Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

Location-aware brokering for consumers in multi-cloud computing environments

Leonard Heilig[a,b,*], Rajkumar Buyya[b], Stefan Voß[a,c]

[a] Institute of Information Systems (IWI), University of Hamburg, Germany
[b] Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, University of Melbourne, Australia
[c] Escuela de Ingeniería Industrial, Pontificia Universidad Católica de Valparaíso, Chile

ARTICLE INFO

Keywords:
CloudSim
Cloud consumer brokering
Multi-cloud
Multi-criteria decision making
Metaheuristics

ABSTRACT

The variety and complexity in cloud marketplaces is growing, making it difficult for cloud consumers to choose cloud services from multiple providers in an economic and suitable way by taking into account multiple objectives and constraints. In this paper, we present an extension of *CloudSim* implementing cloud management functionality to enable the assessment of consumer-oriented brokering schemes. The underlying discrete-event simulation framework allows evaluating their performance in more realistic operating conditions in a repeatable manner. We integrate brokering mechanisms to support a multi-criteria location-aware selection of virtual machines in multi-cloud environments by implementing a greedy heuristic and two large neighborhood search metaheuristics. Based on microbenchmarks of real cloud offerings and a diverse set of scenarios and workloads, we conduct simulation experiments to assess the performance of our approaches. The results show that approximately 10 – 12% of the total costs can be saved by using a large neighborhood search approach compared to the greedy heuristic. Finally, we analyze and discuss the trade-off between costs and latency as well as the impact of region constraints, showing, e.g., that latency improvements often come at a high price and a greater regional flexibility can lead to latency improvements while solely optimizing costs. Using real data of cloud marketplaces, we show that the proposed *CloudSim* extension can support decision makers as a tool for assessing cloud portfolios and market dynamics.

1. Introduction

The market of cloud services is rapidly growing and increasingly attracts new market participants offering and consuming cloud services (Cisco, 2015). This leads to a wide range of cloud providers and services entailing various features, pricing models, and service levels, standing in fierce competition to each other (Do et al., 2016). To maximize the utility of consumers using cloud services, costs and risks need to be minimized while business- and application-related requirements are met. As one cloud provider may not fulfill all requirements in the most economic way, the concept of having multiple clouds has been intensively discussed in the literature. In recent years, research focused on the federation of cloud providers collaborating and forming an inter-cloud to maintain all properties of the paradigm, including the impression of unlimited resources (see, e.g., Assis and Bittencourt, 2016; Toosi et al., 2014). Only some works have studied multi-clouds from a consumer perspective. That is, consumers simultaneously use cloud services of different cloud providers to further increase the business value. The main difference to federated clouds is that cloud providers not necessarily collaborate with each other and that the consumer is aware of using services from multiple providers. Although technological barriers are slowing down the development of multi-cloud applications in practice, current research approaches are promising (Petcu, 2013). As the decision making process is already difficult when adopting different cloud services from one cloud provider, multi-cloud environments will increase the need for brokers interacting with cloud providers on behalf of consumers to match consumer requirements with available cloud services. According to Gartner (2009), consumers depend on cloud brokers to unlock the potential of publicly available cloud services, for example, in terms of costs, quality, and flexibility.

In the area of cloud computing, one of the first definitions of a cloud broker is presented in Buyya et al. (2009). According to their definition, brokers mediate between consumers and cloud providers by purchasing and subleasing capacities to consumers. The role of a broker is not limited to a third-party organization or platform, but can also appear as a tool to coordinate and assign various resource requests within organizations. Classifications of cloud brokering schemes (e.g., pro-

* Corresponding author.
E-mail addresses: leonard.heilig@uni-hamburg.de (L. Heilig), rbuyya@unimelb.edu.au (R. Buyya), stefan.voss@uni-hamburg.de (S. Voß).

http://dx.doi.org/10.1016/j.jnca.2017.07.010
Received 8 January 2017; Received in revised form 5 June 2017; Accepted 9 July 2017
Available online 11 July 2017

posed by Gartner in Plummer and Kenney, 2009) pay little attention to the optimization of cloud service discovery and selection notwithstanding the fact that it is one of the key elements to fully benefit from cloud marketplace offerings. A recent survey of existing broker solutions (Verginadis et al., 2014) identifies only a few tools with decision support capabilities, such as *RightScale* and *DBCE*. The business model of those tools is to solely provide a common platform for comparing different cloud service options (e.g., capacities, prices) based on individual consumer requirements. The same applies to real cloud marketplaces, which may provide tools to roughly estimate costs, but lack decision support for users aiming to select appropriate combinations and configurations of cloud services. Solving strategic and operational decision problems, however, typically involves multi-criteria objectives and requires efficient optimization techniques (Heilig and Voß, 2014).

In this paper, we consider a cloud brokering scheme aimed at optimizing the selection and utilization of virtual machine (VM) types offered in a cloud marketplace by multiple cloud providers. In this environment, brokers act on behalf of consumers intending to execute application and task requests in the cloud. The contributions of the paper are described as follows.

- We extend the Cloud Service Purchasing Problem (CSPP), proposed in Heilig et al., 2016, to consider not only costs when assigning cloud resources, but also the network latency between the locations of consumers and cloud locations.
- To simulate different brokering scenarios, we propose novel extensions of *CloudSim*. The extensions provide multi-cloud management functionality for embedding consumer-centric brokering schemes.
- Given those extensions, we develop an optimization component for *CloudSim* to solve the extended CSPP by embedding two large neighborhood search metaheuristics.
- Using real VM type descriptions and prices of three leading cloud providers and different workloads, we conduct a number of simulations for evaluating the performance of our approaches. The results reveal a competitive performance of the large neighborhood search approaches.
- Finally, we demonstrate and discuss the effects of different decision making preferences towards costs and latency optimization and analyze the impacts of region constraints by slightly modifying the optimization problem.

The paper is structured as follows. In Section 2, we provide a brief overview on related work focusing on optimization approaches from a consumer perspective. The extended version of the tackled optimization problem is presented in Section 3. In Section 4, we briefly explain the extensions of *CloudSim*. The adapted large neighborhood search algorithms for solving the multi-criteria optimization problem are described in Section 5. Section 6 first describes the applied methodology for data collection as well as the simulation setup. Subsequently, the results of the conducted simulation experiments are presented and discussed. Finally, conclusions and directions for future research are presented in Section 7.

2. Related work

In recent years, only a few optimization approaches have been presented to facilitate the assignment and scheduling of tasks and applications in cloud environments from a consumer perspective. While earlier works focus on a single objective, namely cost optimization (e.g., Pandey et al., 2010; Van den Bossche et al., 2010; Chaisiri et al., 2012), recent works also aim to address performance aspects in the objective function (e.g., Lucas Simarro et al., 2011; Tordsson et al., 2012; Coutinho et al., 2015; Heilig et al., 2016).

Pandey et al. (2010) propose a particle swarm optimization (PSO) approach to optimize execution and data transmission costs of workflow applications for a single-cloud environment. Van den Bossche et al. (2010) consider deadline constrained task workloads in hybrid cloud environments and propose an integer programming model to solve the related scheduling problem. Chaisiri et al. (2012) take into account both on-demand and reservation pricing. They formulate a stochastic integer programming model to support cloud resource provisioning decisions that are subject to demand and price uncertainties. The authors further propose Benders decomposition and sample-average approximation algorithms to solve the problem. Shen et al. (2013) propose online hybrid scheduling policies that minimize costs for cloud resources by making use of both on-demand and reserved pricing.

Tordsson et al. (2012) cover resource provisioning problems in multi-cloud environments based on integer programming formulations. In their model, the number of VM type instances is limited and load balancing constraints are used to balance the number of purchased VMs among different cloud locations, besides typical hardware configuration constraints. The authors emphasize the lack of studies investigating price-performance trade-offs.

Lucas Simarro et al. (2011) investigate scheduling strategies in multi-cloud environments taking into account cost and performance aspects. However, the two objectives are either taken into account separately or by considering the performance aspect as a restriction. Besides, the authors consider load balancing constraints as previously seen in Tordsson et al. (2012).

Coutinho et al. (2013) investigate the Cloud Resource Management Problem (CRMP) which is a multi-criteria optimization model aimed at reducing the financial cost and the execution time of consumer applications when selecting cloud resources. They propose an integer programming model as well as a heuristic based on the Greedy Randomized Adaptive Search Procedure (GRASP). This work is extended in Coutinho et al. (2015), where the authors consider federated cloud environments by including the communication costs between tasks deployed in different clouds. The authors take into account both costs and execution times in a weighted sum objective function. Heilig et al. (2016) apply a Biased Random Key Genetic Algorithm (BRKGA) for solving the CRMP in multi-cloud environments, providing feasible solutions in the millisecond range with an excellent quality and suitable for being included as a real-time decision support tool in related deployment processes. In our work, we take into account the model structure of those previous works and adapt it for considering network latencies as well as the specific constraints of the CSPP, proposed in Heilig et al. (2016), such as by differentiating the application requests, allowing the sharing of VMs, and including location and operating systems requirements. At this point, it should be noted that the approaches in Coutinho et al. (2015), Coutinho et al. (2013) bundle all the requirements of the consumers without differentiating between individual requirements and further limit the number of available VMs.

In general, none of those optimization approaches has been integrated into a simulation framework as it is commonly done in works covering optimization problems of cloud providers. A reason could be a lack of functionality in the *CloudSim* framework, which is the de-facto standard for modeling and evaluating provisioning and scheduling algorithms in the area of cloud computing research. The use of a simulation tool may help to better capture the complexity and stochasticity of cloud environments, such as given by different demand patterns, price variations, fluctuating resource performances, and changing preferences of decision or policy makers. In this work, we present extensions of *CloudSim* providing a foundation to evaluate current and future approaches based on brokering functionality and simulated cloud environments. Due to the popularity of *CloudSim* among scholars, the extensions and embedding of a consumer brokering optimization framework may promote future research in this area.

Moreover, the fact that cloud services are offered through different regions and locations has not been considered by current optimization approaches. In this work, we address this aspect by measuring the latency generated by placing applications and tasks into cloud data centers (DCs) around the globe. We present a multi-criteria problem formulation and different heuristics to solve it. As proposed in previous works, we further analyze the cost-performance trade-off based on a large number of simulations. Before presenting the numerical results of the study, we explain the extended problem settings and implemented *CloudSim* functionality in the next two sections.

3. Extended cloud service purchasing problem

In this section, we explain the extended Cloud Service Purchasing Problem (CSPP) representing a multi-criteria optimization problem in the area of cloud computing. As such, it is an extension of the CSPP proposed in Heilig et al. (2016) to additionally consider the network performance between consumers and cloud DCs. Therefore, we refer to the mathematical model presented in Heilig et al. (2016) and focus on pointing out the main assumptions, constraints, and differences.

Generally, a broker is able to execute consumer tasks and applications (i.e., requests) on VMs of different cloud providers. Each request involves different resource, performance, and legal requirements. The resource requirements of those requests may vary over time (e.g., through different workloads, execution time requirements) and thus need to be monitored constantly. To model the performance requirements of requests or to limit the time an application is executed in the cloud, for instance in hybrid clouds, a deadline can be defined for each request. Generally, several approaches have been proposed in literature to profile, model, and analyze resource requirements of requests, for instance, using regression analysis and machine learning techniques (see, e.g., Wood et al., 2008). Regarding mathematical modeling approaches, the assignment of VMs to requests can be modeled as a multidimensional vector bin packing problem (MVBPP; Chekuri and Khanna, 2012), a multidimensional knapsack problem (MKP; Fréville, 2004), or as a multidimensional multiple-choice knapsack problem (MMKP; Moser et al., 1997), where the knapsacks represent the VMs and the requests are the items. We consider the model structure of previous works (in particular, Coutinho et al., 2015, 2013), which is similar to the MKP, and adapt it for supporting the multi-criteria optimization of costs and latencies as well as for accommodating specific constraints given by the CSPP, such as by differentiating between requests, allowing the sharing of VMs, and including location and operating systems requirements. In this regard, it should be noted that Coutinho et al., (2013, 2015) bundle all the requirements of consumers without differentiating between individual requests.

More formally, we define the following characteristics of requests.

- A set of requests A of cloud consumers is given, where each request $a \in A$ has individual resource and system requirements. Namely, the processing capacity g_a in million instructions (MI), hard disk capacity d_a, memory capacity m_a, and the operating system o_a.
- The consumer can specify an execution deadline t_d and a specific deployment region r_a for each request to guarantee a certain performance level and to fulfill legal requirements (e.g., place of jurisdiction), respectively.
- Requests are homogeneous in terms of the computing architecture.
- We further assume that each request can use the full processing capacity of a VM while being executed.

On the other hand, a diverse set of VM types offered by different cloud providers can be used to execute the individual requests of cloud consumers.

- That is, each cloud provider offers a set of VM types V, where each VM of type $v \in V$ provides a maximum amount of processing capacity G_v in million instructions per second (MIPS), a fixed amount of memory M_v, and storage capacity D_v.
- For each VM of type $v \in V$, a set of available regions L_v is defined, which depends on the available regions the cloud provider of this VM is offering.
- For each VM of type $v \in V$, a set of available operating systems S_v is defined, which depends on the available operating systems the cloud provider of this VM is offering.
- Before a VM can be utilized, it must be provisioned with a specific operating system in a specific region according to the demand. Once the region and operating system are fixed for a VM, it cannot be changed during its use.
- A price per time unit of use c_{vj}^{i} (e.g., hours) is defined for each VM of type $v \in V$ dependent on the installed operating system $j \in S_v$ and deployment region $i \in L_v$.
- It is possible to execute multiple requests of a consumer on the same VM. Requests are executed sequentially corresponding to the allocation order.

To gain a benefit for consumers, the CSPP aims at minimizing the total costs for purchasing VMs used to execute the given requests, as formulated in the following objective function.

$$\text{(CSPP)}\ \textit{minimize} \sum_{v \in V} \sum_{j \in S_v} \sum_{i \in L_v} c_{vj}^{i}\, y_{vj}^{i} \tag{1}$$

where the overall costs of purchasing VMs is the sum of the costs per time unit c_{vj}^{i} multiplied by the number of VMs of type $v \in V$, denoted by the decision variable y_{vj}^{i}. The number of VMs reflects the sum of purchased time units of respective VMs. That is, the time dimension is indirectly considered by accumulating the number of VMs over all time periods, whereby the number is increased by newly purchased VMs as well as by each additional time unit a purchased VM is used. The following additional constraints are considered in the CSPP.

- *Fixed Assignment*: Each request $a \in A$ must be assigned to exactly one instance of a VM type.
- *Processor Sharing*: The overall volume of MIPS resulting from the sum of purchased VM hours of type $v \in V$, determined by y_{vj}^{i}, and the processing capacity G_v must be greater or equal to the processing requirements of assigned requests.
- *Memory Sharing*: During execution, a request $a \in A$ can exclusively use the full memory capacity M_v of the assigned VM.
- *Disk Sharing*: Available disk capacity D_v of the VM needs to be shared among assigned requests.
- *System sharing*: The chosen operating system O_v of the VM must correspond to the operating system requirement o_a of all assigned requests.
- *Region*: If specified, a request $a \in A$ has to be executed in a specific region r_a. Ergo, the chosen deployment region R_v of the VM must correspond to the region requirements of all assigned requests.
- *Deadline*: Each request $a \in A$ must be executed before a given deadline t_d to fulfill performance requirements.

In this work, we extend the CSPP by additionally considering the network latency between consumers and assigned VMs. The network latency is measured between a consumer-specified location and a cloud DC of the region (see Section 6.1.2). The latency determines the time duration necessary to transmit a signal, usually in the form of a data packet, from a network origin point to a destination point (Liotine, 2003). Basically, it can be defined as the cumulative delay posed by all network elements in a transmission path (Liotine, 2003). As opposed to network bandwidth, determining the data throughput, the options to influence the network latency (i.e., network delay) are very limited. The total network latency is caused by several non-changeable factors and elements in the transmission path, including the physical distance between origin and destination, network components, and routing queues. Although it is not possible to influence the network latency, a

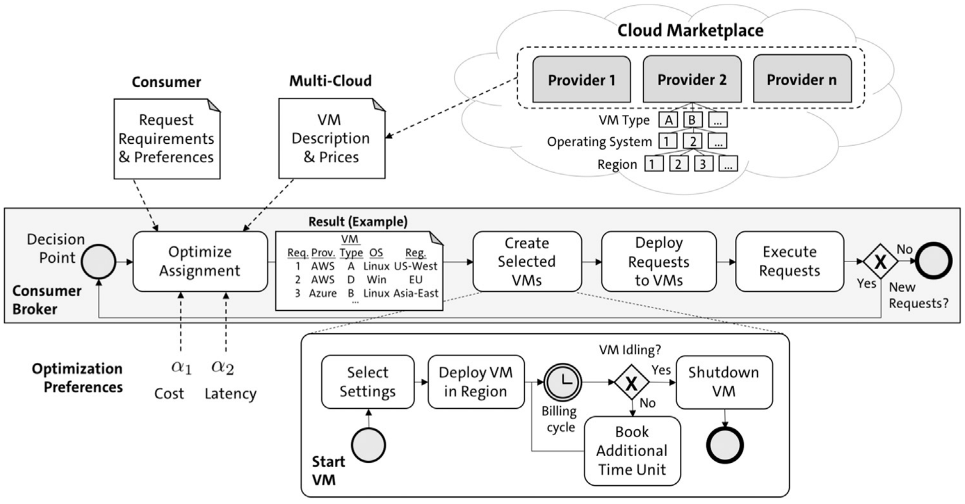

Fig. 1. Concept of the consumer broker.

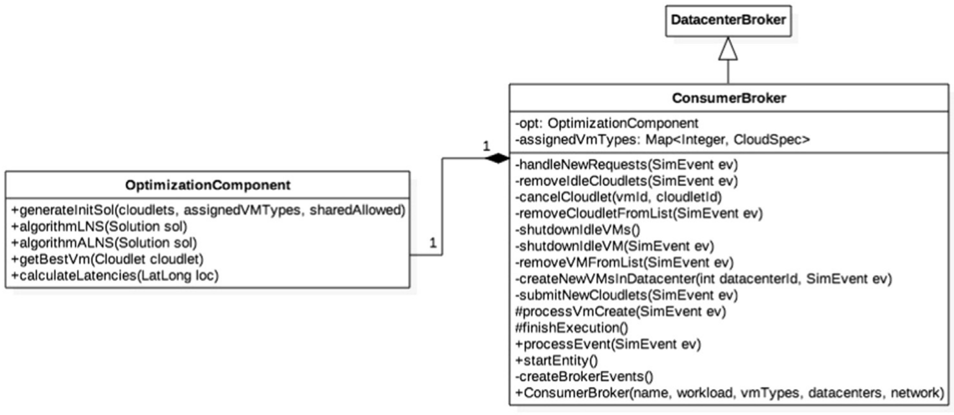
DatacenterBroker
ConsumerBroker
-opt: OptimizationComponent
-assignedVmTypes: Map<Integer, CloudSpec>
-handleNewRequests(SimEvent ev)
-removeIdleCloudlets(SimEvent ev)
-cancelCloudlet(vmId, cloudletId)
-removeCloudletFromList(SimEvent ev)
-shutdownIdleVMs()
-shutdownIdleVM(SimEvent ev)
-removeVMFromList(SimEvent ev)
-createNewVMsInDatacenter(int datacenterId, SimEvent ev)
-submitNewCloudlets(SimEvent ev)
#processVmCreate(SimEvent ev)
#finishExecution()
+processEvent(SimEvent ev)
+startEntity()
-createBrokerEvents()
+ConsumerBroker(name, workload, vmTypes, datacenters, network)
1
1
OptimizationComponent
+generateInitSol(cloudlets, assignedVMTypes, sharedAllowed)
+algorithmLNS(Solution sol)
+algorithmALNS(Solution sol)
+getBestVm(Cloudlet cloudlet)
+calculateLatencies(LatLong loc)

Fig. 2. Excerpt of the extensions implemented in *CloudSim* to support consumer-related decision making in cloud environments.

dedicated selection of VM types offered through various DC locations allows to reduce the network latency experienced between consumers and cloud locations. For this purpose, we extend the objective function proposed in Heilig et al. (2016) as follows:

$$\text{(Extended CSPP)}\ \mathit{minimize}\ (\alpha_1(\sum_{v\in V}\sum_{j\in S_v}\sum_{i\in L_v} c^{i}_{vj}\,y^{i}_{vj}) + \alpha_2(\sum_{v\in V}\sum_{a\in A}\sum_{j\in S_v}\sum_{i\in L_v} z^{ji}_{av} l_{ai})) \tag{2}$$

where $(\alpha_1 + \alpha_2) = 1$. Compared to the CSPP, the objective function seeks both the minimization of total costs and overall network latency.

As considered in the CSPP, an additional decision variable z determines whether a request $a \in A$ is executed within a VM of type $v \in V$ under operating system $j \in S_v$ in region $i \in L_v$. By measuring the network latency l_{ai} between the consumer-specified location of a request $a \in A$ and the deployment region $i \in L_v$, we are able to determine the overall network latency. Our approach to determine the network latency l_{ai} a priori is given in Section 6.1.2.

As a multi-criteria decision making method, we apply the weighted sum approach. That is, we introduce target weights α_1 and α_2 enabling the broker to specify the relevance of each objective in a way that different dimensions can be put into perspective treating them as a single objective. Thus, for values of α_1 close to 1, the broker prefers low-budget solutions, which may result in higher latencies. For values of α_2 close to 1, on the other hand, the broker aims at placing VMs as close as possible to the consumer in terms of network latency, which may result in higher overall costs. To handle the different dimensions in one objective function, a normalization of objective values is necessary. In this work, we normalize the objectives to values between zero and one. Before analyzing the trade-off in Section 6.2, we give an overview on the implemented *CloudSim* extensions and optimization approaches in the next section.

4. Extension of *CloudSim*

As one of the major research goals is the support of consumer-centric brokering schemes in *CloudSim*, this section explains the implemented extensions for supporting the management of cloud resources in multi-cloud environments. In general, the *CloudSim* toolkit supports modeling and simulation of cloud computing environments and has been used for investigations by a large part of the research community (Calheiros et al., 2011). *CloudSim* contains a collection of packages and libraries written in Java, which can be used to configure and run discrete-event simulations. In its current version, the main purpose of *CloudSim* is to allow the evaluation of policies and techniques for cloud resource provisioning and VM placement (see, e.g., Masdari et al., 2016) by providing various cloud system components, workload models, and provisioning algorithms that can be extended with ease and limited effort (Calheiros et al., 2011). In practice, it represents a tool for cloud infrastructure providers to assess new ways for a cost- and energy-efficient management of one or multiple cloud DCs, such as for organizing the resource management among different providers in a federated cloud (for an overview on resource management in federated clouds, see, e.g., Liaqat et al., 2017).

From the perspective of cloud consumers and related cloud brokers, however, the management of DCs is of little interest as long as service level agreements are fulfilled. While the latter can not be strongly influenced by consumers, they have the power to choose services from different cloud providers. Having a broad range of cloud services from different cloud providers, the main aim of the consumer is to maintain an economic virtual infrastructure setup in a way that a good trade-off between costs and performance is achieved besides fulfilling important requirements of users and taking into account the reputation and reliability of providers.

In this work, we extend *CloudSim* by implementing another layer representing the consumers' perspective on cloud environments. As in real cloud environments, the consumer is able to choose from a range of preconfigured VM types. As described before, the broker has to select and start images of those VM types to execute consumers' requests exposing specific requirements, for example, regarding computing capacities, execution deadlines, and other constraints. In a pay-per-use mode, the consumer has to pay per unit of time the VM is used (e.g., per hour, per minute). However, techniques to manage the on-going process of decision making and methods to implement decisions (e.g., in form of deployment processes) in cloud environments are yet missing in *CloudSim*. This involves, for example, to shut down a VM after one billing cycle, if it is not utilized anymore, and to calculate resulting usage costs. To better grasp the process and broker functionality in terms of optimization and management tasks, Fig. 1 illustrates the main concept and activities of the consumer broker. For a detailed description of cloud provisioning and deployment processes, the interested reader is referred to Heilig et al. (2015).

To support the evaluation of related broker decision support techniques over multiple periods, taking into account, e.g., dynamic pricing schemes of cloud providers, workloads and requirements, we have extended *CloudSim* by several classes and discrete event types. An excerpt of the class diagram describing the extended version of *CloudSim* is shown in Fig. 2 to illustrate the most important extensions (see Calheiros et al., 2011 for comparison with the initial version). All extensions have been implemented in Java. In the following subsection, we explain the relevant classes and parts of the implementation.

4.1. *ConsumerBroker* class

The class *ConsumerBroker* is a new type of broker in *CloudSim* handling the selection and allocation of VM types for requests on behalf of cloud consumers. Before the broker is initiated, the simulation environment and a workload scenario must be configured.

4.1.1. *Simulation setup*

To simulate cloud environments in *CloudSim*, DCs have to be defined. The perception of “unlimited resources” is one of the key characteristics of cloud computing from the consumer perspective. As physical layers of cloud environments are hidden for the consumer broker, it is assumed that physical components, used to simulate the execution of VMs, are provisioned with enough resources. Instead, the broker has access to a range of VM types of one (single-cloud) or multiple cloud providers (multi-cloud), which can be specified using the new class *CloudSpec*. Each object of *CloudSpec* specifies the available virtual resources, an operating system, and the prices per region. We furthermore define a network structure to consider the network performance between consumers and cloud DCs.

Moreover, the workload of requests to be assigned is generated prior to the simulation using a new class *WorkloadGenerator*. Currently, this class generates a number of requests per period following a Poisson distribution. The number of simulation periods, the interarrival time, and the parameter $\lambda > 0$, defining the Poisson distribution, can be specified prior to the simulation. Each request defines its resource requirements, an execution deadline, and the location of origin given by a city name and geographical coordinates (e.g., derived from the IP address of the sender).

4.1.2. *Simulation process*

A new broker is initiated by providing the data of the simulation setup. In general, the broker integrates the core functionality of *CloudSim* by extending and modifying functionality of the class *DatacenterBroker* (see Calheiros et al., 2011). That is, we implement new functionality to manage the process of purchasing and releasing VM types from different cloud providers and integrate core functionality of *CloudSim* in order to control simulation processes. We add the following features.

- *Decision Points*: A discrete-time interval can be defined to indicate when an allocation of new requests and, optionally, a reallocation of existing requests shall be performed. As default, the interval corresponds to the interarrival time of requests.
- *VM Creation*: In each decision point, the broker creates a new set of VMs according to the optimization results (see Sections 4.2 and 5). The requests in form of a *CloudSim Cloudlet* are assigned to those machines and submitted to the DC by using the method *handleNewRequests()*.
- *VM Shutdown*: By extending the event processing of the *DatacenterBroker*, it is possible to shutdown idle VMs during the simulation. After each billing unit of a VM, the method *shutdownIdleVM()* checks the status of cloudlets being executed on it and sends an event to the DC to shutdown the VM in case it is idle. After a VM has been stopped, the overall costs of using the VM are calculated according to the billing cycle.
- *Request Reallocation*: It is possible to reassign cloudlets that are still in a waiting queue at the time of a new decision point. The method *removeIdleCloudlets()* is used to remove the idle cloudlets from the previously assigned VM in the DC. The rationale behind this is that it may be beneficial to assign waiting cloudlets to new VMs, assigned in this decision point, in case sharing is allowed.
- *Cost Assessment*: By incorporating real VM type descriptions, prices, and pricing models, it is possible to mimic and evaluate the cloud service portfolio of real cloud providers, intended to be helpful in related decision making processes. Therefore, reports of the different results are generated.

Overall, the new extensions of *CloudSim* provide basic broker functionality to manage and evaluate automated decision making processes with respect to the use of virtual cloud resources from one or multiple cloud providers. The design of the broker follows a modular structure and can be extended to consider specific aspects found in real cloud marketplaces. By using existing *CloudSim* functionality, it is possible to model and simulate dynamics of the machines’ performance, for example, by considering performance statistics from monitoring tools. The core of the consumer broker lies in the embedded optimization component, which is explained in the next subsection.

4.2. *OptimizationComponent* class

The class *OptimizationComponent* contains decision support functionalities for consumer-oriented brokers. In the current version, it implements methods and algorithms to solve the extended CSPP described in Section 3. The overall concept of the proposed *CloudSim* extensions, however, is to support cloud brokers and consumers also with additional management tasks during operations (e.g., composition of cloud services, price negotiations, data management, etc.). In this respect, the extensions leave a door open for other consumer broker approaches.

In the current version, a greedy heuristic and two large neighborhood search approaches are implemented (see Section 5). A method for calculating the latencies between consumers of requests and available cloud DCs is implemented as described in Section 6.1.2. Besides, it provides important functions to manage and utilize rented VMs over time as well as some helper functions. For example, a consumer may decide to share a VM among multiple requests. Therefore, it must also be possible to use the VM in subsequent periods if it is still active and has remaining capacities. Incorporating this option into *CloudSim* will allow to evaluate new approaches to improve the utilization of purchased VM hours.

The main structure being used during the optimization process is the new class *Solution*. An object of this class stores the assignment of requests (i.e., *Cloudlets*) to VMs, which can be altered during the optimization to find a best possible solution for the problem. This particularly involves methods to consistently update the use of VMs in each iteration. After the optimization is finished, the solution object is returned to the broker, which is able to read the instructions and perform related activities. As depicted in Fig. 1, the optimization result determines the assignment of requests to VM types, which is used to create new VMs for deploying and executing requests in the cloud environment of the respective providers (see also Heilig et al., 2015).

5. Optimization approaches

In this section, we explain the implemented large neighborhood search approaches designed for solving the extended CSPP in *CloudSim*. With respect to exact approaches, only small problem instances of the single objective CSPP could be solved to optimality (up to 10 requests) in the previous work of Heilig et al. (2016). For those instances, competitive results are achieved when applying the large neighborhood search algorithms, where the gap to the optimal solution is on average 2.88% and 3.85% for the LNS and ALNS, respectively. For larger problem instances, the general-purpose solver CPLEX is either not able to provide an optimal solution within two hours or runs out of memory. This promotes the adaption of those metaheuristics aimed at reducing the computational time for achieving feasible solutions. Therefore, we adapt the algorithms to support multi-period assignment and multi-criteria decision making. In the following, we explain the adapted and integrated heuristics to make this paper self-contained.

5.1. Large neighborhood search

The concept of Large Neighborhood Search (LNS), proposed by Shaw (1998), is to make large changes to current solutions when exploring the solution space by applying a destroy and repair heuristic in each iteration. Here, a solution represents the assignment of VMs to requests, either by purchasing new VMs or reusing existing VMs. The pseudo-code is shown in Algorithm 1.

The input parameter $iter_{max}$ denotes the maximum number of iterations without improving a current solution, used in the stop criterion in line 3. The variable S_{best} is the best solution observed during the search, S denotes the current solution, and S' is a temporary solution. To start with the search procedure, a feasible starting solution is generated by using a construction heuristic (line 1). While the function *destroy(·)* partly destroys the copy of the current solution (line 4), the function *repair(·)* transforms the destroyed solution into a new feasible solution S' (line 5). In the context of CSPP, the destroy heuristic removes VM-to-request assignments while the repair heuristic fills missing (i.e., destroyed) assignments. In line 6 the new solution is evaluated to determine whether to reject or accept it as a new current solution (line 7). In this regard, we only accept improving solutions and update the best solution S_{best} if necessary (line 8).

Algorithm 1. Large Neighborhood Search.

```
   Input: $iter_{max}$
 1 Generate initial solution $S$ using a construction heuristic
 2 $S_{best} \leftarrow S$
 3 while ($i < iter_{max}$) do
 4     $S' \leftarrow destroy(S)$
 5     $S' \leftarrow repair(S')$
 6     if ($f(S) < f(S_{best})$) then
 7         $S \leftarrow S'$
 8         $S_{best} \leftarrow S$
 9         $i \leftarrow 0$
10     else
11         $i \leftarrow i + 1$
12     end
13 end
14 return $S_{best}$
```

5.2. Adaptive large neighborhood search

We furthermore implement an adaptive large neighborhood search (ALNS), which extends LNS by using multiple destroy and repair heuristics and controls the selection of those heuristics in each iteration by measuring their performance during the optimization procedure (Pisinger and Ropke, 2010). The pseudo-code of the ALNS is shown in Algorithm 2.

The parameters used in the ALNS are the neighborhood size in percent (ξ), maximum number of iterations $iter_{max}$, maximum number of iterations in each search segment seg_{max}, score increments (σ_1, σ_2, σ_3), and reaction factor (r). For understanding the main principle of ALNS, we first explain the selection of heuristics during the search procedure. That is, different destroy and repair heuristics can be selected in each iteration of the search procedure. For that purpose, a roulette wheel selection algorithm is implemented, which is influenced by a roulette weight vector $\Omega = [\omega_1, \omega_2, \ldots, \omega_j, \ldots, \omega_{|H|}]$ representing a measure for the previous performance of each heuristic $j \in H$. In the first iteration, this vector is initialized such that the probability of selecting one heuristic is the same for all (line 5). Moreover, a vector $\Theta = [\theta_1, \theta_2, \ldots, \theta_j, \ldots, \theta_{|H|}]$ is initialized in line 8 for counting the number of times heuristic j is selected. The overall search procedure of $iter_{max}$ iterations is divided into segments of seg_{max} iterations. During the iterations of a segment, each heuristic j collects scores that are given dependent on its success of finding a new solution. A vector $\Pi = [\pi_1, \pi_2, \ldots, \pi_j, \ldots, \pi_{|H|}]$ containing the collected scores for each heuristic is initialized in line 7. It is differentiated between three different scores (σ_1–σ_3). If the heuristic j leads to a new best solution S_{best} in an iteration, the highest score σ_1 is added to the heuristic's score π_j (line 20). If the new solution S' is worse than the current solution S, but is accepted, the lowest score σ_3 is added (line 31). As an acceptance function *accept(·)*, we hybridize the algorithm and implement a simulated annealing (SA) algorithm. Score increment σ_2 is added if the new solution S' is better than the current solution S, but does not lead to a new best solution S_{best} (line 16). After each segment of iterations, the updated vectors Π and Θ are used to determine new weights for each heuristic j as described in Eq. (3). The reaction factor r controls how much the roulette weight depends on the performance of a heuristic in the last segment. That is, if r=0, the performance measure is not used to update the roulette weights; otherwise, if $r > 0$, the roulette weights can positively influence the selection of well-performing heuristics in subsequent iterations (Pisinger and Ropke, 2010).

$$\omega_{j,s+1} = \omega_{j,s}(1-r) + r\frac{\pi_j}{\theta_j} \quad (3)$$

The remaining parts of the algorithm implement the general structure of the LNS framework (see Algorithm 1 for comparison). The current solution S is updated in line 29. After a maximum number of $iter_{max}$ iterations, the best known solution S_{best} is returned.

Algorithm 2. Adaptive Large Neighborhood Search for the CSPP.

```
Input: $\xi \in \mathbb{R}^+$, $iter_{max}$, $seg_{max}$, $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{R}^+$, $r \in \mathbb{R}^+$
1  Generate initial solution $S$ using a construction heuristic
2  Initialize set of accepted solutions $K = \{S\}$
3  $S_{best} \leftarrow S$
4  $i \leftarrow 0$
5  $\Omega \leftarrow 1$
6  while $(i < iter_{max})$ do
7      $\Pi \leftarrow 0$
8      $\Theta \leftarrow 0$
9      $seg_{iter} \leftarrow 0$
10     while $(seg_{iter} < seg_{max})$ do
11         Choose one destroy heuristic and one repair heuristic $j \in H$ using a roulette
           wheel selection by means of the roulette weight vector $\Omega$ and then update
           vector $\Theta$
12         Generate a new solution $S'$ by removing and repairing $\xi$ percent of the VM
           assignments in $S$ using the previously chosen heuristics
13         if $(f(S') < f(S))$ then
14             $S \leftarrow S'$
15             if $(S \notin K)$ then
16                 $\pi_j \leftarrow \pi_j + \sigma_2$
17                 $K \leftarrow K \cup S$
18             end
19             if $(f(S) < f(S_{best}))$ then
20                 $\pi_j \leftarrow \pi_j + \sigma_1$
21                 $S_{best} \leftarrow S$
22                 $i \leftarrow 0$
23             else
24                 $i \leftarrow i + 1$
25             end
26         else
27             $i \leftarrow i + 1$
28             if $(accept(S', S))$ then
29                 $S \leftarrow S'$
30                 if $(S \notin K)$ then
31                     $\pi_j \leftarrow \pi_j + \sigma_3$
32                     $K \leftarrow K \cup S$
33                 end
34             end
35         end
36         $seg_{iter} \leftarrow seg_{iter} + 1$
37     end
38     Update weight vector $\Omega$ for the next segment using $\Pi$, $\Theta$, and $r$
39 end
40 return $S_{best}$
```

First of all, an initial solution is generated (line 1), using a constructive heuristic, and added to a set K storing all accepted solutions (line 2). At each iteration, one destroy heuristic and one repair heuristic from the set H are consecutively chosen by applying *roulette wheel selection* with the obtained roulette weights (line 11). In line 12, a new solution S' is generated by alternately applying the selected destroy and repair heuristic to the current solution S. If the new solution S' is better than the current solution S (line 13) and/or better than the best known solution S_{best}, S' and S_{best} are updated in lines 14 and 21, respectively. As indicated, we occasionally select worse solutions in order to avoid being trapped in a local minimum. Following the idea of SA, a worse solution may be accepted according to a probability that depends on the deterioration $\triangle$ of the objective function value. The probability of acceptance is computed as $e^{(-\frac{\triangle}{Temp})}$, where $Temp$ is used as a control parameter and decreased by a cooling rate $0 < c < 1$ during the search procedure such that $Temp \leftarrow Temp \cdot c$. If a worse solution is accepted, the

5.3. Construction, removal and repair heuristics

As explained, the presented algorithms rely on a construction heuristic to generate an initial solution as well as on destroy and repair heuristics to generate new solutions in each iteration. As the construction and repair heuristics generally generate a new feasible solution, we implement two generic methods in *CloudSim*, which can be used to both construct and repair solutions.

- *Random Assignment*: Iteratively assigns a VM to each open request at random. More specifically, a VM is randomly selected out of the pool of existing (i.e., purchased) VMs, in case sharing is allowed, or purchased as an instance of a randomly chosen VM type given that all constraints are satisfied.
- *Greedy Heuristic*: Iteratively assigns the best possible VM – in terms of the objective function value – to each open request. As it

Table 1
Parameter values for the simulation experiments.

Parameter	Values
Number of periods (k)	{5, 10, 15, 20, 25}
Average number of requests per period (λ)	{5, 10, 20, 40, 60, 100, 200, 400}
Decision point interval in seconds	{300, 900, 1800, 2400, 3000}
Maximum number of requests per country	20
Weight of objectives α_1 and α_2	
• Configuration 1	($\alpha_1 = 1, \alpha_2 = 0$)
• Configuration 2	($\alpha_1 = 0, \alpha_2 = 1$)
• Configuration 3	($\alpha_1 = 0.5, \alpha_2 = 0.5$)

Table 2
Ranges of request requirements.

Requirement	Range
Processing capacity in MI	[350,000, 3500,000]
Memory capacity in GB	[1, 8]
Disk capacity in GB	[1, 200]
Operating system	{Windows, Linux}

might happen that different VM types lead to the same objective function value, a second decision criterion is involved. That is, the VM with the better ratio in terms of MIPS, memory, and storage capacity is selected to achieve a higher cost-benefit.

To destroy solutions, we further implement three different removal heuristics in *CloudSim*.

- *Shaw Removal*: Removes a number of assignments from the solution sharing similar characteristics (Shaw, 1998). Thus, a relatedness measure $R(q_1, q_2)$ determines the similarity of VM-to-request assignments. Assignments are similar if related requests a_1 and a_2 have similar processing capacity c and deadline requirements t and are dissimilar if they do not share the same requirements with respect to the operating system o and deployment region r, as formulated in Eq. (4) where HV is defined as a high enough value. Algorithm 3 shows the pseudo-code of the Shaw removal procedure. The parameter S is the solution to be destroyed, $\xi \in \mathbb{N}$ denotes the percentage of assignments to be removed, and parameter p with $p \geq 1$ is used to control the randomness of selecting assignments to be removed. First, the heuristic randomly picks one assignment r from a given solution S (line 1) and adds it to the set D, initialized in line 2 to contain all assignments to be removed. Starting with the currently chosen assignment q, all remaining assignments are added

Table 3
Performance assessment of simulation results for $\alpha_1 = 1$, $\alpha_2 = 0$ (*Config. 1*).

Scenario		Greedy			LNS				ALNS			
(λ, k)	# Req.	Cost ($)	Lat. (ms.)	t (s.)	Cost ($)	Lat. (ms.)	t (s.)	Impr. (%)	Cost ($)	Lat. (ms.)	t (s.)	Impr. (%)
(5, 5)	24	5.22	4627.96	0.02	5.22	4627.96	5.47	0.00	5.22	4619.70	5.60	0.00
(5, 10)	42	9.73	8176.06	0.00	7.55	8176.06	5.77	22.34	7.55	8176.06	5.98	22.34
(5, 15)	66	14.96	12,665.77	0.01	14.42	12,574.87	4.95	3.61	14.42	12,574.87	5.16	3.61
(5, 20)	88	19.86	16,988.15	0.01	19.29	16,988.15	5.13	2.87	19.29	16,979.88	5.47	2.87
(5, 25)	112	25.43	21,571.53	0.01	24.57	21,571.53	5.30	3.36	24.57	21,550.87	5.54	3.36
(10, 5)	43	9.19	6888.01	0.02	8.84	6921.18	8.37	3.79	8.83	6945.97	9.14	3.85
(10, 10)	91	19.52	15,133.98	0.01	14.85	15,132.55	11.10	23.92	14.85	15,132.55	11.58	23.92
(10, 15)	140	29.77	23,914.26	0.01	28.54	23,972.93	9.35	4.13	28.50	23,956.40	9.37	4.28
(10, 20)	189	40.67	32,326.75	0.01	38.08	32,449.99	9.71	6.38	38.20	32,408.26	10.19	6.08
(10, 25)	247	53.08	42,821.45	0.01	49.51	43,007.14	10.06	6.74	49.66	43,212.73	10.66	6.45
(20, 5)	87	18.38	16,365.37	0.03	17.64	16,334.58	17.24	4.01	17.64	16,326.31	16.77	4.00
(20, 10)	185	40.31	35,052.94	0.02	30.31	35,004.14	22.18	24.79	30.35	34,972.16	22.36	24.71
(20, 15)	293	63.79	55,487.65	0.02	60.07	55358.86	19.00	5.83	59.92	55,316.73	19.66	6.07
(20, 20)	412	90.19	78,126.97	0.03	84.99	77,998.77	21.40	5.76	85.34	78,022.35	21.41	5.38
(20, 25)	534	116.97	101,131.51	0.03	110.14	101,053.49	21.93	5.84	110.35	101,136.13	22.40	5.66
(40, 5)	205	42.35	41,497.90	0.07	39.44	41,479.97	44.01	6.87	39.58	41,494.93	49.05	6.54
(40, 10)	398	82.47	80,126.57	0.05	61.53	80,043.41	54.26	25.39	61.38	80,011.51	50.69	25.57
(40, 15)	606	125.49	122,103.60	0.05	116.71	122,012.70	43.52	7.00	116.54	121,915.67	38.83	7.13
(40, 20)	795	164.76	159,830.71	0.05	152.43	159,764.96	46.02	7.48	153.67	159,637.77	41.19	6.73
(40, 25)	995	204.98	199,720.66	0.05	190.93	199,640.03	48.55	6.86	192.23	199,529.74	43.22	6.22
(60, 5)	306	61.94	61,180.91	0.10	56.33	61,158.93	106.15	9.06	57.51	61,120.34	60.86	7.16
(60, 10)	599	119.22	120,275.50	0.08	88.37	120,244.58	93.61	25.88	90.45	120177.62	73.30	24.14
(60, 15)	939	188.78	188,592.69	0.08	173.16	188,619.60	89.61	8.27	176.94	188,641.36	66.86	6.27
(60, 20)	1223	246.21	245,969.60	0.08	225.93	246,138.59	92.58	8.24	229.56	245,952.21	62.11	6.76
(60, 25)	1534	309.78	308,182.68	0.08	283.53	308,291.30	96.86	8.47	288.69	307,965.41	63.15	6.81
(100, 5)	508	106.54	106,737.62	0.14	93.97	106,848.72	249.30	11.80	100.00	106,823.62	87.58	6.14
(100, 10)	999	210.22	209,298.18	0.13	148.15	209,630.71	238.55	29.53	156.60	209,524.75	116.27	25.51
(100, 15)	1514	321.24	316,202.22	0.13	279.82	316,717.51	234.80	12.90	296.31	316,517.23	87.86	7.76
(100, 20)	2019	430.17	421,429.36	0.13	373.38	422,095.37	233.14	13.20	394.68	421,866.78	88.16	8.25
(100, 25)	2512	534.81	524,657.20	0.13	465.59	525,456.56	231.06	12.94	490.11	525,075.77	95.84	8.36
(200, 5)	1013	211.46	223,275.35	0.27	185.55	223,243.59	521.83	12.26	197.98	223,099.10	178.18	6.38
(200, 10)	2002	426.05	441,600.36	0.26	295.01	441,472.83	471.33	30.76	315.20	441,277.42	228.54	26.02
(200, 15)	3024	649.34	666,196.20	0.26	558.06	665,982.48	455.32	14.06	594.66	665,802.75	177.34	8.42
(200, 20)	4013	860.96	883,792.23	0.26	739.84	883,520.96	439.29	14.07	788.41	883,298.46	173.24	8.43
(200, 25)	5021	1079.71	1,105,762.43	0.26	925.43	1105,503.19	464.90	14.29	990.54	1105,033.49	174.39	8.26
(400, 5)	2020	393.97	459,968.84	0.53	354.87	460,033.39	852.87	9.93	374.69	459,946.53	379.88	4.89
(400, 10)	4005	805.34	910,742.54	0.53	571.29	910,966.14	768.06	29.06	601.58	910,749.58	463.65	25.30
(400, 15)	6037	1221.41	1,371,490.83	0.53	1072.37	1,371,688.00	921.08	12.20	1137.16	1,371,674.58	364.31	6.90
(400, 20)	8022	1632.26	1,821,229.00	0.53	1431.00	1,821,433.40	880.32	12.33	1516.52	1,821,489.62	353.41	7.09
(400, 25)	10033	2046.45	2,279,135.72	0.53	1787.88	2,279,564.77	876.58	12.64	1898.18	2,279,217.83	359.45	7.25
Average	1572.38	325.82	343,506.93	0.14	279.61	343,568.10	218.26	11.97	294.35	343,479.38	101.46	9.77

Table 4
Performance assessment with the simulation results for $\alpha_1 = 0.5$, $\alpha_2 = 0.5$ (*Config. 2*).

Scenario		Greedy			LNS				ALNS			
(λ, k)	# Req.	Cost ($)	Lat. (ms.)	t (s.)	Cost ($)	Lat. (ms.)	t (s.)	Impr. (%)	Cost ($)	Lat. (ms.)	t (s.)	Impr. (%)
(5, 5)	24	9.38	4110.85	0.02	9.38	4110.85	6.13	0.00	9.38	4110.85	6.20	0.00
(5, 10)	42	16.08	7249.92	0.01	16.08	7249.92	4.80	0.00	16.08	7249.92	5.05	0.00
(5, 15)	66	24.77	11,214.57	0.01	24.77	11,214.57	5.00	0.00	24.77	11,214.57	5.11	0.00
(5, 20)	88	33.54	15,102.47	0.01	33.54	15,102.47	5.22	0.00	33.54	15,102.47	5.49	0.00
(5, 25)	112	42.99	19,210.06	0.01	42.99	19,210.06	5.12	0.00	42.99	19,210.06	5.41	0.00
(10, 5)	43	13.91	6399.18	0.02	13.91	6399.18	9.05	0.00	13.91	6399.18	9.64	0.00
(10, 10)	91	29.43	14,055.00	0.01	29.31	14,055.08	8.99	0.39	29.31	14,055.08	9.53	0.39
(10, 15)	140	46.03	22,274.93	0.01	45.55	22,275.18	8.93	1.05	45.55	22,274.93	9.47	1.04
(10, 20)	189	62.36	30,090.91	0.01	61.04	30,087.64	9.91	2.12	61.05	30,087.33	9.95	2.11
(10, 25)	247	81.69	39,892.99	0.01	79.54	39,885.66	9.88	2.63	79.54	39,885.51	9.94	2.63
(20, 5)	87	27.19	15,221.74	0.03	26.48	15,220.94	16.94	2.59	26.48	15,220.94	19.12	2.59
(20, 10)	185	56.42	32,680.31	0.02	54.67	32,678.29	17.00	3.10	54.71	32,678.29	19.85	3.02
(20, 15)	293	88.45	51,799.00	0.02	85.37	51,795.77	18.46	3.48	85.28	51,795.77	18.59	3.59
(20, 20)	412	123.93	73,186.44	0.03	119.70	73,181.99	21.08	3.41	119.71	73,181.99	21.63	3.40
(20, 25)	534	160.48	94,890.05	0.03	154.98	94,884.38	21.52	3.43	155.36	94,884.38	23.01	3.19
(40, 5)	205	61.84	38,680.35	0.07	56.65	38,681.11	40.31	8.40	56.59	38,684.71	39.70	8.50
(40, 10)	398	119.60	74,710.36	0.05	109.27	74,710.78	40.23	8.63	109.60	74,711.80	36.91	8.36
(40, 15)	606	181.31	113,850.61	0.05	166.54	113,853.05	39.57	8.15	168.84	113,851.77	37.34	6.88
(40, 20)	795	238.72	149,158.63	0.05	219.24	149,162.74	42.19	8.16	221.60	149,162.10	38.32	7.17
(40, 25)	995	295.66	186,437.51	0.05	274.35	186,442.35	40.87	7.21	274.24	186,446.67	38.00	7.25
(60, 5)	306	87.33	56,893.57	0.09	77.53	56,898.77	110.05	11.22	79.46	56,898.77	68.16	9.01
(60, 10)	599	171.41	111,810.68	0.08	153.77	111,820.35	80.15	10.29	157.00	111,822.16	55.22	8.41
(60, 15)	939	266.56	175,418.41	0.08	240.40	175,435.32	85.54	9.81	244.27	175,434.42	57.42	8.36
(60, 20)	1223	347.66	228,844.55	0.08	316.18	228,866.26	84.36	9.05	321.51	228,868.55	57.11	7.52
(60, 25)	1534	433.78	286,629.04	0.08	395.57	286,653.36	83.92	8.81	399.72	286,654.83	54.33	7.85
(100, 5)	508	144.43	99,543.37	0.14	123.89	99,544.45	176.58	14.23	132.86	99,558.52	96.29	8.02
(100, 10)	999	283.74	195,283.47	0.13	246.20	195,286.86	155.80	13.23	258.67	195,310.81	84.43	8.84
(100, 15)	1514	429.63	294,901.71	0.13	376.62	294,910.29	152.16	12.34	393.39	294,940.26	85.25	8.43
(100, 20)	2019	573.61	393,128.12	0.14	502.46	393,143.60	163.44	12.40	522.68	393,182.98	87.05	8.88
(100, 25)	2512	712.41	489,501.70	0.13	625.41	489,519.95	157.38	12.21	647.87	489,565.39	86.43	9.06
(200, 5)	1013	281.46	208,443.72	0.27	238.46	208,446.25	481.78	15.28	252.98	208,462.18	191.09	10.12
(200, 10)	2002	564.30	412,185.58	0.27	473.07	412,198.85	575.95	16.17	509.69	412,218.25	177.80	9.68
(200, 15)	3024	858.53	621,909.24	0.26	712.34	621,922.27	549.68	17.03	772.76	621,947.70	172.70	9.99
(200, 20)	4013	1146.00	824,962.17	0.27	946.54	824,982.87	535.48	17.40	1030.29	825,018.91	173.73	10.10
(200, 25)	5021	1436.18	1,032,132.57	0.26	1186.74	1,032,155.16	591.97	17.37	1292.65	1,032,196.98	170.96	9.99
(400, 5)	2020	531.23	428,268.57	0.54	452.15	428,246.18	1001.35	14.88	492.21	428,278.57	409.74	7.34
(400, 10)	4005	1085.88	848,096.50	0.54	915.39	848,047.93	976.12	15.70	991.63	848,077.96	342.70	8.68
(400, 15)	6037	1648.10	1,277,272.52	0.53	1383.46	1,277,208.81	958.54	16.06	1497.40	1,277,284.76	348.15	9.14
(400, 20)	8022	2205.95	1,696,096.29	0.53	1841.88	1,695,992.81	917.82	16.50	2000.39	1,696,073.03	343.30	9.32
(400, 25)	10033	2763.07	2,122,630.44	0.53	2304.54	2,122,502.45	970.36	16.59	2499.63	2,122,597.62	336.94	9.53
Average	1572.375	442.13	320,104.20	0.14	378.40	320,099.62	229.49	8.48	403.14	320,115.02	94.18	5.96

to an array L and sorted in ascending order according to their relatedness to r (line 7) where the first element represents the one with the highest relatedness. A random number between zero and one is used to select one assignment from L (line 8), which is added to D. By increasing the value of the parameter p, the likelihood of selecting dissimilar assignments can be reduced. The procedure is repeated until $\lceil \xi |S| \rceil$ assignments have been selected to be removed. Note that we use a quickselect algorithm for the sort and select procedure in lines 7 and 8, respectively.

- *Worst Removal*: The heuristic follows the idea of the Shaw removal. Instead of using a relatedness measure, assignments are sorted based on the additional costs they produce. In this context, we define the additional costs of an assignment as $costdiff(q, s) = f(s) - f_{-q}(s)$ where $f_{-q}(s)$ denotes the total cost of the solution without assignment q.
- *Random removal*: Removes a number of assignments at random.

$$R(q_1, q_2) = \begin{cases} HV & \text{if } r_{a_1} \neq r_{a_2} \ \| \ o_{a_1} \neq o_{a_2} \\ \dfrac{|c_{a_1} - c_{a_2}|}{\max(c_{a_1}, c_{a_2})} + \dfrac{|t_{a_1} - t_{a_2}|}{\max(t_{a_1}, t_{a_2})} & \text{if } r_{a_1} = r_{a_2} \ \| \ r_{a_2} = 0 \ \| \ o_{a_1} = o_{a_2} \end{cases} \tag{4}$$

Algorithm 3. Shaw Removal

```
   Input: $S$, $\xi \in \mathbb{N}$, $p \in \mathbb{R}_+$
1  Randomly select an assignment $q$ from $S$
2  Initialize set of requests $D \leftarrow \{q\}$
3  while ($|D| < \lceil \xi |S| \rceil$) do
4      Randomly select an assignment $q$ from $D$
5      Initialize an array $L$ containing all assignments from $S$ not in $D$
6      Choose a random number $y$ in the interval [0,1)
7      Sort $L$ such that $i < j \Rightarrow R(q, L[i]) < R(q, L[j])$
8      $D \leftarrow D \cup \{L[y^p |L|]\}$
9  end
10 Remove the assignments in $D$ from $S$
```

In the implemented LNS, we use the Shaw removal as the removal heuristic and the greedy heuristic as the constructive and repair heuristic. ALNS uses all three removal heuristics and the greedy heuristic for constructing and repairing solutions.

6. Performance evaluation

Using the implemented *CloudSim* extensions, we conduct multiple simulation experiments to evaluate the proposed algorithms and to

Table 5
Performance assessment with the simulation results for $\alpha_1 = 0$, $\alpha_2 = 1$ (*Config. 3*) and sensitivity analysis.

Scenario		Greedy (*Config. 3*)			*Config. 3*/*Config. 1*		*Config. 3*/*Config. 2*		*Config. 2*/*Config. 1*	
(λ, k)	# Req.	Cost ($)	Lat. (ms.)	t (s.)	Cost (%)	Lat. (%)	Cost (%)	Lat. (%)	Cost (%)	Lat. (%)
(5, 5)	24.00	12.29	4107.04	0.02	135.61	−11.18	30.99	−0.09	79.87	−11.09
(5, 10)	42.00	20.50	7244.24	0.01	171.49	−11.40	27.50	−0.08	112.94	−11.33
(5, 15)	66.00	32.67	11,204.51	0.01	126.52	−10.90	31.88	−0.09	71.76	−10.82
(5, 20)	88.00	44.12	15,088.68	0.01	128.72	−11.16	31.55	−0.09	73.86	−11.08
(5, 25)	112.00	56.48	19,191.98	0.01	129.85	−10.99	31.39	−0.09	74.93	−10.90
(10, 5)	43.00	20.90	6390.54	0.02	136.55	−7.83	50.20	−0.13	57.49	−7.71
(10, 10)	91.00	42.79	14,038.19	0.01	188.15	−7.23	45.99	−0.12	97.37	−7.12
(10, 15)	140.00	65.55	22,250.89	0.01	129.86	−7.15	43.91	−0.11	59.72	−7.05
(10, 20)	189.00	88.29	30,054.86	0.01	131.47	−7.32	44.62	−0.11	60.05	−7.22
(10, 25)	247.00	116.62	39,843.86	0.01	135.20	−7.58	46.63	−0.10	60.41	−7.48
(20, 5)	87.00	38.37	15,210.74	0.03	117.51	−6.86	44.90	−0.07	50.10	−6.79
(20, 10)	185.00	80.34	32,656.72	0.02	164.89	−6.66	46.91	−0.07	80.31	−6.60
(20, 15)	293.00	126.81	51,761.24	0.03	111.37	−6.46	48.62	−0.07	42.23	−6.40
(20, 20)	412.00	179.88	73,133.54	0.03	111.22	−6.25	50.26	−0.07	40.56	−6.19
(20, 25)	534.00	233.67	94,820.83	0.03	111.96	−6.21	50.59	−0.07	40.75	−6.14
(40, 5)	205.00	85.90	38,657.77	0.07	117.43	−6.82	51.72	−0.06	43.31	−6.76
(40, 10)	398.00	170.29	74,666.76	0.05	177.10	−6.70	55.61	−0.06	78.08	−6.64
(40, 15)	606.00	258.53	113,783.08	0.05	121.68	−6.71	54.17	−0.06	43.78	−6.65
(40, 20)	795.00	337.71	149,070.32	0.05	120.65	−6.66	53.21	−0.06	44.02	−6.60
(40, 25)	995.00	423.86	186,328.16	0.05	121.25	−6.64	54.53	−0.06	43.18	−6.58
(60, 5)	306.00	126.70	56,869.25	0.11	122.59	−6.98	61.40	−0.05	37.91	−6.94
(60, 10)	599.00	246.94	111,764.06	0.08	176.19	−7.03	58.92	−0.05	73.79	−6.98
(60, 15)	939.00	387.90	175,349.51	0.08	121.60	−7.04	60.07	−0.05	38.44	−7.00
(60, 20)	1223.00	506.31	228,753.49	0.08	122.32	−7.03	58.80	−0.05	40.00	−6.98
(60, 25)	1534.00	633.61	286513.63	0.08	121.46	−7.01	59.34	−0.05	38.98	−6.97
(100, 5)	508.00	199.41	99,518.96	0.15	105.60	−6.85	55.33	−0.03	32.36	−6.82
(100, 10)	999.00	395.76	195,234.98	0.13	159.73	−6.84	56.78	−0.03	65.66	−6.81
(100, 15)	1514.00	596.72	294,831.89	0.13	107.15	−6.88	54.99	−0.03	33.65	−6.85
(100, 20)	2019.00	797.34	393,036.52	0.13	107.63	−6.86	55.56	−0.03	33.47	−6.83
(100, 25)	2512.00	992.00	489,385.73	0.13	107.60	−6.83	55.82	−0.03	33.23	−6.80
(200, 5)	1013.00	382.93	208,393.58	0.26	99.69	−6.62	55.84	−0.03	28.14	−6.59
(200, 10)	2002.00	761.00	412,093.03	0.26	149.42	−6.63	54.87	−0.03	61.05	−6.61
(200, 15)	3024.00	1159.43	621,766.64	0.26	101.17	−6.63	56.14	−0.03	28.83	−6.60
(200, 20)	4013.00	1544.80	824,771.22	0.26	102.17	−6.64	56.29	−0.03	29.35	−6.61
(200, 25)	5021.00	1938.15	1,031,895.77	0.26	102.32	−6.64	56.34	−0.03	29.41	−6.61
(400, 5)	2020.00	741.93	428,089.05	0.53	103.39	−6.94	57.13	−0.04	29.44	−6.90
(400, 10)	4005.00	1496.38	847,735.55	0.53	155.17	−6.93	56.93	−0.04	62.59	−6.89
(400, 15)	6037.00	2257.87	1,276,731.59	0.52	104.38	−6.92	56.75	−0.04	30.38	−6.88
(400, 20)	8022.00	3008.03	1,695,380.98	0.53	104.11	−6.92	56.58	−0.04	30.36	−6.89
(400, 25)	10033.00	3773.31	2,121,735.60	0.53	104.73	−6.92	57.08	−0.04	30.33	−6.88
Average	1572.38	609.55	319,983.88	0.14	126.67	−7.40	50.90	−0.06	51.05	−7.34

explore the trade-off between costs and latency in multi-cloud environments. In the following, we first briefly describe the methodology for collecting and preparing input data as well as the configurations and scenarios that are applied in the different simulation experiments. Finally, we present the simulation results of the different experiments and discuss major outcomes.

6.1. Simulation setup

6.1.1. VM type microbenchmarks

The simulations are performed with real datasets collected in August 2016 from three leading cloud providers: AWS, Microsoft, and Google.[1] Overall, we consider 49 different VM types configured for general, compute-intensive, memory-intensive, and storage-intensive purposes (18 VM types of Amazon EC2, 15 VM types of Microsoft Azure, 16 VM types of Google Compute Engine). These VM types can be deployed in several regions of respective cloud providers. In our dataset, we consider 8 AWS locations, 4 Google locations, and 14 Azure locations. To determine the speed of virtual CPUs in terms of MIPS, we perform a benchmark experiment using CMIPS v1.0.4,[2] a program written in C++ executing 200 threads performing a fixed number of operations, for each VM type. Although the provisioning of the processor can vary between Intel Xeon and AMD Opteron in case of Microsoft Azure, we focused on Intel processors to better compare the different providers. In this study, we assume that the performance of the disk and network is the same for all VM types. We further use real prices dependent on the selected operating system and deployment region. In this regard, we use prices per hour of use in our simulation experiments.

6.1.2. Location and latency measurement

In a real environment, the location of cloud consumers is either resolved from a respective IP address or specified by the consumers using geographical coordinates (i.e., longitude and latitude). To simulate and determine the location of IP addresses of consumers, we use the free GeoLite2[3] IP geolocation databases. The location of the different DCs of cloud providers is determined as accurately as possible using public available company data and news articles. As proposed in Grozev and Buyya (2016), we further use PingER,[4] an Internet Performance Monitoring Service (IEPM) maintained by the Stanford University, to receive network metrics between hundreds of distributed server nodes that periodically ping each other in a worldwide network.

[1] Amazon EC2: https://aws.amazon.com/ec2/instance-types; Microsoft Azure: https://azure.microsoft.com/de-de/services/virtual-machines; Google Compute Engine: https://cloud.google.com/compute/pricing.
[2] https://github.com/cmips/cmips.
[3] https://dev.maxmind.com/geoip/geoip2/geolite2/.
[4] http://www-iepm.slac.stanford.edu/pinger/.

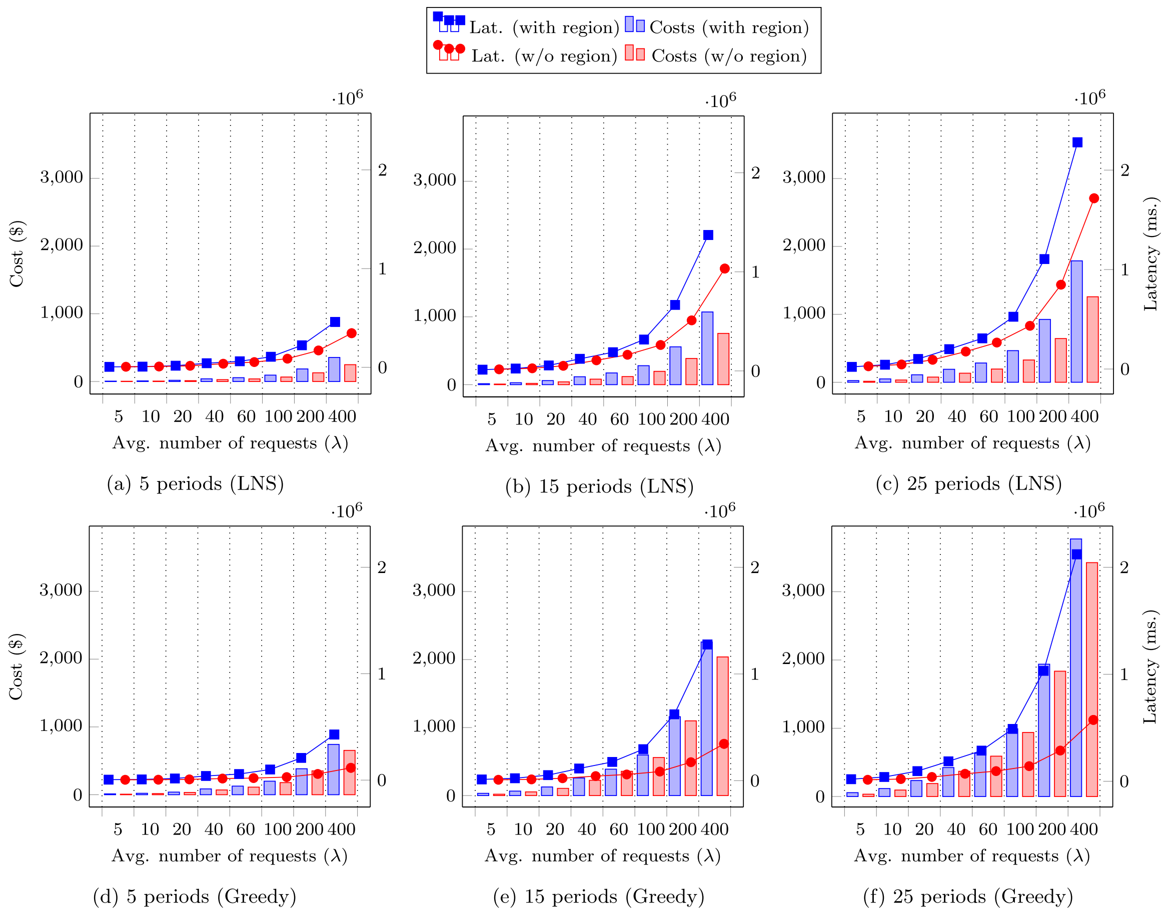

Fig. 4. Comparison of results of the extended CSPP with and without region constraints: cost optimization (first row) with $\alpha_1 = 1$, $\alpha_2 = 0$ and latency optimization (second row) with $\alpha_1 = 0$, $\alpha_2 = 1$.

We export those performance measures for one year in the time between August 2015 and July 2016. The coordinates of each PingER node are provided, too.

For estimating the network latency between worldwide distributed consumers and DCs of cloud providers, we take into account the latency between three pairs of PingER nodes. As depicted in Fig. 3, those pairs connect adjacent nodes of consumers and DCs. As not all nodes are linked to each other, we only consider combinations for which network performance statistics are available.

To determine the geographically closest nodes to consumer and DC locations (i.e., target locations), we measure the distance between each pair of coordinates, connecting nodes and target locations, using the haversine formula. Although the well-known Vincenty's formulae are more accurate taking into account the ellipsoidal shape of the Earth, the haversine formula, assuming a simple sphere Earth shape, provides a sufficient distance approximation in less computational time, which is more suitable for (near) real-time brokering schemes. Given the coordinates (ϕ_1, λ_1) and (ϕ_2, λ_2) and a mean sphere radius of $r = 6371$ km, the distance d is measured as follows.

$$dist = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_1 - \phi_2}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{5}$$

More formally, the latency between two locations t_1 and t_2 is approximated by choosing three possible connections of nodes (n_{11}, n_{12}), (n_{21}, n_{22}), (n_{31}, n_{32}) that minimize the distance d for t_1 and t_2. The latency between t_1 and t_2 is approximated using the following weighted sum.

$$latency(t_1, t_2) = \left(\sum_i \frac{d_{min} \cdot l_i}{d_i}\right) / \left(\sum_i \frac{d_{min}}{d_i}\right) \tag{6}$$

where d_i is the overall distance of a pair i between the chosen nodes n_{i1}, n_{i2} and the target locations t_1, t_2, respectively (Grozev and Buyya, 2016). The smallest distance among the three pairs is denoted as d_{min} and the measured latency between the respective nodes n_{i1} and n_{i2} is represented by l_i. Using this formula, the latency between a consumer and all available cloud DC locations is estimated. In the example shown in Fig. 3, the links between adjacent nodes of a customer t_1 and a DC t_2 are shown (dashed line). While in the first case the three closest nodes can be used to connect t_1 with t_2, it is not possible to use all of the closest nodes adjacent to *DC3* as one is not directly connected to a node adjacent to t_1.

6.1.3. Algorithm parameter settings

By taking into account the indications provided in Pisinger and Ropke (2010) and preliminary simulation runs, we identified the following parameters for LNS and ALNS (see Section 5): $\xi = 0.3$, $seg_{iter} = 300$, $iter_{max} = 2000$, $p = 4$, $\sigma_1 = 33$, $\sigma_2 = 13$,σ_3.

$= 9$, $r = 0.1$

6.1.4. Simulation settings and workload scenarios

In this work, we assume that the resource requirements of the requests are known a priori through means of statistical analysis. Therefore, we generate a large set of different scenarios to comprehensively assess the performance of the proposed multi-criteria decision making approaches and the cost-latency trade-off for multiple periods. First of all, we define different parameter values that are used in each simulation experiment (Table 1). Consequently, one simulation experiment consists of several simulations by combining the given parameter values.

As we could not find workloads fitting our purposes in the literature, we generate a suite of 40 different workloads for each combination of the number of periods and average number of requests per period. For this purpose, we generate a random number of requests per period from the Poisson distribution with a given mean λ. For each request, we generate resource requirements in terms of processing, memory, and disk capacity. Moreover, an execution deadline, required deployment region, and the location of origin are defined. Table 2 contains the ranges of values used for generating the requirements based on a uniform distribution. The location of origin is generated by randomly choosing a country and by randomly selecting a location in this country using data from GeoLite2 (see Section 6.1.2). The deadline of a request is a uniformly distributed value between a minimum reflecting the time it takes to execute the request with the smallest VM type and a maximum denoting the required time with the largest VM type in terms of the processing capacity. To better compare the different scenarios, dynamic aspects are not yet considered and are left for future research.

6.2. *Simulation results*

In this section, we present the results of the simulation experiments and compare the performance of the presented algorithms based on the different simulation settings and workload scenarios (see Section 6.1.4). We analyze the simulation results with regards to two main aspects:

- *Experiment 1*: Assessment of computational performance of the different algorithms and implications of multi-criteria simulation scenarios.
- *Experiment 2*: Assessment of the region constraints impact in the decision making process.

To put the focus on primary results, we show only the results for decision point intervals of 1800 s (0.5 h) as it exhibits comparatively better results for nearly all scenarios.

6.2.1. *Experiment 1: performance assessment*

Using the implemented *CloudSim* simulation framework extension, proposed in Section 4, we conduct several simulations and simulation runs devoted to assessing the presented algorithms for the extended CSPP. To ensure comparability among the simulations, we execute each simulation on the same machine, which is equipped with an Intel Xeon E5-2667 3.3 GHz and 128 GB of RAM. Each scenario has been executed ten times.

In Tables 3–5, we present the average simulation results pertaining the objective function values in terms of both cost and latency. Overall, we assess 40 different scenarios as described in Table 1. Moreover, the tables show the average computational time per simulation period *t(s.)* and the percentage of improvement *Impr. (%)* provided by the large neighborhood approaches compared to the greedy approach. Table 3 presents the results of a sole cost optimization with (*Config. 1*: $\alpha_1 = 1$, $\alpha_2 = 0$), whereas results of a sole latency optimization with (*Config. 3*: $\alpha_1 = 0$, $\alpha_2 = 1$) are presented in Table 5. Regarding the latter, we do not include the results of the LNS and ALNS algorithms. The rationale behind is that the greedy approach already chooses the best possible location, also for a higher price, when costs play no role. In Table 4, we consider the cases where weights are specified for the different objectives. Namely, we consider the case where equal weights (*Config. 2*: $\alpha_1 = 0.5$, $\alpha_2 = 0.5$) are specified to treat the two objectives equally. Given the region constraints, this implies that the algorithms search for a compromise between a close DC in terms of the network latency (within the given region) and a low price. In this work, the two objectives are normalized due to their distinct range by dividing the objectives by the maximum costs and maximum latency times the number of assigned requests, respectively, so that both objective values are always between 0 and 1. Note that we use these values to evaluate solutions during the optimization and only present the corresponding real values in terms of costs and latency, respectively.

Given the computational results, we first observe that the results of the greedy approach can be significantly improved by using the presented large neighborhood search approaches. The average improvement over all scenarios lies approximately between 10% and 12% regarding the cost optimization (*Config. 1*; Table 3) and between 6% and 9% when both objectives are taken into account (*Config. 2*; Table 4). In this regard, the results of LNS exhibits a slightly better

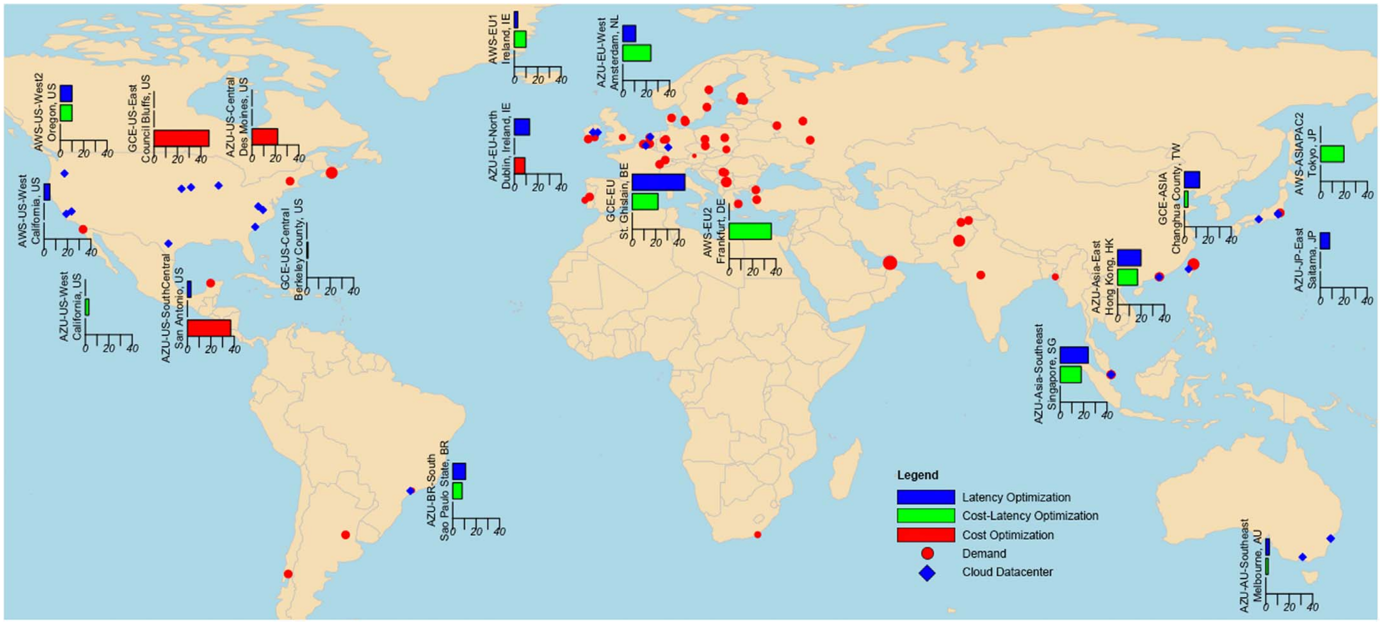

Fig. 5. Extended CSPP without region constraints: example of multi-criteria decision making visualization with LNS for 25 periods, λ= 60. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

solution quality. This indicates that a more complex removal heuristic, without the option of accepting worse solutions, is preferable for the selected settings. Moreover, we observe that the percentage of improvement is increasing the more requests are handled. This is due to the fact that the potential of bundling requests, if the consumer allows sharing VM hours among requests, is higher. Moreover, the results slightly indicate that the potential for improvements is growing by the number of periods. This emphasizes that a more frequent planning is highly beneficial for better utilizing idling VMs.

With regards to the multi-criteria case (*Config. 2*; Table 4), we observe that the greedy approach already finds a good compromise between costs and latency for a small number of requests so that it is not possible to further improve the results. This can be explained by the fact that for smaller problem instances, it is more likely that most of the requests are assigned to a dedicated VM type. Since the capacities are provided and charged on an hourly basis, remaining requests are assigned to booked VMs without changing the overall costs in one case when solely optimizing the costs (see Table 3, $\lambda = 5$, $k = 5$). This effect is intensified when the latency is equally considered in the objective function since it gets more difficult for the large neighborhood approaches, or becomes even impossible, to find better assignments when an additional objective is considered. However, we see that, especially for an increasing number of requests, there is a large potential to improve the assignment even if the region is restricted. In Section 6.2.2, the effects of the region constraints are covered more specifically. Compared to the results of the cost optimization (*Config. 1*), we see that the costs are considerably higher while the latency is lower, providing an indication of the compromise for decision makers, which is further analyzed in the following part.

Regarding the third configuration of preferences ($\alpha_1 = 0$, $\alpha_2 = 1$; Table 5), we analyze the impact of solely optimizing latency on the overall costs and latency. For this purpose, we compare the results of the greedy algorithm, providing optimal results for minimizing the latency, with the results of the other two weight configurations (*Config. 1* and *Config. 2*). Moreover, the table includes the comparison between those two configurations (*Config. 1/Config. 2*). Thereby, the general impact of the α weights can be assessed. It should be noted that we average the results of LNS and ALNS per problem instance to perform the comparison.

In general, it can be observed that the latency improvements come at a high price. By comparing the results with the ones of the sole cost optimization (*Config. 1*), we observe that the costs are more than doubled. The latency improvement of 7.4%, on average, leads to an increase of costs by 126.67%. Comparing the results to the second configuration ($\alpha_1 = 0.5, \alpha_2 = 0.5$) indicates that removing the cost objective ($\alpha_1 = 0$) only leads to a slight improvement in terms of latency. While the latency can be improved by only 0.06% on average, the overall costs increase by 50.90%. In this regard, the comparison between *Config. 1* and *Config. 2* shows that the improvement of the latency is particularly achieved by incorporating latency as a second objective in the objective function. As reported in the table, the average improvement of 7.35% still comes at a high cost increase of 51.05%. In particular for smaller problem instances, we can observe considerable differences between the weight configurations. This can be again explained with the individual requirements of requests. As the percentage of requests using a dedicated VM is usually higher for smaller problem instances, changes regarding the assignment towards a lower latency consequently have a higher impact on the latency.

The overall explanation for the huge sensitivity partly lies in the considerable price differences of VM types offered by the various cloud providers in different regions. Considering the region constraints, the brokering mechanism assigns requests to the cheapest VM type deployed in a DC within the allowed region. By incorporating the latency objective in the objective function, a compromise is achieved by assigning requests to DCs within the allowed region, where the required VM types might be more expensive. Removing the cost objective completely (*Config. 3*) allows to assign the requests to the DCs implying the lowest latency. This leads to the fact that each request is assigned to a dedicated VM type in the closest DC within the predefined region. While this leads to a high cost increase, the latency is not markedly improved. Thus, we see that the compromise already provides a good solution if decision makers aim to economically reduce network latency.

Finally, we analyze the computational time of the algorithms. In general we see that, given the relatively small number of requests per period, an initial solution can be found very quickly by the greedy approach. Both large neighborhood approaches need approximately between 94 and 230 seconds per period on average for providing a solution. For the smaller scenarios, e.g., 5 – 40 requests, the solution is provided in less than one minute. As the CSPP is an $\mathcal{NP}$-hard problem (see Heilig et al., 2016), it can be stated that both algorithms exhibit satisfying computational times that can be accepted in real cloud deployment processes. While providing almost the same solution quality, ALNS greatly outperforms LNS in terms of computational time, in particular for an increasing number of requests. This is mainly due to the complex removal heuristic used in LNS, as indicated above.

6.2.2. Experiment 2: impact of region constraints

As the deployment region is an important aspect in cloud-related decision processes, not only because of legal constraints, but also because of performance issues, we separately assess the impact of the region constraints in this suite of experiments. That is, we compare the simulation results considering the region requirement of each request with the results of the extended CSPP by removing related constraints. That means that the algorithms are free to choose any region (and DC) for executing requests. Thereby, it is possible to evaluate the impact of the region specification on both costs and latency.

In Fig. 4, we compare the results of the extended CSPP with and without region constraints for a cost optimization problem ($\alpha_1 = 1, \alpha_2 = 0$) and for a latency optimization problem ($\alpha_1 = 0, \alpha_2 = 1$). In all cases, we see that the region constraints have a huge impact on the costs, on average 29.01% for the cost optimization problem (using LNS) and 15.29% for the latency optimization problem (using the greedy approach). Regarding the latency, we observe an interesting pattern considering the cost optimization problem. While the latency is, due to the larger cost-saving potential, coming with a greater flexibility, higher than before for a small number of requests (5 – 10 per period), we observe that the latency improves for scenarios with a larger number of requests ($\lambda \geq 20$). This is due to the fact that with a greater flexibility, it is also possible to bundle more requests with similar locations of origins (i.e., closeness), which, even though latency is decreasing, leads to better cost results.

In case of a sole latency optimization, the results show that, besides the cost improvement of 19.83%, latency can be significantly reduced by 59.63% on average. Thus, the impact of removing the region constraints is larger in the latency optimization than in the cost optimization due to the fact that it allows to choose always the closest locations in terms of latency. However, as consumers need to act economically, the truth usually lies somewhere in between, emphasizing the importance of multi-criteria decision support.

Overall, our simulation results illustrate the importance of analyzing the regional requirements of requests, in particular whether the deployment region needs to be restricted to a particular region – those decisions have a major impact on cost and experienced latencies. In this sense, a higher awareness will postulate to make a clear distinction between requests (e.g., processing non-sensitive and sensitive information) instead of simply restricting a bunch of requests to a particular region without reviewing legal requirements and privacy concerns.

6.3. Multi-criteria decision support visualization

The presented multi-criteria optimization approaches support location-aware multi-cloud brokering from the consumer perspective. Depending on the requirements and preferences of consumers, requests can be automatically assigned to cloud DCs of different cloud providers around the globe. To reduce the complexity of those results, a visualization helps to better understand implications of solutions. Fig. 5 depicts an example of a visualization for comparing different locations with respect to optimization preferences for the extended CSPP without region constraints. In the figure, the red dots depict requests and their size the total amount of requests. The blue squares indicate the locations of DCs of different cloud providers. For each location that has been used for deploying VMs, a bar chart shows the number of requests that have been assigned to this location with respect to the different preferences. For problems with a sole cost preference ($\alpha_1 = 1$, $\alpha_2 = 0$), it can be observed that most of the requests are allocated to DCs in middle and southern regions of the United States. We see that this is changing when the preference goes towards a compromise solution taking into account both cost and latency, or even more drastically, when considering only the latency. In those cases, the visualization shows that geographically closer DCs, which were not selected before, are more often considered.

In this sense, optimization approaches and visualizations may help to better control current deployments and to understand the dynamics in cloud marketplaces, e.g., by additionally analyzing the impact of price changes, performance variations, and demand patterns. The proposed extension of *CloudSim* as well as the optimization approaches build a common foundation to establish such a decision support tool for consumer-related multi-cloud brokering schemes.

7. Conclusions and future directions

With the rapid adoption of cloud computing and the growing range of cloud marketplace offerings of different cloud providers, consumer-oriented brokering schemes supporting an economically viable selection and utilization of suitable cloud services have become essential. In this paper, we present novel extensions of *CloudSim* allowing to model, simulate, and evaluate brokering schemes that support cloud consumers in selecting and utilizing cloud resources in multi-cloud environments. Thus, the extensions allow to evaluate approaches aiding the use of cloud computing from a consumer perspective. In this regard, one of the core new components is an optimization module containing decision support functionality for multi-cloud brokers acting on behalf of the consumer. Given a recent optimization problem, referred to as the Cloud Service Purchasing Problem (CSPP), we propose a multi-criteria problem formulation extension. That is, besides costs for using VM types of different providers, we consider the network latency between consumers and cloud DCs. To solve this problem, we integrate a greedy heuristic and two large neighborhood search approaches in *CloudSim*. By conducting VM performance microbenchmarks, collecting real VM type descriptions and prices of leading cloud providers, and generating different workloads, we provide a well-defined set of simulation scenarios. Using these scenarios, we conduct a large number of simulations to evaluate the performance of our approaches and to assess the trade-off between costs and latency as well as to analyze the impact of region constraints on both objectives.

The results demonstrate a promising performance of our large neighborhood search approaches, both in terms of solution quality and computational time. Approximately 10 – 12% of the costs can be saved compared to the greedy approach. Comparing the two large neighborhood search heuristics, LNS exhibits a slightly better solution quality for the tested settings and scenarios, while ALNS implies a better scaling behavior in terms of the computational time, solving all tested scenarios in less than two minutes per period. An explanation of this outcome is that LNS only uses a sophisticated and complex removal heuristic and does not allow to accept worse solutions, which seems to be particularly important in the beginning of the search process. The results further indicate that the potential cost and latency benefits increase by the number of requests to be handled by the broker. Moreover, we analyze the relationship between cost and latency preferences and discuss the conflicting nature of these two objectives. In this sense, the results demonstrate that improvements of the latency usually come at a high price. When solely optimizing the latency by 7.4%, the costs increase by over 126%, on average, for the tested scenarios. We further demonstrate that removing the cost objective from the objective functions leads to undesirable cost increases as the latency cannot be drastically improved compared to the compromise solution where both costs and latency are considered. Moreover, we analyze the impact of region constraints as the deployment location is a major concern of cloud adopters. The results strongly indicate the cost and latency benefits of removing the region constraints. A particularly interesting pattern has been observed within cost optimization scenarios. Whereas the latency usually increases when solely optimizing costs, indicating the conflict of those objectives, for a larger number of requests, we see that the latency is improving. This results from a greater flexibility to bundle requests in more adjacent cloud DCs. We furthermore provide an example how results can be visualized to better understand the dynamics in cloud marketplaces. Overall, the simulation experiments emphasize the importance of multi-criteria decision analysis, where decision makers need to analyze the effects of their preferences, as well as the need for corresponding decision support functionality, such as in form of the proposed algorithms and visualizations. In this context, the proposed *CloudSim* extensions provide a foundation to develop and evaluate novel static and dynamic optimization approaches. Consequently, our approach may promote research activities in the business-oriented direction of cloud computing research.

Several important aspects have been left for future research. First, we aim to further extend the functionality of *CloudSim* to consider additional characteristics of cloud marketplaces and multi-clouds. This includes modeling specific aspects such as additional pricing models (e.g., reservation pricing and bidding) and tools for rating and considering the performance of cloud providers based on user-generated feedback and statistics. Moreover, we intend to extend simulation experiments by considering dynamic aspects, such as performance variations as well as different workload distributions in order to better analyze the scaling behavior and resulting economic and performance implications.

Acknowledgments

We would like to thank Adel Nadjaran Toosi for his constructive suggestions on this article.

References

Assis, M., Bittencourt, L., 2016. A survey on cloud federation architectures: identifying functional and non-functional properties. J. Netw. Comput. Appl. 72, 51–71.
Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I., 2009. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. 25 (6), 599–616.
Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R., 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw.: Pract. Exp. 41 (1), 23–50.
Chaisiri, S., Lee, B.-S., Niyato, D., 2012. Optimization of resource provisioning cost in cloud computing. IEEE Trans. Serv. Comput. 5 (2), 164–177.
Chekuri, C., Khanna, S., 2012. On multi-dimensional packing problems. SIAM J. Comput. 33 (4), 837–851.
Cisco, 2015. Cisco Global Cloud Index: Forecast and Methodology, 2014–2019, URL ⟨http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf⟩.
Coutinho, R.d.C., Drummond, L.M.A., Frota, Y., 2013. Optimization of a cloud resource management problem from a consumer perspective. In: Euro-Par Parallel Processing Workshops. LNCS 8374. Springer, Berlin. pp. 218–227.
Coutinho, R.d.C., Drummond, L.M.A., Frota, Y., de Oliveira, D., 2015. Optimizing virtual machine allocation for parallel scientific workflows in federated clouds. Future Gener. Comput. Syst. 46, 51–68.
Do, C.T., Tran, N.H., Huh, E.-N., Hong, C.S., Niyato, D., Han, Z., 2016. Dynamics of service selection and provider pricing game in heterogeneous cloud market. J. Netw.

Comput. Appl. 69, 152–165.
Fréville, A., 2004. The multidimensional 0–1 knapsack problem: an overview. Eur. J. Oper. Res. 155 (1), 1–21.
Gartner, 2009. Gartner Says Cloud Consumers Need Brokerages to Unlock the Potential of Cloud Services. URL ⟨http://www.gartner.com/newsroom/id/1064712⟩.
Grozev, N., Buyya, R., 2016. Regulations and latency-aware load distribution of web applications in multi-clouds. J. Supercomput. 72 (8), 3261–3280.
Heilig, L., Voß, S., Wulfken, L., 2015. Building clouds: An integrative approach for an automated deployment of elastic cloud services. In: Chang, V., Walters, R., Wills, G. (Eds.), Delivery and Adoption of Cloud Computing Services in Contemporary Organizations, IGI. Hershey, PA. pp. 269–290.
Heilig, L., Voß, S., 2014. Decision analytics for cloud computing: a classification and literature review. In: A. Newman, J. Leung (Eds.), Tutorials in Operations Research, INFORMS. Catonsville. pp. 1–26.
Heilig, L., Lalla-Ruiz, E., Voß, S., 2016. A cloud brokerage approach for solving the resource management problem in multi-cloud environments. Comput. Ind. Eng. 95, 16–26.
Heilig, L., Lalla-Ruiz, E., Voß, S., 2016. Cloud service purchasing in multi-cloud environments. Tech. rep., Working Paper. Institute of Information Systems, University of Hamburg.
Liaqat, M., Chang, V., Gani, A., Ab Hamid, S.H., Toseef, M., Shoaib, U., Ali, R.L., 2017. Federated cloud resource management: review and discussion. J. Netw. Comput. Appl. 77, 87–105.
Liotine, M., 2003. Mission-Critical Network Planning. Artech House, Boston.
Lucas Simarro, J.L., Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M., 2011. Dynamic placement of virtual machines for cost optimization in multi-cloud environments. In: Proceedings of the International Conference on High Performance Computing and Simulation (HPCS). pp. 1–7.
Masdari, M., Nabavi, S.S., Ahmadi, V., 2016. An overview of virtual machine placement schemes in cloud computing. J. Netw. Comput. Appl. 66, 106–127.
Moser, M., Jokanovic, D.P., Shiratori, N., 1997. An algorithm for the multidimensional multiple-choice knapsack problem. IEICE Trans. Fundam. Electron., Commun. Comput. Sci. 80 (3), 582–589.
Pandey, S., Wu, L., Guru, S.M., Buyya, R., 2010. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA). pp. 400–407.
Petcu, D., 2013. Multi-cloud: expectations and current approaches. In: Proceedings of the International Workshop on Multi-cloud Applications and Federated Clouds (MultiCloud). pp. 1–6.
Pisinger, D., Ropke, S., 2010. Large neighborhood search. In: Gendreau, M., Potvin, J.-Y. (Eds.). Handbook of Metaheuristics, 2nd Edition. Springer, New York. pp. 399–419.
Plummer, D.C., Kenney, L.F., 2009. Three Types of Cloud Brokerages Will Enhance Cloud Services, URL ⟨https://www.gartner.com/doc/973412/types-cloud-brokerages-enhance-cloud⟩.
Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.-F. (Eds.), Principles and Practice of Constraint Programming. LNCS 1520. Springer, Berlin. pp. 417–431.
Shen, S., Deng, K., Iosup, A., Epema, D., 2013. Scheduling jobs in the cloud using on-demand and reserved instances. In: Wolf, F., Mohr, B., an Mey, D. (Eds.), Euro-Par Parallel Processing. LNCS 8097. Springer, Berlin. pp. 242–254.
Toosi, A.N., Calheiros, R.N., Buyya, R., 2014. Interconnected cloud computing environments: challenges, taxonomy, and survey. ACM Comput. Surv. 47 (1), 7:1–7:47.
Tordsson, J., Montero, R.S., Moreno-Vozmediano, R., Llorente, I.M., 2012. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Gener. Comput. Syst. 28 (2), 358–367.
Van den Bossche, R., Vanmechelen, K., Broeckhove, J., 2010. Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD). pp. 228–235.
Verginadis, Y., Patiniotakis, I., Mentzas, G., 2014. D20.1 State of the Art and Research Baseline. Tech. rep., URL ⟨http://www.broker-cloud.eu/documents/deliverables/d2-1-state-of-the-art-and-research-baseline⟩.
Wood, T., Cherkasova, L., Ozonat, K., Shenoy, P., 2008. Profiling and modeling resource usage of virtualized applications. In: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware). pp. 366–387.

Leonard Heilig is a Ph.D. candidate at the Institute of Information Systems at the University of Hamburg. He holds a B.Sc. (University of Münster, Germany) and a M.Sc. (University of Hamburg, Germany) in Information Systems. His current research interest is centered around the adoption and utilization of cloud computing in contemporary enterprises with the focus on IT governance, decision support, mobile cloud applications, and its application in the maritime sector. He spent some time at the University of St Andrews (Scotland, UK) and, most recently, at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. Currently he serves as guest editor for the Information Technology & Management journal special issue on information systems and big data in maritime logistics and seaports. His industry experiences in cloud computing include working as a software engineer at Adobe Systems participating in the development of their PaaS environment and at Fiducia & GAD IT, one of the largest IT providers in Germany, focusing on cloud-based banking applications. Further experiences include positions at Beiersdorf Shared Services and Airbus Group Innovations on security management.

Dr. Rajkumar Buyya is a Fellow of IEEE, Professor of Computer Science and Software Engineering, Future Fellow of the Australian Research Council, and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored over 525 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He also edited several books including "Cloud Computing: Principles and Paradigms" (Wiley Press, USA, Feb 2011). He is one of the highly cited authors in computer science and software engineering worldwide (h-index=106, g-index=221, 53,500+ citations). Recently, Dr. Buyya is recognized as "2016 Web of Science Highly Cited Researcher" by Thomson Reuters. Software technologies for Grid and Cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. Dr. Buyya has led the establishment and development of key community activities, including serving as foundation Chair of the IEEE Technical Committee on Scalable Computing and five IEEE/ACM conferences. These contributions and international research leadership of Dr. Buyya are recognized through the award of "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE Computer Society TCSC. Manjrasoft's Aneka Cloud technology developed under his leadership has received "2010 Frost & Sullivan New Product Innovation Award". He served as the founding Editorin- Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Co-Editor-in-Chief of Journal of Software: Practice and Experience, which was established over 45 years ago. For further information on Dr. Buyya, please visit his cyberhome: http://www.buyya.com

Dr. Stefan Voß is professor and director of the Institute of Information Systems at the University of Hamburg. He also holds a visiting position at PUCV in Valparaiso, Chile. Previous positions include full professor and head of the department of Business Administration, Information Systems and Information Management at the University of Technology Braunschweig (Germany) from 1995 up to 2002. He holds degrees in Mathematics (diploma) and Economics from the University of Hamburg and a Ph.D. and the habilitation from the University of Technology Darmstadt. His current research interests are in quantitative/information systems approaches to supply chain management and logistics including public mass transit and telecommunications. He is author and co-author of several books and numerous papers in various journals. Stefan Voß serves on the editorial board of some journals including being Editor of Netnomics and Editor of Public Transport. He is frequently organizing workshops and conferences. Furthermore, he is consulting with several companies.