Cluster Computing manuscript No.
(will be inserted by the editor)

Performance Tradeoffs of Energy-Aware Virtual Machine Consolidation

Gergő Lovász · Florian Niedermeier · Hermann de Meer

Received: date / Accepted: date

Abstract Increasing power consumption of IT infrastructures and growing electricity prices have led to the development of several energy-saving techniques in the last couple of years. Virtualization and consolidation of services is one of the key technologies in data centers to reduce overprovisioning and therefore increase energy savings. This paper shows that the energy-optimal allocation of virtualized services in a heterogeneous server infrastructure is NP-hard and can be modeled as a variant of the multidimensional vector packing problem. Furthermore, it proposes a model to predict the performance degradation of a service when it is consolidated with other services. The model allows considering the tradeoff between power consumption and service performance during service allocation. Finally, the paper presents two heuristics that approximate the energy-optimal and performance-aware resource allocation problem and shows that the allocations determined by the proposed heuristics are more energy-efficient than the widely applied maximum-density consolidation.

1 Introduction

Increasing power consumption of IT infrastructures, growing electricity prices and ecological awareness are major reasons for a change towards green IT. Several approaches have been developed over the last years to reduce the power consumption of IT equipment. Some of the approaches address the development of energy-efficient hardware components with special energy-saving features [1][7][18], whereas others focus on the energy-efficient management of such components [10][17]. However, one of the main reasons for the low energy efficiency of today's data centers is overprovisioning. The uncertainty of long-term future demands and the unexpected occurrence of peak loads force data center operators to oversize their IT equipment even when most of the time these resources are utilized between 10 and 50 percent [2]. Meisner et al. show in [16] that idle or lowly utilized servers consume up to 70% of their maximum power consumption. Hence, a major part of the resources is unutilized and at the same time responsible for a significant amount of energy consumption.

Virtualization and consolidation are key technologies to dynamically deal with overprovisioning. Services can be encapsulated in virtual machines (VMs), which allow a flexible and transparent resource management. At times of low utilization, VMs are consolidated on a subset of the physical IT infrastructure and unused hardware can be turned off. When load increases, powered-off servers can be powered on again and the VMs are migrated from the overloaded servers back to servers with sufficient available resources. However, it is common knowledge that sharing of IT resources among different VMs leads to contention, e.g. multiple VMs demand central processing unit (CPU) time at the same time instant. This behavior leads to a performance degradation of the virtualized services during consolidation, even if the resources are not fully utilized. Therefore, the consolidation of virtualized services boils down to a tradeoff between performance and energy consumption. This leads to the first question tackled in this paper:

University of Passau
Innstrasse 43
94032 Passau, Germany
Tel.: +49-851-5093029
Fax: +49-851-5093052
E-mail: lovasz@fim.uni-passau.de

1. *What is the effect of consolidation on the performance of a virtualized service?*

A mathematical model is needed that describes the performance degradation of consolidated services. Such a model allows predicting the influence of consolidation on the performance of an individual service, based on known parameters, like the number of consolidated services on the server, the server's hardware characteristics or the server's overall resource utilization. Furthermore, the model has to be generic so that it can be applied using different performance metrics depending on the virtualized service. Knowing the performance degradation of a consolidated service, a second question arises:

2. *How can service consolidation be applied to reduce the energy consumption of a heterogeneous server infrastructure while respecting performance constraints of individual services?*

As described in Section 2, several approaches exist in literature that model the energy-aware consolidation of services in data centers. However, these models do not focus on the performance constraints of individual services. An energy-aware service consolidation model is needed that includes the performance model of a service so that both aspects, energy consumption and performance are considered while allocating virtualized services.

In our paper we present a performance model for consolidated services and an energy-aware service consolidation model that is able to consider performance constraints. We propose two heuristics that approximate the solution of this newly formulated energy-optimal and performance-aware virtual machine allocation problem. The evaluation of the developed algorithms is carried out by discrete event simulation.

The remainder of this paper is structured as follows: Section 2 gives an overview on related work, Section 3 discusses the tradeoff between energy consumption and performance and gives a model for the prediction of performance degradation of consolidated services. Section 4 models the energy-aware resource allocation as a variant of the NP-hard multidimensional vector packing problem and extends the model by performance awareness. In Section 5 different algorithms are presented that solve the energy- and performance-aware resource allocation problem. In Section 6 the performance degradation model is evaluated as well as the algorithms that are compared to each other and to other widely used algorithms for resource allocation. The paper is concluded with Section 7.

2 Related Work

Different models have been developed recently to describe the energy-efficient allocation of resources. Some of the approaches model the resource allocation problem as a packing problem [20], [3], [4]. In the models, services are modeled as multidimensional hypercubes that have to be allocated to multidimensional hyperbins that represent the servers. In [3] and [20] the authors assume a homogeneous server infrastructure and in [20], the multidimensional binpacking problem is reduced to only 1 dimension, representing the CPU. In [12] the resource allocation problem is modeled as a generalized assignment problem. The authors assume a heterogeneous server infrastructure and multiple tasks that have to be allocated to the servers. As optimization method, methods of game theory are used. Similarly, in [13] game theory is used to determine an optimal resource allocation in high performance data centers. However, in [12] and [13] the problem is modeled as a one dimensional allocation problem. In [13] only the memory allocation is optimized whereas in [12] only the CPU is considered. Genetic and evolutionary algorithms are used in [5] and [15] to determine optimal VM allocations. In [5], the resource allocation problem is modeled as a knapsack problem. However, the optimization goal is not specifically energy saving but the compliance with the constraints in the service level agreements. Several approaches use service consolidation to reduce the energy consumption of a data center infrastructure [20],[14],[6],[3],[19],[4]. However, only limited analysis exists on the impact of consolidation on the performance of the consolidated services. The authors of [20] and [14] consolidate services in a homogeneous server infrastructure, however the performance of the consolidated services is not taken into account. In [6] and [3], the effect of consolidation on the performance of services is only considered regarding the provisioning of requested resources. Srikantaiah et al. analyze in [19] the inter-relationship between energy consumption, performance and resource utilization of consolidated workloads. Based on a given energy efficiency metric they find optimal operation points. However, the results are highly dependent on the chosen energy efficiency metric and do not allow to predict the performance degradation of a single service when it is consolidated. In [4], the impact of consolidation on the performance is considered, however, the chosen performance metrics only evaluate the performance of a data center as a whole.

3 Performance Impact of Consolidation

Operators of data center infrastructures (e.g. clouds) face the challenge of two contradicting goals, namely reduction of infrastructure energy consumption while complying with Service Level Agreements (SLAs). On one hand, the reduction of energy consumption helps to reduce the total cost of ownership whereas on the other hand, a violation of SLAs may lead to penalties. In the following, we describe a use case of a cloud computing environment in which the cloud provider offers to its customers virtual servers with certain performance levels that have to be guaranteed. Based on this use case we discuss the possibility of saving energy through the consolidation of the virtual servers and the effect of such a consolidation on the performance of the services running on the virtualized servers.

3.1 Use Case: Cloud Computing

In cloud computing environments, for instance, the Amazon EC2[1], users can usually choose between virtual servers with different performance levels. In the case of Amazon EC2, there are three different standard virtual servers user can choose from: a *small*, a *large* and an *extra large* one. The different instances provide different performance levels. E.g. a small virtual server has only one virtual core whereas an extra large virtual server provides its user with four virtual cores. There is also a difference between the performance levels of the cores. The small virtual server's virtual CPU core provides 1 *EC2 Compute Unit* whereas the extra large virtual server's cores computational power is 2 *EC2 Compute Units* for each core. Based on the type of the applications, the users of the cloud can even choose application-specific virtual servers. Amazon EC2, for instance, provides two different types of *High-CPU* virtual servers which are well suited for compute-intensive applications. Behind the virtual servers there is a physical server infrastructure. It is the responsibility of the cloud provider to determine the amount of physical resources (e.g., CPU cycles, RAM, bandwidth) that have to be assigned to the virtual servers in order to provide the promised performance level.

3.1.1 Energy Saving through Consolidation

Consolidation of virtualized servers is a widely applied measure to decrease energy consumption since it significantly reduces the percentage of idle power in the overall infrastructure. Such a consolidation can be done either statically or dynamically at runtime. In the static approach, the mapping of the virtual servers to the physical infrastructure is done beforehand and cannot be changed at runtime. In the case of a cloud provider, the static consolidation approach assigns to each virtual server a certain amount of physical resources based on the performance level of the virtual server. However, such a static assignment of physical resources does not solve the problem of overprovisioning. In the static consolidation approach, the assignment of the physical resources has to be done based on the resource requirements of a fully loaded virtual server in order to ensure that in times of high load the performance level of the virtual server can be guaranteed. On the other hand, a dynamic consolidation of virtual servers allows the reassignment of physical resources at runtime when the load on the virtual servers changes. If there is low load on the virtual server an assignment of less physical resources could be sufficient to provide a certain performance level. If the load on the virtual server increases, more physical resources can be assigned to the virtual server. The virtual server might be even migrated to another physical host if the current physical host gets overloaded. Therefore, the dynamic consolidation approach allows for an over-commitment of the physical servers and helps to reduce the overprovisioning of physical resources. A dynamic consolidation of virtual servers mandates the cloud provider to monitor the resource utilization of the virtual servers in order to determine how many physical resources have to be assigned to it. There are several monitoring solutions that can be used for this purpose. Several hypervisors, e.g. the KVM[2] hypervisor, are based on a Linux operating system. The Linux operating system allows for monitoring the exact resource utilization of single processes by reading the corresponding entries of the proc file system. Since in such systems VMs are ordinary Linux processes, they can easily be monitored. Furthermore, the proc file system contains also information on utilization of hardware resources like the CPU or the hard drive disk. More sophisticated monitoring solutions are even able to monitor multiple physical and virtual servers in a data center infrastructure. Such a monitoring tool is e.g. Nagios[3]. In our use case, we assume that the cloud environment provides monitoring information on the resource utilization of the virtual servers. Furthermore, the cloud environment provides the possibility to dynamically consolidate virtual servers at runtime based on their resource utilization. Especially the over-commitment of physical servers is allowed so that over-provisioning of resources can be reduced.

[1] http://aws.amazon.com/en/ec2/instance-types/

[2] http://www.linux-kvm.org

[3] http://www.nagios.org/

3.1.2 Impact of Consolidation on the Performance of Virtual Servers

Running multiple virtualized servers on the same physical hardware intensifies the contention for hardware resources. Even if a hardware resource, e.g. the CPU, is lowly utilized, concurrent requests to it can lead to delays due to context switching. Increased delays have a negative effect on the performance of the virtual servers and have to be considered when applying consolidation. In our use case, the cloud provider has to consider the impact of dynamic virtual server consolidation on the performance of the virtual servers. If several virtual servers are lowly utilized it is self-evident to assign less physical resources to them by consolidating more virtual servers on the same physical host in order to reduce the power consumption of the physical infrastructure. However, in order to find the optimal balance between energy savings and performance the cloud provider needs to know how much the performance of a virtual server will be degraded if the virtual server shares physical resources with other virtual servers. The performance baseline of a virtual server is always its performance level that is defined in the SLAs. This performance level can always be ensured when a certain amount of physical resources (e.g. number of assigned CPU cycles, size of RAM, I/O rate) is assigned exclusively to the virtual server. The amount of physical resources that corresponds to a certain performance level has to be defined by the cloud provider. Allowing non-exclusive resource usage of virtual servers by consolidating them on the same physical server will lead to a certain performance degradation. How much performance degradation is allowed is determined by the SLAs. E.g. the performance of the Amazon EC2 virtual servers is given in *EC2 Compute Units*. "One *EC2 Compute Unit* provides the equivalent CPU capacity of a 1.0 - 1.2 GHz 2007 Opteron or 2007 Xeon processor." [4] By defining a range for the performance of a CPU instead of a single value allows for some performance degradation. For an *EC2 Compute Unit*, a performance degradation of 16.6% is acceptable (1.0 GHz instead of 1.2 GHz). A model that predicts the performance degradation of a virtual server regarding a performance metric would allow for the cloud provider to find a resource allocation that reduces infrastructure energy consumption and considers performance levels at the same time.

[4] http://aws.amazon.com/en/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it

3.1.3 Performance Degradation Model for Consolidation

In this section, we provide a model that predicts the performance degradation of a virtual server as described in the use case in the previous subsections. As performance metric we take the mean response time, assuming *High-CPU* virtual servers that are hosting CPU-intensive services. In a first step, a performance degradation model p_{no_virt} is developed that predicts the performance degradation of CPU-intensive services in a non-virtualized environment (service = linux process) when it is competing for the CPU with other CPU-intensive services. In a second step, the model is extended by considering the overhead that is caused when a service is encapsulated in a VM (Section 3.7). In the evaluation section of the paper we show that our models are able to predict the performance degradation of consolidated services in non-virtualized and virtualized environments. In the following the term service denotes a single application running as a process on a Linux system in the non-virtualized case or running inside a VM in the virtualized scenario. Defining a service in this way does not mean that no higher level services can exist which use multiple of these individual services. In such a case the performance degradation of the higher level service is determined based on the performance degradations of the individual services. However this is out of the scope of this paper.

Intuitively, besides the process scheduler, there are three parameters that have an impact on the performance of a service v. These are: 1) $v_{\#}$: the number of services competing for the same CPU core as v, 2) s^{CPU}: the overall load on the CPU core of the server s on which v is running and 3) v^{CPU}: the required CPU cycles of v itself.

The impact of these three parameters is analyzed by applying a sensitivity analysis. In the performance measurements, described in the Sections 3.3, 3.4 and 3.5, one out of the three parameters is fixed whereas the other two parameters are varied in order to determine their impact on the performance of v. As service, we choose a load generator which generates load on the CPU by performing mathematical operations ($sin(x)$, $cos(x)$, $pow(x, y)$, $x \cdot y$, $x + y$). The load generator is able to put an arbitrary load on a predefined core of the CPU. The desired load is achieved by periodically suspending the computation of the mathematical operations with sleep phases. The duration of a sleep phase is determined based on the duration of the preceding computational phase so that the generated load is equal to the desired load. Independently from the chosen load level, v performs a predefined number of mathematical operations. Measurement of the execution time of these operations allows for determining performance of v. As performance metric we choose the execution time of 1 loop iteration of the computational phase in which each of the previously mentioned mathematical operations are executed once. The performance measurements were carried out on a server with 16 GB RAM and 2 Intel Xeon E5420 processors, each 2.5 GHz. The measurements in the Subsections 3.2, 3.3, 3.4 and 3.5 were executed 30 times. The given confidence intervals have a confidence level of 99%.

3.2 Reference Measurements

In order to determine the performance degradation of the service v when it is consolidated with other services, reference measurements are needed. The performance degradation of a service is the additional execution time that is caused by the consolidation with other services. Therefore, we consider as baseline performance the execution time of v in an unconsolidated scenario when it has exclusive access to the CPU. In such a scenario v is the only service using the CPU.

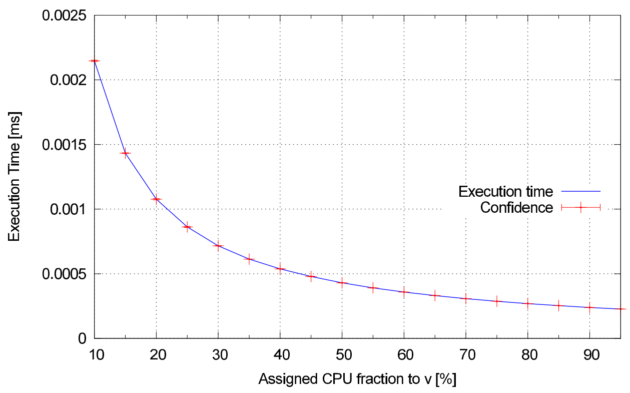

Fig. 1 Performance of v in an unconsolidated scenario

Figure 1 shows the execution time of v for different load levels in an unconsolidated scenario, normalized by the number of loop iterations in the computational phase. As expected, the execution time is halved when the number of CPU cycles assigned to the service is doubled. The performance degradation of v, when it is consolidated with other services, is determined by comparing its execution time to the reference values in Figure 1.

3.3 Constant Number of Services

In the first measurement of the performance degradation of v in a consolidated scenario, the number of services is kept constant. In Figure 2, v is consolidated with an additional service w. v and w utilize the CPU core at the same load levels. The load of the services varies between 10% and 50% in 5% steps so that the overall load is 20%, 30%, ..., 100%. The y-axis shows the average performance degradation for v. E.g. at 50% overall load (service load = 25%), the performance degradation for v is approxemately 15%, compared to its performance when it utilizes 25% of the CPU core in the unconsolidated scenario.

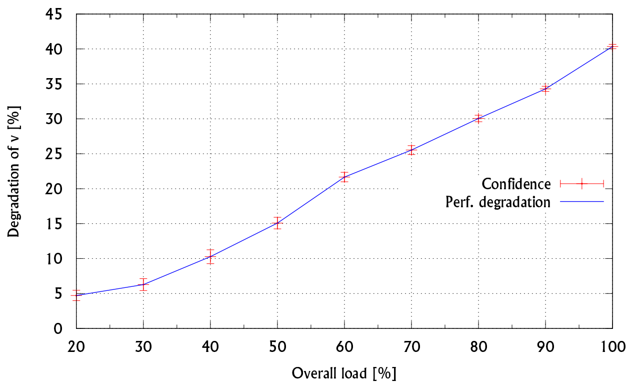

Fig. 2 Constant number of services, varying overall load and process load

According to the graph in Figure 2, the performance degradation of a consolidated service grows linearly with the overall load and the service load.

3.4 Constant Overall Load

In the measurement of Figure 3, the overall utilization of the CPU core is kept constantly at 90%. This 90% load is achieved by dividing the load equally between 1, 2, ..., 7 services. The x-axis shows the number of services, the y-axis the performance degradation of a single service.

According to the graph in Figure 3, the performance degradation of v grows with the number of consolidated services. The number of consolidated services has a higher impact on the performance degradation than the load of the single services since a doubling of the number of services leads to a bisection of the service load but the performance degradation is still increasing.

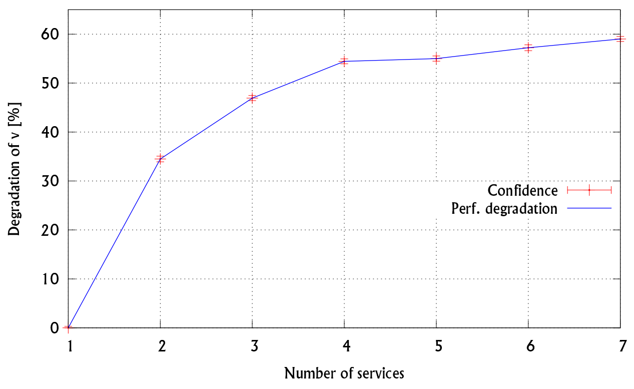

Fig. 3 Constant overall load, varying number of services and service load

3.5 Constant Service Load

In the last measurement, the service load is kept constant whereas the overall load and the number of processes is varying. In Figure 4, each service puts 10% load on the CPU core so that the overall load is 10%, 20%, ...,100% for 1, 2, ..., 10 services.

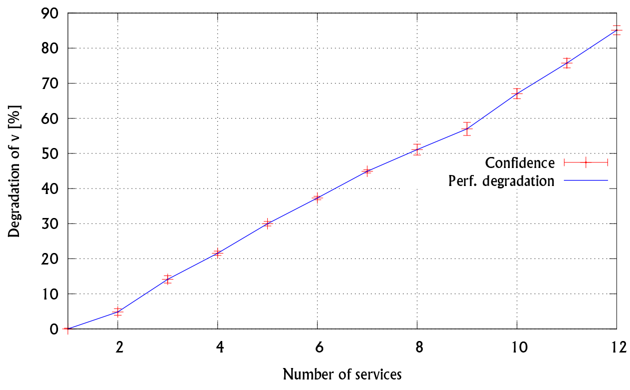

Fig. 4 Constant service load, varying number of services and overall load

With growing overall load and growing number of services the performance degradation of v increases linearly.

3.6 Performance Degradation Function

In order to find the relationship between the number of services, the overall load on the CPU core and the load of a single service symbolic regression is used to discover mathematical equations. As input, the parameters of the measurements in the Sections 3.2, 3.3, 3.4 and the corresponding performance loss are used. Based on the linearity of the performance degradation that can be observed in the measurements of the Sections 3.3 and 3.5 the assumption is made that the performance degradation of a service can be approximated by a linear formula. The overall goal is to define a function $p_{no\_virt} : (\mathcal{V} \times \mathcal{S}) \rightarrow \mathbb{R}$, with $\mathcal{V}$ denoting the set of virtualized services and $\mathcal{S}$ denoting the set of servers, that estimates the performance degradation of a service $v \in \mathcal{V}$ when it is allocated to a server $s \in \mathcal{S}$. The computed formula is presented in Equation (1).

$$p_{no\_virt}(v, s) = 0.802 \cdot s^{CPU} - 0.762 \cdot v^{CPU} - 3.6 \qquad (1)$$

In Equation (1) it can be seen that $v_{\#}$, s^{CPU} and v^{CPU} are dependent. Because of the constraint that all services utilize the CPU core at the same level, the relationship between $v_{\#}$, s^{CPU} and v^{CPU} is given by $s^{CPU} = v_{\#} \cdot v^{CPU}$. For different service load levels, Equation (1) can only be applied in combination with a round robin scheduler. The round robin scheduler ensures that each service gets the same amount of CPU time, independently from its CPU requirements.

3.7 Consideration of Virtualization

The performance degradation function p_{no_virt} only estimates the performance degradation for processes in a non-virtualized environment. Based on p_{no_virt} we develop a performance degradation function p_{virt} that considers the overhead caused by virtualization. We assume that the overhead caused by virtualization consists of two factors: First, the number of VMs plays a role since for each consolidated VM on the server there will be an additional load on the CPU which influences the overall server performance. Second, the higher the load on the VM the higher the overhead. As a consequence of our assumptions we define p_{virt} as follows:

$$p_{virt}(v, s) = p_{no\_virt}(v, s) + v_{\#} \cdot (\lambda_1 + \lambda_2 v^{CPU}) \qquad (2)$$

In Equation 2 λ_1 stands for the performance degradation that is caused by each single VM. Since our second assumption was that the overhead of a VM is also depending on the load on the VM, we modify λ_1 with a fraction of the load on the VM, given by $\lambda_2 v^{CPU}$.

Experimentally we determined λ_1 to be 1 and λ_2 to be 0.1. In Section 6 we will evaluate p_{virt} and p_{no_virt} through measurements in virtualized and non-virtualized scenarios and will show that our assumptions regarding the overhead due to virtualization are correct.

3.8 Applicability and Limits of our Approach

In this section we want to discuss briefly the applicability and the limits of the presented performance degradation function. The benchmark that was chosen to develop the performance degradation function performed CPU-intensive operations. Therefore the response time was mainly determined by the performance of the CPU. Hence the performance degradation function is limited to predict the performance degradation of CPU intensive workload, like the workload that can be found on the *High CPU* virtual servers of the Amazon EC2 cloud. Furthermore we present a performance degradation function that considers the overhead of virtualization which allows its application in real world virtualization environments. Finally, we evaluate the performance degradation function on servers with different Intel CPUs. At least this indicates the independence of the model from specific hardware. Limit of the model is the choice of the scheduler, which is a Round Robin Scheduler. The impact of other schedulers on the performance degradation remains future work.

4 Energy-Optimal and Performance-Aware Resource Allocation Model

In this section a model for the energy-optimal and performance-aware VM allocation is given. The model is developed step by step. First, a model is presented that minimizes the power consumption of the physical server infrastructure but does not consider performance issues. In a second step, this model is extended by the performance degradation function that is described in Section 3. The novelty of our resource allocation model is that it reduces infrastructure power consumption while considering performance requirements of single services. In the following, the terms *VM* and *virtualized service* are used equivalently. We assume that each VM contains only 1 service. The virtualized service is therefore the service and the VM in which the service is encapsulated.

4.1 Energy-Optimal Resource Allocation

The energy-optimal resource allocation problem consists of a set $\mathcal{V} = \{v_1, ..., v_n\}$ of n, $n \in \mathbb{N}$, virtualized services that has to be mapped unto a set $\mathcal{S} = \{s_1, ..., s_m\}$ of m, $m \in \mathbb{N}$, heterogeneous servers, so that the following conditions are fulfilled:

Condition 1	The resource requirements of all $v \in \mathcal{V}$ on CPU, random access memory (RAM), hard disk drive (HDD) and network interface card (NIC) are fully met
Condition 2	The utilizations of the CPU, RAM, HDD and NIC for all $s \in \mathcal{S}$ are always less than or equal to 100%
Condition 3	The sum of the power consumption for all $s \in \mathcal{S}$ is minimal compared to all other mappings.

In the case of CPU, the requirements of a virtualized service are the number of needed CPU-cycles per second. In the case of the RAM, the requirement is the size of the RAM (in *MB*) that is needed by the virtualized service, whereas in the case of the HDD and NIC, it is the I/O rate and the bandwidth in *mbps*, respectively. These requirements can be determined based on the service's profile, if existing, and monitoring data. Similar to the resource requirements of services, the utilizations of CPU, RAM, HDD and NIC represent the number of used CPU-cycles, the size of allocated RAM, the extent of used I/O rate and the used bandwidth. While the first two conditions ensure that the consolidated services do not suffer from resource starvation, the latter conditions' goal is to minimize the overall energy consumption of the physical server infrastructure.

4.2 Assumptions

For the energy-optimal resource allocation model that is described in Section 4.3, the following assumptions are made:

1. **Service requirements**: A virtualized service has 4 different resource requirements which are the requirements on CPU, RAM, HDD and NIC, as previously described. Although these four resources model the most energy consuming demands of a service, more resources may be added later to further increase the model accuracy.
2. **Additivity of service requirements**: Let $v^{CPU}(s)$, $v^{RAM}(s)$, $v^{HDD}(s)$, $v^{NIC}(s)$ denote the CPU, RAM, HDD and NIC requirements of a virtualized service v on the server s. If two virtualized services v and v' are consolidated on the same server s, the resource requirements of the two services together are the sum of the resource requirements of v and v': $(v + v')^{CPU}(s) = v^{CPU}(s) + v'^{CPU}(s)$, $(v + v')^{RAM}(s) = v^{RAM}(s) + v'^{RAM}(s)$, $(v + v')^{HDD}(s) = v^{HDD}(s) + v'^{HDD}(s)$ and $(v + v')^{NIC}(s) = v^{NIC}(s) + v'^{NIC}(s)$. However, running multiple services on the same hardware will of course lead to additional overhead due to context switching of the CPU, which is also considered by explicitly modeling the performance degradation.
3. **Requirement adaption**: Let s and s' be two servers having different CPU frequency, RAM size, I/O rate and bandwidth for data communication.

For the RAM, HDD and NIC requirements of a virtualized service v we assume that the requirements of v are the same on s and s': $v^{RAM}(s) = v^{RAM}(s')$, $v^{HDD}(s) = v^{HDD}(s')$, $v^{NIC}(s) = v^{NIC}(s')$. In the case of the CPU, regarding the required CPU cycles we assume $v^{CPU}(s) = v^{CPU}(s')$. For different CPU clock frequencies of s and s', the same number of required CPU cycles will result in a different load on the CPUs of s and s'. A service, e.g., that requires $5 \cdot 10^8$ CPU cycles per second puts 50% load on a 1GHz CPU and 25% load on a 2GHz CPU. First measurements on an Intel Xeon E5420 (2.5 GHz), Intel Core 2 Duo E6400 (2.13 GHz) and AMD Phenom II X4 955 (3.2 GHz) show that the mean error of the suggested requirement adaptation for CPUs is approximately 5%. If this assumption does not hold, it will increase the overall model error, however the general findings are not invalidated.

4. **Server power consumption model**: The power consumption of a server is calculated using the linear, CPU load dependent model proposed by Fan et al. [8]. The model consists of a per server constant idle power and the dynamic power consumption caused by CPU and fans. The server power is modeled as $P_{idle} + u_{CPU} \cdot (P_{max} - P_{idle})$.

4.3 Formal Problem Description

In order to describe the energy-aware resource allocation problem in a formal way that allows finding an algorithmic solution, we introduce the following mathematical notation:

$\mathbf{v} = (v^{CPU}, v^{RAM}, v^{HDD}, v^{NIC})$, $\mathbf{v} \in \mathbb{R}^4$	Vector, representing a virtualized service with v^{CPU} denoting the number of required CPU cycles, v^{RAM} denoting the size of required RAM in MB, v^{HDD} denoting the required I/O rate in *mbps* and v^{NIC} denoting the needed data rate for network communication in *mbps*.
$V = \{\mathbf{v_1}, ..., \mathbf{v_n}\}$	Set of n, $n \in \mathbb{N}$, vectors representing n virtualized services
$\mathbf{s} = s^{CPU} \times s^{RAM} \times s^{HDD} \times s^{NIC}$	4-dimensional hyper bin in $\mathbb{R}^4$, representing a server with s^{CPU} denoting the number of overall CPU cycles, s^{RAM} denoting the overall RAM size in MB, s^{HDD} denoting the overall I/O rate in *mbps* and s^{NIC} denoting the data rate in *mbps* of the communication link to which the server represented by s is connected.
$S = \{\mathbf{s_1}, ..., \mathbf{s_m}\}$	Set of m, $m \in \mathbb{N}$, hyper bins representing m servers.
$\alpha : \mathcal{V} \to \mathcal{S}$	Resource allocation function that assigns to each $v \in \mathcal{V}$ a server $s \in \mathcal{S}$.

With the given notation, the energy-optimal resource allocation problem can be modeled as a variant of the multidimensional vector packing problem. In the following, the formal problem description is developed step by step.

– First, a definition of the multidimensional vector packing is given (Section 4.3.1).
– It is shown that a resource allocation that considers only Condition 1 and Condition 2 can be transformed to the multidimensional vector packing problem as given in Definition 1 (Section 4.3.2).
– The multidimensional vector packing problem is then adapted by assigning a cost function to each bin and by changing the optimization goal to the minimization of the sum of all cost functions (Section 4.3.3).
– It is shown that the energy-aware resource allocation that considers all 3 conditions can be transformed to the newly defined vector packing problem with cost functions (Section 4.3.4).
– In a last step, the energy consumption of migration is integrated into the model (Section 4.3.5).

4.3.1 Multidimensional Vector Packing

In this subsection the multidimensional vector-packing problem is defined. Simply described, the goal of the multidimensional vector packing is to pack a finite number of multidimensional vectors into a finite number of multidimensional bins with variable sizes so that the number of non-empty bins is minimal. A more formal description is given in Definition 1.

Definition 1 (Multidimensional Vector Packing) Let $V = \{\mathbf{v_1}, ..., \mathbf{v_n}\}$, $n \in \mathbb{N}$, be a set of n d-dimensional vectors, with $d \in \mathbb{N}$ and $d \geq 2$. Let $S = \{\mathbf{s_1}, ..., \mathbf{s_m}\}$,

$m \in \mathbb{N}$, be a set of m d-dimensional bins. Furthermore, let $a : V \to S$ be a function that assigns to each $\mathbf{v} \in V$ a bin $\mathbf{s} \in S$ and $S^a = \{\mathbf{s} \in S | a^{-1}(\mathbf{s}) \neq \emptyset\}$ be the set of non-empty bins after the assignment of the vectors in V to the bins in S by a. The goal of the multidimensional vector packing is to determine a, so that:

1. $\forall \mathbf{s} \in S$ the vector $\sum_{\{\mathbf{v} \in V | a(\mathbf{v}) = \mathbf{s}\}} \mathbf{v}$ fits into $\mathbf{s}$
2. the number of non-empty bins is minimal: $\forall a', a' : V \to S$, $|S^a| \leq |S^{a'}|$.

4.3.2 Resource Allocation Modeled by Multidimensional Vector Packing

A resource allocation function $\alpha : \mathcal{V} \to \mathcal{S}$ that only aims at fulfilling Condition 1 and Condition 2, without considering the overall energy consumption, can be transformed to the multidimensional vector packing problem as described in Definition 1. In Definition 1, we choose $d = 4$ to be the dimension of the vectors and the bins in V and S, respectively. Furthermore, we set $V := \mathcal{V}$ and $S := \mathcal{S}$. Then the function a computes by definition an assignment of the vectors in V to the bins in S that respects the requirements as given in Definition 1. We set $\alpha := a$. The way how the vectors and the bins are modeled, as well as the properties of a ensure that for α, Condition 1 and Condition 2 are fulfilled. Regarding Condition 3, the minimization of the number of non-empty bins, as it is done by a, corresponds to the minimization of the number of powered on servers by α. These servers host the consolidated virtualized services. However, in a heterogeneous server environment, the minimization of the number of powered on servers is not necessarily equivalent to the minimization of the overall power consumption.

4.3.3 Multidimensional Vector Packing with Dynamic Costs

To consider the overall power consumption, Definition 1 has to be adapted by assigning a dynamic cost function to the bins and by changing the optimization goal. The multidimensional vector packing problem with dynamic costs is defined in Definition 2.

Definition 2 (Multidimensional Vector Packing with Dynamic Costs) Let $V = \{\mathbf{v_1}, ..., \mathbf{v_n}\}$, $n \in \mathbb{N}$, be a set of n d-dimensional vectors, with $d \in \mathbb{N}$ and $d \geq 2$. Let $S = \{\mathbf{s_1}, ..., \mathbf{s_m}\}$, $m \in \mathbb{N}$, be a set of m d-dimensional bins. Furthermore, let $a : V \to S$ be a function that assigns to each $\mathbf{v} \in V$ a bin $\mathbf{s} \in S$. Let $V^{\mathbf{s}} = \{\mathbf{v} \in V | a(\mathbf{v}) = \mathbf{s}\}$ and $\mathbf{v}^{\mathbf{s}}_{\mathbf{sum}} := \sum_{\mathbf{v} \in V^{\mathbf{s}}} \mathbf{v}$. We assign to each bin $\mathbf{s} \in S$ a cost function $f_{\mathbf{s}} : \mathbb{R}^d \to \mathbb{R}$. $f_{\mathbf{s}}$ is required to be strictly increasing in each dimension and $f_s(\mathbf{0}) > 0$. Furthermore, let $S^a = \{\mathbf{s} \in S | a^{-1}(\mathbf{s}) \neq \emptyset\}$ be the set of non-empty bins after the assignment of the vectors in V to the bins in S by a. The goal of the multidimensional vector packing is to determine a, so that

1. $\forall \mathbf{s} \in S$ the vector $\sum_{\{\mathbf{v} \in V | a(\mathbf{v}) = \mathbf{s}\}} \mathbf{v}$ fits into $\mathbf{s}$
2. the sum of the cost functions of all non-empty bins with $\mathbf{v}^{\mathbf{s}}_{\mathbf{sum}}$ as input is minimal: $\forall a', a' : V \to S, a' \neq a : \sum_{\mathbf{s} \in S^a} f_{\mathbf{s}}(\mathbf{v}^{\mathbf{s}}_{\mathbf{sum}}) \leq \sum_{\mathbf{s} \in S^{a'}} f_{\mathbf{s}}(\mathbf{v}^{\mathbf{s}}_{\mathbf{sum}})$.

4.3.4 Resource Allocation Modeled by Multidimensional Vector Packing with Dynamic Costs

A resource allocation $\alpha : \mathcal{V} \to \mathcal{S}$ that fulfills Condition 1, Condition 2 and Condition 3, can be transformed to the multidimensional vector packing problem with dynamic costs as described in Definition 2. In Definition 2, we choose $d = 4$ to be the dimension of the vectors and the bins in V and S, respectively. Furthermore we set $V := \mathcal{V}$ and $S := \mathcal{S}$. We assign the power consumption function pow_s of a server $s \in \mathcal{S}$ to be the cost function $f_{\mathbf{s}}$ of the corresponding bin $\mathbf{s} \in S$ that represents the server s. The server power consumption functions fulfill the requirements of a cost function as given in Definition 2. Then the function a computes by definition an assignment of the vectors in V to the bins in S that respects the requirements as given in Definition 2. We set $\alpha := a$. The way how the vectors and the bins are modeled, as well as the properties of a ensure that α fulfills Condition 1, Condition 2 and Condition 3.

4.3.5 Consideration of Migration

The additional energy consumption caused by the migration of virtualized services has not been considered in the model yet. Every time a changed allocation of virtualized services is calculated by α, some of the virtualized services have to be migrated from one host to another. This leads to an increased power consumption of the overall infrastructure. The resource allocation function a or α, respectively, as defined in the Sections 4.3.3 and 4.3.4 only compare the power consumption of different allocations after the migration. In order to consider the costs of migration, we modify the model in Definition 2 by extending it with a cost function $f^a_{mig} : V \to S$ representing the migration. $f^a_{mig}(\mathbf{v})$, $\mathbf{v} \in V$, computes the power consumption overhead that is caused by the reallocation of $\mathbf{v}$ by a. To compute this overhead, we use a simple model. Let v_{size} denote the size of the data of the virtualized service v which has to be migrated in $Mbit$ and d denote the available data rate for the migration in $Mbps$. Then the duration of the migration t can be approximated by $t \approx \frac{v_{size}}{d}$. Furthermore, we assume that during the data transfer the power consumption of the participating servers increases by 1 W [11]. Knowing t and the power consumption overhead of $2 \cdot 1\ W$ it is possible to estimate the energy consumption of the migration, which is $t \cdot 2\ W$. However, to integrate the migration costs into Definition 2, the energy consumption of the migration has to be transformed to power consumption. The envisioned resource management checks periodically the necessity of service reallocations. If the duration of such a period is T seconds, the average power consumption increase that is caused by a migration that lasts t seconds is $\frac{t \cdot 2\ W}{T}$. We define f^a_{mig} as follows:

$$f^a_{mig}(\mathbf{v}) = \begin{cases} 0, & if\ \mathbf{v}\ is\ not\ migrated\ by\ a \\ \frac{t \cdot x}{T}, & if\ \mathbf{v}\ is\ migrated\ by\ a \end{cases} .$$

We change the optimization goal of Definition 2 to determine a so that

$$\forall a', a' : V \to S, a' \neq a : \sum_{s \in S^a} f_s(\mathbf{v}^{\mathbf{s}}_{\mathbf{sum}}) +$$

$$\sum_{v \in V} f^a_{mig}(\mathbf{v}) \leq \sum_{s \in S^{a'}} f_s(\mathbf{v}^{\mathbf{s}}_{\mathbf{sum}}) + \sum_{v \in V} f^{a'}_{mig}(\mathbf{v})$$

4.4 Complexity

In order to show the NP-hardness of the modified multidimensional vector packing problem, the NP-complete [9] binpacking problem is reduced to it. The binpacking problem is given in Definition 3

Definition 3 (Binpacking) Let $I = \{i_1, ..., i_n\}$, with $n \in \mathbb{N}$ be a set of n items. To each item $i \in I$ a size $l(i) > 0$ is assigned. Furthermore, let $B = \{b_1, ..., b_m\}$ with $m \in \mathbb{N}$ be a set of m bins. Each bin $b \in B$ has the same size l_{bin}. The goal of the binpacking problem is to pack all items in I in as few bins as possible so that for each bin $b \in B$, $l(i^b_1) + ... + l(i^b_j) \leq l_{bin}$, where $i^b_1, ..., i^b_j$ denote the items in I that were packed into the bin b.

4.4.1 Reduction of the Binpacking Problem to the Modified Multidimensional Vector Packing

Let a be a function as defined in Definition 2. For each item $i \in I$ we define a vector $\mathbf{v} = (v_1, ..., v_d)$ with $v_1 = l(i)$ and $v_2 = ... = v_d = 0$. Furthermore, for each one dimensional bin b we define a d-dimensional bin $s = \{s_1, ..., s_d\}$ with $l(s_1) = l_{bin}$ and $l(s_2) = ... = l(s_d) = 0$, with $l(s_i)$ denoting the length of the hyper bins' edge in the i-th dimension. For each d-dimensional bin s we choose a cost function $f(\mathbf{w}) = |\mathbf{w}| + c$, with $\mathbf{w} \in \mathbb{R}^d$ and $c \in \mathbb{R}^+$. Then f is strictly increasing and $f(\mathbf{0}) = c > 0$. The function a has the property to minimize $\sum_{i=1}^k f_{s_i}(\mathbf{v}^{\mathbf{s_i}}_{\mathbf{sum}})$, the sum of the cost functions of all non-empty bins $s_1, ..., s_k$ with $\mathbf{v}^{\mathbf{s_i}}_{\mathbf{sum}}$ as input for s_i. $\sum_{i=1}^k f_{s_i}(\mathbf{v}^{\mathbf{s_i}}_{\mathbf{sum}})$ can also be written as $k \cdot c \cdot \sum_{i=1}^k \mathbf{v}^{\mathbf{s_i}}_{\mathbf{sum}} \cdot \mathbf{e_1}$, with $\mathbf{e_1} \in \mathbb{R}^d$ and $\mathbf{e_1} = (1, 0, 0, ..., 0)$. But in this case $\sum_{i=1}^k \mathbf{v}^{\mathbf{s_i}}_{\mathbf{sum}} \cdot \mathbf{e_1} = \sum_{i=1}^n v_i$ and is therefore independent from the number of non-empty bins. Thus, the minimization of $\sum_{i=1}^k \mathbf{v}^{\mathbf{s_i}}_{\mathbf{sum}} \cdot \mathbf{e_1}$ is equivalent to the minimization of $c \cdot k$ and therefore to the minimization of k which is the number of non-empty bins.

4.5 Extension of the Energy-Optimal Resource Allocation by Performance Awareness

The resource allocation function α, as presented in the previous sections minimizes the overall server power consumption $\sum_{s \in S^a} pow_s(\mathbf{v}^{\mathbf{s}}_{\mathbf{sum}})$ but does not consider performance constraints of the consolidated services. Therefore α can be regarded as a pure energy-aware resource allocation function. By replacing the cost function pow_s of s by $P : S \to \mathbb{R}$, $P(s) = \sum_{\{\mathbf{v} \in V | a(\mathbf{v}) = s\}} p(v, s)$ a resource allocation function that minimizes the performance degradation of the allocated services is obtained. Basically, this new resource allocation maximizes the performance of the virtualized services without considering energy consumption. Combining these two extremes, a third category of resource allocation function can be derived which is energy-optimal and performance-aware at the same time.

4.5.1 Energy-Optimal Resource Allocation with Performance Constraints

A first approach is to minimize the power consumption of the infrastructure considering certain performance constraints. The limitation of service degradation to maximal $x\%$ could be such a constraint. We define a new cost function $f_p : \mathcal{S} \to \mathbb{R}$ and assign it to the servers in $\mathcal{S}$:

$$f_p(s) = \begin{cases} pow_s(\mathbf{v}^{\mathbf{s}}_{\mathbf{sum}}), & if\ \forall v \in V^s : p(v, s) \leq x\% \\ \infty, & else \end{cases} .$$

With the new cost function f_p, only allocations are taken that do not degrade the performance of the services more than $x\%$.

4.5.2 Weighted Energy- and Performance-Aware Resource Allocation

A further approach can be the combination of the energy-optimal cost function pow_s and the performance-aware cost function P by the assignment of weights. We define a new cost function $f_w : \mathcal{S} \to \mathbb{R}$ and assign it to the servers in $\mathcal{S}$:

$$f_w(s) = w_1 \cdot pow_s(\mathbf{v^s_{sum}}) + w_2 \cdot P(s)$$

The higher the weighting factor of a cost function the more its impact on the determination of the allocation.

5 Algorithms

In this section, three different heuristics are presented that approximate the optimal solution of the energy- and performance-aware virtual machine allocation problem. We also present the NP-hard algorithm that computes the optimal allocation. For small scenarios, the results of the heuristics will be compared to the optimal solution in Section 6.

Some of the following algorithms sort the list of servers and VMs before the execution of the allocation algorithm. The sorting is done based on a sorting factor that is assigned to each server or VM, respectively. The definitions 4 and 5 define the sorting factors for servers and VMs.

Definition 4 (Sorting Factor for Servers) Let $S = \{\mathbf{s_1}, ..., \mathbf{s_n}\}$ be a set of n vectors that represent n corresponding servers. The amount of CPU, RAM, HDD and NIC resources of a server s_i are represented by the 4 components of its representing vector: $s_i = (s_i^{CPU}, s_i^{RAM}, s_i^{HDD}, s_i^{NIC})$. Furthermore, let f_i^{max} be the power consumption of s_i when all its resources are fully utilized. Then the sorting factor s_i^{sort} of the server s_i is given by: $s_i^{sort} = (s_i^{CPU} \cdot w_{CPU} + s_i^{RAM} \cdot w_{RAM} + s_i^{HDD} \cdot w_{HDD} + s_i^{NIC} \cdot w_{NIC})/f_i^{max}$

Thereby, w_{CPU}, w_{RAM}, w_{HDD}, w_{NIC} are weights for the CPU, RAM, HDD and NIC part of the formula. The weights reflect for how much energy consumption a specific resource is responsible. Therefore we choose as weights $w_{CPU} = 4$, $w_{RAM} = 1$, $w_{HDD} = 1$, $w_{NIC} = 1$.

Definition 5 (Sorting Factor for VMs) Let $V = \{v_1, ..., v_m\}$ be a set of m VMs. Furthermore, let v_i^{CPU}, v_i^{RAM}, v_i^{HDD} and v_i^{NIC} denote the required CPU, RAM, HDD and NIC resources of the VM v_i. Then the sorting factor v_i^{sort} of a VM v_i is given by

$$v_i^{sort} = \frac{v_i^{CPU}}{\sum_{j=1}^m v_j^{CPU}} \cdot w_{CPU} + \frac{v_i^{RAM}}{\sum_{j=1}^m v_j^{RAM}} \cdot w_{RAM} + \frac{v_i^{HDD}}{\sum_{j=1}^m v_j^{HDD}} \cdot w_{HDD} + \frac{v_i^{NIC}}{\sum_{j=1}^m v_j^{NIC}} \cdot w_{NIC}$$

Thereby, w_{CPU}, w_{RAM}, w_{HDD}, w_{NIC} are again the weights for the CPU, RAM, HDD and NIC part of the formula.

5.1 Algorithm for Optimal Allocation

The optimal algorithm computes a VM allocation that exactly fulfills all 3 requirements on an energy- and performance-aware virtual machine allocation, as defined in Section 4. However the computation of the optimal solution is NP-hard. Therefore the algorithm for optimal VM allocation can only be applied to small scenarios with few servers and VMs. The algorithm compares the overall power consumption of all possible allocations and chooses the one with the lowest power consumption. The exact operation of the algorithm is given in Algorithm 1.

Algorithm 1 Optimal VM Allocation

```
Input: $serverList$, $vmList$
Output: Optimal VM allocation stored in $optimalAllocation$
  $vm \leftarrow vmList.removeFirst()$
  for all $s \in serverList$ do
    if $vm$ fits in $s$ then
      $s.add(vm)$
      if $vmList = \emptyset$ and $newAllocation.power() < optimalAllocation.power()$ then
        $optimalAllocation := newAllocation$
      end if
    else
      $optimal(serverList, vmList)$
    end if
    $s.remove(vm)$
  end for
  $vmList.add(vm)$
```

The runtime of Algorithm 1 is in $O(m^n)$, since there are m^n different possible allocations of n VMs to m servers.

5.2 Best from Random Allocation

Algorithm 2 computes 1000 random valid allocations that consider VM requirements and do not overload the servers. For each allocation the power consumption is computed. The allocation causing the lowest power consumption is taken as solution. The pseudo code for the BestFromRandom heuristic is given in Algorithm 2.

Algorithm 2 BestFromRandom Heuristic

```
Input: $serverList$, $vmList$
Output: Best VM allocation stored in $bestAllocation$
  $bestAllocation := allocateRandomly()$
  for $i = 1 \rightarrow 1000$ do
    $vmList.shuffle()$
    for all $v \in vmList$ do
      $vmList.remove(v)$
      while $vm$ does not fit in $s$ do
        $s = serverList.getNext()$
        $s.add(vm)$
      end while
    end for
    if $newAllocation.power() < bestAllocation.power()$ then
      $bestAllocation := newAllocation$
    end if
  end for
```

The complexity of the BestFromRandom Heuristic is $O(n \cdot m)$. The cost of $vmList.shuffel()$ (line 5) is $O(n)$. Then the list of VMs is traversed and for each VM a server in $serverList$ has to be found. In the worst case, for each VM the whole $serverList$ has to be traversed. This costs $O(n \cdot m)$. The whole procedure is repeated 1000 times so that the overall complexity is $1000 \cdot (O(n) + O(n \cdot m)) = O(n \cdot m)$.

5.3 Greedy Heuristic

The approach of the Greedy Heuristic is to sort the VMs according their sorting factor that indicates the 'size' of the VM. Size in this context means the resource usage and the contribution to the overall power consumption that are combined in the sorting factor. The VMs are sorted in decreasing order. The 'biggest' VM v is then removed from the list and the increase of the power consumption of each server $s \in serverList$ is calculated in case v would be allocated to s. v is then allocated to the server with the lowest increase of power consumption.

Algorithm 3 Greedy Heuristic

```
Input: $serverList$, $vmList$
Output: Best VM allocation stored in $serverList$
  for all $v \in vmList$ do
    $v.calculateSortingFactor$
  end for
  $v.sortDecreasing()$
  for all $v \in vmList$ do
    for all $s \in serverList$ do
      /* How much does the power consumption of server $s$
      increase? */
      $deltaP_s = s.additionalPower(v)$
    end for
    /* place $v$ on server $s_{min}$ with smallest $deltaP$ */
    $s_{min}.add(v)$
  end for
```

Sorting $vmList$ can be done in $O(n \log n)$. Then the list of VMs has to be traversed in $O(n)$ and for each VM the list of servers has to be traversed too. The cost for this is $O(n \cdot m)$. The overall complexity of the Greedy Heuristic is $O(n \log n) + O(n \cdot m)$. $O(n \log n) \subset O(n \cdot m)$ since m, the number of servers, is usually much greater than $\log n$. Therefore the overall complexity is $2 \cdot O(n \cdot m) = O(n \cdot m)$

5.4 Modified FirstFit Heuristic

The ModifiedFirstFit Heuristic is similar to the Greedy Heuristic. It first sorts the VMs and servers according their sorting factors. The VMs are sorted in decreasing, the servers in increasing order. The first VM in the list is then removed. It is allocated to the first server in the sorted list of the servers where it fits. With other words, the most expensive VM, in terms of resource and power usage, is allocated to the most energy-efficient and 'smallest' server. The algorithm for the Modified-FirstFit Heuristic is given in Algorithm 4

Algorithm 4 ModifiedFirstFit Heuristic

```
Input: $serverList$, $vmList$
Output: Best VM allocation stored in $serverList$
  for all $v \in vmList$ do
    $v.calculateSortingFactor$
  end for
  $v.sortDecreasing()$
  for all $s \in serverList$ do
    $s.calculateSortingFactor$
  end for
  $s.sortIncreasing()$
  for all $v \in vmList$ do
    for all $s \in serverList$ do
      if $v$ fits in server $s$ then
        $s.add(v)$
        break
      end if
    end for
  end for
```

The runtime of the ModifiedFirstFit Heuristic is $O(n \log n) + O(m \log m)$ for the sorting of $vmList$ and $serverList$ and in the worst case $O(n \cdot m)$ for the VM allocation, when for each VM the whole list of servers has to be traversed. The overall complexity is $O(n \cdot m)$.

6 Evaluation

6.1 Evaluation of the Performance Degradation Functions

We evaluate the performance degradation functions of Section 3 in two different evaluations. First, we evaluate the performance degradation function p_{no_virt} that does not consider virtualization. Second, we evaluate p_{virt} which is a modification of p_{no_virt} that considers the overhead caused by encapsulating a service within a VM.

Figure 5 shows the evaluation of the performance degradation function p_{no_virt}. A randomly chosen number of processes with random load generation are run in parallel on a single server (Intel Xeon E5420, 2.5 GHz, 16GB RAM). No virtualization is applied, the processes are ordinary Linux processes. In the figure, the measured and predicted performance degradation of 16 randomly chosen samples is compared. The mean error of the predictions of p_{no_virt} is 4.99% with a variance of 0.56%.

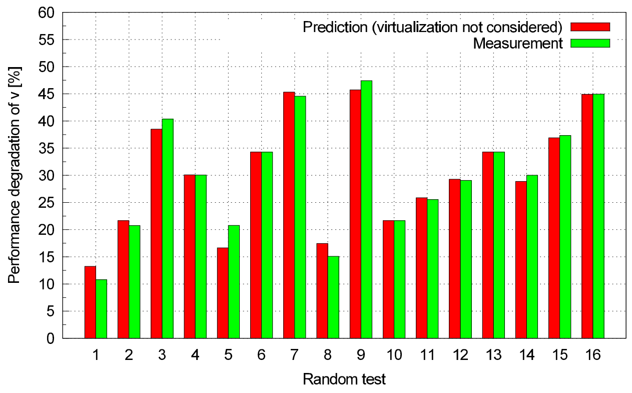

Fig. 5 The measured and predicted performance degradations of v in 16 random measurements (non-virtualized scenario)

Figure 6 shows the evaluation of the performance degradation function p_{virt}. Also in this case, a randomly chosen number of processes with random load generation are consolidated on a single server (Intel Xeon X5650, 2.67 GHz, 20GB RAM). However, this time the processes are encapsulated in a VM so that each process has its own VM. For 12 randomly chosen samples, Figure 6 shows the measured performance degradations and compares them to the results of the prediction functions p_{no_virt} and p_{virt}. Since p_{no_virt} does not consider the overhead caused by the virtualization, it permanently underestimates the performance degradation. It can be seen that the error of p_{no_virt} grows if the measured performance degradation is high. This is usually the case when there are multiple VMs consolidated on the same server or the load of the VMs is high. In both cases the overhead caused by the virtualization grows. On the other hand, p_{virt} extends p_{no_virt} by a correction factor for the virtualization that considers both, the number of VMs and the load generated by the VMs. The mean error of the predictions of p_{virt} is 4.71% with a variance of 0.2%.

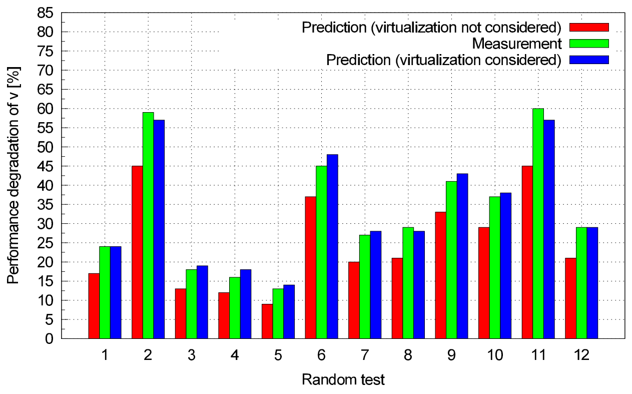

Fig. 6 The measured and predicted performance degradations of v in 12 random measurements (virtualized scenario)

The evaluation shows that the model for the performance degradation of CPU-intensive applications works for both virtualized and non-virtualized environments. Furthermore the evaluation was carried out on servers with different CPU types which affirms the applicability of the model.

6.2 Simulation

To produce meaningful results and to evaluate the algorithms in a larger scale scenario, a discrete event simulation was used. This way, it was possible to test each algorithm under exactly identical conditions. However, to limit simulation complexity, the following assumptions were made:

– The infrastructure uses a shared storage system for VMs, i.e. data that has to be transfered during migrations is limited to the memory size of the VMs.
– The servers are interconnected with Gigabit Ethernet connections.
– Servers are divided into three performance classes: small, medium and large. Each of these classes offers a certain performance level, however each class is in turn subdivided into three energy-efficiency levels: low, medium and high. Energy efficiency of a server is considered higher the closer its power consumption is to energy proportionality, meaning linearly increasing with its load.
– The resource demand of services is limited to CPU and RAM dimensions in the simulation.
– As an abstraction VM memory demands have to be satisfied from physical RAM, virtual memory is not considered.

– VMs are migrated sequentially, virtual machines waiting for migration enter a queue and continue running on their current host until migration is completed.
– VM resource demand is constant during its runtime.

6.3 Evaluated algorithms

Following is a short description of the algorithms that were evaluated in the simulation. For all algorithms, the energy consumption of the infrastructure optimized by the algorithm was compared.

– **Load balancing**
  To acquire a baseline power consumption, the virtual machines were distributed in a load balancing pattern. More detailed, the VMs from the pool were sequentially placed on the server which showed the least utilization after VM placement.
– **Maximum density consolidation**
  This represents a common consolidation strategy, where servers are utilized to the maximum in order to minimize the number of running physical servers. Servers are sorted by their VM hosting capabilities, with the most powerful servers being at the front of the list. After that, for each VM the list of servers is traversed until the first server is found which is able to host the current VM. The VM is then allocated to this server.
– **Exhaustive optimal allocation**
  This algorithm implements a naive optimal allocation by evaluating all possible mappings of virtual machines to physical servers. From all possible mappings, the energy minimal mapping is chosen.
– **Greedy heuristic**
  The greedy heuristic, as described in Algorithm 3, tries to minimize the power consumption increase for each newly allocated VM. The VMs are ordered according to their sorting factor. Then, the increase in power consumption that arises from the simulated allocation of the current VM to each server is evaluated. The VM is then allocated to the server that shows the smallest power consumption increase.
– **Modified first fit**
  This algorithm is an adaption of the first fit decreasing algorithm. First the VMs are ordered in decreasing, the servers in increasing order according to their respective sorting factors. Then, for each VM the servers are traversed in order until the first server which can host the current VM is found. The VM is then allocated to this server.
– **Best from random allocation**
  This algorithm evaluates the power consumption of a fixed number of random VM allocations. In the simulation, 1000 random allocations were performed. In case the energy-minimal of these allocations is more energy saving than the current allocation, it was performed.

6.3.1 Simulation Scenario Setup

The simulation was programmed using the Desmo-J framework[5]. Desmo-J is a Java based discrete event simulation framework. To evaluate the different algorithms, multiple simulation scenarios were set up. Especially since large differences in algorithm runtime have to be expected, the scenarios have to vary in the number of physical servers and VMs. To compare the different heuristics to the optimal solution, a small problem instance was used. Additionally, for the non optimal algorithms, nine larger scale scenarios with 18, 36 and 72 physical servers and 10,20 and 40, 20,40 and 80 and 40,80 and 160 VMs respectively were evaluated. To keep the simulation as generic as possible non homogeneous hardware is assumed, server specifications belong to a class small, medium or large, which differ in the number of CPUs and size of memory. All CPUs and RAM sticks are assumed to be of the same type. Each server class is subdivided into three energy efficiency classes: low, medium and high. These classes mainly differ in their energy proportionality, meaning that the difference of idle power and maximum power is higher. A similar number of all server types is assumed for the simulation. The VM resource demand is randomly assigned, with a uniform distribution in CPU demand (between 0.1 and 4.0 GHz) and RAM demand (between 0.5 and 4.0 GB). The maximum allowed performance degradation of a virtual machine is assigned following a normal distribution with mean 50 and a standard deviation of 25. The random specifications of the VMs are created once for each scenario and then used for the evaluation of all algorithms. The simulated time is set to 24 hours, with the optimizer running every 10 minutes. For server power consumption modeling, the linear, CPU utilization based model proposed by Fan et al. [8] was used.

6.4 Results

Figure 7 shows a comparison of the evaluated algorithms against the optimum allocation, in terms of both energy consumption of the infrastructure and algorithm execution time. It is noteworthy that the numbers in Figure 7 show the results only for a small problem size,

[5] http://desmoj.sourceforge.net/home.html

as an optimal solution was not computable for high numbers of servers or VMs.

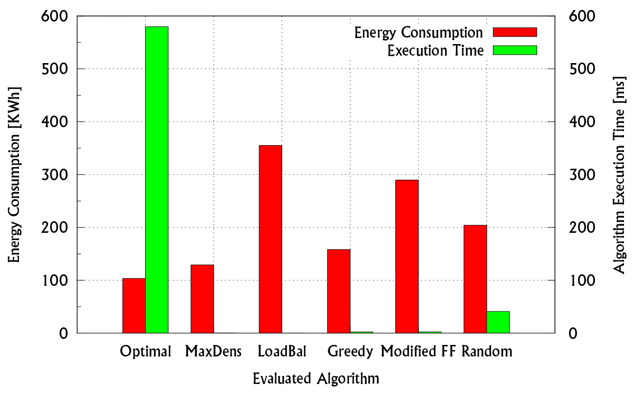

Fig. 7 Algorithm performance compared to optimal solution

Figure 7 shows that in the evaluated simulation only the maximum density consolidation and the greedy heuristic approximate the optimal solution with a quality > 0.5. However, the execution time of the optimal solution is more than 1000% higher than for any of the heuristics. The execution time of the optimal solution algorithm quickly approaches an infeasible duration at small problem instances of around 10 VMs. When averaging the results for bigger problem instances, the overall performance of the algorithms becomes clearer. This is shown in Figure 8.

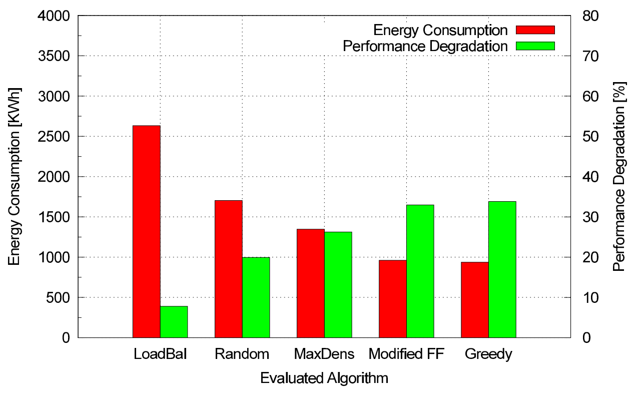

Fig. 8 Average heuristics performance

The results show that both heuristics that explicitly aim at energy savings can perform significantly better than their competitors. On average the greedy heuristic performs best in terms of finding an energy saving allocation. The modified first fit scores nearly similar. Compared to the widely used maximum density consolidation, the greedy heuristic offers an additional energy saving potential of more than 30%. However, as expected the higher energy savings come at the price of higher performance degradation. It has to be mentioned though that all algorithms keep the performance degradation constraints of all virtual machines. Certain algorithms, e.g. Load Balancing, just do not exploit the allowed degradation to its maximum. Using the greedy algorithm as an example, Figure 9 shows the tradeoff between performance and energy savings. Especially in the range from $0 - 50\%$ a relaxation of performance degradation constraints leads to a significant increase in energy savings of more than 50%. A further relaxation does not yield significant increases in energy saving potential.

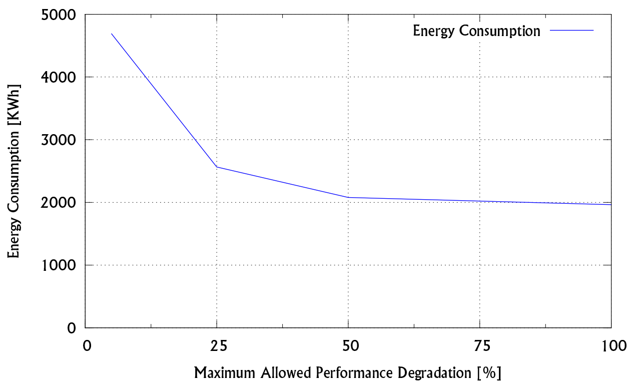

Fig. 9 Energy consumption depending on the maximum allowed performance degradation

7 Conclusion and Future Work

Recent research has often been applying virtualization and consolidation approaches to save energy in data center environments. However, the suggested approaches are often limited to an application in homogeneous server environments. Even in heterogeneous ones, energy savings are usually achieved by minimizing the number of powered-on servers, which is not necessarily an optimal solution with respect to energy consumption. The main drawback of current approaches is the negligence of performance constraints of single services. Performance constraints can be violated when multiple services are consolidated on the same physical server. This paper has introduced a model that addresses the drawbacks of previous approaches. The presented model describes the energy- and performance-aware VM allocation in heterogeneous server environments as a variant of the multidimensional vector packing problem. Traditional vector packing is a well-known method to model task allocation problems. However, in order to apply vector packing to model energy- and performance-aware VM allocation in heterogeneous server environments, the traditional vector packing has been extended by a model for performance degradation of services and a model for server power consumption. In the paper a Greedy heuristic and a Modified First-Fit heuristic have been described which approximate an energy-optimal and performance-aware VM allocation. The presented approach assumes a round robin process scheduler, CPU-intensive services, and response time as performance metric. As future work it would be interesting to analyze the impact of different scheduling strategies on the performance degradation of a service. A further extension would be the development of performance degradation models for different performance metrics. This would allow for using service-dependent performance metrics, e.g., based on service level agreements between data center operators and customers.

8 Acknowledgments

The research leading to these results was supported by the German Federal Government BMBF in the context of the G-Lab_Ener-G project and by the EC's FP7 framework program in the context of the EuroNF Network of Excellence (grant agreement no. 216366).

References

1. AMD: AMD cool'n'quiet technology. http://www.amd.com/us/products/technologies/cool-n-quiet/Pages/cool-n-quiet.aspx. Accessed 15 November 2011
2. Barroso, L.A., Hölzle, U.: The case for energy-proportional computing. IEEE Computer **40**(12), 33–37 (2007)
3. Borgetto, D., Costa, G.D., Pierson, J.M., Sayah, A.: Energy-aware resource allocation. In: GRID, pp. 183–188 (2009)
4. Buyya, R., Beloglazov, A., Abawajy, J.H.: Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. CoRR **abs/1006.0308** (2010)
5. Campegiani, P.: A genetic algorithm to solve the virtual machines resources allocation problem in multi-tier distributed systems. In: In Proceedings of the Second International Workshop On Virtualization Performances: Analysis, Characterization and Tools (VPACT'09) (2009)
6. Cardosa, M., Korupolu, M.R., Singh, A.: Shares and utilities based power consolidation in virtualized server environments. In: Integrated Network Management, pp. 327–334 (2009)
7. Ekker, N., Coughlin, T., Handy, J.: Solid state storage 101: An introduction to solid state storage. https://members.snia.org/apps/group_public/download.php/35796/SSSI%20Wht%20Paper%20Final.pdf (2009). Accessed 15 November 2011
8. Fan, X., Weber, W.D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. In: Proceedings of the 34th annual international symposium on Computer architecture, ISCA '07, pp. 13–23. ACM, New York, NY, USA (2007)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
10. Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation: Advanced configuration and power interface specification. http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf (2010). Accessed 15 November 2011
11. Intel Corporation: Intel 82541ei gigabit ethernet controller. http://download.intel.com/design/network/products/lan/prodbrf/25251103.pdf (2005). Accessed 15 November 2011
12. Khan, S.U., Ardil, C.: Energy efficient resource allocation in distributed computing systems. In: Proceedings of the 2009 International Conference on Distributed, High-Performance and Grid Computing (DHPGC), pp. 667–673 (2009)
13. Khargharia, B., Hariri, S., Szidarovszky, F., Houri, M., El-Rewini, H., Khan, S.U., Ahmad, I., Yousif, M.S., Yousif, M.S.: Autonomic power & performance management for large-scale data centers. In: IPDPS, pp. 1–8 (2007)
14. Kusic, D., Kephart, J.O., Hanson, J.E., Kandasamy, N., Jiang, G.: Power and performance management of virtualized computing environments via lookahead control. Cluster Computing **12**(1), 1–15 (2009)
15. Mark, C.C.T., Niyato, D., Tham, C.K., Tham, C.K.: Evolutionary optimal virtual machine placement and demand forecaster for cloud computing. In: AINA, pp. 348–355 (2011)
16. Meisner, D., Gold, B.T., Wenisch, T.F.: Powernap: eliminating server idle power. In: ASPLOS, pp. 205–216. ACM (2009)
17. NVIDIA Corporation: Introducing hybrid sli technology. http://www.nvidia.com/content/includes/images/us/product_detail/pdf/hybrid_sli_0308.pdf (2008). Accessed 15 November 2011
18. Pallipadi, V.: Enhanced intel speedstep technology and demand-based switching on linux. http://software.intel.com/en-us/articles/enhanced-intel-speedstepr-technology-and-demand-based-switching-on-linux/ (2009). Accessed 15 November 2011
19. Srikantaiah, S., Kansal, A., Zhao, F.: Energy aware consolidation for cloud computing. In: Proceedings of the 2008 conference on Power aware computing and systems, HotPower'08, pp. 10–10. USENIX Association, Berkeley, CA, USA (2008)
20. Subramanian, C., Vasan, A., Sivasubramaniam, A.: Reducing data center power with server consolidation: Approximation and evaluation. In: High Performance Computing (HiPC), 2010 International Conference on, pp. 1 –10 (2010)

9 Appendix

Table 1 gives an overview on the used symbols and describes them.

Symbol	Description
$\mathcal{V}$	Set of virtualized services
v	Virtualized service
$\mathcal{S}$	Set of servers
$v^{CPU}(s)$	CPU-cycles required by v when allocated to s
$v^{RAM}(s)$	Amount of RAM required by v when allocated to s
$v^{HDD}(s)$	I/O rate required by v when allocated to s
$v^{NIC}(s)$	Bandwidth required by v when allocated to s
$\mathbf{v}$	4-dimensional vector representing a virtualized service
$\mathbf{s}$	4-dimensional hyper bin representing a server
α	Function assigning a server to a VM
a	Function assigning a bin to a vector
S^a	Set of non-empty bins regarding the assignment with a
$V^{\mathbf{s}}$	Set of vectors assigned to $\mathbf{s}$
$\mathbf{v}^{\mathbf{s}}_{\mathbf{sum}}$	Sum of the vectors assigned to $\mathbf{s}$
$f_{\mathbf{s}}$	Cost function assigned to bin $\mathbf{s}$
f_s	Power consumption function of server s
pow_s	Power consumption function of the server s
f^a_{mig}	Function computing the power consumption overhead for the migration of a VM under a
v_{size}	Amount of data that has to be migrated when v gets reallocated
d	Available data rate for migration
T	Time interval between two reallocations
I	Set of items
$l(i), i \in I$	Size of the item i
B	Set of bins with dimension 1
l_{bin}	Size of the bins in B
i^b	Item that is packed into the bin b
$p(v, s)$	Function estimating the performance degradation of a VM v when it is running on a server s
$v_{\#}$	Number of VMs consolidated on the same CPU
s^{CPU}	CPU load of the server s
v^{CPU}	The required CPU cycles of v

Table 1 List of symbols and their descriptions