Information Systems 52 (2015) 83–95

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

More than bin packing: Dynamic resource allocation strategies in cloud data centers

Andreas Wolke *, Boldbaatar Tsend-Ayush, Carl Pfeiffer, Martin Bichler

Department of Informatics, Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany

ARTICLE INFO

Article history:
Received 27 February 2015
Received in revised form
13 March 2015
Accepted 14 March 2015
Recommended by: D. Shasha
Available online 3 April 2015

Keywords:
Cloud computing
Capacity planning
Resource allocation

ABSTRACT

Resource allocation strategies in virtualized data centers have received considerable attention recently as they can have substantial impact on the energy efficiency of a data center. This led to new decision and control strategies with significant managerial impact for IT service providers. We focus on dynamic environments where virtual machines need to be allocated and deallocated to servers over time. Simple bin packing heuristics have been analyzed and used to place virtual machines upon arrival. However, these placement heuristics can lead to suboptimal server utilization, because they cannot consider virtual machines, which arrive in the future. We ran extensive lab experiments and simulations with different controllers and different workloads to understand which control strategies achieve high levels of energy efficiency in different workload environments. We found that combinations of placement controllers and periodic reallocations achieve the highest energy efficiency subject to predefined service levels. While the type of placement heuristic had little impact on the average server demand, the type of virtual machine resource demand estimator used for the placement decisions had a significant impact on the overall energy efficiency.

1. Introduction

Modern data centers are increasingly using virtualization technology and provide virtual machines (VMs) for their customers rather than physical servers. Actually, IT service managers worldwide ranked virtualization and server consolidation as one of their top priorities in the recent years [1,2]. We focus on IaaS (Infrastructure-as-a-Service) as the most basic cloud-service model, in which IT service providers offer computers or virtual machines as a service to their customers. In virtualized data centers of IaaS providers, many VMs are allocated on a single server where dedicated servers were required before. VMs can be allocated and deallocated within seconds using nowadays migration technology in VM hypervisor software.

With the adoption of virtualization technology the demand for new physical servers decreased while the demand for VMs has grown considerably. At the same time, server administration costs increased as many virtual machines need to be managed [3]. This leads to new resource allocation problems, which require decision support and ultimately automation to bring down administration costs and achieve high energy efficiency. While there has been a substantial literature on capacity planning and business models [4–6], the literature on dynamic resource allocation in clouds in combination with incoming and outgoing VMs has received less attention. In particular, there is little experimental literature comparing different methods with respect to their energy efficiency in a lab environment, despite their potentially high impact on the total cost of IT service provisioning [4,7]. Most research is based on discrete event simulations only.

* Corresponding author.
E-mail addresses: wolke@tum.de (A. Wolke), ayush@cs.tum.edu (B. Tsend-Ayush), ce07c3@gmail.com (C. Pfeiffer), bichler@in.tum.de (M. Bichler).

http://dx.doi.org/10.1016/j.is.2015.03.003
0306-4379/

Today, *cloud management tools* such as OpenStack[1] and Eucalyptus[2] are used in many IaaS cloud environments for resource allocation. Incoming VMs are placed on servers via simple bin packing heuristics and remain there until they are deallocated. Such heuristics have also been analyzed in the academic literature (see Section 2). Because such *placement heuristics* do not consider future allocation and deallocation requests, servers might be underutilized and operated at a low energy efficiency.

VM live-migrations allow to move VMs between servers during runtime. The technology has matured to a state where it is a viable option not only for emergency situations [8], but also for routine resource allocation tasks. At this point, none of the existing cloud management tools uses reallocation after the initial placement in their resource allocation strategies. Rather they rely on the original placement decisions using simple bin packing algorithms.

In contrast to previous research we contribute results of an extensive set of lab experiments covering an effective runtime of more than 100 days. Lab experiments are expensive as they require a dedicated infrastructure to run controlled experiments including specific VM management, monitoring, and metering tools. Also, a single experiment takes several hours. However, the experiments provide us with adequate estimates for system latencies and migration costs, which we use afterwards for the simulation of larger infrastructures. The parameter estimates from our lab experiments lead to high external validity of our simulations, which would be difficult to parametrize otherwise.

Technically, our lab infrastructure closely resembles a private IaaS cloud that one would find in small to medium sized enterprise environments. In addition a proprietary cloud manager and monitoring solutions were developed in order run controlled experiments that allow a proper analysis. The very same controller software was used for the simulations. Workload traces were simulated from log data in real-world data centers from industry partners. In addition, we used wide-spread benchmark applications such as SPECjEnterprise[3] to emulate real enterprise applications.

There are many possibilities how reallocation controllers can be implemented and combined with placement algorithms. We systematically analyzed a large variety of resource allocation strategies in different workload environments in simulations as a first step. In particular, we analyzed different resource allocation strategies, simple *placement controllers* based on bin packing, and advanced controllers combining placement and reallocation algorithms for server consolidation and high energy efficiency. We will refer to such combined resource allocation strategies as *reallocation controllers*. While placement controllers have been the focus of prior literature, the efficiency gains of reallocation controllers is the focus of this study. In a second step, we compared selected placement and reallocation controllers in lab experiments on a physical data center infrastructure. These lab experiments allowed us to derive results on various efficiency and quality-of-service metrics with high external validity, because we do not need to make estimates about migration overheads or system latencies. The scope and scale of the experiments is beyond what has been reported in the literature, and the results provide tangible guidelines for data center managers.

We found that reallocation controllers have a substantial positive impact on the energy efficiency of a data center. We achieved the highest energy efficiency with placement controllers computing a very dense allocation from the start. In case of overload in such allocations, VMs were migrated later. If the placement decisions were based on the actual demand on a server rather than the reservations for the VMs on a server, the density of the packing could be increased and therefore also the energy efficiency of the servers. Interestingly, the type of bin packing heuristic for the initial placement had little impact on the energy efficiency. Periodic reallocation by reallocation controllers, however, had substantial impact on the energy efficiency overall.

In Section 2, we discuss related literature. Section 3 introduces the experimental setup, while Section 4 describes the results. Finally, Section 5 provides a summary and conclusions.

2. Related work

Our research draws on different literature streams. First, we will introduce relevant literature on bin packing problems, as this is the fundamental problem for the VM placement. Next we will discuss the literature on static allocation and on dynamic allocation algorithms, where VMs arrive and depart continuously. There is significant research on power distribution in cloud data centers [9], which we consider complementary to our work.

2.1. Bin packing

The bin packing problem has been subject of extensive theoretical research. A wide selection of heuristics based on different packing strategies exists [10]. The bin packing problem that needs to be solved for VM placement decisions can be solved by well known heuristics like Next-Fit, First-Fit, Best-Fit, or many others. Vector bin packing algorithms can be used if multiple resources such as CPU and memory need to be considered over time [4].

Coffman [11] analyzed dynamic bin packing where new items arrive and existing ones depart. He proves a competitive ratio that compares fully dynamic on-line packing heuristics with optimal solutions for the offline problem. The competitive ratio was improved successively, most recently by Wong et al. [12] to a value of 8/3. Of course, competitive ratios are worst case bounds and average results are typically better than this.

Finally, Ivkovic et al. [13] introduced fully dynamic bin packing where items can be reallocated *before* placing a new VM on a server. Their algorithm is 5/4-competitive. This theoretical analysis indicates that there are gains from reallocation from the worst-case perspective assumed in the theoretical literature. It is interesting to understand, if reallocation has a positive impact on the average utilization in a data center and the order of magnitude of these differences on average in a controlled lab experiment.

[1] https://www.openstack.org/.
[2] https://www.eucalyptus.com.
[3] http://www.spec.org/jEnterprise2010/.

2.2. Static and dynamic allocation

A number of papers have addressed the problem of server consolidation where a set of VMs is allocated to a minimal set of servers in order to increase resource utilization. Sometimes VMs can be migrated between servers over time, but the set of VMs is stable as it is often the case in corporate data centers.

Some solutions to these static allocation problems leverage bin-packing heuristics, others use mixed integer programming (MIP) to find optimal or near-optimal solutions [4,14,6]. Bobroff et al. [15] predict future resource utilization by autoregressive forecasting models which are used by bin-packing heuristics. pMapper [16] computes a new allocation based on actual resource utilization measurements and triggers iterative VM migrations to change the allocation considering costs entailed with VM migrations. Gmach et al. [5] proposed a fuzzy logic based dynamic controller that load balances servers. Thresholds on memory and CPU are used to detect overloaded servers. The evaluation is based on a simulation.

So far, only a few authors evaluated their resource allocation algorithms in lab experiments. Wood et al. [17] proposed Sandpiper as a dynamic VM allocation approach for load balancing VMs rather than minimizing total server uptime. Migrations are triggered if server overload thresholds are reached. For our experiments, we adapted Sandpaper in one of the treatments such that it minimizes total server uptime in addition to load balancing as described in Section 3.1.2. Sandpiper was evaluated on a hardware infrastructure, but not compared to alternative strategies regarding energy efficiency.

vGreen [18] is another dynamic allocation approach that leverages different controllers depending on the resource considered. VMs on servers with low utilization are moved to other servers heuristically such that some servers can be turned off. vGreen was also evaluated in a small test bed with 2 physical servers.

2.3. Fully dynamic allocation

The work discussed so far did not consider fully dynamic scenarios where VMs arrive and depart over time, as it is the focus of this paper. A few recent papers have focused on related problems.

Calcavecchia et al. [19] proposed a two stage VM placement strategy called backward speculative planning. Demand risk is a metric characterizing the unfulfilled demand of VMs running on a server. Incoming VMs are placed by a Decreasing Best-Fit strategy on the server providing the minimal demand risk after placing the VM, assuming that the new VM was fully utilized in the past. VM migrations are only triggered if unfulfilled demand exists and a threshold on the number of migrations for a period is not exceeded.

Mills et al. [20] simulated initial placement strategies in an IaaS environment without live-migrating VMs. For each incoming VM, a cluster and a server are selected in this order. Three cluster selection algorithms and 6 server selection heuristics were evaluated in 32 workload scenarios. Workloads were generated randomly based on a configuration of 6 parameters.

Both papers analyze VM resource allocation strategies in fully dynamic scenarios. In addition, simple placement policies such as the minimization of migrations or policies allowing for growth have been compared in simulations [21].While these papers are important first steps, we extended these studies in important ways.

First, we analyze different strategies not only in simulations, but also in lab experiments. This is expensive, but important for the external validity of such experiments. Our workload data used in the simulations and experiments is based on monitoring data of actual enterprise data centers, and VM arrival and departure rates are based on statistics described by Peng et al. [22] based on multiple data centers operated by IBM.

Second, we significantly increased the scale of the studies and analyzed a large number of VM placement and reallocation strategies separately and in combination to take interaction effects into account. This allows us to make recommendations for data center managers based on a large number of treatment combinations. In particular, we can shed light on the benefits of reallocation in fully dynamic environments. While the number of possible heuristics to allocate resources dynamically is huge, we argue that we have considered the most prominent approaches discussed to adequately reflect the state-of-the-art as well as new and promising strategies for reallocation controllers.

3. Experimental setup

In what follows, we describe the experimental setup and the technical infrastructure used for the experiments.

3.1. Technical environment

In our infrastructure VMs are created and removed automatically by means of a network service. New VMs are allocated and existing ones are removed continuously. VM lifetimes vary between a couple of minutes and months. We do not assume prior knowledge about the type or applications running within VMs. An allocation request includes the VM resource reservation and a reference to a network attached disk. A *reservation* describes the size of a VM in terms of allocated CPU cores, memory, network, and disk capacity.

For a new VM allocation request, the cloud management tool has to decide on which server the new VM should be placed. This decision is taken by a *placement controller*. Already running VMs might get migrated to another server. Migrations are triggered by *reallocation controllers* that either run in regular intervals or closely monitor the infrastructure and respond to inefficiencies of the current allocation.

In this paper, we attempt to analyze a wide variety of controllers for placement and reallocation. The functionality of the controllers under consideration is outlined in the following.

3.1.1. Placement controllers

Let us first introduce two ways how parameters for placement controllers can be computed. We describe *residual capacity* as the amount of unused resources on a server. It is

expressed as a vector. Each component of the vector represents a resource like CPU or memory. There are two types of residual capacity. *Reservation-based residual capacity* subtracts VM resource reservations from a server's total capacity. *Demand-based residual capacity* is the measured amount of free resources on a server. Both can be used for placement controllers. Interestingly, all available cloud management tools that we are aware of (including OpenStack and Eucalyptus) use a reservation-based allocation, which guarantees each VM its resource reservation. Many VMs are not utilized to 100% of the capacity that was reserved leading to underutilized servers. Demand-based allocation leverages information about the actual VM resource utilization and increases server utilization by overbooking. However, they can only be used in conjunction with reallocation controllers as otherwise servers might get overloaded. Reallocation controllers mitigate overloaded servers using VM migrations.

Any-Fit placement controllers including First-Fit, Best-Fit, Worst-Fit, and Next-Fit assign an arriving VM* to a server [10]. These are regularly used in cloud management software and based on established and simple bin packing algorithms [23]; First-Fit iterates over the decreasingly sorted server list by their load and picks the first one that fits the VM* reservation. Best-Fit computes the delta of residual capacity and VM* reservation vector norms and picks the server with minimal difference that fulfills the reservation. Worst-Fit works the same way but selects the server with the maximum vector norm difference. Next-Fit holds a pointer on a server. Each incoming VM is placed on this server as long as its residual capacity allows for the VM*'s reservation. Otherwise, the pointer is moved to the next server in a round-robin fashion until a feasible server is found.

on-line?

first是第一个，next是轮着开始

Apart from these standard algorithms, there have been specific proposals for VM placement in the literature. Dot-Product [24] is a First-Fit-Decreasing approach where server weight is calculated by the dot product of server residual capacity $\hat{s}^{-1}$, and the reservation $\dot{s}$ for *VM** as $\hat{s}^{-1} \cdot \dot{s}$. Similarly, cosine is also a First-Fit-Decreasing algorithm [25] that uses the cosine as a weight function: $\cos(\hat{s}^{-1}, \dot{s})$. L2 [24] leverages First-Fit-Increasing with the difference of the vector norms as a weight $\|\hat{s}^{-1}\| - \|\dot{s}\|$.

3.1.2. Reallocation controllers

Reallocation controllers are executed regularly and trigger VM migrations in order to re-optimize the allocation. We found two approaches in the literature to compute a schedule of reallocations over time. DSAPP describes an optimization model to compute an optimal schedule of reallocations over time and it was introduced by [4,26]. Alternatively, KMControl and TControl are two heuristics proposed by [17] for the Sandpiper system. In the following, we will briefly describe these controllers such that the paper is self-contained, but refer the reader to the original papers for details.

DSAPP: The DSAPP controller gets executed every 60 s. Each time it recomputes the allocation of VMs to servers according to the optimization problem shown in Eq. (1). Let us discuss the underlying mixed integer program used in the controller.

Suppose we are given a set of servers $i \in I$ and VMs $j \in J$. A server's size is denoted by $\hat{s}_i$ describing its resource capacity, e.g. CPU or available memory. The total planning horizon is divided into two discrete periods $t \in \{1,2\}$, the current one x_{ij1} and the upcoming one x_{ij2}. Values for x_{ij1} are passed as a parameter to the model. $y_i \in \{0,1\}$ tells whether a server i is active in period 2. $\dot{u}_{j2}$ describes the expected CPU utilization of VM j during period 2. The allocation matrix x_{ijt} of period t indicates whether VM j is assigned to server i. Migrations of VMs from period 1 to 2 are indicated by slack variables z_{ij}^- for outgoing and z_{ij}^+ for incoming ones. The objective function minimizes the sum of total server operation costs $\hat{c}$ and migration costs c_j^-, c_j^+. The first constraint makes sure that each VM is allocated in the next period. The second constraint enforces upper bounds on the capacity of each server. Constraints three and four set the variables z_{ij}^+ and z_{ij}^- used in the objective function if a VM is incoming or outgoing:

应该没有考虑一台一台的过程

$$\min \sum_{i=1}^{I} \left(\hat{c} y_{i2} + \sum_{j}^{J} c_j^+ z_{ij}^+ + \sum_{j}^{J} c_j^- z_{ij}^- \right)$$

s.t.

$$\sum_{i=1}^{I} x_{ij2} = 1, \quad \forall j \in J$$

$$\sum_{j=1}^{J} \dot{u}_{j2} x_{ij2} \le \hat{s}_i y_i, \quad \forall i \in I$$

$$-x_{ij1} + x_{ij2} - z_{ij}^+ \le 0 \quad \forall i \in I, \forall j \in J$$

$$x_{ij1} - x_{ij2} - z_{ij}^- \le 0 \quad \forall i \in I, \forall j \in J$$

$$y_i, x_{ijt}, z_{ij}^- \in \{0,1\}, \quad \forall i \in I, \forall j \in J, \forall t \in \{1,2\} \tag{1}$$

After each execution, the new allocation x_{ij2} is implemented and the migrations are triggered. A migration scheduler ensures that each server is running only one migration in parallel, either an outgoing or an incoming one. As many live-migrations as possible are executed in parallel without overloading servers or their network interfaces.

For parametrization, server operation costs and migration costs are estimated by execution period and average migration duration. In our lab experiments, the execution period was 60 s while live-migrations took 25 s on average which provides values for the parameters $\hat{c} = 60$ and $c_j^+ = c_j^- = 12.5$ in the objective function.

KMControl and TControl: The controllers used in Sandpiper [17] provide an alternative heuristic to recompute the allocations in order to balance load across servers. We extend these controllers in the following way.

The controller is executed periodically every 5 min. There are three phases: (1) detect and mark overloaded and underloaded servers; (2) compute server and VM load rankings; (3) mitigate overloaded and underloaded servers by migrations.

KMControl and TControl differ in phase one: KMControl checks if M out of the last K server CPU utilization measurements $\hat{u}_{it}$ are above an threshold T_o or below T_u to mark a server i as overloaded or underloaded. TControl is based on a single-sided t-test that compares the mean of

the M recent utilization measurements starting at time t against T_o and T_u: $T_o \leq (1/M)\sum \hat{u}_{i(t-M+1)}, .., \hat{u}_{it}\} \leq T_u$. If H_0 is rejected at a p-level of 0.001, a significant difference is assumed and a server is either marked as overloaded or underloaded.

Phase two computes a volume (VOL) and a volume-size ratio (VSR) (Eq. (2)) for each server and VM according to [17] with u^r denoting the utilization of resource r and s^{mem} the memory capacity of a server or VM individually. Both computations are performed for physical servers and VMs. Servers are sorted by VOL in decreasing order so that servers with a high resource utilization are found at the top:

$$\mathrm{VOL} = \frac{1}{\prod_{\forall r}(1-u^r)} \tag{2}$$

$$\mathrm{VSR} = \frac{\mathrm{VOL}}{s^{mem}} \tag{3}$$

Phase three triggers migrations such that underloaded servers are emptied and overloaded ones are relieved. VMs running on an overloaded or underloaded server are sorted by their VSR in a decreasing order. VM migration overhead depends on the combination of memory size and utilization. VSR puts memory size into perspective of utilization. To reduce migration costs, VMs with a low memory size compared to utilization are considered first for migration.

3.2. Experimental design

For our experiments we generated schedules of VM arrivals and departures that we could evaluate with different controllers. This allows for a fair comparison of different controller combinations. A schedule defines arrival and departure times as well as VM reservations (illustrated in Fig. 1).

There are many possible schedules that one could explore in experiments. A single lab experiment takes 13–15 h. Due to time restrictions a large number of schedules was analyzed by simulations first (see http://cloudcontrol.in.tum.de/) and a subset was then analyzed in the lab. These schedules vary lifetime, inter-arrival time, and resource reservations of VMs.

Inter-arrival times were taken from Peng et al. [22], who published cumulative density functions (CDF) for inter-arrival times of VMs. Data originated from real world measurements in enterprise data centers operated by IBM. Our schedules leverage two CDFs, one with short (A1) and one with longer inter-arrival times (A2).

Lifetimes are based on two CDFs as well. L1 is a CDF of VM life-time over all data centers, L2 is a mixture of CDFs for unpopular, average, and popular VMs with probabilities (0.2, 0.6, 0.2). Both, VM inter-arrival- and life-times are scaled by factor 30 to keep the total time for a lab experiment below 15 h. Table 1 provides an overview of the schedule configurations in our experiments.

Five schedule instances were generated for each schedule configuration shown in Table 1. A schedule runs up to 20 VMs in parallel. Three VM sizes were used in the experiments: Small VMs (S) are configured with 1 vCPU core and 2 GByte of memory. Medium sized VMs (M) have

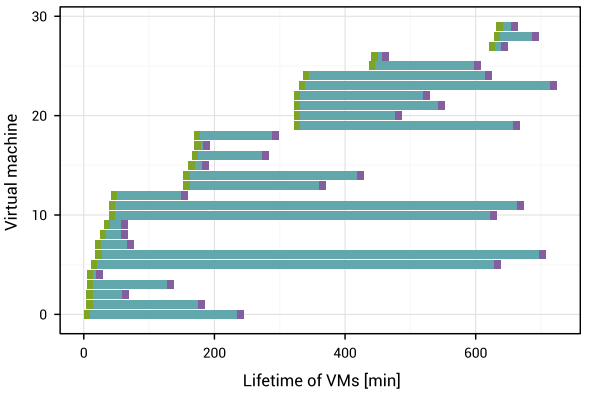

Fig. 1. Sample of a VM arrival and departure schedule.

Table 1
Five schedule instances are generated for each of the four schedule configurations.

Schedule config.	Arrival time	Lifetime
1	A1	L1
2	A1	L2
3	A2	L1
4	A2	L2

2 vCPU cores and 4 GByte of memory. Large ones (L) are assigned 3 vCPU cores and 6 GByte of memory. VM sizes are picked based on probabilities $(S, M, L) = (0.6, 0.3, 0.1)$ for each new VM in a schedule.

While the demand patterns for our VMs are representative and based on real-world demand traces, there can also be VMs with specific demand patterns. In particular, if a VM has regular demand peaks such as batch jobs that are executed every 15 min, this can trigger frequent migrations. Such VMs should be treated differently and allocated to dedicated servers with enough capacity to handle these demand peaks without migrations. Monitoring systems are able to detect such specific demand patterns. These topics are outside the scope of our analysis. For long-running VMs where the controller can learn regular demand patterns, algorithms have been developed to deal with regular demand patterns [6].

3.3. Hardware infrastructure

The architecture of our hardware infrastructure is shown in Fig. 2. It consists of six identical servers and 90 VMs of three sizes described in the previous section. Fedora Linux 16 is used as operating system with KVM as hypervisor. Each server is equipped with a single Intel Quad CPU Q9550 2.66 GHz, 16 GByte memory, a single 10k disk and four 1GBit network interfaces.

The VM disk files are located on two separate NFS storage servers as qcow2 files. The first one is equipped with an Intel Xeon E5405 CPU, 16 GByte memory and three 1GBit network interfaces in a 802.3ad LACP bond. The second storage server has a Intel Xeon E5620 CPU, 16 GByte memory and three GBit network interfaces in a LACP bond. Disks are set up in a RAID 10 configuration.

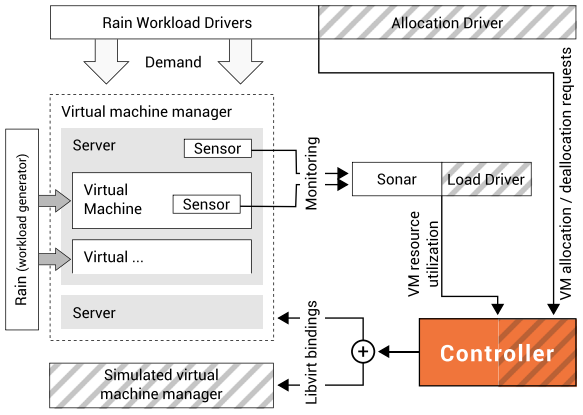

Fig. 2. IaaS environment that supports simulations and experiments.

Both, the network and storage infrastructure had sufficient capacity such that they did not result in bottlenecks.

A Glassfish[4] application server with the SPECjEnterprise2010[5] (SPECj) application and a MySQL database server were installed on each VM. SPECj was chosen because it is widely used in industry to benchmark enterprise application servers. It is designed to generate a workload on the underlying hardware and software that is very similar to the one experienced in real world business applications.

Two additional servers are used as workload drivers. Each one is equipped with an Intel Core 2 Quad Q9400 CPU with 12 GByte main memory and two 1GBit network interfaces in an LACP bond. A modified version of the Rain[6] workload framework is used to simulate varying workload scenarios.

Rain simulates a varying amount of users over time. This user demand is specified by a set of 32 time series that describe the workload for a given time in values $0 \le x_t \le 1$. Depending on the VM size, the time series is multiplied with 100 (S), 150 (M), and 200 (L). The resulting time series describes the number of users to simulate by Rain over time on a single VM.

A monitoring system called Sonar[7] captured relevant resource utilizations in three second intervals of servers and VMs. All relevant metrics such as CPU, memory, disk and network utilization are monitored. Rain drivers reported three second averages of the response time for each service individually. Sonar allows a complete replication of an experiment for analytical purposes.

3.4. Focus variables of the experiment

Our primary objective is to minimize the total server operation hours subject to a service quality above 99%. In the following, we discuss how we measure server operation hours and server demand of an experiment. Assume a list of VM allocation and deallocation requests $L = (q_1, \ldots, q_t, \ldots, q_\tau)$ with $q_t = (a_t, n_t)$ describes the arrival time a_t and the number of running servers n_t. Each request can affect n_t as shown in Fig. 3. Besides that, VM migrations might also affect n_t. In this case, requests are logged at the beginning and at the end of each live-migration to ensure that the number of running servers is correct during migrations.

Total server operation hours (OH) is the area under the curve and average server demand ($\overline{\text{SD}}$) is the server demand divided by operation time as shown in the following equation:

$$\text{OH} = \sum_{t=1}^{\tau-1} n_t(a_{t+1} - a_t)$$
$$\overline{\text{SD}} = \frac{\text{OH}}{a_\tau - a_1} \tag{4}$$

CPU utilization serves as a metric for a server's energy consumption that can be predicted by Eq. (5), $\hat{u} \in [0, 1]$ being the CPU utilization, and P the energy demand in watts [27]:

$$P_{idle} + \hat{u}(P_{busy} - P_{idle}) \tag{5}$$

Tests on one of our servers (2 Intel Xeon CPUs, 64 GB RAM, 6 disks, 2 PSUs) yielded power consumption values for $P_{idle} = 160$ W when idling and $P_{busy} = 270$ W when the server was at 100% CPU utilization. Based on Eq. (5), energy consumption at $\hat{u} = 0.30$ is 193 W. Consolidating two such servers to a single one with an aggregated load of $\hat{u} = 0.60$ would yield energy savings of forty percent (41%), and energy consumption drops from 386 to 226 W.

Fan [27] found that for CPU intensive workloads, dynamic voltage scaling (DVS) can decrease a server's energy demand by more than 20% depending on the application and the aggressivity of the algorithm. For our scenario, we assume an aggressive DVS configuration with 50% reduction on CPU's energy demand (see Equation 5 [27]). Both servers with $\hat{u} = 0.3$ will now consume 176 W each, a saving of 8% due to DVS. DVS cannot be applied to the consolidated server as its utilization is above 50%. Still, consolidating those two servers with active DVS to a single one without DVS reduces energy consumption by 36%. Hence, energy savings achieved by consolidation of VMs to a lower number of servers are considerable, even if DVS is enabled.

4. Results

The experiments compare different combinations of placement and reallocation controllers. Due to the large number of possible controller combinations one can analyze, we first conducted simulations and ranked them based on core metrics: migrations, average, and maximum server demand. Subsequent experiments in the lab were then conducted on the most promising controllers.

4.1. Simulation study

Simulations were conducted for all combinations of 5 allocation controllers with their demand- and reservation-based implementation as well as 4 reallocation controllers including scenarios without reallocation.

[4] http://glassfish.java.net/
[5] http://www.spec.org/jEnterprise2010/
[6] https://github.com/yungsters/rain-workload-toolkit
[7] https://github.com/jacksonicson/sonar

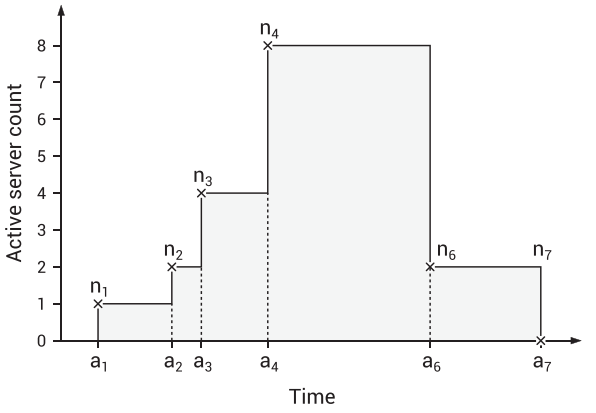

Fig. 3. A datapoint is recorded with each event q_t at time a_t the server demand n_t changes.

A simulation was conducted for each of the 20 schedules described in Section 3.2. The simulation framework and the schedules used in these simulations are described in http://cloudcontrol.in.tum.de/.

Controllers are named by an approach similar to the Kendall notation with three variables separated by a slash, e.g. D/FF/KM. The first element declares if a demand (D) or reservation (R) based placement controller was used. The placement controller used is described in the second element with FF=First-Fit, BF=Best-Fit, WF=Worst-Fit, DP=Dot-Product, L2=L2. Additionally, a reservation based RD=Random controller was tested for control purpose, resulting in 11 combinations of demand and reservation based placement controllers. The dynamic controller is described in the last element with KM=KMControl, TC=TControl, DP=DSAP+, and – if no dynamic controller was used. In total, 44 controller combinations were evaluated, considering this 4 dynamic controllers.

Based on the simulations, controllers are ranked using the MRR (Mean Reciprocal Rank) metric. An MRR rank is computed by Eq. (6). Larger values indicate better rankings. MRR was used because it is not entirely fair to sort controllers based on their average server demands or normalized values for average and maximum server demand. Normalizing migrations is not an option as there are no upper bounds on migrations. MRR ranks controllers individually for each metric and aggregates these into an overall ranking:

$$\mathrm{MRR}(c) = \frac{1}{|P| \cdot |Q|} \sum_{p \in P} \sum_{q \in Q} \frac{1}{R_{cpq}} \tag{6}$$

For each controller $c \in C$ and schedule instance $q \in Q$, three ranking values $p \in P$ exist: migrations (MG), average ($\overline{\mathrm{SD}}$), and maximum server demand ($\lceil \mathrm{SD} \rceil$).

Rankings R_{cpq} are calculated for each metric and schedule separately. For each schedule instance, controllers are sorted in increasing order by a metric (MG, $\overline{\mathrm{SD}}$, $\lceil \mathrm{SD} \rceil$). A controller's list position gives its ranking. In total, each controller is assigned $|P| \cdot |Q|$ rankings.

Controllers are sorted by the MRR ranking over all metrics in Table 2. Some combined controllers using placement and dynamic controllers seem to be equally good than controllers without dynamic strategy, still pure static controllers outperformed all dynamic controllers by the MRR ranking. Obviously, placement-only controllers do not trigger VM migrations. This is an advantage in the combined ranking as they get the best possible score on the migrations metric. Despite this advantage, they often performed worse than combinations of placement and dynamic controller for the MRR rankings $\overline{\mathrm{SD}}$ and $\lceil \mathrm{SD} \rceil$.

Fig. 4 shows a heatmap of the average server demand for each controller and schedule. Controllers are sorted by their MRR ranking on average server demand only, not considering other metrics as for Table 2. Darker areas indicate high values and bright ones low values.

In the heatmap the controllers can be clustered following the type of controller combination. Controllers without dynamic strategy performed worst and are found on the left-hand side (Reservation Static). Demand-based controllers outperformed reservation-based ones. Controllers leveraging some kind of dynamic strategy usually delivered a higher allocation density. Again, combined controllers that leverage demand-based placement controllers outperformed reservation-based ones (Reservation + Dynamic and Mostly Reservation + Dynamic vs. Demand + Dynamic).

Reservation-based controllers performed worst because they are most conservative. Reservations do not reflect the actual resource demand of a VM. Adding a dynamic controller overall improved allocation density. Using a demand-based instead of a reservation-based controller takes advantage of the actual resource demand and achieves a denser allocation in the first place. Again, adding a dynamic controller improved allocation density. This combination performed even slightly better as a combination of reservation-based and dynamic controllers. Reasons are, demand-based controllers achieve a denser allocation right after placement and they achieve a denser allocation faster than reservation-based ones due to less VM migrations.

With respect to VM migrations, controllers without dynamic strategy did not trigger any migrations and performed best. Demand-based controllers consistently triggered less migrations than reservation-based ones. Demand-based controller combinations outperformed reservation-based ones for the same reason as before. Leveraging VM demands leads to a denser allocation in the first place. Fewer or even zero migrations are required to establish a dense allocation after allocating a new VM.

Table 3 summarizes the average server demand and number of VM migrations clustered by the controller type. This provides additional evidence that demand-based placement controllers lead to a superior VM allocation in contrast to reservation-based ones, and that combining them with dynamic controllers reduces the number of migrations substantially in contrast to reservation based placement controller combinations.

Overall, simulations suggest that combinations of demand-based placement with dynamic controllers are most efficient.

4.2. Controller selection for lab experiments

We perform lab experiments in order to understand if the main results of the simulations carry over. In addition, the lab experiments provide information about response times and

Table 2
$\overline{MRR}$ – average MRR ranking of all metric MRR rankings (migrations, average, and maximum server demand), $\overline{SD}$ – average server demand, $\lceil SD \rceil$ – maximum server demand, $\overline{MG}$ – average number of VM migrations. Standard deviation of average simulation results is reported in parentheses.

	MRR rankings				Average simulation results		
Controller	$\overline{MRR}$	$\overline{SD}$	$\lceil SD \rceil$	$\overline{MG}$	$\overline{SD}$	$\lceil SD \rceil$	$\overline{MG}$
D/WF/–	0.52	0.04	0.52	1.00	3.33 (0.37)	5.30 (0.57)	0.00 (0.00)
D/BF/–	0.48	0.03	0.45	0.95	3.34 (0.45)	5.42 (0.61)	0.00 (0.00)
D/L2/–	0.48	0.03	0.44	0.95	3.36 (0.44)	5.47 (0.61)	0.00 (0.00)
D/FF/–	0.47	0.03	0.42	0.95	3.37 (0.44)	5.53 (0.51)	0.00 (0.00)
D/DP/–	0.47	0.03	0.41	0.95	3.39 (0.46)	5.58 (0.51)	0.00 (0.00)
R/RD/–	0.46	0.03	0.36	1.00	4.10 (0.43)	6.00 (0.00)	0.00(0.00)
R/WF/–	0.46	0.03	0.36	1.00	4.52 (0.35)	6.00 (0.00)	0.00 (0.00)
R/FF/–	0.46	0.03	0.36	1.00	4.53 (0.45)	6.00 (0.00)	0.00 (0.00)
R/DP/–	0.46	0.03	0.36	1.00	4.52 (0.39)	6.00 (0.00)	0.00 (0.00)
R/BF/–	0.46	0.03	0.36	1.00	4.54 (0.38)	6.00 (0.00)	0.00 (0.00)
R/L2/–	0.46	0.03	0.36	1.00	4.57 (0.36)	6.00 (0.00)	0.00 (0.00)
D/L2/DP	0.44	0.46	0.73	0.12	2.31 (0.20)	4.65 (0.49)	13.05 (3.35)
D/BF/TC	0.42	0.26	0.85	0.14	2.44 (0.36)	4.35 (0.81)	10.75 (2.43)
D/L2/TC	0.42	0.31	0.80	0.14	2.43 (0.37)	4.45 (0.69)	10.80 (2.84)
D/BF/KM	0.40	0.09	0.79	0.33	2.60 (0.30)	4.50 (0.69)	5.30 (2.11)
D/FF/TC	0.40	0.29	0.78	0.15	2.44 (0.35)	4.50 (0.69)	10.40 (2.39)
D/WF/TC	0.39	0.20	0.82	0.14	2.43 (0.32)	4.40 (0.68)	10.60 (2.44)
D/L2/KM	0.39	0.07	0.71	0.39	2.70 (0.34)	4.70 (0.66)	4.60 (1.70)
D/FF/KM	0.38	0.08	0.69	0.36	2.64 (0.31)	4.70 (0.57)	4.75 (1.68)
D/FF/DP	0.38	0.31	0.68	0.13	2.37 (0.35)	4.75 (0.44)	12.45 (3.52)
D/DP/KM	0.36	0.06	0.66	0.38	2.76 (0.37)	4.80 (0.70)	4.75 (1.71)
D/BF/DP	0.35	0.28	0.66	0.13	2.34 (0.22)	4.80 (0.52)	13.00 (3.43)
D/DP/TC	0.35	0.12	0.80	0.14	2.49 (0.35)	4.45 (0.69)	10.70 (2.27)
D/WF/DP	0.35	0.26	0.67	0.14	2.42 (0.33)	4.85 (0.75)	12.75 (4.34)
D/DP/DP	0.35	0.27	0.65	0.14	2.45 (0.37)	4.85 (0.75)	12.45 (3.83)
D/WF/KM	0.35	0.05	0.66	0.34	2.75 (0.36)	4.80 (0.52)	5.05 (1.70)
R/RD/TC	0.24	0.08	0.56	0.08	2.67 (0.36)	5.20 (0.52)	17.85 (3.31)
R/RD/DP	0.21	0.10	0.44	0.10	2.63 (0.42)	5.70 (0.47)	15.70 (3.87)
R/L2/DP	0.21	0.08	0.46	0.10	2.75 (0.61)	5.70 (0.47)	18.95 (5.84)
R/WF/DP	0.21	0.07	0.46	0.09	2.72 (0.62)	5.70 (0.47)	19.80 (5.05)
R/FF/DP	0.21	0.08	0.46	0.08	2.74 (0.48)	5.70 (0.47)	19.60 (5.07)
R/RD/KM	0.21	0.04	0.43	0.15	3.22 (0.40)	5.75 (0.44)	10.60 (2.58)
R/BF/DP	0.20	0.08	0.46	0.07	2.64 (0.40)	5.70 (0.47)	20.85 (3.87)
R/DP/DP	0.20	0.07	0.46	0.08	2.82 (0.54)	5.70 (0.47)	19.05 (4.87)
R/WF/TC	0.19	0.04	0.47	0.06	2.91 (0.38)	5.70 (0.47)	23.95 (2.95)
R/WF/KM	0.18	0.03	0.37	0.13	3.74 (0.48)	5.95 (0.22)	11.40 (3.12)
R/FF/TC	0.18	0.04	0.42	0.06	2.90 (0.36)	5.70 (0.47)	23.75 (4.54)
R/DP/KM	0.18	0.03	0.36	0.14	3.78 (0.42)	6.00 (0.00)	11.55 (2.91)
R/L2/KM	0.18	0.03	0.36	0.14	3.65 (0.44)	6.00 (0.00)	11.90 (2.88)
R/L2/TC	0.17	0.04	0.42	0.06	2.91 (0.35)	5.75 (0.44)	23.10 (3.86)
R/FF/KM	0.17	0.03	0.36	0.13	3.71 (0.46)	6.00 (0.00)	11.40 (2.68)
R/BF/TC	0.17	0.04	0.42	0.06	2.94 (0.39)	5.75 (0.44)	24.25 (3.48)
R/DP/TC	0.17	0.04	0.41	0.06	2.87 (0.33)	5.80 (0.41)	24.55 (4.12)
R/BF/KM	0.17	0.03	0.36	0.12	3.73 (0.37)	6.00 (0.00)	12.25 (2.17)

service level violations of each controller, which cannot be obtained by simple simulations. Experiments were conducted with the following controller combinations:

- Good performing controllers in simulations
  - First Fit Demand with KMControl (D/FF/KM)
  - L2 Demand with KMControl (D/L2/KM)
  - Worst Fit Demand without reallocation (D/WF/–)
- Poor performing controllers in simulations
  - Worst Fit with TControl (R/WF/TC)
  - Worst Fit with KMControl (R/WF/KM)

Controller combinations for experiments were chosen to cover a set of good and poor performing controllers based on the MRR ranking shown in Table 2. We selected D/FF/KM and D/L2/KM as they performed well for average, maximum server demand, and migrations. R/WF/TC and R/WF/KM represent poor performing controllers. In each case two controllers were picked to see whether they perform similarly well in experiments as suggested by simulations. Considering all metrics and data center requirements there is no clear winner. In addition D/WF/– was tested because it performed best according to the global MRR ranking if migrations are a limiting factor and average server demand is less of a concern.

Experimental results can be found in Tables 4 and 5. For each controller and schedule configuration, there is one result line. It describes the average experimental results over 5 schedule instances of one schedule configuration. Differences

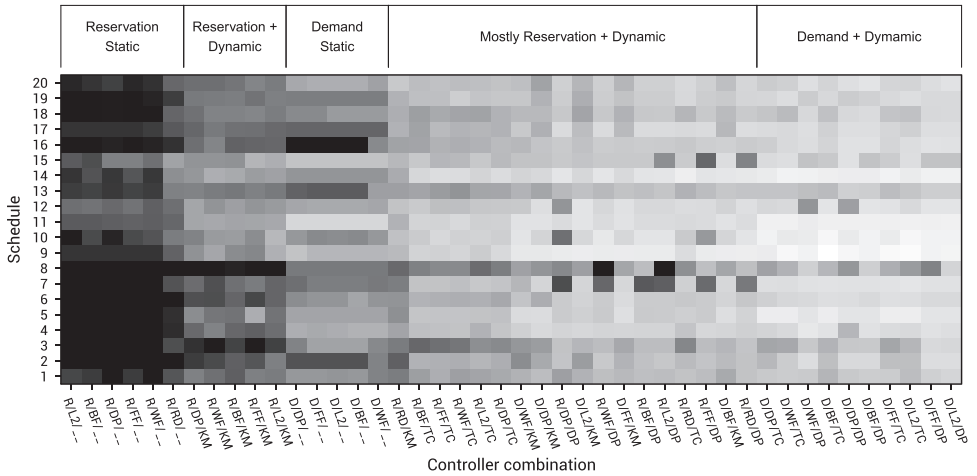

Fig. 4. Heatmap of average server demand over controllers and schedule instances.

Table 3
Statistics for results clustered by the controller type. $\overline{SD}$ and $\overline{MG}$ – average server demand and migrations, σSD and σMG – standard deviation for average server demand and VM migrations, ΔSD and ΔMG – difference between min and max values.

Cluster	$\overline{SD}$	σSD	ΔSD	$\overline{MG}$	σMG	ΔMG
Reservation Static	4.46	0.42	2.03	0.00	0.00	0.00
Demand Static	3.36	0.42	1.70	0.00	0.00	0.00
Reservation + Dynamic	3.07	0.60	2.80	17.81	6.27	28.00
Demand + Dynamic	2.51	0.35	1.81	9.43	4.30	18.00

between min and max results are reported in square brackets while variances are reported in parentheses.

For migrations, we found that combinations of reservation-based placement and reallocation controllers triggered more migrations than ones using demand-based placement controllers. This confirms simulation results and can be explained by the higher resource demands of reservation-based controllers. Demand-based controller combinations consistently triggered more migrations than pure static controllers and less than reservation-based combinations.

Service quality could be maintained by almost all controller combinations except D/L2/KM which fell below a desired level of 99% service quality for all schedule configurations. For some schedule configurations controller combinations R/WF/TC and D/WF/– also fell below a service quality of 99%.

Average server demand showed that D/FF/KM delivered the best performance and consistently required the least number of servers while maintaining the desired service level. Overall R/WF/KM had the highest average server demand and the highest peak server demand. These results are in line with simulation results. All other controllers can be found between D/FF/KM and R/WF/KM.

Server CPU utilization varied between 20% and 45% depending on the controller configuration and remained well below 70%. Average memory as a second constraining resource remained below a 50% utilization level for all controller combinations. A reason for this low average CPU utilization was that most VMs were only utilized below 30% on average. In order to increase overall server CPU utilization many VMs would have to be allocated to a single server. However, this was not possible due to a fixed memory limit for all VMs on a server, which was set to 85% of its total capacity to keep enough space for the hypervisor as shown in Fig. 5. Server utilization increased to 90% and above during times with higher workload on the VMs.

CPU utilization will also be low if there are too few VMs to fully utilize a single or all active servers. For example an average CPU utilization of 50% is caused if two servers are active, one running at 80% and the second one at 20% utilization. Average utilization cannot be increased because resource limitations prevent a migration of the VM on the second server to the first one, and there are not enough VMs available.

Average response time of operations was slightly sensitive to denser allocations. For the two controllers D/FF/KM, R/WF/TC, and D/L2/KM that produced the lowest average server demand, average response time was slightly increased compared to other controllers. However, the difference is not significant due to the high variance in the response times. Interestingly, D/L2/KM yielded the highest response times and worst service quality while delivering low average and maximum server demands without triggering too much migrations.

Table 6 summarizes experimental results over all schedules for each controller. While the numbers are slightly different, the ranking of controllers is in line with the simulation results. Average server demand was generally a bit lower as predicted by simulations while more VM migrations were triggered. Demand-based controllers always triggered less migrations than reservation-based controllers as already found for simulations.

5. Conclusion

Much research on resource allocation in virtualized data centers has focused on efficient heuristics for the initial placement. Typically, bin packing heuristics are used in wide-spread cloud management tools like OpenStack or

Table 4
Experimental results on placement and dynamic controllers. (+/−) – based on simulation results, either a poor or good performing controller. *Control* – controller combination with notation [(D)emand- or (R)eservation-based]/[placement controller]/[dynamic controller], *Sched* – VM allocation/deallocation schedule configuration, $\overline{SD}$ – average server demand, ⌈SD⌉ – maximum server demand, $\overline{CPU}$ – average CPU load, $\overline{MEM}$ – average memory load, $\overline{RT}$ – average response time (ms), ⌈RT⌉ – maximum response time (ms).

Control		Sched	$\overline{SD}$	⌈SD⌉	$\overline{CPU}$	$\overline{MEM}$	$\overline{RT}$	⌈RT⌉
+	D/FF/KM	20 000	2.53 (0.17)	3.67 (0.58)	42	47	654.50 (49.86)	44 847
+	D/L2/KM	20 000	2.74 (0.27)	4.25 (0.96)	42	43	900.19 (187.90)	50 262
+	D/WF/–	20 000	3.15 (0.25)	4.75 (0.96)	33	39	617.55 (26.44)	23 045
−	R/WF/KM	20 000	3.69(0.17)	6.00 (0.00)	32	34	637.24 (32.40)	36 326
−	R/WF/TC	20 000	2.97 (0.27)	5.50 (0.53)	40	41	679.35 (87.81)	61 091
+	D/FF/KM	20 100	2.41 (0.39)	4.00 (0.82)	41	48	627.23 (52.04)	32 832
+	D/L2/KM	20 100	2.88 (0.51)	4.50 (0.58)	41	41	1427.40 (518.85)	61 127
+	D/WF/–	20 100	3.38 (0.41)	5.25 (0.50)	28	35	616.68 (26.65)	21 060
−	R/WF/KM	20 100	3.60 (0.66)	6.00 (0.00)	30	33	615.77 (50.38)	34 873
−	R/WF/TC	20 100	2.77 (0.48)	5.75 (0.46)	39	42	641.82 (39.44)	46 271
+	D/FF/KM	20 200	2.29 (0.22)	4.50 (0.58)	38	43	662.26 (61.48)	44 190
+	D/L2/KM	20 200	2.34 (0.29)	4.25 (0.50)	42	43	1578.25 (611.89)	204 359
+	D/WF/–	20 200	2.97 (0.45)	4.75 (0.50)	28	35	635.31 (59.12)	28 491
−	R/WF/KM	20 200	3.21 (0.26)	6.00 (0.00)	30	31	639.47 (31.31)	40 645
−	R/WF/TC	20 200	2.63 (0.16)	5.43 (0.53)	37	37	674.82 (50.44)	58 384
+	D/FF/KM	20 300	2.38 (0.14)	4.00 (0.00)	41	48	670.62 (33.76)	59 608
+	D/L2/KM	20 300	2.68 (0.07)	4.33 (0.58)	41	45	1091.45 (152.11)	82 731
+	D/WF/–	20 300	3.29 (0.16)	5.00 (0.00)	28	36	644.83 (40.58)	31 294
−	R/WF/KM	20 300	3.62 (0.15)	6.00 (0.00)	30	32	648.37 (25.09)	48 188
−	R/WF/TC	20 300	2.78 (0.15)	5.25 (0.46)	39	42	686.48 (34.21)	46 496

Table 5
Experimental results on placement and dynamic controllers. (+/−) – based on simulation results, either a poor or good performing controller. *Control* – controller combination with notation [(D)emand- or (R)eservation-based]/[placement controller]/[dynamic controller], O – total operation count, $\overline{MG}$ – average VM migrations, SQ – service quality (percentage of successful requests with a response time below 3 s).

Control		O	$\overline{MG}$	SQ
+	D/FF/KM	2 460 676	07 [05/08]	99.12 (0.273)
+	D/L2/KM	2 399 470	06 [05/08]	95.52 (1.360)
+	D/WF/–	2 576 691	00 [00/00]	99.60 (0.100)
−	R/WF/KM	2 634 552	20 [16/22]	99.61 (0.049)
−	R/WF/TC	2 555 583	26 [25/27]	97.70 (0.814)
+	D/FF/KM	2 528 618	08 [06/09]	99.48 (0.095)
+	D/L2/KM	2 210 333	06 [04/07]	91.66 (2.033)
+	D/WF/–	2 535 460	00 [00/00]	99.49 (0.139)
−	R/WF/KM	2 535 183	19 [16/21]	99.50 (0.121)
−	R/WF/TC	2 522 725	27 [24/29]	99.18 (0.200)
+	D/FF/KM	1 923 776	07 [04/09]	98.77 (0.316)
+	D/L2/KM	1 672 268	06 [04/09]	94.58 (1.515)
+	D/WF/–	1 932 057	00 [00/00]	99.21 (0.204)
−	R/WF/KM	1 931 438	20 [17/24]	99.16 (0.217)
−	R/WF/TC	1 849 433	27 [23/33]	98.68 (0.274)
+	D/FF/KM	2 365 682	09 [08/10]	99.23 (0.136)
+	D/L2/KM	2 270 676	07 [07/08]	97.30 (0.284)
+	D/WF/–	2 383 054	00 [00/00]	91.90 (3.625)
−	R/WF/KM	2 370 031	19 [15/20]	99.38 (0.041)
−	R/WF/TC	2 360 173	26 [24/29]	96.80 (0.928)

Eucalyptus. We could not find substantial differences in the energy efficiency of different bin packing heuristics in the placement controllers.

However, there were substantial differences in the energy efficiency if additional reallocation controllers were used. There was no substantial difference among the types of reallocation controllers used, but whether one was used or not had a considerable impact on the average server number. Surprisingly, reallocation has not been a concern in the related literature so far, nor is it used in cloud management tools used in industry practice.

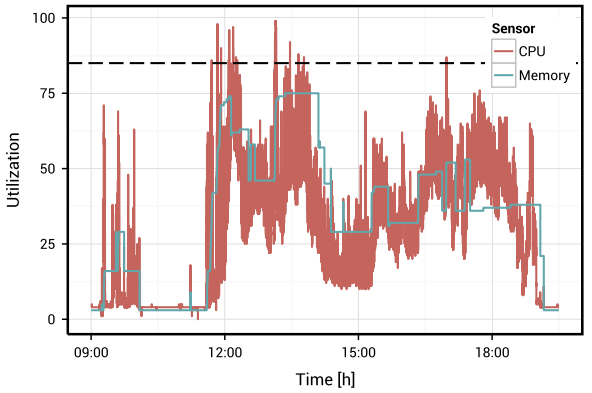

Fig. 5. Exemplary utilization of server CPU and memory for a single experiment.

Table 6
Statistics over all experiments sorted by MRR ranking in experiments. $\overline{SD}$ and $\overline{MG}$ – average server demand and migrations, σSD and σMG – standard deviation for average server demand and VM migrations, ΔSD and ΔMG – difference between min and max values.

Cluster	$\overline{SD}$	σSD	ΔSD	$\overline{MG}$	σMG	ΔMG
D/FF/KM	2.40	0.24	0.95	7.60	1.68	6
D/L2/KM	2.66	0.37	1.32	6.07	1.53	5
R/WF/TC	2.79	0.31	1.32	26.42	2.25	10
D/WF/–	3.20	0.34	1.37	0.00	0.00	0
R/WF/KM	3.54	0.38	1.53	19.59	2.43	9

In addition, the parameters used for the placement algorithms have an impact on the average server demand. Both simulation and experimental results indicate that a controller should aim for a dense allocation from the start for high energy efficiency. Demand-based placement controllers lead to dense packing, because they take the actual demand on a server into account and not the reserved capacity. However, demand-based placement controllers need to be used in conjunction with reallocation controllers to avoid overloads.

Nowadays, reservation-based placement controllers are state-of-the-practice, which is probably due to risk considerations of IT service managers. Our study shows that combinations of demand-based placement controllers with reallocation controllers actually lead to fewer migrations than reservation-based placement controllers and lower server demand, while maintaining service quality.

Overall, demand-based placement controllers in combination with a reallocation controller appear to be the most energy-efficient solution. Experiments and simulations indicate significant savings in average server demand of about 20% to 30% compared to placement-only control strategies.

Acknowledgments

This project is supported by the Deutsche Forschungsgemeinschaft (DFG) (BI 1057/4-1).

Appendix A. Simulating IaaS cloud environments

A discrete event simulation framework resembles our test bed infrastructure. Simulations were conducted on a large number of different placement and reallocation controllers. Promising setups were transferred to the experimental infrastructure for further verification. Experiments and simulations are based on the same software framework so that controller implementations used for experiments and simulations are identical. Therefore, all software interfaces are either implemented to control a physical or simulated infrastructure as shown in Fig. 2.

During an experiment, resource utilization data on all VMs and servers is provided by the Sonar monitoring system. For simulations, a workload driver component replaces Sonar, simulating the CPU and memory utilization readings. A time series is attached to each VM. For simulations this time series describes the resource utilization of the VM. For experiments a Rain workload driver leverages the time series to vary the workload generated on the VM which eventually causes a resource utilization that is very similar to the original time series.

Application requests (e.g. HTTP or CORBA queries) are not simulated by the framework. Simulations are based on resource utilizations of servers and VMs according to predefined time series data. A time series describes the workload over a certain time, e.g. 12 h. This time is simulated in discrete 3 second steps. For each step the VM utilization is determined using the time series data. Server utilizations are calculated by summing up VM utilizations. A server is over-subscribed if its CPU utilization exceeds 100%.

Simulated service quality is calculated by dividing the number of simulated time steps where a server was over-subscribed and dividing it by the total number of simulated time steps. Contrary, *experimental service quality* is calculated dividing the number of failed or late HTTP requests by the total number of triggered HTTP requests.

Rain accesses an IaaS network service to allocate and deallocate VMs during an experiment. Simulations leverage an VM allocation driver substituting Rain. It directly issues method calls to the IaaS service layer by using to the same VM allocation schedules as used by Rain.

VM migrations are simulated by waiting a negative-exponentially distributed time with $\mu = 3.29, \sigma = 0.27$. In addition an increased CPU utilization of 8% is simulated on both, the migration target and source server during the migration. Parameters are based on findings of our previous work.

Sonar closely monitors the whole infrastructure during experiments. All server utilization measurements of all monitored resources as well as all controller actions are recorded by Sonar as well. An analysis step following each experiment reads all relevant data from Sonar. It allows a complete reproduction of an experiment. A set of core metrics gets calculated: VM migrations, service quality, average, and peak server demand, CPU, and memory utilization.

Appendix B. VM arrival and departure schedules

Simulations and experiments are based on schedules of VM arrivals and departures. A schedule is generated based on four random variables.

- *Lifetime* of a VM (see black bars in Fig. 1).
- *Inter-arrival time* of the VMs.
- *VM launches* describes the total number of arriving VMs (the total number of black bars in Fig. 1).
- *VM sizes* describes the VMs' resource demands.

Not all possible schedules could be evaluated, considering multiple levels for each variable. To cut the number of schedules used for simulatiosn and experiments, we analyzed which variables influence the performance of controllers. A 2^k fully factorial design was used to address his question. Table B1 summarizes the factors and levels. For each schedule configuration 25 schedules were generated randomly.

Each schedule was evaluated with nine different placement controllers: First-Fit, Best-Fit, Worst-Fit, Dot-Product with a demand based and reservation based implementation. A

Table B1
Factors and levels for schedule configuration sensitivity analysis.

Factor	Low	High
Lifetime (h)	1	6
Inter-arrival time (min)	5	20
VM launches	400	500
VM sizes	2^x	x

random controller was used as a control group. In total $16 \cdot 25 \cdot 9 = 3600$ simulations were conducted.

For each simulation the allocation efficiency of the controller is calculated based on average and peak server demand during the simulation and a lower bound estimate. A dedicated ANOVA analysis for each controller found lifetime and inter-arrival time as well as their interaction effect significant in all cases, with levels of $p < 0.001$. In rare cases, the factor launches was significant, with level of $p < 0.01$. Q–Q plots and residual plots showed no abnormalities.

Based on these findings we focused on schedules with different levels for variables: inter-arrival time and lifetime.

Appendix C. Reactive controller parametrization

A design of experiments approach (DoE) [28] was chosen to find a good parametrization for KMControl and TControl controllers. ANOVA and contour plots were used to determine parametrization that minimize the average number of servers while keeping migrations and service level violations in a desired operation range.

The simulation framework as described in Appendix A was used. Initially, all VMs are allocated to the servers in a round robin approach where VM with index i is assigned to server $i \bmod n$ with n servers. Simulations were conducted for a scenario size of 390 VMs. The number of VMs in the infrastructure did not change during a simulation.

Metrics returned for each simulation were: migrations (number of triggered live-migrations), servers (average number of active servers), violations T^v (number of simulation periods where server load was above 100%), and T^a (total number of simulation periods). A service level is calculated by $1 - T^v/T^a$. It is important to understand that calculated service level differs from service levels of real infrastructures as explained in Appendix A.

Simulations were conducted according to a 2^k fully factorial design. Factor levels are shown in Table C1 for both controllers. Each factor is one controller parameter and each treatment was replicated 20 times with different CPU workload traces. In total, 640 simulations were conducted for each controller, discarding invalid factor level combinations for KMControl where $k < m$.

The desired operation range of both controllers was defined based on our experience with European data center operators as follows: Service level above 95%, VM migrations below $1.5 \cdot m$ for m VMs, and minimal average server demand.

Table C1
KM and TControl parametrization factors and levels.

	KMControl		TControl	
Factor	Low	High	Low	High
Overload threshold, T_o	80	95	80	95
Underload threshold, T_u	10	40	10	40
Executing interval I (s)	60	900	60	900
k value	10	100	50	200
m value	8	80	–	–
α	–	–	0.05	0.1

C.1. Results for KMControl

Initial ANOVA analysis over all simulations included all factors and their interactions. All factors were coded to design variables in a range of $[-1, 1]$. Data was log-transformed as initial results indicated heteroscedasticity in the residual plots and deviations from the normality assumption. Some factors were found to be significant at $p < 0.05$. A second linear ANOVA model for each target variable was constructed based on significant factors only. QQ-plots of the model residuals indicate a normal distribution with slight deviation in the tails. Scatter plots of residuals vs. fitted values did not show any patterns or heteroscedasticity. Contour plots were generated based on the linear ANOVA models to find a global optimal parametrization. Based on these plots we gained a number of insights:

1. The average server demand can be decreased by choosing a large T_o, a large T_u, and a large m in conjunction with a small k. It is independent to the interval length I.
2. VM live-migrations can be decreased by increasing T_o, m, and I. However, k and T_u should be small. Especially the demand for a small T_u is in contradiction with the goal to decrease average server demand with a high T_u. We chose the highest level for T_u so that the migration constraint is satisfied in order to minimize average server count.
3. Violations can be decreased by increasing T_o, I, and m while decreasing T_u, and k. These requirements are closely aligned with the requirements to achieve a low number of migrations.

Based on the contour plots, we determined a controller parametrization with $T_o = 95$, $T_u = 27$, $k = m = 65$, and $I = 750$.

C.2. Results for TController

The same analysis approach was chosen as for the KMControl controller. Based on the contour plots for the TControl controller we gained the following insights:

1. The average server demand can be decreased by increasing both T_o and T_u thresholds and by decreasing factors k and I.
2. Migrations can be minimized by increasing T_o, k and I and T_U should be minimized. Increasing T_o and k is in contradiction with the goals to minimize average serve demand. We choose values so that average servers is minimized and migrations remain within the desired operation range.
3. Violations are minimized by decreasing all factors while T_o and T_u are the strongest contributors.

Based on the contour plots, we determined a controller parametrization with $T_o = 90$, $T_u = 25$, $k = 170$, and $I = 400$.

C.3. Differences KMController and TController

We conducted additional simulations to compare the performance of both controllers. Both were configured with the settings of the previous analysis. Comparison is based on 20 simulations with a different set of workload traces. Key metrics were average server demand, migrations, and violations. For each metric the mean, medium, maximum, minimum, first-, and third-quartile was calculated.

For average server demand, we found TControl to outperform KMControl significantly based on a two-sided t-test at $p=0.007529$. In all scenarios TControl was better regarding server demand. Both controllers performed equally good for migrations while TControl has a slight tendency to trigger more migrations. A two-sided t-test found significant differences at $p=0.01888$. For a scenario with 100 VMs over 20 simulation runs, KMControl performed 136 migrations on median vs. 155 migrations for TControl. We could not find significant differences for violations between both controllers with $p=0.4264$.

References

[1] A. Hios, T. Ulichnie, Top 10 Data Center Business Management Priorities for 2013 about the Uptime Institute Network, Technical Report, Uptime Institute, 2013.

[2] NASCIO, State Cio Priorities for 2013, Technical Report, NASCIO, 2013.

[3] C. Ingle, S. Group, Beyond Organisational Boundaries: Answering the Enterprise Computing Challenge Chris Ingle, Consulting and Research Director, Systems Group Agenda Current Economic Conditions and the Cio Investing for Efficiency: Datacentre Design and Operations, Technical Report, IDC, 2009.

[4] B. Speitkamp, M. Bichler, A mathematical programming approach for server consolidation problems in virtualized data centers, IEEE Trans. Serv. Comput. 3 (4) (2010) 266–278.

[5] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, Resource pool management: reactive versus proactive or let's be friends, Comput. Netw. 53 (17) (2009) 2905–2922.

[6] A. Wolke, M. Bichler, T., Setzer, Planning vs. dynamic control: resource allocation in corporate clouds, IEEE Transactions on, Cloud Computing, vol. 99, 2015, p. 1–1, http://dx.doi.org/10.1109/TCC.2014.2360399.

[7] S. Brumec, N. Vrcek, Cost effectiveness of commercial computing clouds, Inf. Syst. 38 (4) (2013) 495–508.

[8] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, M. Nelson, B.-H. Lim, G. Hutchins, Fast transparent migration for virtual machines, in: M. Nelson, B.-H. Lim, G. Hutchins (Eds.), Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, vol. 2, ATEC 05, USENIX Association, Berkeley, CA, USA, 2005, pp. 273–286.

[9] Y. Choi, S. Lee, J. Kim, Y. Kim, H. Pak, G. Moon, J. Ra, Y.-G. Jung, The method to secure scalability and high density in cloud data-center, Inf. Syst. 48 (2015) 274–278.

[10] E.G., Coffman, Jr., M.R. Garey, D.S., Johnson, Approximation algorithms for bin packing: a survey, in: Approximation Algorithms for NP-hard Problems, Dorit S. Hochbaum, (Ed.) PWS Publishing Co., Boston, MA, USA, 1997, pp. 46–93, 〈http://dl.acm.org/citation.cfm?id=241938.241940〉.

[11] E.G. Coffman Jr, M.R. Garey, D.S. Johnson, Dynamic bin packing, SIAM J. Comput. 12 (2) (1983) 227–258.

[12] Wong, W.H. Prudence Yung, C.C. Fencol Mihai, Burcea, An 8/3 lower bound for online dynamic bin packing, in: Chao, Kun-Mao and Hsu, Tsan-sheng and Lee, Der-Tsai, (Eds.) Algorithms and Computation, Lecture Notes in Computer Science, vol. 7676, pp. 2012, 44-53, Springer, Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-642-35261-4_8.

[13] Z. Ivkovic, E. Lloyd, Fully dynamic algorithms for bin packing: being (mostly) myopic helps, in: T. Lengauer (Ed.), Algorithms – ESA '93, Lecture Notes in Computer Science, vol. 726, Springer, Berlin, Heidelberg, 1993, pp. 224–235.

[14] T. Setzer, M. Bichler, Using matrix approximation for high-dimensional discrete optimization problems: server consolidation based on cyclic time-series data, Eur. J. Oper. Res. 227 (1) (2013) 62–75.

[15] N. Bobroff, A. Kochut, K. Beaty, Dynamic placement of virtual machines for managing SLA violations, in: Proceedings of the 10th IFIP/IEEE International Syposium on Integrated Network Management, 2007, pp. 119–128, http://dx.doi.org/10.1109/INM.2007.374776.

[16] A. Verma, P. Ahuja, A. Neogi, pmapper: Power and migration cost aware application placement in virtualized systems, in: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, Middleware '08, Springer-Verlag New York, Inc., New York, NY, USA, 2008, pp. 243–264.

[17] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, Sandpiper: black-box and gray-box resource management for virtual machines, Comput. Netw. 53 (17) (2009) 2923–2938.

[18] G. Dhiman, G. Marchetti, T. Rosing, vgreen: A system for energy efficient computing in virtualized environments, in: Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED '09, ACM, New York, NY, USA, 2009, pp. 243–248.

[19] N. Calcavecchia, O. Biran, E. Hadad, Y. Moatti, VM placement strategies for cloud scenarios, in: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), 2012, pp. 852–859.

[20] K. Mills, J. Filliben, C. Dabrowski, An efficient sensitivity analysis method for large cloud simulations, in: 2011 IEEE International Conference on Cloud Computing (CLOUD), 2011, pp. 724–731.

[21] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768. Special Section: Energy efficiency in large-scale distributed systems.

[22] C. Peng, M. Kim, Z. Zhang, H. Lei, Vdn: virtual machine image distribution network for cloud data centers, in: INFOCOM, 2012 Proceedings IEEE, 2012, pp. 181–189.

[23] S.S. Seiden, On the online bin packing problem, J. ACM 49 (5) (2002) 640–671.

[24] R. Panigrahy, K. Talwar, L. Uyeda, U. Wieder, Heuristics for Vector Bin Packing, Technical Report, 2011.

[25] X. Li, A. Ventresque, N. Stokes, J. Thorburn, J. Murphy, ivmp: an interactive vm placement algorithm for agile capital allocation, in: IEEE Conference on Cloud Computing, 2013.

[26] T. Setzer, A. Wolke, Virtual machine re-assignment considering migration overhead, in: Network Operations and Management Symposium (NOMS), 2012 IEEE, 2012, pp. 631–634.

[27] X. Fan, W.-D. Weber, L.A. Barroso, Power provisioning for a warehouse-sized computer, in: Proceedings of the 34th Annual International Symposium on Computer Architecture, ISCA '07, ACM, New York, NY, USA, 2007, pp. 13–23.

[28] Montgomery, C. Douglas, Design and Analysis of Experiments, John Wiley & Sons, 2006.