Allocating Compute and Network Resources under Management Objectives in Large-Scale Clouds

Fetahi Wuhib · Rerngvit Yanggratoke · Rolf Stadler

August 23, 2013

Abstract We consider the problem of jointly allocating compute and network resources in a large Infrastructure-as-a-Service (IaaS) cloud. We formulate the problem of optimally allocating resources to virtual data centers (VDCs) for four well-known management objectives: balanced load, energy efficiency, fair allocation, and service differentiation. Then, we outline an architecture for resource allocation, which centers around a set of cooperating controllers, each solving a problem related to the chosen management objective. We illustrate how a global management objective is mapped onto objectives that govern the execution of these controllers. For a key controller, the Dynamic Placement Controller, we give a detailed distributed design, which is based on a gossip protocol that can switch between management objectives. The design is applicable to a broad class of management objectives, which we characterize through a property of the objective function. The property ensures the applicability of an iterative descent method that the gossip protocol implements. We evaluate, through simulation, the dynamic placement of VDCs for a large cloud under changing load and VDC churn. Simulation results show that this controller is effective and highly scalable, up to 100'000 nodes, for the management objectives considered.

Keywords

Cloud computing, distributed management, resource allocation, gossip protocols, management objectives

Fetahi Wuhib
CDN & Cloud Computing, Ericsson Research
Färögatan 6, 164 80 Kista, Sweden
Tel: +46-1-0715-6349. E-mail: fetahi.wuhib@ericsson.com
Work done while the author was with KTH Royal Institute of Technology

Rerngvit Yanggratoke (✉)
ACCESS Linnaeus Center, KTH Royal Institute of Technology
Osquldas väg 10, SE-100 44, Sweden
Tel: +46-8-790-4256. E-mail: rerngvit@kth.se

Rolf Stadler
ACCESS Linnaeus Center, KTH Royal Institute of Technology
Osquldas väg 10, SE-100 44, Sweden
Tel: +46-8-790-4250. E-mail: stadler@kth.se

1 Introduction

We consider the problem of jointly allocating compute (i.e., processor and memory) resources and network resources in a large-scale cloud environment. We focus the discussion on an *Infrastructure-as-a-Service* (IaaS) cloud, a popular service concept that is offered by many public cloud providers, including Amazon and RackSpace, and is supported by public-domain solutions, such as OpenStack, OpenNebula, and Eucalyptus. An IaaS cloud makes ICT (Information and Communication Technology) infrastructure available to customers in a virtualized way, including computation in form of virtual machines (VMs), storage in form of, e.g., virtual disks, and networking in form of, e.g., virtual links between VMs. IaaS clouds have proved suitable for running interactive applications, such as websites and social networks, media streaming services, such as audio or video on-demand services, as well as analytics and data mining applications.

The IaaS service model has two principal stakeholders: the *IaaS provider* who owns and operates the cloud infrastructure and a set of *customers* who run applications on the cloud. (An IaaS customer can, in turn, serve end users when running a web site for instance. This customer-user relationship however is not relevant to this paper and not discussed further.) A customer typically has a service-level agreement (SLA) with the provider, which specifies how its application is executed and which cloud resources are made available.

We call the service abstraction, provisioned by the provider and offered to a customer to run its applications, the *virtual data center* (VDC). A customer specifies the resource requirement of a VDC in terms of the number and capacity of VMs, as well as the communication requirements among VMs and between VMs and the (public) Internet. The provider chooses a *management objective* according to which the physical resources of the cloud infrastructure are allocated to VDCs. A management objective typically includes (a) requirements related to the infrastructure use, such as balancing the load across machines or consolidating the load onto a minimal set of machines, (b) requirements regarding SLAs, such as supporting different classes of service or ensuring that the (changing) demand of a VDC is periodically evaluated and satisfied, and (c) requirements reflecting business goals of the provider, e.g., maximizing revenue, or using a maximum amount of green energy.

In this paper, we formulate the problem of optimally allocating compute and networking resources to VDCs for four well-known management objectives: balanced load, energy efficiency, fair allocation, and service differentiation. We outline an architecture for resource allocation under management objectives, which centers around a set of cooperating controllers, each solving a problem related to the chosen management objective. We illustrate how a global management objective is mapped onto objectives that govern the execution of these controllers. For a key controller, the Dynamic Placement Controller, we give a detailed distributed design. It is based on a gossip protocol that computes a heuristic solution to a placement problem that is NP-hard for most management objectives. The protocol is instantiated with the objective function associated with the chosen management objective. We evaluate, through simulation, the dynamic placement of VDCs for a large cloud under changing load and VDC churn. We show through simulation that this controller is effective and highly scalable for the management objectives considered.

Note that, while we provide a scalable solution for dynamic placement regarding compute and networking resources, we do not explicitly consider storage resources, which applications need as well. Coming up with a scalable solution for (shared) storage is a research issue in itself [1].

We make the following contributions with this work. First, our design considers a large *class of management objectives*, as exemplified by the four chosen objectives for this paper. The class consists of management objectives that are characterized through objective functions that have a specific property $\mathcal{P}$. (This property ensures the applicability of an iterative descent method that the gossip protocol implements.) In fact, virtually all objective functions related to VM placement we found in the literature satisfy this property. Our approach differs from most reported results on resource allocation (more specifically: application or VM placement) in clouds, where the allocation problem is studied only for a single objective. From a technological viewpoint, we believe it is significant to develop generic solutions that cover a range of management objectives, which gives choices to providers. Second, we present the gossip protocol for computing VM placement that supports all management objectives in the considered class. To switch between management objectives, the protocol simply switches between the objective functions f associated with those objectives. We show that the protocol converges as long as the objective function f has the property $\mathcal{P}$. The protocol implements a *distributed heuristic to compute a placement* and is highly scalable in the sense that, for the same protocol overhead, its effectiveness does not noticeably worsen when the system size increases from 800 to 100,000 machines, as simulation results show. Third, our work considers *joint allocation of compute and network resources*, which we believe is essential for ensuring predictable performance of large cloud-based applications. We assume that the network interconnecting the machines has full-bisection bandwidth (see below), which allows for an elegant distributed solution to the placement problem, but limits our approach to a single data center, where full bisection bandwidth technology is feasible and preferred. Fourth, while the core contribution of this work is on application placement, we show how our *solution seamlessly fits into a complete resource management architecture.*

In our prior work, we have developed a gossip protocol for adaptive placement that achieves min-max fairness [2, 3], as well as a version of the protocol that achieves energy efficiency [4]. This paper is a significant extension and generalization of that work in that the same protocol—not different versions of the protocol—computes placements for a class of management objectives, including the two mentioned above. In addition, the solution presented here covers compute and networking resources, not only compute resources. The architecture outlined in this paper has been introduced in [5], where we focused on an OpenStack implementation for cloud resource management.

The paper is organized as follows. Section 2 introduces our model for resource allocation under management objectives. Section 3 presents our architecture for resource management. Section 4 presents our solution for dynamic VDC placement under management objectives, while we present the results of its evaluation in Section 5. Section 6 surveys related research, and Section 7 concludes this paper.

2 Modeling Resource Allocation under Management Objectives

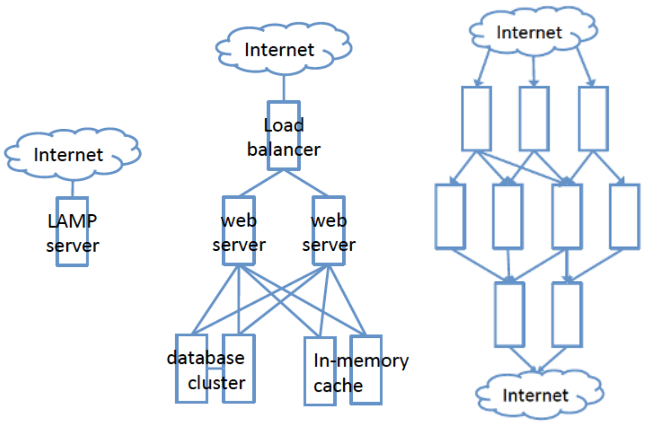
2.1 Examples of virtual data centers (VDCs)

The service abstraction considered in this work, i.e., the virtual data center (VDC), is suitable for running a wide range of applications. We give two examples that are common use cases for IaaS cloud services. The first type of application is an interactive, multi-tiered website. Amazon, for example, hosts more than 9 million websites in its cloud computing facility [6]. Figure 1a shows two example architectures for websites hosted on an IaaS cloud. On the left is a small self-contained website based on a LAMP (Linux, Apache, MySQL and PHP) VM; on the right is a larger website that includes a load balancer (e.g., a nginx reverse proxy), a set of web servers (e.g., Apache servers), a database cluster (e.g., MySQL cluster) and a set of in-memory cache servers (e.g., Memcached servers)—whereby all of these components run in VMs. Requests to a website arrive at the load balancer, which forwards them to web servers. A web server serves a request for information in the following order: (1) from the local logic/cache/disk (2), from an in-memory cache server, and (3) from the database cluster.

The second type of application is written using a programming framework for processing large data sets, such as MapReduce [7] or Dryad [8]. (Dryad generalizes the more popular MapReduce framework.) Consider a Dryad application shown in Figure 1b. The application is modeled as a directed acyclic graph, whereby links represent the dataflow of the application, while vertices define the operations that are to be performed on the data. Each such operation runs in a VM.

2.2 Modeling the VDC abstraction and the Cloud Infrastructure

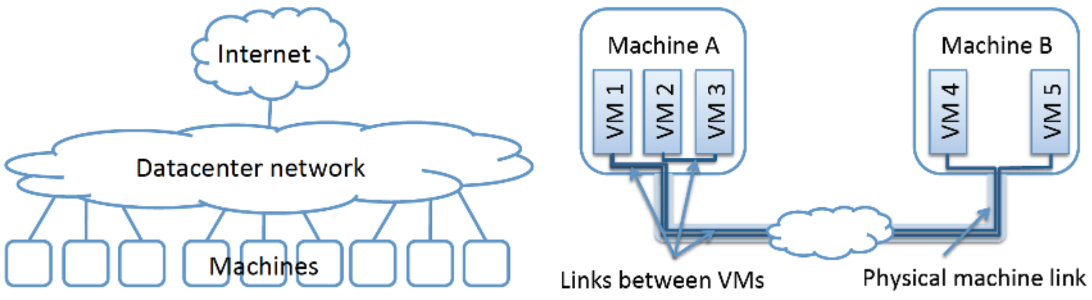
We model a VDC, i.e., the service abstraction visible to the customer, as a graph whose nodes represent virtual machines (VMs) running the customer application and whose edges represent bi-directional links through which the VMs communicate. A VDC is specified by giving, for each VM, a CPU demand limit ω^l_m and a memory demand limit γ^l_m. (For instance, a 'Large' Amazon EC2 [9] instance m has a ω^l_m of four Compute units and a γ^l_m of 7.5GB.) A link between two VMs i and j is specified by giving the link bandwidth demand limit $\lambda^l_{i,j}$. During the operation of a VDC, we denote the CPU demand of VM m at time t by $\omega^d_m(t) \leq \omega^l_m$ and the memory demand by $\gamma^d_m(t) \leq \gamma^l_m$. In addition, we denote the bandwidth demand of the link between VMs i and j by $\lambda^d_{i,j}(t) \leq \lambda^l_{i,j}$. The set of all VMs is represented by $M(t)$ and the set of all links by $L(t) \subseteq M(t) \times M(t)$.



(a) Examples of website setups (b) A Dryad job

Fig. 1: Customer applications deployed on VDCs

We model the physical infrastructure as a set of machines N that are interconnected by the data center network and connected to the public Internet as shown in Figure 2a. We assume that the data center network has full bisection bandwidth, which implies that, for each network interface on a machine, the aggregate outgoing bandwidth and the aggregate incoming bandwidth is limited only by the interface capacity. We justify this assumption by the emerging consensus about future data center network architectures (cf. [10]) as well as a measurement study involving the Amazon EC2 cloud [11]. In our model, each machine $n \in N$ has an associated CPU capacity ω_n^c, a memory capacity γ_n^c, a network interface capacity λ_n^c and a maximum power consumption p_n^{max}.



(a) The data center network (b) Interconnecting VMs

Fig. 2: A simple model of the cloud infrastructure and deployment of VDCs

A VDC is deployed on the cloud infrastructure as follows. Each VM m is placed on a machine n, which reserves CPU capacity $\omega_m^r(t_0)$ and memory capacity $\gamma_m^r(t_0)$. (The reservation follows the initial demand of the VM at deployment time t_0 and is continuously updated as the demand changes.) We denote the set of VMs placed on machine n by $A_n(t)$. The resource allocation for links between communicating VMs is more complex to model, since a communication task requires CPU and memory. For a link between VM $i \in A_n(t)$ and VM $j \notin A_n(t)$, interface bandwidth $\lambda_{i,j}^r$ must be allocated on machine n. In case i and j are placed on the same machine, no interface bandwidth must be reserved. (See Figure 2b.) In addition to bandwidth resources, the communication between VMs i and j requires CPU and memory capacity. We denote the communication cost for machine n as $\omega_n^{net}(t)$ for CPU and $\gamma_n^{net}(t)$ for memory, and we model these variables as increasing proportional to the aggregate bandwidth consumed by inter-VM communication. ($\omega_n^{net}(t)$ and $\gamma_n^{net}(t)$ are incurred by both inter-machine as well as intra-machine communication of VMs). For formal reasons we set $\lambda_n^{net}(t) = 0$.

$N, M(t), \mathcal{B}(t), \mathcal{G}(t)$	the set of machines, VMs, VMs with best-effort service, and VMs with guaranteed service
$\theta_m^d, \theta_m^l, \theta_m^r, \theta_m^v$	demand, demand limit, reservation and relative reservation for VM m and resource type $\theta \in \{\omega, \gamma, \lambda\}$
$\lambda_{i,j}^d, \lambda_{i,j}^l, \lambda_{i,j}^r, \lambda_{i,j}^v$	network bandwidth demand, demand limit, reservation and relative reservation for a link between VMs i and j
$\theta_n^c, \theta_n^u, p_n^{max}, A_n(t)$	capacity and utilization of resource $\theta \in \{\omega, \gamma, \lambda\}$, maximum power consumption, and set of VMs of machine n
$\theta_n^{net}(t)$	communication cost of resource $\theta \in \{\omega, \gamma, \lambda\}$ on machine n; we define $\lambda_n^{net}(t) = 0$ for notational simplicity

Table 1: Summary of notations

Aggregate resource reservation is less than capacity $$\theta_n^{net}(t) + \sum_{m \in A_n(t)} \theta_m^r(t) \leq \theta_n^c \text{ for } \theta \in \{\omega, \gamma, \lambda\}, n \in N \quad (1)$$	Each VM is placed on exactly one machine $$\bigcup_{n \in N} A_n(t) = M(t) \text{ and } A_i(t) \cap A_j(t) = \emptyset \; \forall i \neq j \quad (2)$$
Reserved resources are at least equal to demand $$\theta_m^r(t) \geq \theta_m^d(t) \text{ for } \theta \in \{\omega, \gamma, \lambda\} \quad (3)$$	Reserved network resources are at least equal to demand $$\lambda_m^r(t) \geq \lambda_m^d(t) \quad (4)$$
(3) holds specifically for guaranteed service VMs $$\theta_m^r(t) \geq \theta_m^d(t) \text{ for } \theta \in \{\omega, \gamma, \lambda\} \text{ and } m \in \mathcal{G}(t) \quad (5)$$	

Table 2: Constraints of the optimization problems for resource allocation

To simplify notation, we use for a VM m in machine n, its network bandwidth demand and its network bandwidth reservation respectively as $\lambda_m^d(t) = \sum_{i \notin A_n(t)} \lambda_{m,i}^d(t)$ and $\lambda_m^r(t) = \sum_{i \notin A_n(t)} \lambda_{m,i}^r(t)$.

2.3 Optimizing Resource Allocation for Various Management Objectives

Taking into account the objectives of the cloud provider, who owns the infrastructure, and the objectives of the customer, who has resource demands on the VDC, we model resource allocation as an optimization problem with a real-valued objective function f and a set of constraints. We associate a management objective with an instance of such an optimization problem and give four specific examples of management objectives in this section, which refer to balanced load, energy efficiency, fair resource allocation, and service differentiation. Variations of these objectives, as well as additional objectives, for instance, revenue maximization, can be expressed using our framework.

The input variables to such an optimization problem include the set of VMs $M(t)$ with their associated resource demands (CPU, memory and network) and the set of machines N with their associated capacities. The solution of the optimization problem includes, for each machine n, the set $A_n(t)$ of VMs placed on that machine, and the resource reservations for each of these VMs. Note that for the modeling of the allocation problem, the VDCs are not visible, since resource demands and reservations are expressed in terms of VMs.

The objective function of an optimization problem corresponds to a specific management objective. Table 2 lists the constraints for the management objectives discussed in this paper. In this table, (1) states that the aggregate resource reservation on a machine must be smaller than its capacity; (2) requires that each VM is placed on exactly one machine; (3) requires that the resources reserved (CPU, memory, network) for a VM be at least its demand; (4) requires that specifically network resources reserved be at least its demand; (5) requires that (3) holds specifically for VMs belonging to a guaranteed service class $\mathcal{G}$.

Balanced Load

We define the management objective of balanced load as that of balancing the utilization of resources across all machines for the provider, while reserving resources sufficient to the demand of the customers' VMs.

We define the utilization of resource $\theta \in \{\omega, \gamma, \lambda\}$ on machine n at time t as the sum of the reserved resources for all VMs on machine n, including the resources needed for communication, normalized by the machine capacity: $\theta_n^u(t) = \frac{\theta_n^{net}(t) + \sum_{m \in A_n(t)} \theta_m^r(t)}{\theta_n^c}$. Note that $\lambda_n^{net}(t) = 0$ for formal reasons of notational simplicity. The optimization problem for the management objective is:

minimize:

$$f = \sum_{\theta \in \{\omega,\gamma,\lambda\}} k^\theta \sum_{n \in N} \theta_n^c \theta_n^u(t)^2 \qquad (6)$$

subject to: (1), (2), (3)

The objective function is minimized when the utilization θ_n^u is identical on all machines n and for all resource types θ. The parameters $k^\theta \geq 0, \theta \in \{\omega, \gamma, \lambda\}$ are weights that indicate the relative importance as to whether the utilization of a specific resource is balanced or not. Setting $k^\theta = 0$ means that utilization of resource θ does not need to be balanced. The constraints (1,2,3) relate to capacity limit, placement of VMs and resource demands of VMs. (See Table 2.)

Energy Efficiency

We define the management objective of energy efficiency as that of minimizing the power consumption of the machines in the cloud, while reserving resources sufficient to the demand of the customers' VMs. Following the recent literature (e.g., [12]), we model the power consumption p_n of machine n as a linear function of its CPU utilization ω_n^u, the relative power consumption when idle I_n, and the maximum power consumption of the machine p_n^{max}:

$$p_n(t) = \begin{cases} p_n^{max}(I_n + (1 - I_n)\omega_n^u(t)) & \text{if } \omega_n^u(t) > 0 \\ 0 & \text{otherwise} \end{cases} \qquad (7)$$

The optimization problem for the management objective is:

minimize:

$$f = \sum_{n \in N} p_n(t) \qquad (8)$$

subject to: (1), (2), (3)

The objective function is minimized when the aggregate power consumed by the non-idle machines is minimal. For a homogeneous system, this is the case when the demand of the VMs is consolidated on a minimal set of machines. The constraints are identical to the management objective of balanced load.

Fair Resource Allocation

We define the management objective of fair resource allocation as that of maximizing the aggregate reservation of CPU and memory resources of the machines, while sharing the resources fairly among the customers' VMs. We say that VMs share resources fairly, if for each VM m the resource reservation $\theta_m^r(t)$ divided by the demand for the resource $\theta_m^d(t)$ is equalized. Let $\theta_m^v(t) = \frac{\theta_m^r(t)}{\theta_m^d(t)}$. Then, the optimization problem for the management objective becomes:

minimize:

$$f = -\left(\sum_{\theta \in \{\omega,\gamma\}} k^\theta \sum_{m \in M(t)} \theta_m^d(t) \sqrt{\theta_m^v(t)} \right) \qquad (9)$$

subject to: (1), (2), (4)

The objective function is minimized when the relative resource reservation $\theta^v_m(t)$ is identical for all VMs m and resources $\theta \in \{\omega, \gamma\}$. The parameters $k^\theta \geq 0, \theta \in \{\omega, \gamma\}$ are weights that indicate the relative importance of allocating a specific resource fairly. Setting $k^\theta = 0$ means that resource θ does not need to be allocated fairly. The constraints (1,2,4) relate to capacity limit, placement of VMs and network resource demands of VMs.

Service Differentiation

We define the management objective of service differentiation as that of maximizing the aggregate reservation of CPU and memory resources of the machines, while supporting two service classes for the customers' VMs. The first class of service is a *guaranteed service* class $\mathcal{G}$ whereby each VM receives the resources it demands or more. The second class is a *best-effort service* class $\mathcal{B}$ whereby the unreserved CPU and memory resources of the machines are fairly shared by the VMs in this class. (The notion of fairness is the same as above). The optimization problem for the management objective is:

minimize:

$$f = -\left(\sum_{\theta \in \{\omega,\gamma\}} k^\theta \sum_{m \text{ of } \mathcal{B}(t)} \theta^d_m(t)\sqrt{\theta^v_m(t)}\right) \qquad (10)$$

subject to: (1),(2),(4),(5)

The objective function is similar to (9) and is minimized when the relative resource reservation $\theta^v_m(t)$ is identical for all VMs m of service class $\mathcal{B}$ and resources $\theta \in \{\omega, \gamma\}$. The parameters $k^\theta \geq 0, \theta \in \{\omega, \gamma\}$ are weights that indicate the relative importance of allocating a specific resource fairly. The constraints (1,2,4,5) relate to capacity limit, placement of VMs, network resource demands of all VMs and the resource demands of VMs in service class $\mathcal{G}$.

3 An Architecture for Resource Allocation and Management

Figure 3 shows an architecture for resource allocation in an IaaS cloud, which we have adapted from [5]. The management station provides an interface for the cloud manager, which displays key management metrics and allows to set and adjust management objectives at run-time. The Resource Allocation System is designed around a set of cooperating controllers and state estimator components. When a customer sends a request for new VDC to the customer-provider interface, the request first passes through the Request Flow Profiler, which gathers statistics. The Admission Controller then decides whether to accept or reject the VDC request. If accepted, the request is passed on to the Placement Scheduler, which places the VMs of the VDC onto machines of the cloud. Then, the Local Scheduler allocates physical resources to each VM. Finally, the Power Manager puts on standby unused machines and reactivates them as needed.

This work focuses on the design and evaluation of the Placement Scheduler and the Local Scheduler. The functionality of the other components is outlined.

3.1 State Estimators

The lightly colored components in the architecture perform state estimation and prediction. The Request Flow Profiler collects request statistics, which are used by the admission controller to decide whether to accept or reject a VDC request. The Utilization Estimator monitors the resource utilization of the machines and the VMs. The Demand Profiler processes current and historical utilization estimates, in order to produces demand forecasts for the running VMs. These forecasts are then used by all controllers to make resource allocation decisions.

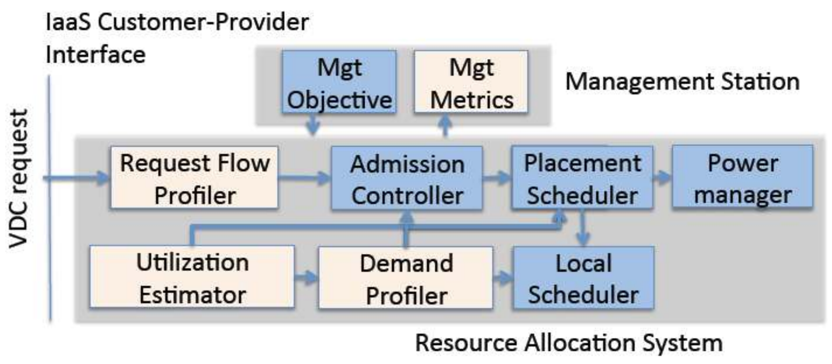


Fig. 3: Architecture for resource allocation in an IaaS cloud

3.2 Admission Controller

The Admission Controller accepts a VDC request if it can find a feasible solution to the problem of placing the VMs of the VDC onto the machines under the currently active management objective.

3.3 Local Scheduler

The Local Scheduler runs on each machine. Following the management objective of the cloud, it solves the optimization problem for the set $A_n(t)$ of VMs running on the local machine n. It takes as input the resource demands $\theta_m^d(t)$ of the VMs in $A_n(t)$ and the capacity of the machine n and computes the resource reservations $\theta_m^r(t)$ for the local VMs m. For example, for the management objective of balanced load, the local scheduler solves the following. (Recall that $\theta_n^u(t) = \frac{\theta_n^{net}(t)+\sum_{m\in A_n(t)}\theta_m^r(t)}{\theta_n^c}$)

minimize:

$$f = \sum_{\theta\in\{\omega,\gamma,\lambda\}} k^\theta \theta_n^c \theta_n^u(t)^2 \tag{11}$$

subject to: (1),(2),(3)

The constraints (1),(2),(3) express local constraints. For the other management objectives discussed in Section 2, the local optimization problems can be obtained in a similar way from the global optimization problem expressed in (8), (9) and (10). In contrast to the global optimization problems, the local problems are not hard to solve.

3.4 Placement Scheduler

Following our approach in [5], we split the functionality of the Placement Scheduler in Figure 3 into two components: an Initial Placement Controller, which performs placement for new VDC requests, and a Dynamic Placement Controller, which dynamically adapts an existing placement in response to change in VDC demand, cloud configuration, etc.

The Initial Placement Controller takes a VDC request and places the associated VMs on the machines of the cloud, following the active management objective. To compute the allocation, it solves the optimization problem associated with the management objective, with the additional constraint that the VM placed prior to this VDC request are not moved.

The Dynamic Placement Controller dynamically adapts the VM placement, such that the management objective is achieved at all times. Changing an existing placement can be realized through live-migration, which is resource-intensive [13]. Specifically, moving a VM from one machine to another machine, without interrupting its execution, consumes a significant amount of network bandwidth along the communication path between the two machines. Also, to a lesser extent, CPU resources on the source machine and memory resources on the destination machine are required for live-migration.

In this work, we take the approach that a small fraction k_θ^{LM} of machine resources is always available for live-migration, even under machine overload, since this allows the system to move to a better state, i.e., to achieve the management objective. If $k_\theta^{LM} = 0$, then the Dynamic Placement Controller will be unable to improve the state of overloaded machines.

We view the system as evolving at discrete points in time $t = 0, 1, \dots$ whereby a change occurs at each time t, e.g., a change in demand captured by $\theta^d(t)$. In response to such a change, the Dynamic Placement Controller computes a new placement $\{A_n(t)\}_{n \in N}$ that minimizes the objective function f for the currently active management objective. In addition to the constraints for the optimization problem discussed in Section 2, the new placement must be reachable within a given time period from the previous placement $\{A_n(t-1)\}_{n \in N}$ with the resources available for live-migration, which are for machine n

$$\max(k_\theta^{LM} \theta_n^c, \theta_n^c(1 - \theta_n^u(t-1))), \theta \in \{\omega, \gamma, \lambda\}. \tag{12}$$

In the following, we refer to this additional constraint as the *constraint for live migration.*

4 Placement under Management Objectives

In this section, we describe the design and functionality of the Placement Scheduler in our architecture (see Figure 3), whose task is to continuously solve an optimization problem that is associated with a specific management objective (see Section 3).

We provide a generic solution that is applicable to a large class of management objectives, including those specified in Section 2. This class contains the objectives whose associated objective functions f satisfy the following property:

$$\boxed{\begin{array}{l} f \text{ is a function on the state space of } n \text{ machines, i.e., } f = f_n : S^n \to \mathbf{R}. \\ \text{For any } i < j \in \{1, \dots, n\} \text{ and any } s_i, s_j, t_i, t_j \in S, \text{ the following implication holds} \\ f_2(s_i, s_j) > f_2(t_i, t_j) \to f_n(s_1, \dots, s_i, \dots, s_j, \dots, s_n) \geq f_n(s_1, \dots, t_i, \dots, t_j, \dots, s_n) \end{array}} \quad (\mathcal{P})$$

If f satisfies $\mathcal{P}$ for the protocol that underlies the Dynamic Placement Controller, then the protocol converges (see below). The protocol implements an iterative descent method, and $\mathcal{P}$ ensures that an iteration can be performed using two machines only for the descent method to work on the state space of n machines.

It is straightforward to verify that, for positive state values, some commutative and associative functions including *sum*, *product*, *max*, and *min* satisfy $\mathcal{P}$. Also, if both f and g satisfy $\mathcal{P}$, so do $f + g$, $f * g$, $\alpha * f$ with $\alpha > 0$, and $sum(h(s_1), \dots, h(s_n))$ with $h : S \to \mathbf{R}$. Furthermore, statistical functions, such as *average* and *variance*, as well as the objective functions of the management objectives discussed in Section 2 have this property. $(-1)^n * sum(s_1, \dots, s_n)$ is an example of a function that does not satisfy $\mathcal{P}$ for $n = 3, 5, 7, ...$

4.1 Algorithm for Initial Placement Controller

A search-based solution for the problem of placing a VDC onto a set of N machines can be shown to have complexity that is polynomial with the number of machines and exponential with the size of the VDC. Such a solution may be usable in situations where the size of the VDCs and $|N|$ are small and where the placement of VDCs does not occur very often. However, the solution becomes unfeasible when the size of the cloud and that of the VDCs grows to sizes corresponding to today's and future cloud environments. We propose the following simple and scalable algorithm instead.

The algorithm is inspired by the so-called 'power of two random choices' [14]. In this algorithm, placement of a VDC is done one VM at a time. (VMs with colocation constraints are merged into one bigger VM with the aggregate resource demands of those VMs). For each VM that is to be placed, the controller selects d random machines, and out of these machines, places the VM on the machine that minimizes the value of the objective function as computed over the d machines. If no feasible placement is found, the algorithm tries further d random machines a preconfigured number of times, before rejecting the VDC. The Initial Placement Controller can run on all machines of the cloud for reasons of scalability.

4.2 Algorithm for Dynamic Placement Controller

We propose the use of GRMP [4], a generic and scalable gossip protocol that we developed in earlier work in order to implement the functionality of the Dynamic Placement Controller. GRMP is an *asynchronous* protocol that executes continuously, improving the allocation of cloud resources according to specified management objectives, while adapting the state of the cloud, e.g., the set $M(t)$ or the demand of VMs.

The pseudocode of GRMP is listed in Algorithm 1. The protocol follows the push-pull interaction pattern, where a machine pushes its state to another machine and pulls that machine's state, which we implement with an active and a passive thread on each machine. During an interaction between two machines, they exchange their states (including, e.g., the set of VMs placed), and each machine computes a new state and realizes the new state (step 4-6 in the active thread, step 3-5 in the passive thread) through live-migration and updating the resources allocated to the VMs it runs.

Algorithm 1 Protocol GRMP runs on each machine i

```
initialization
1: initInstance()
2: start passive and active threads;

passive thread
1: while $true$ do
2:     read current state $s_i$
3:     $s_j$ =receive($j$); send($j, s_i$)
4:     updateState($s_i, s_j$)
5:     realize $s_i$ through live-migration and
       updating resources reserved to VMs
6: end while

active thread
1: while true do
2:     read current state $s_i$
3:     $j$ =choosePeer()
4:     send($j, s_i$); $s_j$ =receive($j$)
5:     updateState($s_i, s_j$)
6:     realize $s_i$ through live-migration and
       updating resources reserved to VMs
7:     sleep until end of round
8: end while
```

GRMP is *generic* in the sense that its three abstract methods, namely `initInstance()`, `choosePeer()` and `updateState()` must be instantiated in order to implement the Dynamic Placement Controller functionality for a specific management objective. The generic functions are instantiated as follows.

4.2.1 initInstance()

This method initializes the local state s_i of the protocol on node $i \in N$. The local state includes the resource demands $\theta_m^d(t)$ of VMs $m \in A_i(t))$ and the resource capacities θ_i^c of the machine. Other objective-specific states such as the service classes or colocation/anti-colocation constraints for those VMs may also be included.

4.2.2 choosePeer()

This method returns an identifier of a machine, which the current node executes a gossip round. For all our management objectives, this function returns a machine selected uniformly at random from the set of all machines. Such a function can be implemented in a distributed manner by protocols such as [15, 16].

4.2.3 updateState(s_i, s_j)

This method is specific to the management objective in the sense that it depends on the objective function f and the constraints of the optimization problem. In addition, the resources available for live-migration (12) are taken into account. Algorithm 2 lists the pseudocode.

Given the state (s_i, s_j) of the two machines i and j, the algorithm attempts to find a new state (t_i, t_j) that *better* fits the management objective. (Whether (t_i, t_j) is a better state depends on (s_i, s_j) and the optimization problem.) The algorithm considers two situations. The first situation is one where both s_i and s_j are feasible states with respect to the local optimization problem. In this situation, (t_i, t_j) is a better state if $f(t_i, t_j) < f(s_i, s_j)$. The second situation is one where either one or both machines are in an infeasible state due to overload (i.e., in violation of either constraint (3) or (4)). We measure the overload on a machine i as $\theta_i^\delta = (\omega_i^\delta, \gamma_i^\delta, \lambda_i^\delta)$, which is the additional capacity needed to satisfy the aggregate demand on that machine, and we define the overload on (s_i, s_j) as $O(s_i, s_j) = \max(\|\theta_i^\delta\|_2, \|\theta_j^\delta\|_2)$. In this situation, (t_i, t_j) is a better state if $O(t_i, t_j) < O(s_i, s_j)$. In both situations, the algorithm attempts to compute the best state that is reachable under the constraint for live-migration. As can be seen from Algorithm 2, the algorithm contains two consecutive loops. The first loop (line 2-5) is executed if there is overload (line 1), and the algorithm attempts to minimize the overload under the constraint of live-migration. The second loop (line 8-11) is executed if there is no overload, and the algorithm attempts to minimize the objective function f under the constraint of live-migration. (The term 'attempts' refers to the fact that the problem is often NP-hard, and the complexity of its state space grows exponentially with the number of VMs.)

We now give a detailed description of the algorithm. In line 1, the algorithm determines whether the machines are in overload. The loop (lines 2-5) implements a greedy heuristic that attempts to minimize $O(s_i, s_j)$. In this loop, for each VM m on machines i and j, we compute, for the case that m is migrated from the current machine, say i, to the other machine, say j, the difference in the available capacity on the destination machine , $\theta_j^\Delta(m)$ respectively, as well as the difference of the overload metric, $\Delta O(m) = O(s_i, s_j) - O(s_i^{-m}, s_j^{+m})$. In step 3, we select a VM m^* out of all VMs m that maximizes $\frac{\Delta O(m)}{|\theta_j^\Delta(m)|_2}$, under the constraints that (1) $\Delta O(m) > 0$ holds and (2) the state is reachable with resources available for live-migration $(\theta_i^{LM}, \theta_j^{LM})$. If such a VM is found, the resources available for live-migration are reduced accordingly, and the process is repeated in the new state $(s_i^{-m^*}, s_j^{+m^*})$ (step 4). Otherwise, the loop terminates (step 5). (This loop eventually terminates since $\Delta O(m^*) > 0$, and the set of possible placements is finite.)

The second loop is executed only if there is no overload on both machines (step 7). The loop (lines 8-11) implements a greedy heuristic, similar to the one above, that attempts to minimize $f_2(s_i, s_j)$. When a VM m on machine i is migrated to machine j, we denote the difference in objective function with $\Delta f(m) = f_2(s_i, s_j) - f_2(s_i^{-m}, s_j^{+m})$. In line 9, a VM m^* that maximizes $\frac{\Delta f(m)}{|\theta_j^\Delta(m)|_2}$ is selected, under the constraints that (1) $\Delta f(m) > 0$ holds, (2) the migration leads to a feasible state (s_i^{-m}, s_j^{+m}), and (3) the state is reachable with resource available for live-migration. If such a VM is found, the resources available for live-migration are updated accordingly, and the process is repeated in the new state (step 10). Otherwise, the loop terminates (step 11).

Algorithm 2 `updateState()`

```
1: if $O(s_i, s_j) > 0$ then
2:   repeat
3:     find a VM $m$ from $A_i \cup A_j$ that maximizes $x = \frac{\Delta O(m)}{|\theta_j^\Delta(m)|_2}$ under the condition that $x > 0$ and
       $\min(\min(\theta_i^{LM}(m)), \min(\theta_j^{LM}(m))) > 0$
4:     move $m$ and set $(\theta_i^{LM}, \theta_j^{LM}) = (\theta_i^{LM}(m), \theta_j^{LM}(m))$
5:   until no such $m$ is found
6: end if
7: if $O(s_i, s_j) = 0$ then
8:   repeat
9:     find a VM $m$ from $A_i \cup A_j$ that maximizes $x = \frac{\Delta f(m)}{|\theta_j^\Delta(m)|_2}$ under the condition that $x > 0$,
       $\min(\min(\theta_i^{LM}(m)), \min(\theta_j^{LM}(m))) > 0$ and the resulting state is feasible
10:    move $m$ and set $(\theta_i^{LM}, \theta_j^{LM}) = (\theta_i^{LM}(m), \theta_j^{LM}(m))$
11:  until no such $m$ is found
12: end if
```

4.3 Convergence of the protocol

It is not difficult to see that the protocol converges under reasonable assumptions to a local optimum. Using a distributed heuristic, the protocol implements, through a sequence of gossip interactions, a greedy approach to optimization.

We assume that the system starts in a feasible state, that the resource demand does not change, and that no VDC churn occurs. During execution, the protocol performs a sequence of gossip interactions between pairs of machines, chosen uniformly at random, at discrete times $t = 1, 2, \cdots$. The method `updateState()` defines the semantics of such an interaction. When machines i and j interact at time t, either the state of the machines changes, in which case $f_2(s_i(t), s_j(t)) < f_2(s_i(t-1), s_j(t-1))$, or it remains the same, in which case $f_2(s_i(t), s_j(t)) = f_2(s_i(t-1), s_j(t-1))$.

As f has property $\mathcal{P}$, we conclude

$$f_n\left(s_1(t-1), ..., s_i(t-1), ..., s_j(t-1), ...\right) \geq f_n\left(s_1(t), ..., s_i(t), ..., s_j(t), ...\right) \tag{13}$$

whereby $s_k(t) = s_k(t-1), \forall k \neq i, j$. Therefore, the sequence $\{f_n(t)\}_t$ $t = 1, 2, \cdots$ is monotonically decreasing, and it converges as it is lower bounded.

Under stronger (and often less realistic) assumptions, one can prove exponential convergence of the protocol, as we have done in [2], considering CPU resources only and assuming that the demand for a software module can be split among machines running the same module.

5 Evaluation through Simulation

We evaluate, through simulation, the dynamic placement of VDCs for large clouds under changing load and VDC churn. The simulation environment is based on PeerSim [17], a simulator for large P2P systems. For this evaluation, we extended PeerSim, to support IaaS cloud components including physical machines, virtual machines, virtual machine links, VDCs, as well as components of our architecture (Figure 3). We generate the load for the simulation, as well as, the machine capacities of the cloud infrastructure, based on an analysis and statistics from an extensive data set from a Google data center [18]. This data set includes figures about jobs (which consists of a number of tasks), the task execution times, the estimated maximum CPU and memory demand of the tasks, as well as the CPU and memory capacities of the cluster machines. For the purpose of our simulation, we associate the jobs of the Google study with applications on a cloud and the tasks of the study with VMs of the VDCs that run these applications.

5.1 Simulation setup

5.1.1 Architectural components

For the simulation studies, the components of our architecture work as follows. The function of the Request Flow Profiler does not influence the simulation. The Utilization Estimator is perfect in the sense that it produces the correct resource utilization of the machines and VMs at all times. The Demand Profiler simply returns the current demand of the VDCs as the forecasted demand. The Admission Controller accepts all requests. (As a consequence, the IaaS infrastruture can become overloaded during experiments.) The Power Manager puts on standby those machines that do not run VMs; it reactivates standby machines when a VM is placed on it. The Local Scheduler (Section 3.3), the Initial Placement Controller (Section 4.1) and the Dynamic Placement Controller (Section 4.2) are implemented as described above. For the Initial Placement Controller, we choose $d = 2$ and always accept a better allocation for a VM. During a gossip round of the dynamic placement controller, a machine select another machine uniformly at random from the set of all machines. Such a function can be implemented on top of gossip-based overlays like [15, 16].

5.1.2 Applications

We assume that two types of applications are run on the VDCs (see Section 2.1). First, we consider interactive applications, whereby the CPU and network demand changes overtime, while the memory demand remains constant. The set of interactive applications does not change during a simulation run. Second, we consider compute-intensive applications, whereby the demand for CPU, memory, and network bandwidth does not change. The lifetime of such an application follows a truncated power-law distribution with exponent 2 (cf. [18]), incremented by 10 minutes (to account for the time of copying the VM image as well as booting up and powering down a VM), and truncated to 24 hours (the length of a simulation run). The requests for these applications arrive following a Poisson process, whereby the rate is computed, using Little's formula, in such a way that the expected number of VDCs on the cloud has a desired value. The type of application that is requested, i.e., interactive or compute-intensive, is chosen with equal probability.

5.1.3 VDC Types

For the purpose of this evaluation, we assume that the above applications are run on multi-tiered VDCs, which consist of several layers of identical VMs, whereby each VM is connected to the VMs in adjacent layers. In the course of a simulation run, VDCs are randomly created as follows. The number of VMs in a VDC follows a truncated power-law distribution with exponent 2, minimum 1, and maximum 30 (choice of exponent follows recommendation in [18]). The number of tiers is 1, 2, or 3, with equal probability, and the number of VMs in each tier is the total number of VMs divided by the number of tiers, rounded down to an integer.

5.1.4 Resource Demand

We express the resource demand of applications in terms of the CPU and memory demand of the VMs that run these applications, as well as the network demand of the links connecting the VMs. To each VM m, we assign an expected maximum CPU and memory demand, $\overline{\omega}_m$ and $\overline{\gamma}_m$, following the distribution of task demands in [18]. (We scale the relative demand with 32-core CPU and 64GB memory.) While we assume that the demand of a VM that runs a compute-intensive application equals the maximum expected demand and does not change over time, the CPU demand of a VM associated with an interactive application changes periodically according to $\omega_m(t) = \frac{\overline{\omega}_m}{2}(1 + u_m \sin(\frac{2\pi t}{86400} - 2\pi s_m))$, whereby $u_m, s_m \in [0, 1]$ is selected uniformly at random; the memory demand remains constant and equals to the expected maximum demand.

For both types of applications, we assume that the maximum network demand of a link depends on the maximum CPU and memory demands of the connected VMs as follows. If $\overline{\omega}_m \leq 0.5$core and $\overline{\gamma}_m \leq 0.5$GB, then the expected maximum network demand $\overline{\lambda}_l$ for all links l connected to VM m is chosen as 1 Mb/sec. Otherwise, $\overline{\lambda}_l = 10$Mb/sec with probability 0.9 and 100Mb/sec with probability 0.1. Then, the actual network demand is assumed to be equal to the maximum relative CPU demand of the two VMs connected by the link, multiplied with $\overline{\lambda}_l$.

5.1.5 Capacities of cloud machines

In our simulation, we consider four types of physical machines. The relative capacities of these machine, together with the probability according to which they are chosen for a configuration, are given in Table 3.

Type	CPU	RAM	Network	Probability
Type 1	0.5	0.25	1	31%
Type 2	0.5	0.5	1	55%
Type 3	0.5	0.75	1	8%
Type 4	1	1	1	6%

Table 3: Relative capacities of cloud machines

The distribution of the machine capacities, except network capacities, follows the statistics from [18]. We assume that the maximum power consumption of a machine is proportional to its CPU capacity. We also assume that the CPU communication cost of machine n, $\omega_n^{net}(t)$, is proportional to the network utilization λ_n^u and at most one core, while the memory communication cost $\gamma_n^{net}(t)$ is 0.

5.1.6 Load on the cloud infrastructure

For the simulation study, the concept of the aggregate load on the cloud infrastructure is useful. We define the CPU load factor (CLF) of the cloud as the ratio of the total CPU demand of the VMs to the total CPU capacity of the machines in the cloud. We define the memory load factor (MLF) and network load factor(NLF) in an analogous fashion.

5.1.7 Resource consumption during live-migration

Our simulation captures the resource consumption of VM migration. We assume that migrating a VM requires (1) CPU resources at the sender machine, namely $\overline{\omega}_m/8$, (2) memory resources at the receiver machine, namely $\overline{\gamma}_m$, and (3) network resources on both machines, namely twice the memory size of the VM, divided by the transfer time (which we assume to be 1 minute).

We assume that up to 10% of the resource capacities of a machine are available for live-migration (see Section 3.4 and (12)), and that live-migration of a VM can be executed within a protocol round. If the resources are not available, a VM is not migrated.

5.1.8 Evaluation metrics

During each simulation run, we measure the following two metrics. The first metric is *effectiveness*, which captures how well the management objective is achieved by the resource allocation system. For the objective of balanced load, the effectiveness is measured as the mean of three components, namely the coefficient of variation of CPU, memory, and network utilization of the physical machines. (The coefficient of variation of a set of values is the standard deviation divided by the mean.) We call this metric also *unbalance metric.* For the energy efficiency objective, the effectiveness is measured as the aggregate power consumed by all active machines (i.e. the machines that are not on standby), divided by the maximum power consumption of all machines. We call this metric also *relative power consumption metric.* For the objective of fair resource allocation, the effectiveness metric is measured as the mean of two components, namely the coefficient of variation of relative CPU and memory reservations of the virtual machines. We call this metric also *unfairness metric.* For the objective of service differentiation, the effectiveness metric is measured as the unfairness metric applied to the best-effort VMs. We call this metric also *best-effort unfairness metric.*

The second metric is the *constraint violation metric*, which is defined as the fraction of machines that violate the constraint (1) in Table 2, i.e., the fraction of machines that allocate resources beyond their capacities.

5.1.9 Simulation execution

We simulate the evaluation of the resource allocation system over time. The simulation processes a sequence of events, which are of three types: (1) a machine executes a round of the gossip protocol, (2) a new application (i.e., a new VDC) is placed onto the cloud or an application terminates, (3) the load of an (interactive) application changes. Before the simulation starts, we compute an initial placement of a set of applications (i.e., their respective VDCs). The gossip protocol is executed in an asynchronous fashion, which means that the start of a round is not synchronized among machines.

After the start of a simulation run, the system evolves over a warmup period of 30 rounds (of 30 seconds each), after which the evaluation metrics are sampled at the end of each round. A simulation run extends over 24 hours, i.e., 2880 rounds. As measurement result, we report the average of the evaluation metrics over one run. For most of the scenarios, the number of machines is 10,000, which is also the number of initially placed VDCs.

We evaluate the system in five scenarios. In four of them, we evaluate the effectiveness and quality of resource allocation under various load factors, for a specific management objective and system size. In the last scenario, we evaluate the resource allocation system for different system sizes, while keeping the load constant.

5.2 Simulation results

5.2.1 Performance under Balanced-Load Objective

We evaluate the system under the objective of balanced load for $k^\theta = 1$, $\theta \in \{\omega, \gamma, \lambda\}$ (see (6)). We execute 50 runs, for CLF values of {0.15, 0.35, 0.55, 0.75, 0.95}, NLF values of {0.3, 0.6, 0.9, 1.2, 1.5} and MLF values of {0.25, 0.5}.

Figure 4a shows the measurements of the unbalance metric, for MLF=0.5. A value of 0 means that the system has been perfectly balanced throughout the run, and that all allocations produced by the protocol minimize (6). The figure also includes the metric for an ideal system, which can be imagined as one where resource demands can be arbitrary split across machines. The ideal system gives a lower bound to optimal metric (we cannot compute the optimal metric due to the combinatorial nature of the optimization problem).

The figure suggests that the unbalance metric does not significantly change with change in CLF values. This is the result of our design, which reserves resources on each machine for VM migration, enabling live-migration even under high loads. The figure also shows that the unbalance is low when the network is underloaded ($NLF \leq 0.9$), and the unbalance increases when the network is overloaded ($NLF \geq 1.2$). This is expected, since balancing the load is generally more difficult, when the network is overloaded. Finally, Figure 5a shows that, when the network is overloaded, constraint violation approaches 1.

The results for MLF=0.25 (not shown here) are qualitatively similar to MLF=0.5 discussed above, with unbalance and constraint violation metric values generally smaller. The 95% confidence intervals are within 0.01% for all results stated above.

We draw the conclusion that our resource allocation system performs well under the objective of balanced load for all CLF values investigated and $NLF \leq 0.9$.

5.2.2 Performance under Energy Efficiency Objective

We evaluate the system under the objective of energy efficiency for the same set of parameters as in Section 5.2.1. Figure 4b shows the measurements of the relative power consumption metric, for MLF=0.5. A value of 1 means that the system consumes the maximum power possible. The figure also includes the metric for an ideal system, considering only CPU and memory demand. (We do not have a reasonable method for including network demand.)

The figure shows that for NLF=0.3, the power consumed by the system is close to that of the ideal system. However, the distance between the power consumption of the ideal system and that of our system increases with increasing NLF since the ideal system does not take into account network demand. Figure 5b shows that the constraint violation metric has a qualitatively similar behavior to that of the balanced-load objective. The 95% confidence intervals are within 0.01% for all results stated above.

The results for MLF=0.25 (not shown here) are qualitatively similar to MLF=0.5 discussed above, with relative power consumption and constraint violation metric values generally smaller.

We draw the conclusion that our system performs well under the objective of energy efficiency when $NLF \leq 0.9$.

5.2.3 Performance under Fairness Objective

We evaluate the system under the objective of fair resource allocation for $k^{\theta} = 1$, $\theta \in \{\omega, \gamma, \lambda\}$ (see (9)). We execute 50 runs, for CLF values of {0.55, 0.75, 0.95, 1.15, 1.35}, NLF values of {0.3, 0.6, 0.9, 1.2, 1.5} and MLF values of {0.25, 0.5}.

Figure 4c shows the measurements of the unfairness metric, for MLF=0.5. A value of 0 means that the resource allocation is perfectly fair. The figure also includes the unfairness metric for the ideal system. The figure suggests that the unfairness metric does not significantly change with change in CLF. This is the result of our design, which reserves resources on each machine for VM migration, allowing to achieve fairness even under overload conditions. The figure also shows that the unfairness metric increases with NLF beyond NLF=0.9. Figure 5c shows that the constraint violation metric does not increase with increasing CLF as it does with NLF. This is due to the fact that CPU overload does not result in constraint violation but network overload does (see the constraints of (9)).

The results for MLF=0.25 (not shown here) are qualitatively similar to MLF=0.5 discussed above , with unfairness and constraint violation metric values generally smaller. The 95% confidence intervals are within 0.01% for all results stated above.

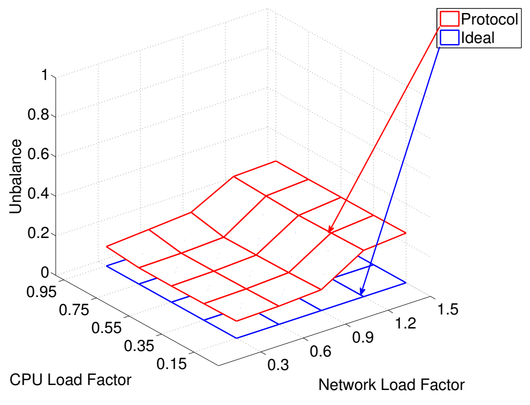
We draw the conclusion that our resource allocation system performs well under the objective of fair resource allocation when the network is not overloaded, i.e., $NLF \leq 0.9$.

5.2.4 Performance under Service Differentiation Objective

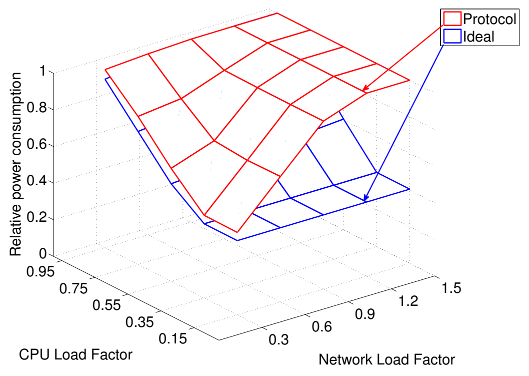
We evaluate the system under the objective of service differentiation for $k^\theta = 1$, $\theta \in \{\omega, \gamma, \lambda\}$ (see (10)). Interactive applications are given guaranteed service while compute intensive applications are given best-effort service. We execute 50 runs, for CLF values of {0.55, 0.75, 0.95, 1.15, 1.35}, NLF values of {0.3, 0.6, 0.9, 1.2, 1.5} and MLF values of {0.25, 0.5}.

Figure 4d shows the measurements of the unfairness metric for best-effort VMs, for MLF=0.5. The figure also includes the unfairness metric for the ideal system. Qualitatively, the figure is similar to that for the fairness objective, with the unfairness metric generally smaller and the curves smoother for the fairness objective. We explain this by the fact that the resources that are shared fairly among best-effort VMs are those that are unused by guaranteed service VMs, which are in generally more uneven and hence difficult to share fairly compared to the case where entire machines are shared. Figure 5d shows the property of constraint violation is also qualitatively similar to that of fairness objective. The results for MLF=0.25 (not shown here) are qualitatively similar to MLF=0.5 discussed above, with unfairness and constraint violation metric values generally smaller. The 95% confidence intervals are within 0.01% for all results stated above.

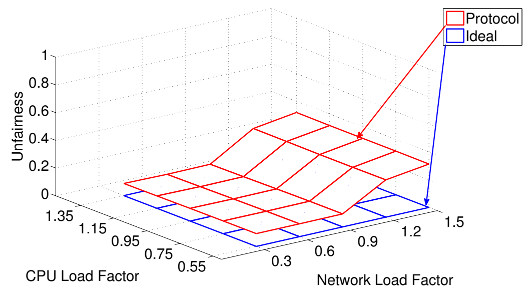
We draw the conclusion that when $NLF \leq 0.9$, our protocol performs well: guaranteed service VMs have their demands satisfied (i.e., no constraint violation) while remaining resources are shared fairly among best effort VMs (i.e., low unfairness metric).



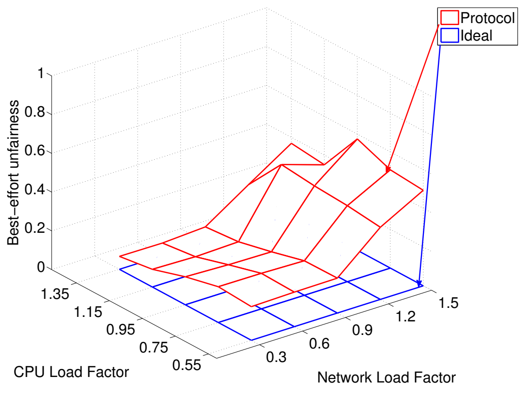
(a) Balanced load objective



(b) Energy efficiency objective

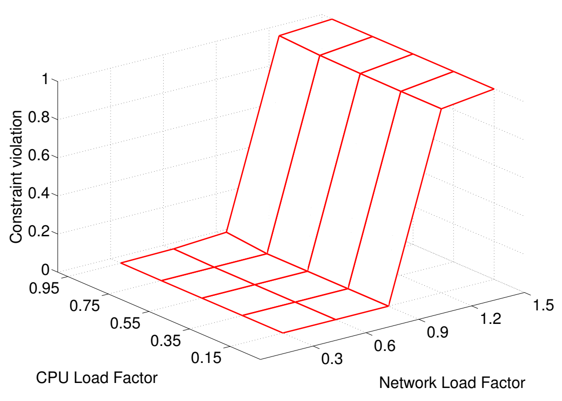


(c) Fairness objective

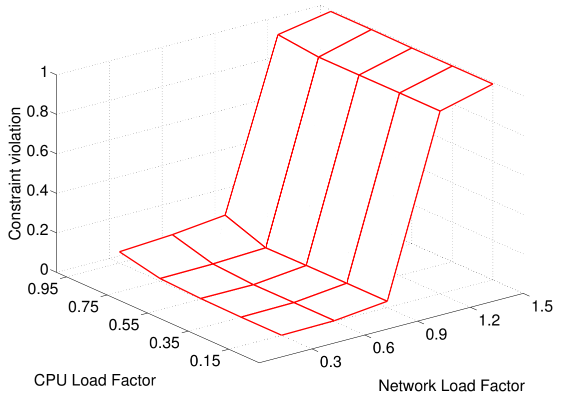


(d) Service differentiation objective

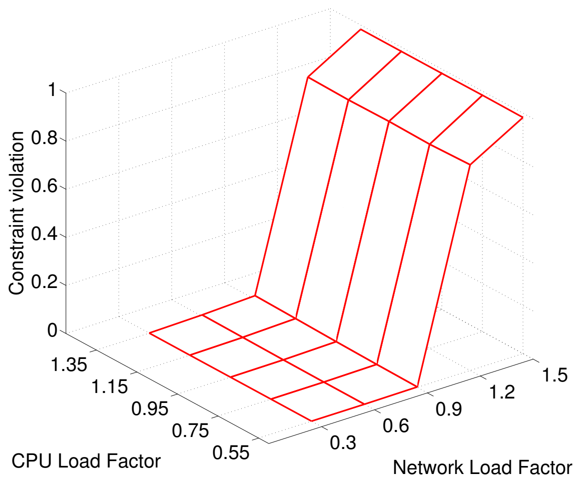
Fig. 4: Effectiveness of the protocols for different management objectives



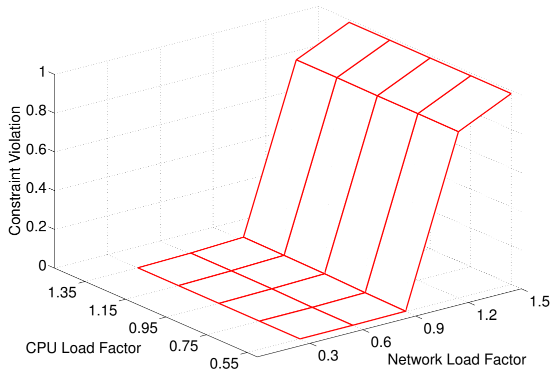
(a) Balanced load objective



(b) Energy efficiency objective



(c) Fairness objective



(d) Service differentiation objective

Fig. 5: Constraint violation of the protocols for different management objectives

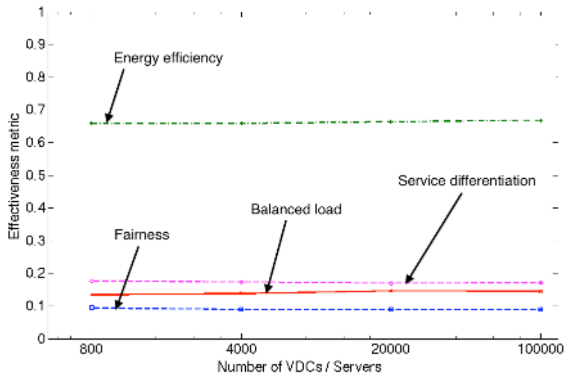
5.2.5 Scalability of the Resource Allocation System

We evaluate the scalability property of the resource allocation system under the various objectives by measuring the corresponding performance metrics for CLF=NLF=MLF=0.5, while varying the number of VDCs (and machines) to 800, 2000, 4,000, 20,000, and 100,000.

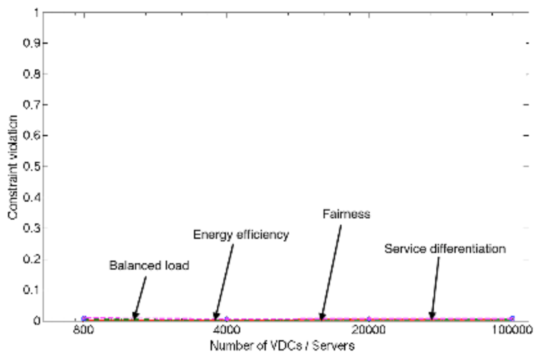
Figure 6 shows the results, which indicates that all metrics considered for all objectives do not change significantly with increasing system size. This is a property of GRMP-based protocols, which we observe in [3, 4] and prove in [2] for specific management objectives. We draw the conclusion that the resource allocation system is scalable to at least 100,000 machines and VDCs, under the four management objectives considered in this work.

6 Related Work

The work on resource allocation to VDCs presented in this paper is unique and novel in the sense that it combines the following properties. The solution supports a broad range of management objectives with a single algorithm that performs adaptive placement. The solution is fully decentralized and highly



(a) Effectiveness metrics



(b) Constraint violations

Fig. 6: Scalability of the protocols for two key metrics

scalable, to at least 100,000 machines. Finally, the placement algorithm jointly allocates both compute and networking resources.

Most works in the literature, for instance [19–27], address only a single management objective. The result in [28] supports multiple management objectives, but each objective with a different algorithm. In contrast, the work in this paper covers multiple objectives with a single algorithm, and switching between management objectives is achieved by switching the objective functions f in the algorithm.

All works discussed above rely on a centralized scheduler for placement, which becomes a bottleneck and prevents the solutions to scale to large systems. [29] proposes a decentralized scheduler but requires an oracle that has knowledge of available resources for all machines in the data center. In contrast, our solution includes a decentralized scheduler based on a gossip protocol, which scales up to 100,000 machines, as our simulation results demonstrate.

The problem of allocating compute resources, i.e., placing VMs, in a cloud has been extensively studied over the last years (e.g., [30–33]). The same is true for networking resources in the cloud environment (e.g., [34–36]). More recently, attention has focused on joint allocation of compute and network resources. In these works, one can find two different approaches to abstract the network for the purpose of resource allocation. Some works consider the link bandwidth between the physical machines as the resource [20, 21, 24], while others model the network access capacity of a machine [22, 23]. In this work, we follow the latter approach, however, while those works base on a hierarchical network, which is a conventional network configuration, we adopt the full bisection bandwidth network, which is an emerging data center technology, because it enables a distributed computation for resource allocation.

As mentioned in Section 1, the solution presented in this paper is limited to a single data center, because we assume a full bisection bandwidth network. As a consequence, our solution is not directly applicable to resource allocation across data centers at different locations, in contrast to [37], which represents resources distributed globally as a graph, and performs resource allocation based on a heuristic solution to the graph-embedding problem.

Most results to date focus on offline (or static) placement of VDCs or VMs [22, 24, 28, 37]. More recently, more attention has been devoted to the problem of online (or adaptive) placement, whereby a sequence of VDC or VM requests is processed in an iterative fashion [26, 27]. Our work falls into the second category.

7 Conclusion and Future Work

We believe that this paper contains significant contributions towards engineering a resource management system for large-scale IaaS environments. First, unlike other results in the literature, the resource allocation system outlined here can support a *broad class of management objectives*, exemplified by the four chosen objectives for this paper. Second, we present an *improved version of GRMP*, a generic, highly-scalable gossip protocol, which dynamically compute VM placements, and which is instantiated with the objective function of the optimization problem. Simulation results shows that the protocol is very effective, except for high load, and that it scales with a high level of effectiveness up to 100,000 server machines. Third, our work considers *joint allocation of compute and network resources* (for a full bisection bandwidth network), which we believe is essential for ensuring predictable performance of cloud-based applications. Finally, we show how our protocol for dynamic adaptation *seamlessly fits into a complete resource management architecture.*

The framework presented in this work is extensible in several directions. First, the resource allocation system can support further management objectives than discussed, such as revenue maximization, with additional constraints regarding, e.g., colocation/anti-colocation of VDCs or license restrictions. Second, the framework can be extended to include additional resources types, such as storage or GPU resources, or it can be adapted to alternative resources models. Further, the architecture allows alternative algorithms to be used to implement the functions of specific controllers.

With respect to future work, we plan to study thoroughly (including analytically) how the choice of the management objective (i.e., the objective function) and how specific load patterns affect the effectiveness and the convergence speed of the protocol. An aspect of this work will be investigating to which extent the effectiveness of the protocol can be improved in the case of high load and overload. Second, we plan to extend our work to an alternative network model that is suitable for resource management across data centers and for telecom clouds. (The full bisection bandwidth model is applicable primarily within data centers.) Third, we plan to implement and evaluate our framework on an OpenStack-based cloud management platform and continue the work described in [5].

References

1. A. Gulati, G. Shanmuganathan, A. Holler, I. Ahmad, in *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing* (USENIX Association, Berkeley, CA, USA, 2011), HotCloud'11, pp. 3–3. URL http://dl.acm.org/citation.cfm?id=2170444.2170447
2. F. Wuhib, R. Stadler, M. Spreitzer, Network and Service Management, IEEE Transactions on (2012)
3. F. Wuhib, R. Stadler, M. Spreitzer, in *International Conference on Network and Service Management* (2010)
4. R. Yanggratoke, F. Wuhib, R. Stadler, in *International Conference on Network and Service Management* (2011)
5. F. Wuhib, R. Stadler, H. Lindgren, in *Proceedings of the 7th International Conference on Network and Services Management* (2012), CNSM '12
6. Netcraft. http://news.netcraft.com/archives/2013/01/07/ (2013)
7. MapReduce. http://www.mapreduce.org (2012)
8. Dryad. http://research.microsoft.com/en-us/projects/dryad/ (2012)
9. Amazon Web Services. http://aws.amazon.com/ec2/ (2012)
10. L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, I. Stoica, in *Proceedings of the 6th International COnference* (ACM, New York, NY, USA, 2010), Co-NEXT '10, pp. 16:1–16:12. DOI 10.1145/1921168.1921189
11. C. Raiciu, M. Ionescu, D. Niculescu, in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing* (USENIX Association, Berkeley, CA, USA, 2012), HotCloud'12, pp. 6–6
12. A. Verma, P. Ahuja, A. Neogi, in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware* (Springer-Verlag New York, Inc., New York, NY, USA, 2008), Middleware '08, pp. 243–264
13. A. Verma, G. Kumar, R. Koller, in *Proceedings of the 11th International Middleware Conference Industrial track* (ACM, New York, NY, USA, 2010), Middleware Industrial Track '10, pp. 11–16
14. M. Mitzenmacher, IEEE Trans. Parallel Distrib. Syst. **12**(10), 1094 (2001). DOI 10.1109/71.963420. URL http://dx.doi.org/10.1109/71.963420
15. S. Voulgaris, D. Gavidia, M. van Steen, Journal of Network and Systems Management **13**(2), 197 (2005)
16. A. Allavena, A. Demers, J.E. Hopcroft, in *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing* (ACM, New York, NY, USA, 2005), PODC '05, pp. 292–301. DOI 10.1145/1073814.1073871
17. A. Montresor, M. Jelasity, in *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on* (2009), pp. 99 –100. DOI 10.1109/P2P.2009.5284506
18. C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch, in *ACM Symposium on Cloud Computing (SoCC)* (San Jose, CA, USA, 2012)
19. V. Shrivastava, P. Zerfos, K.W. Lee, H. Jamjoom, Y.H. Liu, S. Banerjee, in *INFOCOM, 2011 Proceedings IEEE* (2011), pp. 66 –70. DOI 10.1109/INFCOM.2011.5935247
20. C. Guo, G. Lu, H.J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, Y. Zhang, in *Proceedings of the 6th International COnference* (ACM, New York, NY, USA, 2010), Co-NEXT '10, pp. 15:1–15:12. DOI 10.1145/1921168.1921188
21. H. Ballani, P. Costa, T. Karagiannis, A. Rowstron, in *Proceedings of the ACM SIGCOMM 2011 conference* (ACM, New York, NY, USA, 2011), SIGCOMM '11, pp. 242–253. DOI 10.1145/2018436.2018465. URL http://doi.acm.org/10.1145/2018436.2018465
22. M. Wang, X. Meng, L. Zhang, in *INFOCOM, 2011 Proceedings IEEE* (2011), pp. 71 –75. DOI 10.1109/INFCOM.2011.5935254
23. D. Breitgand, A. Epstein, in *INFOCOM, 2012 Proceedings IEEE* (2012), pp. 2861 –2865. DOI 10.1109/INFCOM.2012.6195716
24. X. Meng, V. Pappas, L. Zhang, in *INFOCOM, 2010 Proceedings IEEE* (2010), pp. 1 –9. DOI 10.1109/INFCOM.2010.5461930

25. G. Lee, N. Tolia, P. Ranganathan, R.H. Katz, SIGCOMM Comput. Commun. Rev. **41**(1), 120 (2010). DOI 10.1145/1925861.1925881
26. J. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, in *INFOCOM, 2012 Proceedings IEEE* (2012), pp. 2876 –2880. DOI 10.1109/INFCOM.2012.6195719
27. O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, E. Silvera, in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on* (2012), pp. 498 –506. DOI 10.1109/CCGrid.2012.119
28. D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, E. Snible, in *Services Computing (SCC), 2011 IEEE International Conference on* (2011), pp. 72 –79. DOI 10.1109/SCC.2011.28
29. Y. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, Y. Coady, in *IEEE International Conference on Cloud Computing* (2010), pp. 91 –98. DOI 10.1109/CLOUD.2010.66
30. A.J. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, W. Carithers, in *Proceedings of the International Conference on Green Computing* (IEEE, Chicago, IL, 2010). DOI http://dx.doi.org/10.1109/GREENCOMP.2010.5598294
31. G. Wei, A.V. Vasilakos, Y. Zheng, N. Xiong, J. Supercomput. **54**(2), 252 (2010). DOI 10.1007/s11227-009-0318-1
32. J. Rao, Y. Wei, J. Gong, C.Z. Xu, in *Proceedings of the Nineteenth International Workshop on Quality of Service* (IEEE Press, Piscataway, NJ, USA, 2011), IWQoS '11, pp. 31:1–31:9
33. J.Z. Li, M. Woodside, J. Chinneck, M. Litoiu, in *Proceedings of the 7th International Conference on Network and Services Management* (2011), CNSM '11, pp. 162–170
34. J. Mudigonda, P. Yalagandula, M. Al-Fares, J.C. Mogul, in *Proceedings of the 7th USENIX conference on Networked systems design and implementation* (USENIX Association, Berkeley, CA, USA, 2010), NSDI'10, pp. 18–18
35. C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, in *Proceedings of the ACM SIGCOMM 2011 conference* (ACM, New York, NY, USA, 2011), SIGCOMM '11, pp. 266–277. DOI 10.1145/2018436.2018467
36. J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, Y. Pouffary, SIGCOMM Comput. Commun. Rev. **41**(4), 62 (2011). DOI 10.1145/2043164.2018444
37. G. Koslovski, S. Soudan, P. Goncalves, P. Vicat-Blanc, in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on* (2011), pp. 153 –160. DOI 10.1109/INM.2011.5990686

Author Biographies

Fetahi Wuhib is an experienced researcher at Ericsson Research, Sweden where he conducts research on scalable technologies for large-scale cloud environments. Fetahi received a B.Sc. degree in Electrical Engineering from Addis Ababa University (2000), an M.Sc. degree in Internetworking (2005) and a Ph.D. degree in Telecommunications (2010) from KTH Royal Institute of Technology, Sweden. Prior to joining Ericsson, Fetahi was a post-doctoral researcher with the Network Management group at EES/KTH.

Rerngvit Yanggratoke is pursuing his Ph.D. under the supervision of Prof. Rolf Stadler at the School of Electrical Engineering with KTH Royal Institute of Technology in Stockholm, Sweden. He received his M.Sc. in Security and Mobile Computing from Aalto University, Finland and KTH, Sweden. His major research interests are mobile computing, distributed systems, and management of large systems.

Rolf Stadler (`www.ee.kth.se/~stadler`) is a professor at the KTH Royal Institute of Technology in Stockholm, Sweden. He holds an M.Sc. degree in mathematics and a Ph.D. in computer science from the University of Zurich. Before joining the faculty at KTH in 2001, he held positions at the IBM Zurich Research Laboratory, Columbia University, and ETH Zürich.