidimensional bin packing algorithms. *IBM Journal of Research and Development*, 21(5):443–448, 1977.

[70] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *Proc. Fourth IEEE Workshop Mobile Computing Systems and Applications*, pages 40–46, 2002.

[71] D. Kusic. *Combined power and performance management of virtualized computing environments using limited lookahead control.* PhD thesis, Drexel University, 2009.

[72] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and Guofei J. Power and performance management of virtualized computing environments via lookahead control. In *Proc. Int. Conf. Autonomic Computing ICAC '08*, pages 3–12, 2008.

[73] B. Lubin, J. O. Kephart, R. Das, and D. C. Parkes. Expressive power-based resource allocation for data centers. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 1451–1456, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[74] Microsoft Corporation. Hyper−V: Using Hyper−V and Failover Clustering. `http://technet.microsoft.com/en-us/library/cc732181(WS.10).aspx`.

[75] Microsoft Corporation. Virtualization with Hyper-V. `http://www.microsoft.com/windowsserver2008/en/us/hyperv-main.aspx`.

[76] Microsoft Corporation. Windows Virtual PC. `http://www.microsoft.com/windows/virtual-pc/default.aspx`.

[77] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.

[78] Message passing interface (mpi). `https://computing.llnl.gov/tutorials/mpi/`.

[79] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 229–240, Washington, DC, USA, 2005. IEEE Computer Society.

[80] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for hpc with xen virtualization. In *Proceedings of the 21st annual international conference on Supercomputing*, ICS '07, pages 23–32, New York, NY, USA, 2007. ACM.

[81] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *Proc. Fourth Int. Conf. Autonomic Computing ICAC '07*, page 5, 2007.

[82] W. Nebel, M. Hoyer, K. Schröder, and D. Schlitt. Untersuchung des Potentials von rechenzentrenübergreifendem Lastmanagement zur Reduzierung des Energieverbrauchs in der IKT (analysis of potential energy consumption reduction in ICT by using data center comprehensive load management). http://www.lastmanagement.offis.de/, 2009. chapter 5.

[83] M. Nelson, B. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Proceedings of USENIX 05: General Track*, 2005.

[84] S. Niles. Virtualization: Optimized power and cooling to maximize benefits. Technical report, American Power Conversion Corp. (APC), 2009.

[85] Novell GmbH. PlateSpin Recon 3.7. `http://www.novell.com/products/recon/`.

[86] M. Nuttall. A brief survey of systems providing process or object migration facilities. *SIGOPS Oper. Syst. Rev.*, 28(4):64–80, 1994.

[87] Open Grid Forum. Web Services Agreement Specification (WS-Agreement). `www.ogf.org/documents/GFD.107.pdf`, March 2007.

[88] E. Pakbaznia, M. Ghasemazar, and M. Pedram. Temperature-aware dynamic resource provisioning in a power-optimized datacenter. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 124–129, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[89] P. Petliczew. Reduzierung des Energieverbrauchs von modernen Serverfarmen (reducing the energy consumption of modern server farms). Master's thesis, C.v.O. University of Oldenburg, 2008.

[90] V. Petrucci, O. Loques, and D. Mossé. A dynamic optimization model for power and performance management of virtualized clusters. In *e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 225–233, New York, NY, USA, 2010. ACM.

[91] J. Pierson. Allocating resources greenly: reducing energy consumption or reducing ecological impact? In *e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 127–130, New York, NY, USA, 2010. ACM.

[92] N. Rasmussen. A scalable,reconfigurable, and efficient data center power distribution architecture. Technical report, American Power Conversion Corp. (APC), 2010.

[93] N. Rasmussen and J. Spitaels. A quantitative comparison of high efficiency ac vs. dc power distribution for data centers. Technical report, American Power Conversion Corp. (APC), 2007.

[94] J. A. Rice. *Mathematical Statistics and Data Analysis*, chapter 2. Duxbury Press, 1994.

[95] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak. A capacity management service for resource pools. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 229–237, New York, NY, USA, 2005. ACM.

[96] C. Rusu, A. Ferreira, C. Scordino, and A. Watson. Energy-efficient real-time heterogeneous server clusters. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 418–428, Washington, DC, USA, 2006. IEEE Computer Society.

[97] L. Sachs and J. Hedderich. *Angewandte Statistik - Methodensammlung mit R (Applied Statics - methods in R)*, chapter 5 Zufallsvariablen, Verteilungen (random variables, distributions). Springer, 2009.

[98] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. *SIGOPS Oper. Syst. Rev.*, 36(SI):377–390, 2002.

[99] K. O. Schallaböck and C. Schneider. Im Steigflug in die Klimakatastrophe? (a study about German aviation in 2007.). `http://www.bund.net/fileadmin/bundnet/publikationen/verkehr/20080409_verkehr_luftverkehr_2007_wuppertal_studie.pdf`, April 2008.

[100] G. Schulz. *Regelungstechnik 1: Lineare und Nichtlineare Regelung, Rechnergestützter Reglerentwurf (Feedback control systems 1: Linear and non Linear Feedback Control Loops)*. OLDENBOURG, 2007.

[101] S. Seltzsam, D. Gmach, S. Krompass, and A. Kemper. Autoglobe: An automatic administration concept for service-oriented database applications. In *Proc. 22nd Int. Conf. Data Engineering ICDE '06*, page 90, 2006.

[102] J. Shahabuddin, A. Chrungoo, V. Gupta, S. Juneja, S. Kapoor, and A. Kumar. Stream-packing: Resource allocation in web server farms with a qos guarantee. In *High Performance Computing - HiPC 2001*, volume Volume 2228/2001, pages 182–191. Springer Berlin / Heidelberg, 2001.

[103] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware qos management in web servers. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, RTSS '03, pages 63–, Washington, DC, USA, 2003. IEEE Computer Society.

[104] D. Shen and J. L. Hellerstein. Predictive models for proactive network management: application to a production web server. In *Proc. IEEE/IFIP Network Operations and Management Symp. NOMS 2000*, pages 833–846, 2000.

[105] ACM SIGCOMM. The internet traffic archive. `http://ita.ee.lbl.gov/`. HTTP requests to the NASA Kennedy Space Center WWW server (Jul/1995).

[106] J. Skene, F. Raimondi, and W. Emmerich. Service-level agreements for electronic services. *IEEE Transactions on Software Engineering*, 99:288–304, 2009.

[107] H. Sprang, T. Benk, J. Zdrzalek, and R. Dehner. *Xen, Virtualisierung unter Linux (Virtualization using Linux)*, chapter 2 Einstieg in die Welt der Virtualisierung (Introduction into the World of Virtualization). Open Source Press, 2007.

[108] H. Sprang, T. Benk, J. Zdrzalek, and R. Dehner. *Xen, Virtualisierung unter Linux (Virtualization using Linux)*, chapter 17 CPU-Time Scheduling. Open Source Press, 2007.

[109] H. Sprang, T. Benk, J. Zdrzalek, and R. Dehner. *Xen, Virtualisierung unter Linux (Virtualization using Linux)*, chapter 11 Speichermanagement (Memory Management). Open Source Press, 2007.

[110] H. Sprang, T. Benk, J. Zdrzalek, and R. Dehner. *Xen, Virtualisierung unter Linux (Virtualization using Linux)*, chapter 9 Netzwerkkonzepte und -Konfiguration (Network Configuration Concepts). Open Source Press, 2007.

[111] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova. Resource allocation using virtual clusters. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 260–267, Washington, DC, USA, 2009. IEEE Computer Society.

[112] L. Stobbe, M. Nissen, N.F.and Proske, A. Middendorf, B. Schlomann, M. Friedewald, P. Georgieff, and T. Leimbach. Abschätzung des energiebedarfs der weiteren entwicklung der informationsgesellschaft (assessment of the energy demand by the further development of the information society). `http://publica.fraunhofer.de/documents/N-110231.html`, 2009.

[113] A. S. Tanenbaum. *Modern Operating Systems*, chapter 3. Deadlocks. Prentice Hall International, 2001.

[114] M. Thottan and C. Ji. Adaptive thresholding for proactive network problem detection. In *Proc. IEEE Third Int Systems Management Workshop*, pages 108–116, 1998.

[115] W. Torell. Data center physical infrastructure: Optimizing business value. Technical report, American Power Conversion (APC), 2010.

[116] V. Tosic. *Service offerings for XML web services and their management applications*. PhD thesis, Carleton University, Ottawa, Ontario, Canada, 2004.

[117] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. *SIGOPS Oper. Syst. Rev.*, 36(SI):239–254, 2002.

[118] A. Verma, P. Ahuja, and A. 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.

[119] A. Verma, P. Ahuja, and A. Neogi. Power-aware dynamic placement of hpc applications. In *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, pages 175–184, New York, NY, USA, 2008. ACM.

[120] VMware, Inc. ESX configuration guide. `http://www.vmware.com/pdf/vsphere4/r41/vsp_41_esx_server_config.pdf`. chapter 3 Basic networking with vNetwork standard switches.

[121] VMware, Inc. ESX configuration guide. `http://www.vmware.com/pdf/vsphere4/r41/vsp_41_esx_server_config.pdf`. chapter 4 Basic networking with vNetwork distributed switches.

[122] VMware, Inc. VMware Capacity Planner. `http://www.vmware.com/files/de/pdf/datasheet_capacity_planner_de.pdf`.

[123] VMware, Inc. VMware Distributed Resource Scheduler (DRS). `http://www.vmware.com/products/drs/`.

[124] VMware, Inc. VMware ESXi and ESX Info Center. `http://www.vmware.com/products/vsphere/esxi-and-esx/`.

[125] VMware, Inc. VMware Server 2 - a risk-free way to get started with virtualization. `http://www.vmware.com/files/pdf/server_datasheet.pdf`.

[126] VMware, Inc. vSphere resource management guide. `http://www.vmware.com/pdf/vsphere4/r41/vsp_41_resource_mgmt.pdf`. chapter 4 Managing Storage I/O Resources.

[127] VMware, Inc. vSphere basic system administration. Technical report, VMware, Inc., 2010. `http://www.vmware.com/pdf/vsphere4/r40/vsp_40_admin_guide.pdf`.

[128] T. Wood, L. Cherkasova, K. M. Ozonat, and P. J. Shenoy. Profiling and modeling resource usage of virtualized applications. In *Middleware*, pages 366–387, 2008.

[129] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *NSDI'07: Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*, 2007.

[130] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. *Autonomic Computing, International Conference on*, 0:25, 2007.

[131] E. R. Zayas. Attacking the process migration bottleneck. In *In Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, pages 13–24, 1987.

[132] Q. Zhu, A. Shankar, and Y. Zhou. Pb-lru: a self-tuning power aware storage cache replacement algorithm for conserving disk energy. In *Proceedings of the 18th annual international conference on Supercomputing*, ICS '04, pages 79–88, New York, NY, USA, 2004. ACM.

[133] Q. Zhu and Y. Zhou. Power-aware storage cache management. *IEEE Trans. Comput.*, 54:587–602, May 2005.

[134] D. Zimmer. *VMware & Microsoft Virtual Server*, chapter 1 Einführung (Introduction). Galileo Press, 2005.

[135] D. Zimmer. *VMware & Microsoft Virtual Server*, chapter 4 Auswahl der möglichen virtuellen Maschine (Selection of Appropriate Services for Virtualization). Galileo Press, 2005.

Curriculum Vitae

Personal Data

Name:	Marko Hoyer
Place and date of birth:	January, 28^{th} 1980, Burg
Address:	Alexanderstraße 66 26121 Oldenburg
E-Mail:	Marko.Hoyer@offis.de

Education

06/2000	Gymnasium Georgianum Lingen(Ems), Degree: general university qualification

Alternative Public Service

09/2000 – 08/2001	German Red Cross

Academic Studies

10/2001 – 03/2006	Computer Science at Carl von Ossietzky University Oldenburg, Focus: Embedded Systems
10/2007 – 05/2011	PhD thesis: "Resource Management in Virtualized Data Centers Regarding Performance and Energy Aspects", Reviewers: Prof. Wolfgang Nebel (Univ. Oldenburg, Germany) and Prof. Michael Sonnenschein (Univ. Oldenburg, Germany)

Professional Career

since 04/2006	Research assistant at OFFIS e.V., devision energy
04/2006 – 07/2008	Employee, EC project CLEAN: Controlling Leakage In Nanometer CMOS SoCs
08/2008 – 08/2010	Researcher in an internally funded project: Energy Efficiency in Data Centers (preliminary research, public relations, establishment of this research topic at OFFIS)
since 09/2010	Researcher funded by an internal scholarship (for finishing the PhD thesis)

Teaching

Bachelor's thesis (supervisor and reviewer)

11/2008 – 02/2009	*Development of an Evaluation Environment for the Analysis of Correlated Resource Demand in Data Centers*, Thomas Strathmann

Master's thesis (supervisor and reviewer)

10/2007 – 04/2008	*Development of a Software for Testing and Assessing Power Management Strategies of Desktop and Notebook PCs*, Malte Viet
10/2007 – 03/2008	*Extension of a PC Power Management Strategy by a Learning Component*, Andreas Baumgart
02/2008 – 07/2008	*Reducing the Energy Consumption of Modern Server Farms*, Pierre Petliczew
11/2008 – 04/2009	*Development of a Scheduling Algorithm to Realize Dynamic Virtualization*, Daniel Schlitt

Student Project (supervisor)

10/2007 – 09/2008	Student project *EyeFly - The Flying Eye*, Department Embedded Systems at C.v.O. University Oldenburg

Publications and Talks

Publications

[1] M. Hoyer, D. Schlitt, K. Schröder, and W. Nebel. Proactive Dynamic Resource Management in Virtualized Data Centers. In *e-Energy '11: Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking*, 2011. ACM.

[2] M. Hoyer, K. Schröder, and W. Nebel. Statistical static capacity management in virtualized data centers supporting fine grained qos specification. In *e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 51–60, 2010. ACM.

[3] K. Schroeder, D. Schlitt, M. Hoyer and W. Nebel, Power and Cost Aware Distributed Load Management. In *e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 123–126, 2010. ACM.

[4] D. Schlitt, M. Hoyer, K. Schröder, W. Nebel. Analysis of Attainable Energy Consumption Reduction in ICT by Using Data Center Comprehensive Load Management (Extended Abstract). *Workshop: The Economics of Green IT*, ZEW, 2010

[5] D. Schlitt, K. Schröder, M. Hoyer, W. Nebel. Last- und Powermanagement bei Virtualisierung im Rechenzentrum. *NTZ 7-8/2010*, VDE

[6] W. Nebel, M. Hoyer, K. Schröder, and D. Schlitt. Untersuchung des Potentials von rechenzentrenübergreifendem Lastmanagement zur Reduzierung des Energieverbrauchs in der IKT (analysis of potential energy consumption reduction in ICT by using data center comprehensive load management). `http://www.lastmanagement.offis.de/`, 2009.

[7] M. Hoyer, A. Baumgart, and W. Nebel. Adaptives Powermanagement für Desktop- und Notebooksysteme (adaptive power management for desktop and notebook systems). In *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 32-2:96–104, 2009.

[8] M. Hoyer, D. Helms, W. Nebel. Modelling the impact of high level leakage optimization techniques on the delay of RT-components. *PATMOS'07*, 2007.

[9] D. Helms, O. Meyer, M. Hoyer, W. Nebel. Voltage- and ABB-Island Optimization in High Level Synthesis. *ISLPED'07*, 2007.

[10] D. Helms, M. Hoyer, W. Nebel. Accurate PTV, State, and ABB Aware RTL Blackbox Modeling of Subthreshold, Gate, and PN-Junction Leakage. *PATMOS'06*, 2006.

Patents

[1] M. Hoyer and D. Schlitt. Verfahren zur dynamischen Verteilung von einem oder mehreren Diensten in einem Netz aus einer Vielzahl von Rechnern. 05/2010. state: filed.

[2] M. Hoyer and A. Baumgart. Verfahren zum Optimieren des elektrischen Energieverbrauchs wenigstens einer Datenverarbeitungseinrichtung, insbesondere einer mobilen Datenverarbeitungseinrichtung und elektronische Vorrichtung sowie Datenträger zur Realisierung des Verfahrens. *DE102008036246B4*. 08/2008. state: granted.

[3] D. Helms and M. Hoyer. Method for simulation of circuit, involves simulating reference basic circuits and scaling simulation results of preceding steps in circuit, where step of scaling of reference basic circuits has scaling of channel width of transistors. *DE102006043805A1*. 09/2006. state: granted.

Public Talks

[1] Proactive Dynamic Resource Management in Virtualized Data Centers. Talk at e-Energy'11. Columbia University New York. 06/2011.

[2] Data Centers of The Future – Holistic System Management Solutions, *Workshop: ITOP – IT Operations Play*. University Bern. 11/2010.

[3] Statistical static capacity management in virtualized data centers supporting fine grained QoS specification. Talk at e-Energy'10. University Passau. 04/2010.

[4] Data Centers of The Future – Challenges and Perspectives. *Labs Talks - ICT for a low carbon society*. Deutsche Telekom Laboratories Berlin. 10/2009.

[5] Final presentation: analysis of potential energy consumption reduction in ICT by using data center comprehensive load management. *Expert workshop Green IT*. BMWi Berlin. 09/2009

Erklärung zur Dissertation

Oldenburg, den 18. Februar 2011

Ehrenwörtliche Erklärung zu meiner Dissertation mit dem Titel: „Resource Management in Virtualized Data Centers Regarding Performance and Energy Aspects“

Sehr geehrte Damen und Herren,

hiermit erkläre ich, dass ich die beigefügte Dissertation selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel genutzt habe. Alle wörtlich oder inhaltlich übernommenen Stellen habe ich als solche gekennzeichnet. Die Regeln guter wissenschaftlicher Praxis entsprechend der DFG-Richtlinien wurden eingehalten.

Ich versichere außerdem, dass ich die beigefügte Dissertation nur in diesem und keinem anderen Promotionsverfahren eingereicht habe und, dass diesem Promotionsverfahren keine endgültig gescheiterten Promotionsverfahren vorausgegangen sind. Ferner wurde der Inhalt dieser Dissertation nicht schon für eine Diplom- oder ähnliche Prüfungsarbeit verwendet.

Marko Hoyer