Enabling Cost-Efficient Software Service Distribution in Infrastructure Clouds at Run Time

Ulrich Lampe, Thorsten Mayer, Johannes Hiemer, Dieter Schuller, Ralf Steinmetz
Multimedia Communications Lab (KOM) – Technische Universität Darmstadt
Corresponding Author: Ulrich.Lampe@KOM.tu-darmstadt.de

***Abstract*—Because Infrastructure as a Service in the form of virtual machines only provides a limited supply of resources per discrete instance, approaches are required to compute cost-efficient distribution strategies for Software as a Service instances. Due to its computational complexity, this *Software Service Distribution Problem* is difficult to address at run time where time constraints apply. In the work at hand, we propose a Knapsack-based heuristic approach; an extensive evaluation, based on data from actual cloud systems and software services, shows that the heuristic is able to solve the problem with more than 99.7 % reduction in computation time and only marginal degradation in solution quality compared to a locally optimal approach. Thus, through the minimization of infrastructure leasing costs, our proposed approach enables Software as a Service providers to achieve higher profit margins or more competitive pricing of their services in the market.**

I. Introduction

A. Scenario and Motivation

In recent years, cloud computing has emerged as a novel paradigm in Information Technology (IT). It promises to deliver IT capacities in a utility-like fashion, i. e., via public networks and on-demand, comparable to traditional utilities such as electricity or water [1]. A popular service model in cloud computing is Infrastructure as a Service (IaaS), specifically the provision of Virtual Machines (VMs). A drawback in this model is that VMs constitute discrete compute units that have to be explicitly instantiated and only scale up to the level of the physical machine that hosts them. However, VMs are also highly customizable and may host or execute almost any existing software without prior adaptation [2].

Based on these notions and a three-layered cloud model by Armbrust et al. [3], we assume the following scenario: Existing software is offered in the form of Software as a Service (SaaS) by a *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*. A key characteristic of this scenario is that each requested Software Service Instance (SSI) exhibits a specific resource demand throughout its execution, e. g., in terms of processor power or memory consumption. Likewise, IaaS providers commonly offer different VM types that supply a certain, static quantity of these resources depending on the respective leasing price. We further assume that the SaaS users may request and terminate SSIs at arbitrary points in time 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 exhibits a specific and continuous resource demand on the VM instance that hosts it, while this VM instance is subject to a restricted resource supply.

Thus, the SaaS provider faces the problem of distributing the requested SSIs across leased VM instances. While the provider may pursue various objectives in this process, the most obvious and rational one lies in the minimization of infrastructure leasing costs, because this objective permits a maximization of profit margins or more competitive pricing of services in the market. Accordingly, we also assume cost minimization as objective. The resulting challenge, which constitutes a combined capacitation (number of leased instances of each VM type) and assignment (placement of each SSI on a specific VM instance) problem, is referred to as *Software Service Distribution Problem* (SSDP) and addressed in the work at hand.

B. Problem Statement

As previously outlined, the SSDP concerns the SaaS provider and consists in the distribution of a set of requested SSIs, which have to be continuously executed for an initially unknown duration of time, 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 (i. e., the distribution is *effective*)

2) The overall cost of leasing the required VM instances is minimized (i. e., the distribution is cost-*efficient*).

According to our scenario, SSIs may be requested or terminated at arbitrary points in time. Thus, the SSDP does not constitute a *design time*, but a *run time* problem stretching over multiple subsequent periods. Accordingly, above conditions should also be met across all periods.

In our previous research [4], we have outlined a simple one-period, design time approach to the SSDP and provided a first analysis of the SSDP's computational complexity. In the work at hand, we substantially extend this research through a multi-period, run time model and two corresponding optimization approaches.

The remainder of this paper is structured as follows: In Section II, we present our solution approach, an integrated broker that addresses the SSDP through optimization approaches. Two of these approaches are subsequently introduced. A first prototypical implementation of the broker is evaluated in Section III. Section IV provides an overview of related work. Eventually, Section V concludes the paper with a summary and outlook.

II. Solution Approach: Software Service Distribution Broker

A. Architectural Overview of the Broker

Our research aims at the design and implementation of an integrated *Software Service Distribution Broker*. The broker will be accessible for SaaS providers and IaaS providers through a standardized Web interface.

Through the interface, the broker 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 (so-called *push* model). The tool will also allow the mining of publicly available VM offer descriptions, e. g., through the Amazon EC2 API[1] (so-called *pull* model). Depending on the update frequency, the broker may also collect just-in-time pricing information, for instance from auction systems. Likewise, SaaS providers may submit the SSIs that have been currently requested by their end users to the broker, most notably specifying those SSIs' resource demands.

Based on the submitted information, the Software Service Distribution Broker repeatedly computes an (updated) *SSI distribution strategy* for the SaaS provider using suitable optimization approaches. The SSI distribution strategies specify the number of VM instances of each VM type to lease and the assignment of individual SSIs to these VM instances.

Thus, in accordance with the vision of a future cloud market [1], the broker facilitates the cost-efficient distribution of SSIs through SaaS providers. As previously outlined, SaaS providers profit from this mechanism through lower leasing costs, which not only yields higher profit margins, but may also result in more competitive – i. e., less costly – offers to the SaaS users.

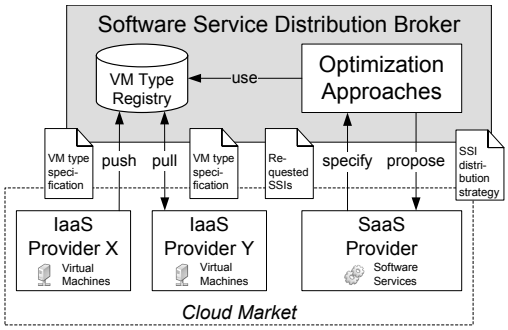

Figure 1: Architectural overview of the Software Service Distribution Broker.

An architectural overview of the Software Service Distribution Broker is depicted in Figure 1. To date, we have created a prototypical implementation of the broker. It features two different optimization approaches as key components. These approaches will be presented in detail in the following.

B. Binary Integer Programming-based Optimization

In accordance with our previous research, we initially provide a mathematical optimization model of the SSDP. Because this model is based on the principles of linear programming, more specifically binary integer programming, it can immediately be transferred into a corresponding optimization algorithm.

As a substantial extension to our previous work [4], we present a multi-period approach. It not only supports the computation of an initial distribution strategy at design time, but yields updated strategies during run time, based on incremental changes with respect to the requested SSIs and information about the already leased infrastructure.

The model is based on two assumptions: First, SSIs may be arbitrarily combined on one VM instance, as long as the aggregated resource demands are satisfied, but may not be split between different VM instances. Second, for each requested SSI, there exists at least one VM type that provides a sufficient resource supply for the SSI's execution; if this condition is not met, the specific SSI cannot be executed on the offered infrastructure.

We further establish that currently executed SSIs may not be moved between different VM instances. While such *live migration* is common in data centers to optimize the load of physical machines, it is usually conducted on the level of VMs, where it is more efficient and less error-prone than for individual SSIs [5]. The process thus lies

[1] http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/

in the control sphere of the IaaS provider, not of the SaaS Provider.

Based on these assumptions, we introduce the following formal notation: Let $\delta \in \mathbb{R}^+$ denote a regular time interval at which optimization steps are initiated, i. e., where a new distribution strategy is computed. Let $t \in \mathbb{R}^+$ ($t' \in \mathbb{R}^+$) further denote the point in time where the current (most recent) optimization period is (was) initiated, i. e., $t' = t - \delta$. In the case that the optimization is initiated for the first time at t, we assume $t' = 0$.

Let V be the persistent set of $|V| > 0$ available VM types, and let R be the set of $|R| > 0$ regarded resource types. Let $M_v \in \mathbb{N}$ be the maximum number of concurrently leaseable instances of each VM type $v \in V$, as specified by the respective IaaS provider.

I_v^t denotes a set of unique IDs for VM type $v \in V$ at time t. Each of these IDs $i \in I_v^t$ represents a leaseable *VM instance*, i. e., a concrete individual instantiation of an abstract VM type. Through this construct, the SSDP – which actually constitutes a combined capacitation and assignment problem – may be modeled in the simpler form of a pure assignment problem; more details will be provided in the course of this section.

Let S_t be a set of $|S^t|$ requested SSIs at time t. For each SSI $s \in S^t$, the demand for each resource type $r \in R$ is formally defined as $RD_{sr} \in \mathbb{R}^+$.

Likewise, $RS_{vr} \in \mathbb{R}^+$ denotes the supply of each resource type $r \in R$ for each VM type $v \in V$. Furthermore, for each VM type $v \in V$, the cost per leased instance $C_v \in \mathbb{R}^+$ is specified, based on a common billing period of length $P \in \mathbb{R}^+$. In the case that a VM type exhibits a different billing period, the cost per instance is interpolated accordingly.

Lastly, if a VM instance $v \in V$ with the index $i \in I_v^t$ is currently leased, $TF_{vi} \in \mathbb{R}^+$ denotes the point in time where this VM instance has been initiated, i. e., where its first billing period has started. If a VM instance is not currently leased, a value of 0 is assumed.

The complete optimization model, based on above formalisms, is split into three parts, which are provided in Equation Sets 1, 2 and 3. The individual parts are successively employed in the optimization process. Prior to providing additional details, we briefly explain the meaning of the contained decision variables: Equation 12 defines x_{svi}^t as binary decision variables, which indicate whether the SSI $s \in S^t$ has been assigned to a VM of type $v \in V$ with the instance index $i \in I_v^t$ at time t or not. Equation 13 further specifies the binary decision variables y_{vi}^t, which determine whether a certain instance $i \in I_v^t$ of VM type $v \in V$ is utilized at time t or not.

In the *pre-optimization computations*, we first initialize all relevant sets from the previous optimization period with default values in the case that no prior optimization has been conducted (Equation 1). Subsequently, we determine those SSIs that were sustained from the previous optimization period, i. e., the SSIs that are still being requested, SS^t, as well as the set of newly requested SSIs, SN^t (Equation 2).

Equations 3 and 4 determine the set of valid IDs I_v^t in the current optimization period, i. e., at the point in time t, for each VM type $v \in V$. In this context, $\mathsf{IdGen}(n)$ denotes a function that generates $n \in \mathbb{N}$ new unique IDs. The number of IDs is based on the notion that, in the worst case, each newly requested SSI may require a VM instance of the same type for execution, while, in addition, the maximum number of concurrent instances may not be exceeded. As previously outlined, each ID represents one concrete leaseable instance of an abstract VM type. The decision whether the VM instance is actually utilized and thus leased is taken in the optimization process. Through this introduction of a maximum number of concrete VM instances, an implicit capacitation decision is taken for each VM type, and thus, the SSDP may be modeled as a pure assignment problem.

Finally, in Equation 5, we determine *marginal costs* MC_{vi} for each VM instance of type $v \in V$ with ID $i \in I_v^t$. The marginal costs correspond to the normal per-instance cost of the VM type if a VM instance has not been previously available. They also assume this value if the VM instance is currently not leased or if the current billing period is bound to end before the next optimization period. If none of these conditions are met, i. e., if the VM instance is currently leased and not bound to be billed before the next optimization period, the marginal cost becomes 0. All computations are based on the start of the most recent billing period, TL_{vi}, which is determined in Equation 6.

For the *actual optimization*, Equation 7 first defines the objective, which consists in the minimization of the Total Marginal Cost (TMC) across all utilized VM instances. Equation 8 links the decision variables, x^t and y^t, by setting the value of the latter (indicating utilization and thus lease of a VM instance) depending on the value of the former (indicating assignments of SSI to VM instances).

The optimization is subject to the following constraints: Equation 9 ensures that each SSI is assigned precisely once to a VM instance. Equation 10 guarantees that the resource constraints are held on each VM instance. Lastly, Equation 11 imposes the constraint that each sustained SSI is assigned to the same VM instance as in the previous optimization, thus preventing the live migration of SSIs.

The *post-optimization computations* solely comprise Equation 14. The equation provides an implicit VM instance management by updating the timestamps TF_{vi}, which indicate the most recent initiation for an VM

Equation Set 1 Pre-Optimization Computations

$$\left.\begin{array}{c} S^{t'} = \emptyset \\ I_v^{t'} = \emptyset \quad \forall v \in V \end{array}\right\} \quad \text{if } t' = 0 \tag{1}$$

$$\begin{aligned} SS^t &= S^t \cap S^{t'} \\ SN^t &= S^t \setminus S^{t'} \end{aligned} \tag{2}$$

$$I_v^t = \begin{cases} \mathsf{IdGen}(\min(M_v, |S^t|)) & \text{if } t' = 0 \\ \hat{I}_v^t \cup \mathsf{IdGen}(\min(M_v - |\hat{I}_v^{t'}|, |SN^t|)) & \text{else} \end{cases} \quad \forall v \in V \tag{3}$$

$$\hat{I}_v^{t'} = \{i \in I_v^{t'} | TF_{vi} > 0\} \tag{4}$$

$$MC_{vi} = \begin{cases} C_v & \text{if } t' = 0 \vee i \notin I_v^{t'} \\ C_v & \text{if } TF_{vi} = 0 \\ C_v & \text{if } TL_{vi} = t \\ C_v & \text{if } TL_{vi} + P < t + \delta \\ 0 & \text{else} \end{cases} \quad \forall v \in V, i \in I_v^t \tag{5}$$

$$TL_{vi} = TF_{vi} + \left\lfloor \frac{t - TF_{vi}}{P} \right\rfloor * P \quad \forall v \in V, i \in I_v^t \tag{6}$$

Equation Set 2 Actual Optimization Computations

$$\text{Minimize } TMC(x^t, y^t) = \sum_{v \in V, i \in I_v^t} y_{vi}^t * MC_{vi} \tag{7}$$

$$y_{vi}^t \geq x_{svi}^t \quad \forall s \in S^t, v \in V, i \in I_v^t \tag{8}$$

$$\sum_{s \in S^t, v \in V, i \in I_v} x_{svi}^t = 1 \quad \forall s \in S^t \tag{9}$$

$$\sum_{s \in S^t} RD_{sr} * x_{svi}^t \leq RS_{vr} \quad \forall v \in V, i \in I_v^t, r \in R \tag{10}$$

$$x_{svi}^t = x_{svi}^{t'} \quad \forall s \in SS^t, v \in V, i \in I_v^t \tag{11}$$

$$x_{svi}^t \in \{0, 1\} \quad \forall s \in S^t, v \in V, i \in I_v^t \tag{12}$$

$$y_{vi}^t \in \{0, 1\} \quad \forall v \in V, i \in I_v^t \tag{13}$$

instance of type $v \in V$ with ID $i \in I_v$. The timestamp is set to the current optimization period t if a VM instance has not been previously available or utilized, but is used in the current period. It is reset to 0 if the VM instance is bound to be billed before the next optimization step, but not currently used; thus, the leasing is terminated. In any other case, the current timestamp is not modified.

Equation Set 3 Post-Optimization Computations

$$TF_{vi} = \begin{cases} t & \text{if } y_{vi}^t = 1 \wedge i \notin I_{vi}^{t'} \\ t & \text{if } y_{vi}^t = 1 \wedge TF_{vi} = 0 \\ 0 & \text{if } y_{vi}^t = 0 \wedge TL_{vi} + P < t + \delta \end{cases} \quad \forall v \in V, i \in I_v^t \tag{14}$$

The model in Equation Sets 1 through 3 constitutes a *Binary Integer Program* (BIP). Such BIP exclusively contains binary decision variables and thus, poses a special form of an Integer Program (IP). IPs can be solved using well-known methods from the field of operations research, e. g., *branch and bound* [6]. However, these methods may become very costly in terms of computational power with increasing problem size. Corresponding findings for single-period optimization have been presented in our past research [4].

It should be noted that the BIP-based algorithm always provides an optimal (i. e., cost-minimal) distribution strategy for each *individual* optimization period. However, because it operates in a myopic (short-sighted) manner, the algorithm does not necessarily compute an optimal distribution strategy across multiple periods; while the choice of a specific VM instance may be cost-optimal in an earlier period, it may unavoidably lead to higher costs in the future. In fact, an optimal distribution strategy across multiple periods can only be computed *ex post*, i. e., for a past period of time for which all requested SSIs with their respective durations of execution are known. Thus, except for single-period problems, the BIP-based algorithm should not be seen as absolute, but only as relative benchmark in terms of solution quality.

C. Knapsack-Based Optimization

Due to the potentially high computational complexity of the BIP-based algorithm, we have developed a heuristic that is inspired by the well-known Knapsack problem. In this problem, a choice has to be made among multiple items with differing weight and utility, such that the utility of the selected items is maximized while a certain weight constraint is held. A common heuristic approach to the Knapsack problem is to determine the relative utility of each item, which corresponds to the item's (absolute) utility divided by its respective weight, and consecutively select the items in the order of descending

utility [7]. Our heuristic is based on the same principle of relative utility, but split into two phases, *VM packing* and *VM selection*. The approach is based on similar pre- and post-optimization computations as the BIP-based algorithm in the previous section, i. e., it utilizes the same pricing scheme based on marginal costs and the identical VM instance management via timestamps.

In the first phase, *VM packing*, we determine the absolute utility AU_v of each VM type $v \in V$. Absolute utility is defined as the maximum number of SSIs that fit onto the respective VM type. To support the optimization across multiple periods, already leased VM instances are introduced as *pseudo VM types*. Pseudo VM types may only be instantiated once, and their resource supply and marginal cost correspond to the respective values of the underlying VM instance; thus, in contrast to the BIP-based approach, marginal costs are specified per (pseudo) VM type, rather than per VM instance.

For each VM type, we scan the set of requested SSIs in linear order. If the currently regarded SSI fits onto the current VM type in terms of resource supply, the VM type's maximum SSI capacity is incremented. In addition, the VM type's residual resource supply is reduced by the resource demand of the SSI. In the process, sustained SSIs are exclusively assigned to the pseudo VM type that hosted them in the previous optimization period, thus preventing a live migration. Once the absolute utility of each VM type has been determined, we compute the VM type's relative utility by dividing the determined SSI capacity through the current marginal cost, i. e., $RU_v = \frac{AU_v}{MC_v} \forall v \in V$. If the marginal cost is 0, we assume a very low price of $\epsilon \in \mathbb{R}^+$ instead.

In the second phase, *VM selection*, we determine the VM type with the highest relative utility. If the maximum number has not been reached yet, a new instance of this VM type is created. The SSIs that have been associated with the respective VM type in the VM packing phase are assigned to the new instance, and removed from the set of unassigned SSIs. In the case that multiple VM types with the same relative utility exist, we pick the VM type with the highest absolute utility. If this condition does not break the tie, the first VM type according to the order of the set is selected. Both phases are repeated until all requested SSIs have been successfully assigned to a VM instance. The Knapsack-based heuristic is schematically depicted in Figure 2.

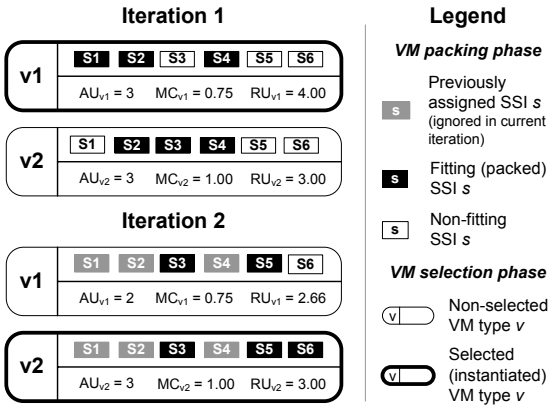

Figure 2: Scheme of the Knapsack-based heuristic.

III. Evaluation

Both the BIP-based and Knapsack-based optimization approaches have been implemented as part of a prototypical broker, which constitutes the basis of our evaluation. The implementation has been conducted in Java and utilizes the *Java ILP*[2] framework. This framework permits a flexible choice between the commercial *ILOG CPLEX*[3] and free *lpsolve*[4] IP solver frameworks, with the first constituting the default choice in our work.

A. Evaluation Methodology

The aim of our evaluation lies in a quantitative assessment of the performance of the two optimization approaches; it thus complements the brief qualitative discussion in Section II-B. We focus on two metrics that are of practical relevance in the context of the proposed broker: First, the metric *computation time* allows to judge the scalability of the proposed approaches, as well as the absolute waiting time that is introduced through the computation of distribution strategies by the broker. Second, the metric *total (marginal) cost* represents the solution quality of the computed distribution strategies with respect to the objective of cost efficiency. It is important to stress that we focus on the *relative* performance of the two approaches, rather than the absolute performance, which would require the computation of an optimal distribution strategy ex post as benchmark.

For the evaluation, we have created a collection of SSDPs with a varying number of requested SSIs (n_s), VM types (n_v), resource types (n_r), and optimization periods (n_p). For the VM and resource types, we used the specifications of the Windows-based, on-demand Amazon EC2 VM offers in the European Union[5]. The specifications yield eight VM types that provide three resource types (CPU, RAM, and HDD space) at differing costs, based on a one-hour billing period. To obtain realistic SSI execution data, we measured the absolute resource demands – in terms of the identical resource types – for five contemporary video games.

[2] http://javailp.sourceforge.net/
[3] http://www.ibm.com/software/integration/optimization/cplex-optimizer/
[4] http://sourceforge.net/projects/lpsolve/
[5] http://aws.amazon.com/ec2/#instance

Based on this information, we created different classes of SSDPs with fixed values for n_s, n_v, n_r, and n_p. Each class contained 200 individual problems (in the following, the terms SSDP and problem are used synonymously). The regarded SSIs and resource types were randomly drawn, based on the previously outlined specifications. n_p and δ were chosen to resemble an initial optimization ($n_p = 1$), as well as multiple subsequent optimization periods stretching over 1 hour ($n_p = 30, \delta = 120$) and 2 hours ($n_p = 60, \delta = 120$). In each incremental version of a problem, a random number of SSIs was terminated and newly initiated compared to the previous problem. The specific numbers are drawn from the range $[0; \frac{n_s}{4}]$, thus resembling the varying load of a SaaS provider. In order to achieve a comparable size across all evaluated problems, the overall number of SSIs was only allowed to fluctuate by 50 % compared to the initial problem.

All generated SSDPs were attempted to solve using the two optimization approaches, BIP-based and Knapsack-based. For both approaches, we measured the computation time and the total infrastructure leasing cost of the obtained solutions. In the process, we imposed a timeout period of 1 minute per problem and optimization approach, representing the maximum permissible waiting time for the computation of a distribution strategy. The evaluation was conducted on a desktop computer, equipped with an Intel Core 2 Duo E7500 processor and 4 GB of memory.

B. Evaluation Results and Discussion

Table I provides an overview of the evaluated SSDP classes. It also depicts the number of problems that could be solved by each optimization approach within the predefined timeout period. Only the data from those problems that could be solved by both approaches constitutes the sample for the detailed results that are presented in Figures 3 through 5.

Table I: Overview of evaluated SSDP classes and solved number of problems per class.

Class / problem size				Solved problems (out of 200)		
n_p	n_s	n_v	n_r	BIP	Knaps.	Both
1	10	4	2	200	200	200
1	20	4	2	194	200	194
1	20	8	3	197	200	197
1	30	8	3	137	200	137
30	10	4	2	200	200	200
30	20	4	2	198	200	198
30	20	8	3	199	200	199
30	30	8	3	119	200	119
60	10	4	2	200	200	200
60	20	4	2	196	200	196
60	20	8	3	197	200	197
60	30	8	3	134	200	134

The numbers of solved problems per class in Table I already indicate a substantial performance difference between the BIP- and Knapsack-based approaches. With an increasing number of regarded SSIs, the BIP-based algorithm fails to solve a growing share of problems due to the occurrence of a timeout. In contrast, the Knapsack-based heuristic does not exhibit such behavior; in fact, the approach is able to solve all of the considered SSDPs in the evaluation.

This difference in performance is further highlighted in Figure 3, which depicts the ratios of computation times between both optimization approaches based on macro-average[6]. On average across all problem classes, the Knapsack-based heuristic requires less than 0.3 % of the computation time, which corresponds to a reduction of 99.7 % compared to the BIP-based approach. According to a Friedman test [8], the difference between the two optimization approaches with respect to computation times is statistically significant at the confidence level of 95 % (i. e., $\alpha = 0.05$).

The implications of this difference in relative performance with respect to the absolute computation times can be seen in Figure 4. While the BIP-based approach generally exhibits mean computation times per optimization period in the magnitude order of seconds, the Knapsack-based heuristic achieves values in the magnitude order of milliseconds. In this context, it is worthy to note that the mean computation times per optimization period shrink with an increasing number of total optimization periods. This effect can be explained by the relatively high computation times for the initial optimization periods, which exhibit the largest number of newly requested SSIs.

Figure 5 depicts the ratios of total costs of the computed distribution strategies between both optimization approaches, again based on macro-average. The results indicate the existence of a trade-off between the computation time and total cost. The effect, i. e., an increase in total cost versus a reduction in computation time, is strongest for the single-period problems. As previously outlined, the BIP-based algorithm computes an optimal solution for these problems and thus may serve as a benchmark for the Knapsack-based heuristic. For the single-period problems, the cost increase amounts to approximately 4.7 % on average. For the remaining multiple-period problems, this figure is slightly less accented and corresponds to about 3.5 %. Across all evaluated classes, the cost increase assumes an average value of circa 3.9 %. A Friedman test shows that the difference between both approaches with respect to the total costs is statistically significant at $\alpha = 0.05$.

[6]Macro-average means that we first determine the ratio of computation times for each individual problem in a class and subsequently compute the mean across all obtained ratios.

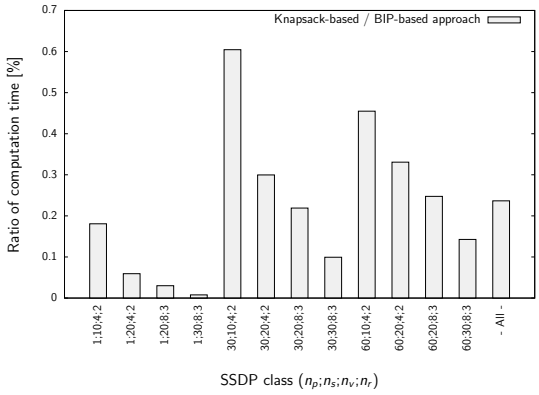

Figure 3: Ratios of computation times between both optimization approaches (based on macro-average).

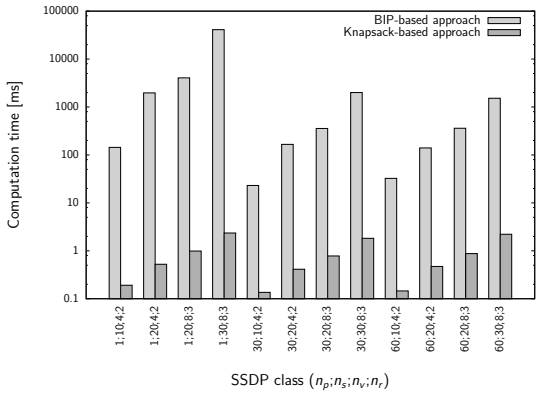

Figure 4: Mean absolute computation times per optimization period for both optimization approaches.

In summary, the results indicate that the BIP-based algorithm, because of its computational complexity, is unsuitable for real-life scenarios, which may involve dozens or even hundreds requested SSIs in parallel and require the computation of distribution strategies at run time, i. e., under narrow time constraints. However, the results show that the Knapsack-based heuristic represents a viable alternative, due to its ability to solve the SSDP with favorable solution quality at very low computational effort. Thus, in accordance with our scenario and to the benefit of SaaS providers, the Knapsack-based approach facilitates a cost-efficient distribution of software services in infrastructure clouds under realistic conditions.

IV. Related Work

Ardagna et al. [9] provide an approach for the distribution of Service Level Agreement (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

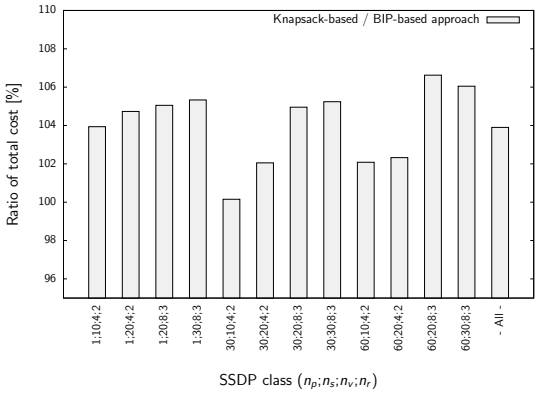

Figure 5: Ratios of total costs between both optimization approaches (based on macro-average).

consider a combination of multiple software services on individual VM instances.

Breitgand and Epstein [10] consider the profit-maximizing 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 (IaaS) provider, rather than a SaaS provider. The authors do not consider specific resource types and only provide a single-period optimization model.

Breitgand et al. [11] extend the previously described work, based on an identical scenario. Their approach regards specific resource demands, as well as resource supplies of different physical machines in a federated cloud. The authors present a multi-period IP model that permits different objectives, including profit minimization, and suggest a heuristic solution based on Linear Program (LP) relaxation. Their work is focused on the role of a IaaS provider, rather than SaaS provider again. In addition, the authors assume SLA-bound, temporally interruptible software services, and do not explicitly consider the management of VM instances depending on discrete billing periods. Lastly, they do not provide evaluation results for their proposed approach.

Kwok and Mohindra [12] present an approach for the optimal placement of multi-tenant SaaS applications in a data center under consideration of different resource types and Quality of Service 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. [13], 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 SaaS instances that are, starting at an instant in time, continuously executed for an initially unknown duration. In contrast, the related work usually assumes job- or task-style requests for software services, where the duration or execution time of these jobs is either known a priori or can be inferred for a specific VM instance based on known resource demands, e. g., in terms of required processor cycles. In addition, our approach considers arbitrary resource types and VM types with discrete billing periods, which permits a more flexible representation of actual cloud systems. We further formulate a mathematical model that permits an optimal solution to the SSDP for individual optimization periods, not only a heuristic approach.

V. Summary and Outlook

In this work, we have addressed the *Software Service Distribution Problem*; this research challenge concerns the cost-efficient distribution of software service instances across leased cloud infrastructure in the form of virtual machines under resource constraints at run time. We have introduced the concept of an integrated Software Service Distribution Broker and proposed a multi-period mathematical optimization model. Based on this model, we have developed two optimization approaches, which address the problem through the computation of distribution strategies. Both approaches have been implemented in a prototypical broker and evaluated based on actual virtual machine and software service execution data.

The evaluation shows that the first approach, which is based based on binary integer programming, exhibits high computational demands. This renders it non-applicable for real-life scenarios that involve a large number of software service instances. We have also found that the second approach, a Knapsack-based heuristic, achieves substantial reductions in computation times. These reductions amount to about 99.7 % on average, while the approach maintains a favorable solution quality in terms of the resulting infrastructure leasing cost, with cost increases amounting to approximately 3.9 % on average. Thus, our proposed heuristic approach provides the means to solve the Software Service Distribution Problem under realistic conditions at run time. Accordingly, it enables Software as a Service providers to minimize their infrastructure leasing costs and achieve higher profit margins or more competitive pricing of services in the market.

Our future work will aim at two objectives: First, the development of an ex post optimization approach, which can serve as absolute benchmark for the proposed run time approaches. Second, the inclusion of additional problem extensions, such as software service dependencies or location constraints [4].

Acknowledgments

This work has partly been sponsored by E-Finance Lab e. V., Frankfurt a.M., Germany (www.efinancelab.de).

References

[1] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] G. Briscoe and A. Marinos, "Digital Ecosystems in the Clouds: Towards Community Cloud Computing," in *3rd Int. Conf. on Digital Ecosystems and Technologies*, 2009, pp. 103–108.

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[4] U. Lampe, "Optimizing the Distribution of Software Services in Infrastructure Clouds," in *7th IEEE World Congress on Services, Ph.D. Symposium*, 2011, pp. 69–72.

[5] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *2nd Symp. on Networked Systems Design & Implementation*, 2005, pp. 273–286 (Vol. 2).

[6] F. Hillier and G. Lieberman, *Introduction to Operations Research*, 8th ed. McGraw-Hill, 2005.

[7] W. Domschke and A. Drexl, *Einführung in Operations Research*, 6th ed. Springer, 2004, in German.

[8] J. Marques de Sá, *Applied Statistics Using SPSS, STATISTICA, MATLAB and R.* Springer, 2007.

[9] D. Ardagna, C. Ghezzi, B. Panicucci, and M. Trubian, "Service Provisioning on the Cloud: Distributed Algorithms for Joint Capacity Allocation and Admission Control," in *Towards a Service-Based Internet*, ser. LNCS. Springer.

[10] D. Breitgand and D. Epstein, "SLA-aware Placement of Multi-Virtual Machine Elastic Services in Compute Clouds (TR H-0287)," IBM Research, Tech. Rep., 2010.

[11] D. Breitgand, A. Maraschini, and J. Tordsson, "Policy-Driven Service Placement Optimization in Federated Clouds (TR H-0299)," IBM Research, Tech. Rep., 2011.

[12] T. Kwok and A. Mohindra, "Resource Calculations with Constraints and Placement of Tenants and Instances for Multi-Tenant SaaS Applications," in *6th Int. Conf. on Service-Oriented Computing*, 2008, pp. 633–648.

[13] L. Wu, S. K. Garg, and R. Buyya, "SLA-based Admission Control for a Software-as-a-Service Provider in Cloud Computing Environments (CLOUDS-TR-2010-7)," U. of Melbourne, Tech. Rep., 2010.