Ulrich Lampe, Melanie Siebenhaar, Dieter Schuller, Ralf Steinmetz:
A Cloud-Oriented Broker for Cost-Minimal Software Service Distribution.
In: Proceedings of the Second Optimising Cloud Services Workshop (OCS 2011), October 2011.

A Cloud-Oriented Broker for Cost-Minimal Software Service Distribution

Ulrich Lampe, Melanie Siebenhaar, Dieter Schuller, and Ralf Steinmetz

Multimedia Communications Lab (KOM)
Technische Universität Darmstadt, Germany
Corresponding Author: Ulrich.Lampe@KOM.tu-darmstadt.de

Abstract. In a cloud computing market, Software as a Service providers face the *Dependent Software Service Distribution Problem*, i. e., the challenge of cost-minimally distributing partially dependent software service instances across infrastructure in the form of discrete virtual machines. In the work at hand, we introduce the *Software Service Distribution Broker* to address this challenge. As integral component of the broker, we propose a mathematical optimization model and corresponding algorithm. An evaluation, which is based on data from actual software services and cloud systems, indicates the applicability of our approach for small-scale scenarios involving approximately 20 software service instances or less, but also illustrates the need for the development of heuristic solution approaches.

1 Introduction

During the past few years, cloud computing has gained tremendous popularity. A common theme of this novel paradigm is the provision of IT capacities in a utility-like fashion: scalable, on-demand, and via public networks [6]. This permits IT users to lease capacities, rather than providing them in-house [10]. A popular service model in this context is the provision of Infrastructure as a Service (IaaS) in the form of Virtual Machines (VMs), such as in Amazon's Elastic Compute Cloud (EC2). A major advantage of this model is that VMs are highly customizable and thus may run most existing software without any specific adaptation. On the downside, VMs are *discrete* compute units, which means that they have to be explicitly instantiated and only scale up to the level of the physical machines that host them [5].

Based on these observations, we assume the following scenario, which is inspired by the three-layered cloud model of Armbrust et al. [2]: Existing software is offered in the form of Software as a Service (SaaS) by an *SaaS provider*. Instances of the resulting software services are requested by *SaaS users* and executed using the leased infrastructure, i. e., VM instances, of various *IaaS providers*. The scenario further exhibits the following characteristics: Each requested Software Service Instance (SSI) has a specific resource demand throughout its execution. Likewise, IaaS providers offer different VM types, which supply a certain limited

amount of these resources at differing costs. In addition, SSIs may have dependencies, which result in resource demands or Quality of Service (QoS) requirements between pairs of SSIs. Likewise, the IaaS providers specify resource supplies or QoS requirements for the links between pairs of VM types. Lastly, we assume that SaaS users may request and terminate SSIs at arbitrary points in time, resulting in an unknown duration of execution, and that the execution of SSIs is not interruptible, i. e., that each SSI has to be *continuously* executed between its initiation and termination.

A practical example of the outlined scenario is cloud gaming: A game provider offers video games, which are executed on VMs that transmit the resulting audio/video stream to end users' playback devices and in turn receive control commands from these devices. Not only may various video games be offered, but these games may also be requested at different visual quality levels depending on the respective playback device. Thus, during its execution, each game – SSI – exhibits a specific and continuous resource demand on the VM instance that executes it, while this VM instance is subject to a restricted resource supply. Additionally, in the case of multi-player games, those SSIs belonging to the identical session will exchange data subject to certain QoS constraints, such as maximal latency.

Under the assumption that the SaaS provider pursues the rational objective of minimizing the infrastructure leasing cost, the research challenge consists in the distribution of a set of requested SSIs – which have to be continuously executed – across a set of leased VMs instances, such that:

1. The resource demands of all SSIs are met by the available resource supplies of the respective VM instances where the SSIs are executed.
2. The QoS requirements between SSIs are met by the QoS guarantees between the respective VM instances where the SSIs are executed.
3. The overall cost of leasing the VM instances is minimized.

This poses a combined capacitation (number of leased instances of each VM type) and assignment (allocation of SSIs to specific VM instances) problem, which we refer to as *Dependent Software Service Distribution Problem* (DSSDP) and address in the work at hand.

The remainder of this work is structured as follows: In Section 2, we present our solution approach, an integrated broker that addresses the DSSDP through optimization approaches, based on a formal mathematical model of the problem. A prototypical implementation of this broker is evaluated in Section 3. Section 4 provides an overview of related work. Eventually, Section 5 concludes the paper with a summary and outlook.

2 Software Service Distribution Broker

2.1 Outline of the Broker

Our research aims at the design and implementation of an integrated *Software Service Distribution Broker*, which addresses the DSSDP in the broader context

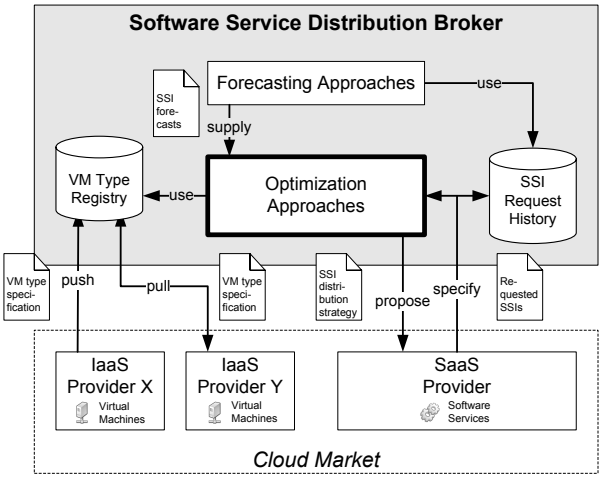

Fig. 1. Architectural overview of the Software Service Distribution Broker.

of a future cloud market. The broker will be accessible for SaaS providers and IaaS providers in this market through a Web-based interface. The interface permits IaaS providers to register their VM type offers by submitting the relevant information regarding, e. g., resource supply and price, in a structured form (*push* model). The tool will also allow to mine publicly available VM offer descriptions, e. g., through the Amazon EC2 API[1] (*pull* model). Likewise, SaaS providers may specify the SSIs that have been requested by their end users.

Based on the submitted information, the Software Service Distribution Broker computes a *SSI distribution strategy* for the SaaS provider using suitable optimization approaches. Such strategy defines how many instances of each VM type should be leased, and on which of those VM instances the requested SSIs should be executed in order to achieve minimal infrastructure leasing cost[2].

In addition, the broker features a forecasting component. It operates on historic SSI request and execution data, which is stored in a central database. Based on different forecasting approaches, the component allows to estimate the most likely duration of execution for currently requested SSIs. It may also predict the expected SSI requests at any future point in time. This, for instance, permits to exploit lower leasing costs through the advanced booking or reservation of VM instances.

Overall, in accordance with the vision of a future cloud market [6], the broker facilitates the efficient distribution of SSIs through SaaS providers. Specifically, it permits the realization of cost savings in leasing infrastructure from the cloud. An architectural overview of the Software Service Distribution Broker is depicted in

[1] `http://aws.amazon.com/documentation/ec2/`

[2] Please note that the term "VM type" denotes an *abstract* VM offer, whereas "VM instance" denotes an *concrete* instantiation of such type.

Figure 1. The component *optimization approaches* has been highlighted, because it constitutes the focus of the work at hand.

2.2 Formal Model of the Dependent Software Service Distribution Problem

In order to address the DSSDP through optimization approaches and thus permit the computation of SSI distribution strategies, we formalize the problem as a mathematical model.

This model is based on the following assumptions: First, there exists at least one leaseable VM type that provides a sufficient overall resource supply for each regarded SSI. If this condition is not met, the SSI cannot be hosted on the offered infrastructure at all. Second, SSIs may be arbitrarily combined on one VM instance, as long as the overall resource demands of all SSIs are satisfied by the overall resource supply of the corresponding VM instance. Third, we assume that the links between pairs of VM types are shared by all instances of the same types, which permits to treat the resource demands and supplies between different VM types as aggregate.

Table 1 provides an overview of the used symbols. The upper part of the table contains the basic entities in the model, i. e., SSIs, VM and resource types, and QoS attributes. The remainder of the table introduces data specifications that are based on these entities.

In accordance with our previously described scenario, resource demands and supplies are specified on an *intra-machine* and *inter-machine* level, i. e., for individual SSIs and VM types, as well as pairs of SSIs and links between VM types. In contrast, QoS requirements and guarantees are only specified on an *inter-machine* level, because they exclusively apply to pairs of SSIs and links between VM types respectively. Without loss of generality, we assume that QoS requirements are expressed in terms of maximal permissible values.

On the *intra-machine* level, billing is defined both on a per-instance and per-unit basis. Per-instance means that a flat fee is billed for an VM instance per (partial) billing period. Per-unit means that a cost applies for each consumed unit of a resource type. On the *inter-machine* level, only the latter scheme is applied. However, we additionally differentiate between *incoming* and *outgoing* resource consumption on this level. Thus, for example, the outgoing network traffic from one VM type can be billed as incoming network traffic on a linked VM type.

The resulting mathematical model is given in Model 1. The decision variables are defined in Equations 16 to 19. The binary decision variables x_{svi} indicate whether the SSI s has been assigned to a VM of type v with the instance index i or not. The integer decision variables y_{vi} indicate the number of subsequent billing periods for which a certain instance i of VM type v is leased. The binary decision variables z_{sv} indicate whether SSI s has been assigned to a VM instance of type v or not. Lastly, the binary decision variables $d_{svs'v'}$ serve the same function as z, but do so for a pair (s, s') of SSIs and corresponding pair of VM types (v, v'). The respective relations between the individual decision variables are established in Equations 12 to 14.

Table 1. Used symbols in the mathematical model.

Symbol	Description
S	Set of requested SSIs.
V	Set of leasable VM types.
$R^{\diamond}$	Set of considered intra-machine resource types.
$R^{\rightleftarrows}$	Set of considered inter-machine resource types.
Q	Set of considered QoS attributes.
$D_s \in \mathbb{R}^+$	Expected duration of execution of SSI $s \in S$ (in time units).
$RD^{\diamond}_{sr} \in \mathbb{R}^+$	Intra-machine resource demand of SSI $s \in S$ for resource type $r \in R^{\diamond}$.
$RD^{\rightleftarrows}_{ss'r} \in \mathbb{R}^+$	Inter-machine resource demand between SSIs $s, s' \in S$ for resource type $r \in R$.
$RS^{\diamond}_{vr} \in \mathbb{R}^+$	Intra-machine resource supply per instance of VM type $v \in V$ for resource type $r \in R^{\diamond}$.
$RS^{\rightleftarrows}_{vv'r} \in \mathbb{R}^+$	Inter-machine resource supply for the link between VM types $v, v' \in V$ for resource type $r \in R^{\rightleftarrows}$.
$QR_{ss'q} \in \mathbb{R}$	QoS requirements between SSIs $s, s' \in S$ regarding QoS attribute $q \in Q$.
$QG_{vv'q} \in \mathbb{R}$	QoS guarantees for the link between VM types $v, v' \in V$ regarding QoS attribute $q \in Q$.
$BI_v \in \mathbb{R}$	Length of an per-instance billing period for instances of VM type $v \in V$ (in time units).
$CI_v \in \mathbb{R}$	Cost per leased instance of VM type $v \in V$ per billing period BI_v.
$CU^{\diamond}_{vr} \in \mathbb{R}$	Intra-machine cost at VM type $v \in V$ per consumed unit of resource type $r \in R^{\diamond}$ per time unit.
$CU^{\rightarrow}_{vv'r} \in \mathbb{R}$	Inter-machine cost for the link between VM types $v, v' \in V$ per outgoing consumed unit of resource type $r \in R^{\rightleftarrows}$ per time unit.
$CU^{\leftarrow}_{vv'r} \in \mathbb{R}$	Inter-machine cost for the link between VM types $v, v' \in V$ per incoming consumed unit of resource type $r \in R^{\rightleftarrows}$ per time unit.

The objective is to minimize the Total Cost (TC) of leasing the selected VM instances in Equation 1. The total cost comprises four components: First, the per-instance cost of leasing VM instances across the respective number of billing periods (Equation 2). Second, the per-unit, intra-machine resource costs of the executed SSIs (Equation 3). Third and fourth, the per-unit, inter-machine, outgoing and incoming resource costs of dependent SSIs (Equations 4 and 5). For the latter, the minimal execution time of the dependent SSIs is decisive, i. e., once one of two dependent SSIs is terminated, the inter-machine resource consumption ceases.

The VM instance indices are defined in Equation 15. The set I consists of $|S|$ elements, each corresponding to a *potential* VM instance. The cardinality of the set is based on the notion that each SSI may be assigned to an individual VM instance of the same type in the worst case. Through this construct, the DSSDP can be treated as a pure assignment problem, rather than combined capacitation and assignment problem.

As far as the constraints are concerned, Equation 6 ensures that each SSI is assigned precisely once to a VM instance. Equation 7 guarantees that the resource constraints are held on the intra-machine level for each VM instance. Equation 8 serves the same purpose on the inter-machine level. Because we assume that links between VM types are shared, the constraints may be checked at an aggregate level. Equation 9 guarantees that the QoS requirements are satisfied for all dependent SSIs. For that matter, pairs of unsuitable VM types for all pairs of SSIs are determined in Equations 10 and 11.

As such, Model 1 only constitutes a *design time* approach that permits the computation of an *initial* SSI distribution strategy. However, at run time, SSIs may be additionally requested or terminated at any instant. This triggers the need for the computation of *incremental* SSI distribution strategies, which define adaptations to the leased infrastructure upon changes in the requested SSIs.

In the work at hand, we do not focus on this aspect, but only outline the principal adaptations that are required in the model: First, terminated SSIs are removed from the set S, and newly requested SSIs are added to it. Subsequently, the expected durations of execution for the sustained SSIs are updated. Under the assumption that sustained SSIs may not be moved between different VM instances, a new constraint is added to the model for each sustained SSI. These constraints enforce the assignment of sustained SSI to the same VM instance as in the previous optimization.

3 Evaluation of the Broker

To date, we have created a prototypical implementation of the Software Service Distribution Broker. As optimization algorithm, the broker implements Model 1, which can be transferred into an *Integer Program* (IP), i. e., a special form of Linear Program (LP) that exclusively contains integer and binary decision variables. Such IP can be solved using well-known methods from the field of operations research, most notably, *branch and bound* [7]. The algorithm will always compute the optimal, i. e., cost-minimal, solution to a DSSDP, but it can be very costly in terms of required computational power.

The prototypical broker has been implemented in Java. In the implementation, we employed *IBM ILOG CPLEX Optimizer*, a commercial solver framework for linear programs[3].

3.1 Evaluation Methodology

To assess the practical applicability of the prototypical broker approach, we have conducted an extensive evaluation. The focus of the evaluation lies on the performance of the employed optimization algorithm, as measured by the computation time for a SSI distribution strategy. Specifically, in order to assess the scalability of our approach, we created 20 different classes of DSSPSs with

[3] `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`

Model 1 Dependent Software Service Distribution Problem

$$\text{Min. } TC(x,y,z,d) = \alpha + \beta + \gamma + \delta \tag{1}$$

$$\alpha = \sum_{v\in V; i\in I} y_{vi} * CI_v \tag{2}$$

$$\beta = \sum_{s\in S; v\in V; i\in I; r\in R^{\diamond}} x_{svi} * RD^{\diamond}_{sr} * D_s * CU^{\diamond}_{vr} \tag{3}$$

$$\gamma = \sum_{s,s'\in S; v,v'\in V; r\in R^{\rightleftarrows}} d_{svs'v'} * RD^{\rightleftarrows}_{ss'r} * min(D_s, D_{s'}) * CU^{\rightarrow}_{vv'r} \tag{4}$$

$$\delta = \sum_{s,s'\in S; v,v'\in V; r\in R^{\rightleftarrows}} d_{svs'v'} * RD^{\rightleftarrows}_{s'sr} * min(D_s, D_{s'}) * CU^{\leftarrow}_{vv'r} \tag{5}$$

$$\sum_{v\in V; i\in I} x_{svi} = 1 \quad \forall s \in S \tag{6}$$

$$\sum_{s\in S} RD^{\diamond}_{sr} * x_{svi} \leq RS^{\diamond}_{vr} \quad \forall v \in V; i \in I; r \in R^{\diamond} \tag{7}$$

$$\sum_{s,s'\in S} RD^{\rightleftarrows}_{ss'r} * d_{svs'v'} \leq RS^{\rightleftarrows}_{vv'r} \quad \forall v, v' \in V; r \in R^{\rightleftarrows} \tag{8}$$

$$z_{sv} + z_{s'v'} \leq 1 \quad \forall (v,v') \in U_{ss'}; s, s' \in S \tag{9}$$

$$U_{ss'} = \cup_{q\in Q} U_{ss'q} \quad \forall s, s' \in S \tag{10}$$

$$U_{ss'q} = \{(v,v') \in V^2 | QR_{ss'q} \leq QG_{vv'q}\} \quad \forall s, s' \in S; q \in Q \tag{11}$$

$$y_{vi} \geq x_{svi} * \left\lceil \frac{D_s}{BI_v} \right\rceil \quad \forall s \in S; v \in V; i \in I \tag{12}$$

$$z_{sv} \geq x_{svi} \quad \forall s \in S; v \in V; i \in I \tag{13}$$

$$d_{svs'v'} \geq z_{sv} + z_{s'v'} - 1 \quad \forall s, s' \in S; v, v' \in V \tag{14}$$

$$I = \{1, ..., |S|\} \qquad I \subset \mathbb{N} \tag{15}$$

$$x_{svi} \in \{0,1\} \quad \forall s \in S; v \in V; i \in I \tag{16}$$

$$y_{vi} \in \mathbb{N} \quad \forall v \in V; i \in I \tag{17}$$

$$z_{sv} \in \{0,1\} \quad \forall s \in S; v \in V \tag{18}$$

$$d_{svs'v'} \in \{0,1\} \quad \forall s, s' \in S; v, v' \in V \tag{19}$$

varying size. Each class contained 200 valid individual problems with a different number of requested SSIs (n_s) and QoS attributes (n_q). In addition, we varied the ratio of SSIs that exhibit dependencies (r).

For the VM and resource types, we used the specifications of the Windows-based, on-demand Amazon EC2 VM offers in the European Union[4]. The specifications yield eight VM types, which provide limited supplies of three different intra-machine resource types (processor, memory, and hard disk storage) at differing per-instance costs. Based on the same specifications, we additionally consider Internet-bound network bandwidth and provider-internal network bandwidth as intra-machine and inter-machine resource type, respectively. Both types of network bandwidth are billed based on actual consumption. In total, four intra-machine and one inter-machine resource types are considered.

In accordance with the scenario of cloud gaming from Section 1, we obtained realistic SSI execution data by measuring the absolute resource demands of five contemporary video games on a local desktop computer. We further estimated the resource demands for Internet-bound and provider-internal network bandwidth of dependent SSIs. These demands result from the audio/video stream to the SaaS user and data exchange between the SSIs in a multi-player session.

Because no actual data for the QoS requirements and guarantees is available, we randomly drew the values from the intervals $[0, 1]$ and $[0, 0.5]$, respectively. Additionally, the expected duration of execution for the SSIs was randomly drawn from the interval $[1800, 9000]$. Assuming seconds as time unit, this range corresponds to a duration of 0.5 to 2.5 hours, which we believe to reflect a reasonable length for a cloud gaming session. In the final Software Service Distribution Broker, this information will be delivered by the forecasting component.

The generated DSSDPs in each class were subsequently attempted to solve using the prototypical broker. For each evaluated DSSDP, the computation time of the optimization algorithm was measured, with a timeout of 1 minute being in place. This timeout represents the maximum waiting time that a SaaS provider would be willing to accept for the computation of a SSI distribution strategy. For the evaluation, we employed a dedicated laptop computer, equipped with an Intel Core i5-450M processor and 2 GB of memory.

3.2 Evaluation Results and Discussion

Table 2 provides an overview of the evaluated DSSDP classes. The table also contains the number of decision variables and constraints in the corresponding optimization models as first indicator of computational complexity[5]. In addition, the number of problems per class that could be solved within the specified timeout

[4] `http://aws.amazon.com/ec2/#instance`

[5] Because the values for the QoS requirements and guarantees are randomly drawn, the number of unsuitable VM types and thus, the number of constraints is non-deterministic. Accordingly, the expected average number of constraints is provided.

Table 2. Overview of the evaluated DSSDP classes with computational complexity (DC: Number of decision variables; Con: Number of constraints) and ratio of solved problems.

Class			DC	Con.	Solved	Class			DC	Con.	Solved
n_s	n_q	r			[%]	n_s	n_q	r			[%]
10	0	0	7360	8394	100.0	20	0	0	29120	32724	100.0
10	1	0.125	7360	8594	100.0	20	1	0.125	29120	33524	100.0
10	2	0.125	7360	8744	100.0	20	2	0.125	29120	34124	98.5
10	1	0.25	7360	8794	100.0	20	1	0.25	29120	34324	99.5
10	2	0.25	7360	9094	100.0	20	2	0.25	29120	35524	100.0
15	0	0	16440	18559	100.0	25	0	0	45400	50889	100.0
15	1	0.125	16440	19009	100.0	25	1	0.125	45400	52139	97.5
15	2	0.125	16440	19347	100.0	25	2	0.125	45400	53077	96.5
15	1	0.25	16440	19459	100.0	25	1	0.25	45400	53389	94.5
15	2	0.25	16440	20134	99.5	25	2	0.25	45400	55264	96.0

period is given[6]. These problems constitute the sample for the mean computation times that are depicted in Figure 2.

As can be seen from Table 2, the algorithm is able to solve practically all problems involving 20 SSIs or less within the specified timeout period. For the largest problem classes involving 25 SSIs, the share of solved problems starts to notably drop, however. In accordance with this finding, Figure 2 shows that the mean computation times of the algorithm quickly grow with increasing problem size, already reaching a magnitude order of seconds for those problems involving 15 SSIs or more. In accordance with the number of decision variables and constraints in the mathematical model (cf. Table 2), the growth of computation times is not linear, but rather follows the squared number of regarded SSIs.

Notably, at least for those problem classes involving 20 SSIs or more, the introduction of SSI dependencies leads to a decline in mean computation times. The effect, which is also statistically significant at a confidence level of 95 % ($\alpha = 0.05$) for the largest problem classes with 25 SSIs, becomes more evident with an increasing ratio of dependent SSIs and growing number of QoS attributes. This finding is interesting, since the determination of pairs of unsuitable VM types (cf. Equations 10 and 11 in Model 1) requires substantial computational efforts. However, it can be reasoned that the process also leads to a substantial reduction of the problem space by restricting the valid assignments, and thus, results in an overall reduction of computation times.

In summary, we conclude that the proposed IP-based algorithm is well applicable for DSSDPs that involve about 20 SSIs or less. Thus, the algorithm

[6] In fact, the figure refers to problems that were *optimally* solved within the timeout period. In some cases, a valid, yet non-optimal solution may be obtained upon the occurrence of a timeout. The non-optimality arises from the fact that the solution space could not be completely examined.

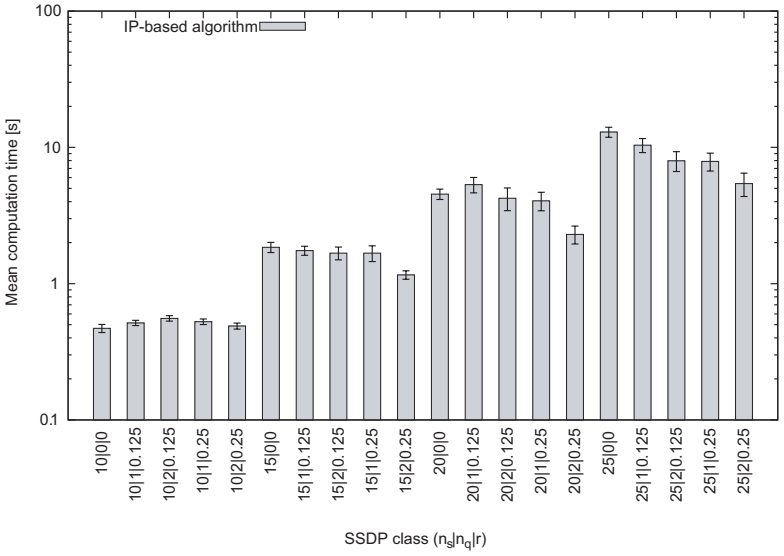

Fig. 2. Mean computation times of the optimization algorithm per evaluated DSSDP class, excluding timed-out problems (with 95% confidence intervals; please note the logarithmic scaling).

provides a viable approach to facilitate cost-efficient SSI distribution at small scale. However, for a scenario such as cloud gaming, which may involve hundreds of SSIs and requires the swift computation of distribution strategies, the algorithm appears unsuitable due to its high computational requirements. This indicates the need for the development of heuristic optimization approaches, which trade potentially large reductions in computation time against preferably small reductions in solution quality, i. e., increases in the cost of the computed distribution strategies. The development of such heuristics is complex due to the introduction of software service instance dependencies, because these dependencies render common solution approaches for assignment problems inapplicable.

4 Related Work

The challenge of distributing software service instances across virtual machines has previously been addressed in our own research [9]. In this past work, we introduced a basic version of the problem – i. e., SSI distribution under intra-machine resource constraints – and provided initial findings on its computational complexity. We substantially extend this research through the consideration of SSI dependencies and different resource pricing schemes in this paper.

Ardagna et al. [1] provide an approach for the distribution of SLA-bound Web services across various IaaS providers and the subsequent admission of

user requests, which is modeled using queuing theory. In contrast to our work, the authors do not distinguish different resource types and do not consider a combination of multiple software services on individual VM instances.

Breitgand and Epstein [3] consider the optimal placement of VMs on physical machines in a data center, with sets of VMs executing a specific software service. Their work is focused on the role of a cloud provider, rather than a SaaS provider. Also, the authors neither consider predefined VM types nor specific resource types.

Breitgand et al. [4] extend the previously described work, based on an identical scenario. Their approach, which is also based on linear programming, regards specific resource demands, as well as resource supplies of different physical machines in a federated cloud. However, the authors assume SLA-bound, temporally interruptible software services and do not regard SSI dependencies. In addition, Breitgand et al. do not provide any evaluation results for their proposed approach.

Kwok and Mohindra [8] present an approach for the optimal placement of multi-tenant SaaS applications in a data center under consideration of different resource types and QoS constraints. However, the authors do not specifically regard different VM types at varying price levels; accordingly, their objective consists of optimal resource exploitation on physical machines, rather than cost minimization.

Wu et al. [11], in accordance with our work, address the distribution of SaaS-style software services to different IaaS providers under SLA restrictions. However, the authors only consider a given set of quantifiable resource demands, such as processor cycles, rather than arbitrary resource types, and assume job-oriented, rather than continuously executed software services. In addition, they only implement and evaluate heuristic solutions, rather than an optimal approach.

In summary, as important distinctive feature, our approach considers continuously executed SaaS instances, rather than interruptible job- or task-style requests for software services. In addition, we address arbitrary resource types and VM types, as well as software service dependencies, which allows a more flexible and detailed representation of actual cloud systems. We further formulate a mathematical model that provides an optimal – rather than non-optimal, heuristic – solution to the DSSDP and thus may serve as a performance benchmark for other approaches.

5 Summary and Outlook

In the work at hand, we have addressed the *Dependent Software Service Distribution Problem* as a research challenge in the context of a cloud computing market. The problem concerns the distribution of partially dependent software service instances across leased cloud infrastructure in the form of virtual machines at minimal cost, with these distribution being subject to resource and quality of service constraints.

As solution approach, we have proposed the Software Service Distribution Broker, which facilitates cost-efficient software service deployment through the

computation of distribution strategies. We have formulated a mathematical model of the software service distribution problem, based on integer programming, and implemented a corresponding optimization algorithm. This algorithm has been evaluated based on actual software service execution and virtual machine data.

The evaluation shows that the proposed optimization algorithm features reasonable computation times for problems involving 20 software service instances or less and thus facilitates cost-efficient software service distribution at small scale. However, due to the rapid, non-linear growth in its computational demand, the algorithm is unsuitable for scenarios involving a higher number of software service instances, such as cloud gaming. Thus, our primary goal currently consists in the development of heuristic approaches, which involve reduced computational demand, but also achieve a favorable solution quality in terms of the resulting infrastructure leasing cost.

Acknowledgments

This work has partly been sponsored by the E-Finance Lab e. V., Frankfurt am Main, Germany (`http://www.efinancelab.de`).

References

1. Ardagna, D., Ghezzi, C., Panicucci, B., Trubian, M.: Service Provisioning on the Cloud: Distributed Algorithms for Joint Capacity Allocation and Admission Control. In: Di Nitto, E., Yahyapour, R. (eds.) Towards a Service-Based Internet, LNCS, vol. 6481, pp. 1–12 (2010)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A View of Cloud Computing. Communications of the ACM 53(4), 50–58 (2010)
3. Breitgand, D., Epstein, D.: SLA-aware Placement of Multi-Virtual Machine Elastic Services in Compute Clouds (TR H-0287). Tech. rep., IBM Research (2010)
4. Breitgand, D., Maraschini, A., Tordsson, J.: Policy-Driven Service Placement Optimization in Federated Clouds (TR H-0299). Tech. rep., IBM Research (2011)
5. Briscoe, G., Marinos, A.: Digital Ecosystems in the Clouds: Towards Community Cloud Computing. In: 3rd Int. Conf. on Digital Ecosystems and Technologies. pp. 103–108 (2009)
6. Buyya, R., Yeo, C., Venugopal, S., Broberg, J., Brandic, I.: Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. Future Generation Computer Systems 25(6), 599–616 (2009)
7. Hillier, F., Lieberman, G.: Introduction to Operations Research. McGraw-Hill, 8th edn. (2005)
8. Kwok, T., Mohindra, A.: Resource Calculations with Constraints and Placement of Tenants and Instances for Multi-Tenant SaaS Applications. In: 6th Int. Conf. on Service-Oriented Computing. pp. 633–648 (2008)
9. Lampe, U.: Optimizing the Distribution of Software Services in Infrastructure Clouds. In: 7th IEEE World Congress on Services, Ph.D. Symp. pp. 69–72 (2011)
10. Walker, E.: The Real Cost of a CPU Hour. Computer 42(4), 35–41 (2009)
11. Wu, L., Garg, S.K., Buyya, R.: SLA-based Admission Control for a Software-as-a-Service Provider in Cloud Computing Environments (CLOUDS-TR-2010-7). Tech. rep., The University of Melbourne (2010)