ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/221293461

SLA-aware placement of multi-virtual machine elastic services in compute clouds

Conference Paper · May 2011

DOI: 10.1109/INM.2011.5990687 · Source: DBLP

CITATIONS
54

READS
156

2 authors:

David Breitgand
IBM Research -- Haifa
49 PUBLICATIONS **1,271** CITATIONS

SEE PROFILE

Amir Epstein
IBM
19 PUBLICATIONS **995** CITATIONS

SEE PROFILE

All content following this page was uploaded by David Breitgand on 22 May 2014.

The user has requested enhancement of the downloaded file.

SLA-aware Placement of Multi-Virtual Machine Elastic Services in Compute Clouds

David Breitgand Amir Epstein
Virtualization Technologies, System Technologies & Services
IBM Haifa Research Lab, Haifa, Israel
Email: {davidbr, amire}@il.ibm.com

***Abstract*—Elastic services comprise multiple virtualized resources that can be added and deleted on demand to match variability in the workload. A Service owner profiles the service to determine its most appropriate sizing under different workload conditions. This variable sizing is formalized through a service level agreement (SLA) between the service owner and the cloud provider. The Cloud provider obtains maximum benefit when it succeeds to fully allocate the resource set demanded by the elastic service subject to its SLA. Failure to do so may result in SLA breach and financial losses to the provider. We define a novel combinatorial optimization problem called elastic services placement problem (ESPP) to maximize the provider's benefit from SLA compliant placement. We observe that ESPP extends the generalized assignment problem (GAP), which is a well studied combinatorial optimization problem. However, ESPP turns out to be considerably harder to solve as it does not admit a constant factor approximation. We show that using a simple transformation, ESPP can be presented as a multi-unit combinatorial auction. We further present a column generation method to obtain near optimal solutions for ESPP for large data centers where exact solutions cannot be obtained in a reasonable amount of time using a direct integer programming formulation. We demonstrate the feasibility of our approach through an extensive simulation study. Our results show that we are capable of consistently obtaining good solutions in a time efficient manner. Moreover, if one is willing to trade precision to gain in computation time, our method allows to explicitly manage this tradeoff.**

I. Introduction

We consider a popular Infrastructure as a Service (IaaS) Cloud Computing paradigm where service providers rent VM instances on-demand from the IaaS provider on a "pay-as-you-go" basis to provide functionality (service) using these resources. In this model, the payment per VM instance comprises an initial fixed fee for ordering an instance and a variable usage based fee where usage is aggregated in each billing period.

To take advantage of the "pay-as-you-go" model, service providers strive to use just the needed capacity to satisfy the target end-user QoS at any given time. This is termed thin provisioning to differentiate it from the traditional over-provisioning methodology that plans capacity for peak workloads. Since workload applied to services varies with time, to match these variations with minimum capacity, VM instance sets comprising the services are *elastic*. The structure of an elastic service remains fixed, but the number of instances and/or size of the instances may vary.

The elastic behavior of the services is programmed using service-specific elasticity policies (also known as elasticity rules) that match workload variations with on demand capacity allotments [1], [2]. In response to executing an elasticity rule, the IaaS provider needs to solve the service placement optimization problem to accommodate new VMs and already deployed ones to maximize profit from service provisioning. In any given billing period, each elastic service contributes to the total revenue of the IaaS provider via the per VM usage based payments aggregated over this period. Currently most IaaS providers offer VM instances from a diversified catalog suggesting a number of discrete hardware configurations. For example, Amazon EC2 offers "small", "large", "extra large" and a few other VM configurations, where each configuration represents a different VM sizing and is charged using different instance hour rate[1]. Usually, VM types are provided under a single standard availability SLA. In RESERVOIR project [3] availability SLA is further diversified and extended as follows. At any given time, the set of VMs mandated by the active elasticity rules has to be placed in its entirety to comply with the availability SLA of the service. We refer to this as *set requirements*.

The main focus of our study is SLA compliant placement of multi-VM elastic services under set requirements and placement constraints. Our proposed approach is general and deals with a variety of placement restriction types. In this work, however, we focus on *anti-collocation* constraints that demand all VMs of the service to be placed on different physical hosts. While deployment of VMs under placement constraints has received a significant attention in the literature, to the best of our knowledge, this problem was not considered in conjunction with the set requirements.

In this paper we present Elastic Service Placement Problem (ESPP) that generalizes the model studied by Urgaonkar *et al.* in [4]. The input to ESPP includes set of hosts and set of services, where each service is composed of a set of VMs. Each VM has size and profit that may depend on its type, SLA

This work was partially funded by the European Commission's 7th Framework Programme ([FP7/2003-2013]) under grant agreement no. 215605.

[1]It should be noted that usually the usage fee paid by the customer does not depend on the actual resource utilization. Under a typical IaaS chargeback scheme, such as that of EC2 or Rackspace, a VM instance that is utilized up to, say, 80% would be charged the same as an instance of the same type utilized, say, up to 1%, as long as both were powered up during the the equal periods of time.

and the provider costs. In general, each VM may have different profit and capacity demands (size) when assigned to different hosts. The goal of this optimization problem is to maximize the profit obtained from the placed VMs, while respecting the set requirements, placement constraints and resource capacity constraints.

Solving ESPP may entail VM migrations which incur various operational costs. In ESPP migration costs are directly incorporated into the placement problem by subtracting them from the profit of a migrated VM. Modeling of migration costs is highly non-trivial due to second order effects migrations might have on the migrated service and other running services. To the best of our knowledge, there is no single model thus far that accounts for all such effects caused by migrations. It should be noted that exact modeling of the migration costs in monetary terms is outside of the scope of this paper. Rather than that we introduce synthetic migration costs used as a management lever to control the total number of migrations and potential profit loss caused by them.

A. Our Contribution and Paper Organization

To summarize, our contributions are as follows.

- We address the problem of maximizing the cloud provider profit from SLA compliant placement in virtualized data centers. We formulate a novel combinatorial optimization problem called elastic services placement problem (ESPP).
- We show that ESPP can be reformulated as a multi-unit combinatorial auction. We further solve this formulation of ESPP using a column generation method to obtain good solutions in a reasonable amount of time.
- Our simulations demonstrate that our approach yields near-optimal solutions for large resource pools efficiently. Moreover, our method allows to explicitly manage the tradeoff between solution quality and computation time.

The rest of this paper is organized as follows. Sections II presents the related work. In Section III we provide background and motivation for ESPP and in Section IV we define our problem. We present a combinatorial auction view of ESPP and give a multi-unit combinatorial auctions formulation for ESPP in section V. The column generation method is presented in section VI. Numerical results for ESPP are presented in Section VII, showing that our method finds near-optimal solutions for large resource pools efficiently. We conclude our paper in Section VIII.

II. Related Work

Autonomic and dynamic optimization of virtual machines placement in a data center backing the cloud service, received considerable attention recently [4]–[8]

Most of this research focuses on selected aspects of placement optimization considering either management of SLAs or migrations costs or energy management or placement restrictions. However, SLA management, migration-related operational costs and set requirements are rarely considered together.

The model studied by Urgaonkar *et al.* in [4] is closest to our work. The authors consider non-elastic sets of application components corresponding to mandatory configurations that should be successfully placed in full on a cluster of physical hosts in order for the application to function. The objective is to maximize the number of successfully placed applications assuming all applications are equally important and generating the same revenue for the provider irrespective of the QoS and application sizing.

This problem is a special case of ESPP. We generalize the model of [4] in the following important aspects.

- We consider variable values of VMs comprising the service sets due to diversified sizing and pricing and different SLA compliance histories.
- We consider variable profit from placing VMs on different hosts due to operational costs such as those incurred by migrations (we consider the use case where the placement optimization problem is solved repeatedly).
- We consider variable capacity requirements of the same VM when placed on different hosts possessing non-uniform processing capabilities.

ESPP extends the generalized assignment problem (GAP), which is its special case where every service contains exactly one item. GAP is a well studied optimization problem that has many practical applications. Chekuri and Khanna [9] observed that the work of Shmoys and Tardos [10] implies a 2-approximation algorithm for GAP. This problem is known to be APX-hard. The best known approximation ratio is $\mathrm{e}/(\mathrm{e}-1)+\epsilon$ for some constant $\epsilon > 0$ by Feige and Vondrák [11].

We observe that ESPP problem can be viewed as a multi-unit combinatorial auction problem. In a combinatorial auction a number of non-identical goods are sold concurrently and bidders express preferences about combinations of goods and not just about single good. In a multi-unit combinatorial auction – that extends this problem – there are multiple copies of each good and the agents are allowed to bid on more than one unit of each good. The combinatorial auction problem is computationally hard and cannot be approximated to within a factor better than $m^{1/2-\epsilon}$ for any constant $\epsilon > 0$. This holds even for the special case of single-minded-bidders [12], which is also a special case of ESPP. Recent work focused on classes of valuations that can be better approximated, e.g., subadditive [13] and submodular [14] valuation functions. For the case of multi-unit combinatorial auctions, where there are at least $B \geq 1$ units of each item and every agent demands at most 1 unit of each item, the best known approximation algorithms have approximation ratio of $O(m^{1/B})$ [15]. This result also follows from a LP-based randomized rounding algorithm for packing integer programs (PIPs)[2] [16], [17]. For the case where each bundle contains at most d goods, results for PIPs with columns-sparse constraint matrices [18] (i.e., no column in the matrix contains more than d nonzero entries) imply that LP-based randomized rounding yields an $O(d^{1/(B-1)})$ approximation.

These approximation guarantees are not sufficiently good for most practical applications. Therefore we explore scalable

[2]A PIP, cf. [16], is an integer linear program of a form $\max\{c \cdot x : Ax \leq b, x \in \{0,1\}^q\}$; where $p, q \in \mathbb{N}$, $c \in \mathbb{R}^q_{\geq 0}$. $A \in [0,1]^{pxq}$, $b \in [1,\infty)^p$.

near optimal solutions by expressing ESPP as a multi-unit combinatorial auction using a standard set packing formulation. Since this formulation may have huge number of variables, we apply a column generation approach, which is a powerful tool for solving large scale integer problems. Instead of enumerating columns explicitly, the column generation method uses only a selected set. This method was first applied by Gilmore and Gomory [19] for solving the cutting stock problem. Later column generation was successfully applied for several applications, such as vehicle routing, crew scheduling and resource allocation with spatial TDMA in ad hoc radio networks [20]–[22].

III. Background and Motivation

To put our discussion into a specific context, we abstract a generic management framework of an IaaS cloud as follows:

- Service Manager: receives service provisioning requests and presents Placement Optimizer with the VM sets that should be placed on the physical infrastructure at any point in time.
- Placement Optimizer: calculates an optimal placement to maximize the provider's profit from service placement (over all services).
- Placement Planner: calculates an optimal schedule of management actions required to move from the current placement to the new placement calculated by the Placement Optimizer.
- Placement Actuator: implements the placement schedule provided by the Placement Planner.

Current practices in cloud computing set customer expectations high, implying that service provisioning requests should be served very fast, usually on the scale of minutes. In this work we focus on the Placement Optimizer component developing efficient and scalable algorithms for a core optimization problem of maximizing profit while maximally satisfying availability SLAs and minimizing network overhead due to migrations.

Each service provisioning request contains the structure of the service including the number of VMs of each type, deployment constraints such as collocation and anti-collocation, elasticity rules that prescribe specific configurations under different environmental conditions, minimum and maximum number of instances of VMs of each type and availability Service Level Objectives (SLO) of the service. It is the responsibility of Service Manager to monitor performance of the service and change resource allotments in accordance with the elasticity rules. These rules may require scaling-up or scaling-down of the service by adding or removing VM instances subject to the maximal and minimum limits defined for the service.

Availability SLO specifies availability percentile that should be achieved for the VMs comprising the service *irrespectively of what elasticity rules are executed and when*, as long as resource demands are between the minimum and maximum limits contracted for the service. It should be stressed that committing to availability SLO of an elastic service is especially challenging because the intervals over which availability of VMs are calculated may not even overlap for different VMs comprising an elastic service. To cope with this challenge, we propose a different approach to the management of availability guarantees. Namely, at any given moment we consider a VM set comprising the service (as mandated by the elasticity policy effective at this moment in time) as an indivisible whole that should be mapped on physical machines subject to capacity constraints and deployment constraints. In other words, elastic service concept implies a placement model where all VMs of the current service configuration should be placed for the service to be considered available[3].

Non-placing a service does not imply an immediate SLO violation as long as availability percentile of the service is not violated in this billing period.

The value generated by VM sets for the provider depends on the number of VMs in the set and VM types of the instances comprising the set. The direct consequence of the availability SLO protection model described above is that VM sets that are structurally the same (in terms of VM types and number of instances) may have different values at different points in time depending on their availability SLO compliance history. Namely, there are three options that should be considered by the Placement Optimizer w.r.t. every VM set:

- VM set is included in the next placement: the provider obtains revenue from the set calculated as a sum of per-VM revenues on the instance-hour basis;
- VM set is excluded from the next placement, but this does not violate availability SLA of the service yet: provider loses revenue from the service;
- VM set is excluded from the next placement, violating availability SLA of the service: the provider loses revenue from the service and also compensates the customer with the service credit for the next billing period, which is a function of the service usage fee accumulated thus far in this billing period.

The tasks performed sequentially by Placement Optimizer, Placement Planner and Placement Actuator form a *management cycle*. At the beginning of each management cycle, Placement Optimizer receives input from the Service Manager in terms of VM of the newly arrived and already provisioned VM sets and their values at this point in time. The Placement Optimizer computes a placement to maximize profit from the service placement. As a part of this objective, Placement Optimizer minimizes the costs associated with availability SLO incompliance and VM migrations.

In case provisioning is performed on an initially empty host pool or on a partially loaded pool where VMs are not allowed to move from their current hosts due to workload sensitivity or other management concerns. There are no operational costs associated with migrations and the value of each VM is the same irrespective of the physical placement.

In many other practical scenarios, provisioning is being performed on the pre-loaded host pools where VMs can move within the pool or across different pools. Current virtualiza-

[3]An alternative approach to this "all-or-nothing" scale-up placement model is to define the levels of compliance as, e.g., as percentage of the VM superset that contains the initially requested configuration. The initially requested configuration still has to be placed in an all-or-nothing manner. We defer discussion of the partial placement model to future work.

tion technologies allow live migrating VMs across disparate physical hosts. The live migration of VMs may stress the data center network, which is a scarce shared resource, and incurs an overhead on both source and destination hosts in terms of CPU and memory. This overhead can be modeled as cost of migration between the source host of a VM and its feasible host destinations. While many factors affect the migration costs, the dominant factor is the cost of communication that depends on the available network bandwidth and network topology. While in this work we are not dealing with modeling the cost of VM migration, our problem formulation explicitly accounts for these costs to maintain policy-based control over them.

It is important to stress that the quality of the placement produced by the Placement Optimizer directly influences duration and operational costs of the subsequent phases implemented by the Placement Planner and Placement Actuator. Obviously, the more migrations are implied by the new placement, the longer will be the scheduling and provisioning phases and higher will be the network overhead.

IV. Elastic Service Placement Problem

We consider the following elastic service placement problem. We are given a set B of m bins and a set A of k services. Each bin i has capacity $c(i)$ and each service j contains n_j items, where each item l of service j has size p^j_{il} and value v^j_{il} to the provider when item l is assigned to bin i. The total value of each service equals the sum of the values of all its items if the whole service is packed and otherwise it equals zero.

The packing may be subject to deployment constraints. One especially challenging type of deployment constraints is given by the anti-collocation requirements where all items of a given service should be assigned to different bins[4]. The goal is to find a feasible assignment of the items' sets to bins to maximize the total value.

It is easy to see that ESPP generalizes the *Generalized Assignment Problem (GAP)*, which is defined as follows. Given a set of bins and a set of items, where the items may have different size and value for each bin. The goal is to find a feasible assignment of the items to bins to maximize the total value.

We now provide a direct integer programming formulation for ESPP. Let y_j be an indicator variable assuming 1 if service j is included into the placement and 0 otherwise. For each item l of service j and bin i, we have a decision variable x^j_{il}, which indicates whether item l is assigned to bin i.

[4]The motivation for this type of constraints arises from the typical management scenarios such maintaining a specific level of fault tolerance mandated by regulations and SLAs.

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{k}\sum_{l=1}^{n_j}\sum_{i=1}^{m} x^j_{il}\cdot v^j_{il} \\
s.t. \quad & \sum_{i=1}^{m} x^j_{il} = y_j && \forall j \in A, \forall l \in \{1,\ldots,n_j\} \\
& \sum_{j=1}^{k}\sum_{l=1}^{n_j} x^j_{il}\cdot p^j_{il} \le c(i) && \forall i \in \{1,\ldots,m\} \\
& \sum_{l=1}^{n_j} x^j_{il} \le 1 && \forall j \in A, \forall i \in B \\
& x^j_{il} \in \{0,1\} && \forall j \in A, \forall i \in B \\
& y_j \in \{0,1\} && \forall j \in A.
\end{aligned}
$$

In the integer program, the objective is maximizing the sum of values. The first set of constraints requires that for each service either the whole service is packed or none of its items is packed. The second set of constraints requires that the load of each bin does not exceed its capacity. The third set of constraints requires that the items of each service are assigned to distinct bins.

It is prohibitively expensive (in terms of running time and computation resources) to solve this IP formulation directly for large problem instances. Therefore, we turn to an alternative integer programming formulation that can be used to find near optimal solutions efficiently using column generation method. In the next section we show the relationship between ESPP and multi-unit combinatorial auctions (also see Section II) and provide an alternative integer program formulation of ESPP using set packing formulation for multi-unit combinatorial auctions. This formulation is naturally amenable to column generation.

V. A Combinatorial Auctions approach to ESPP

A. Combinatorial Auctions Problem Description

In combinatorial auctions a number of non-identical goods are sold concurrently and bidders express preferences about combinations of goods and not just about the singletons. This problem can be viewed as a high level abstraction of complex resource allocation, and is the paradigmatic problem on the interface of economics and computer science [23], [24].

Formally, in combinatorial auctions there is a set of m non-identical goods that are concurrently auctioned to n bidders; in a multi-unit combinatorial auctions each good $i \in \{1,\ldots,m\}$ is available in $c(i) \in \mathbb{N}$ units. The combinatorial character of the auction comes from the fact that bidders have preferences regarding bundles of goods. A bundle of goods is a vector $(d_1,\ldots,d_m)$, where $0 \le d_i \le c(i)$ is the number of units of good i in the bundle. Each bidder j has a valuation function v_j that describes its preferences in monetary terms and assigns non-negative value for each bundle of goods, $v_j : \{0,\ldots,c(1)\} \times \ldots \times \{0,\ldots,c(m)\} \rightarrow R^+$. The goal of the auction is to find an allocation that maximizes the social welfare $\sum_j v_j(S_j)$ where S_j is the bundle of goods allocated to bidder j. Let us now turn to our problem. ESPP can be viewed as a multi-unit combinatorial auction as follows. The

set of bins B is the set of non-identical goods and the capacity $c(i)$ of each bin i is the number of units of each good i. The set of services A is the set of bidders which bid on bundles of bin capacities. The goal is to maximize the sum of values of the bundles allocated to the bidders.

We define the load of a bin as the total size used by items (i.e., VMs) assigned to it. Each bundle S of service j is a load vector of the bins that corresponds to a feasible assignment of service j to the bins. We will abuse notation somewhat and also use S to refer to the corresponding assignment of service j to the bins. We denote by $\mathcal{S}_j$ the set of bundles of service j. Each service j has a profit function v_j, where $v_j(S)$ is the maximum profit of a feasible packing of service j with vector of loads that equals $S \in \mathcal{S}_j$. We denote by $p(S,i)$ the total size of items that are packed in bin i according to bundle S.

B. Integer Programming Formulation of Combinatorial Auctions for ESPP

We now model the ESPP problem using the integer programming formulation for multi unit combinatorial auctions. For a feasible assignment $S \in S_j$, let $x_{j,S}$ be an indicator variable that indicates whether assignment S is chosen for service j. We relax the $x_{j,S}$ variables to be in $[0,1]$ and obtain the following linear programming relaxation (LP2):

$$\max \quad \sum_{j \in A} \sum_{S \in \mathcal{S}_j} x_{j,S} \cdot v_j(S) \tag{1}$$

$$s.t. \quad \sum_{S \in \mathcal{S}_j} x_{j,S} \leq 1 \qquad \forall j \in A \tag{2}$$

$$\sum_{j \in A, S \in \mathcal{S}_j} x_{j,S} \cdot p(S,i) \leq c(i) \quad \forall i \in B \tag{3}$$

$$x_{j,S} \geq 0 \qquad \forall j \in A, S \in \mathcal{S}_j. \tag{4}$$

In the linear program, the objective function is to maximize the sum of profits. Constraint (2) ensures that each service gets at most one assignment and constraint (3) ensures that the load of a bin does not exceed its capacity.

The corresponding dual linear program (DLP) is then:

$$\min \quad \sum_{i \in B} c(i) \cdot y_i + \sum_{j \in A} z_j \tag{5}$$

$$s.t. \quad \sum_{i \in B} y_i p(S,i) + z_j \geq v_j(S) \quad \forall j \in A, \forall S \in \mathcal{S}_j \tag{6}$$

$$z_j \geq 0 \qquad \forall j \in A \tag{7}$$

$$y_i \geq 0 \qquad \forall i \in B. \tag{8}$$

The main reason for reformulating ESPP as a multi-unit combinatorial auction using set packing approach is that it naturally renders itself to column generation approach that can be used for solving the integer problem more efficiently.

VI. Column Generation

Column generation is a powerful technique for solving IP problems with huge number of variables. The idea of column generation is to solve the problem without considering explicitly all the variables. Column generation decomposes an LP relaxation of the problem into a master problem and subproblems. Initially a restricted LP formulation, called restricted master problem (RMP), that contains only a small subset of the variables of the full LP formulation is solved optimally. Then a subproblem called *pricing problem*, which is a separation problem for the dual linear program, is solved repeatedly to identify new variables that have positive reduced cost and can, therefore, potentially increase the objective value of the RMP. This process is referred to as the column generation phase. When no additional variables that can improve the objective value of the RMP can be identified, the column generation phase stops and the problem obtained in the column generation phase is solved as IP.

For (LP2) the pricing problem is to find a service j and assignment $S \in \mathcal{S}_j$ of positive reduced cost $v_j(S) - \sum_{i \in B} y_i p(S,i) - z_j$, where (y,z) is an optimal solution to the dual problem of the restricted master problem (RMP). For ESPP with anti-collocation constraints the pricing problem is the maximum weight bipartite matching problem that can be solved in polynomial time (see [25]). For ESPP without the anti-collocation constraints, the pricing problem is GAP. In this paper we focus on the variant of ESPP with the anti-collocation constraints.

For each $j \in A$, we define a bipartite graph $G_j = (B, I_j, E_j)$ in which we have an edge (i,l), if item l of service j can be assigned to bin i. The weight of edge (i,l) is $w^j_{il} = v^j_{il} - y_i p^j_{il}$. Thus, in order to find the highest reduced cost for every service j we have to solve the maximum weight bipartite matching that can be formulated as an integer program as follows.

$$\max \quad \sum_{(i,l) \in E_j} w^j_{il} x_{il}$$

$$s.t. \quad \sum_{i=1}^{m} x^j_{il} = 1 \qquad \forall l \in \{1, \ldots, n_j\}$$

$$\sum_{l=1}^{n_j} x^j_{il} \leq 1 \qquad \forall i \in \{1, \ldots, m\}$$

$$x^j_{il} \in \{0,1\} \qquad \forall i \in B, \forall l \in \{1, \ldots, n_j\}.$$

If the value of the optimal solution to the above maximum weight bipartite matching problem for service j is greater than z_j then we have found a variable of service j that can be added to the LP. In our implementation, in every iteration of column generation, we solve the pricing problem for every service and if it has a positive reduced cost we add the corresponding column to the restricted master problem (RMP).

A generalization of branch-and-bound with LP relaxations, called branch-and-price [26], is often used for solving large IP problems. This scheme allows applying column generation throughout the branch-and-bound tree. However, our experiments show that simple column generation was capable of producing ESPP solutions very close to those of LP relaxation of ESPP where LP reflect the upper limit on the ESPP objective function for any given problem instance. Consequently we leave a more complex branch-and-price method out of the scope of this work.

VII. Implementation and Simulation Results

The objective of our evaluation study is to compare the direct IP formulation to the proposed column generation method to get insights about the tradeoff between optimality and computational time under reasonable assumptions about the cloud settings.

One of the more challenging issues while evaluating algorithms on complex scenarios is generating representative input data sets. This data can come either from a real production environment or from a cloud simulator. In the cloud environment that we consider, the problem of the input data is exacerbated by having multiple independent dimensions characterizing our problem. Namely, one has to specify distribution of the physical hosts capacities, distribution of the resource demands by VMs, popularity distribution of VM sizes, distribution of the VM set sizes, birth-death stochastic process characterizing elasticity, distribution of value attributed to VM sets in accordance with specific SLA offerings and external workload. While recently some cloud simulation tools have appeared [27], these tools still need some input on the above mentioned aspects to drive the simulation. Given current production practices, we were not able to obtain sufficiently representative production data sets to extract all the needed data.

Therefore we resorted to implementing our own problem instance generator that abstracts the core features of the model important for the algorithmic evaluation. As one can easily verify, varying all aspects of the model described above quickly results in an unwieldy set of sub-instances that would be difficult to interpret and generalize. Thus, we decided to focus on a simple experimental setting that would be easy to follow and present, and which will be sufficiently informative to achieve the objective of this simulation study.

A single algorithm execution is performed on a random problem instance. We generate instances ranging from 10 to 700 bins representing physical hosts. We set the ratio by $\rho = n/m$ between the number of items (representing VMs) and bins in an instance and generate the number of items to satisfy this ratio. Item types are chosen uniformly from "Small", "Medium", "Large" which are the three typical sizes of virtual machine instances, where "Small", "Medium", and "Large" require 1, 3, and 7 compute units, respectively[5]. Bin capacities are expressed in the same compute units as items. We uniformly draw bin capacities from $[20, 40]$. In most of our instances we used $\rho = 10$. The number of items comprising each service is drawn uniformly from $[1, 10]$. We consider "Silver", "Gold" and "Platinum" SLA types with hourly compute unit price of 1, 2, 4, respectively. Each service SLA type is drawn uniformly from "Silver", "Gold" and "Platinum".

We partitioned our experiments into three sets. In the first set we considered small resource pools with the number of bins ranging from 10 to 220 and $\rho = 10$. In this problem size range we explicitly compare our column generation algorithm to direct IP formulation and to the solution obtained by LP relaxation of ESPP. Beyond this range solving direct IP proved to be infeasible. Thus, we conducted the second set of experiments where we varied the number of bins from 250 to 700 with the same item-bin ratio as before. This set of experiments evaluates performance of our proposed method on the large resource pools. In these experiments we compare the results obtained by our solution to the LP relaxation only. In both sets of experiments we considered a use case where we start from an initially empty resource pool.

In the third set of experiments we study the use case where an initial assignment of items to bins already exists. We generate additional item sets simulating newly arrived service provisioning requests. Placement of the new services may cause migrations of items across bins. It is our objective in this set of simulations to validate that our migration costs modeling indeed controls the costs of migrations and get insights on the sensitivity of this mechanism.

In practical settings, timeliness of solving the placement problem is of paramount importance. Therefore in all our experiments we introduce two early stopping criteria. The first one is a time limit on the total computation time and the second one is target optimality gap that instructs the MIP solver to stop when the target is achieved. The optimality gap is defined as

$$100 \cdot \frac{\text{LP optimum - Best known feasible solution}}{\text{Best known feasible solution}}.$$

We implemented the column generation algorithm using ILOG CPLEX version 12.1 [29]. We use CPLEX to solve the direct IP formulation of ESPP to compare performance and scalability of the direct approach to our column generation method. When evaluating the column generation method, we use CPLEX to solve the master problem, the subproblem and the IP resulting from the column generation algorithm.

It is known that the column generation method may suffer from the tailing-off effect where large number of iterations may be required to reach an optimal solution of the master problem LP. To reduce this effect, we impose a stopping condition as follows. The column generation halts when solution improves by less than certain percent between consecutive iterations. Our results show that the early stopping condition only marginally impacts solution optimality.

All our computational results are based on 5 randomly generated instances for each problem instance type. We compute average of these runs and also present the maximum value in the result set. We run all our simulations on a Linux machine with 2.5 GHz processor and 16 GByte of RAM.

A. Small Resource Pools

We present the full experimental results for small resource pools in [30]. Due to lack of space we highlight here the most important observations. Namely, we observe that the column generation algorithm becomes much faster than the direct IP formulation as the item-bin ratio increases. However, the optimality gap of the column generation algorithm is slightly worse than that of the direct IP formulation.

Our results for comparing column generation algorithm and direct IP formulation with time limit stopping condition of 15 minutes for resource pools of size 10 to 220 bins and $\rho = 10$

[5]The concrete definition of a compute unit differs from one provider to another. See, e.g., [28]. We use the term compute unit in a generic manner.

show that if relatively long computation times are permissible, the direct IP formulation is slightly superior to the column generation method.

The big advantage of the column generation approach becomes evident in the next set of experiments where target optimality gap stopping condition is used. These simulations are summarized in Table I.

As one can see the column generation algorithm is significantly faster than the direct IP formulation for the target optimality gap stopping condition. At the same time the optimality gap of the column generation algorithm is only slightly worse than that of the direct IP formulation.

m	n	COLUMN GENERATION				DIRECT IP			
		Gap [%]		Time [s]		Gap [%]		Time [s]	
		Avg	Max	Avg	Max	Avg	Max	Avg	Max
10	100	6.75	8.63	0.18	0.26	3.15	5.61	0.03	0.05
40	400	8.85	9.34	1.56	1.66	5.59	7.62	1.31	2.07
70	700	7.58	9.21	4.72	5.25	3.86	6.9	7.16	7.73
100	1000	8.93	10.11	8.53	10.43	3.15	5.7	25.77	33.22
130	1300	9.13	10.01	17.90	22.03	3.11	4.79	60.04	73.97
160	1600	8.74	10.19	35.63	43.99	2.9	3.5	110.39	138.33
190	1900	8.8	9.29	42.58	45.21	3.0	4.82	256.74	409.91
220	2200	9.53	9.99	66.54	86.39	3.64	4.37	434.59	623.28

TABLE I
TARGET OPTIMALITY GAP OF 10% FOR SMALL RESOURCE POOLS

B. Large Resource Pools

For large resource pools CPLEX was unable to find feasible solutions for the direct IP formulation within the time limit, hence in this section we show computational results only for the column generation algorithm. The results are summarized in Tables II and III. The tables additionally display the average and maximum number of generated columns and the number of column generation iterations.

Our results show that for the time limit stopping condition the optimality gap increases with the problem size. We can see in Table II that for the problem instances of at most 500 bins, the average optimality gap obtained within the time limit is at most 10.29 and for larger instances of size at most 700 bins, the average optimality gap is at most 23.07.

As seen from Table III, the obtained optimality gap of the column generation algorithm may occasionally exceed the target optimality gap of 10 percents. We can see in the table that the average optimality gap is at most 11.95. This happens because we use early stopping condition for column generation to mitigate the tailing-off effect as discussed earlier. Therefore we obtain near optimal LP solutions to the master problem rather than the optimal ones and this contributes to the overall optimality gap.

m	n	Gap [%]		Time [s]		Columns		Iterations	
		Avg	Max	Avg	Max	Avg	Max	Avg	Max
250	2500	6.07	6.62	900	900	2387	2494	9	9
300	3000	6.81	7.11	900	900	2890.6	3102	9.4	10
350	3500	7.26	8.32	900	900	3369.4	3439	10	10
400	4000	8.676	9.63	900	900	4166.6	4434	11.4	12
450	4500	8.97	11.31	900	900	4928.8	5568	13	14
500	5000	10.29	11.92	900	900	5699.8	6084	13.2	14
550	5500	12.62	13.85	900	900	6677.6	7168	15.4	17
600	6000	13.65	14.95	900	900	7441.2	8022	16.6	17
650	6500	16.36	20.16	900	900	8597.6	9021	18.2	19
700	7000	23.07	23.9	900	900	9082.6	9346	17.2	18

TABLE II
TIME LIMIT OF 15 MINUTES FOR LARGE RESOURCE POOLS

m	n	Gap [%]		Time [s]		Columns		Iterations	
		Avg	Max	Avg	Max	Avg	Max	Avg	Max
250	2500	11.10	12.2	62.506	65.19	2276.2	2457	9	9
300	3000	10.52	11.95	96.248	104.44	2947.4	3114	10	10
350	3500	11.04	11.87	141.578	158.07	3662	3805	10.8	11
400	4000	11.95	13.0	226.26	284.46	4307.4	4571	11.8	12
450	4500	10.34	12.32	406.194	459.99	5190.8	5532	13	14
500	5000	11.90	12.6	562.476	749.65	5603.4	5944	13.8	15
550	5500	11.41	13.37	910.94	1028.38	6710.8	7090	15.6	17
600	6000	11.83	13.21	1,167.27	1,250.96	7547.8	7925	16.4	17
650	6500	10.91	11.52	1,987.33	2,836.34	8065.6	8141	17.2	18
700	7000	11.61	12.35	2,072.33	2,364.51	9061.4	9791	17.2	18

TABLE III
TARGET OPTIMALITY GAP OF 10% FOR LARGE RESOURCE POOLS

C. Placement with migrations

In this section we consider the case of replacement, where the placement problem starts from an existing placement. In this case moving from the existing placement to the new placement may require live migrations of VMs. To take into account the operational costs incurred by live VM migrations, we subtract these costs from the value obtained from the migrated VMs. Thus, an already placed VM may produce different values when placed on different hosts.

We model migration cost of a VM with value v as αv, where $0 \leq \alpha \leq 1$. Thus, the real value that results from moving a VM from the original host to another host is $(1-\alpha)v$. In general α may vary for each VM and may depend on the communication cost between hosts in the host pool and between the host pools. In the presented experiments, we consider the homogenous case where communication costs between the hosts in the pool are uniform. We test the case of replacement by generating an instance for 250 to 700 bins. For each problem instance we compute an initial placement. Then, we generate new VMs that amount to $\beta = 0.2$ fraction of the original number of VMs and run the placement algorithm for different values of α. We evaluate the quality of the new placement in terms of the total value to the provider and the number of VM migrations.

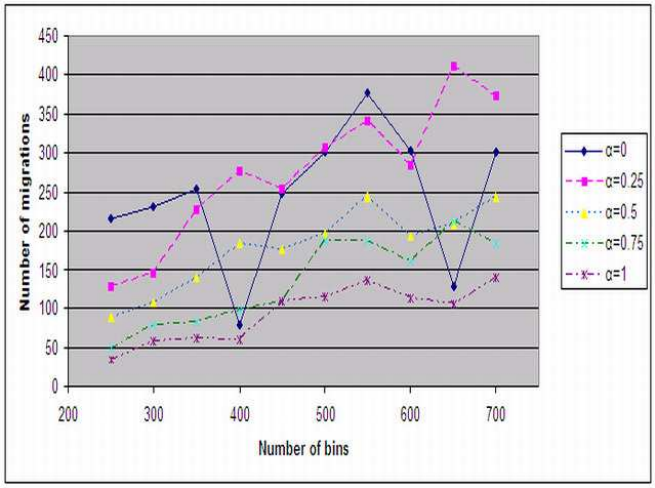

Fig. 1. time limit of 15 minutes

Figure 1 shows the effect of migration cost on the number of migrations performed when moving from the initial placement to the new placement computed by the column generation algorithm with the time limit stopping condition of

15 minutes. As expected, the number of migrations decreases as the migration cost grows, which indicates that the cloud providers can control the operational cost of migrations using this method. Due to the lack of space we do not show the tabular data on the optimality gap. In general, the optimality gap obtained for the time limit of 15 minutes decreased from 10% to 4.5% as migration costs increased.

One may notice from Figure 1 that when the cost of migrations is zero, the number of migrations sharply decreases occasionally. At a first glance this appears to be counter-intuitive since zero migrations costs should have encouraged the solver to use more migrations when seeking for optimal solutions. The explanation for this effect is that when migration costs are non-zero, the solver starts with a smaller preferred candidate set for the already placed items giving placement preference to the bins where these items are placed at the moment. In the absence of migration costs the solver needs to perform much more branching since there is no preferred placement for the items. Therefore 15 minutes might be too short a time limit and the optimality gap achieved in the case of zero migration costs is worse than in the case of non-zero migration costs.

To verify that in [30], we repeated the same experiment with the time limit of 30 minutes. The results of these simulations show that when given more time, the optimality gap in case of zero migration costs improves and more migrations are performed as expected.

VIII. Conclusion

We presented a novel approach to SLA-aware placement of elastic services in a cloud. We formulated a novel optimization problem, ESPP, and showed its relationship to GAP and multi-unit combinatorial auctions. We further developed an efficient computational methodology to solve ESPP using column generation and demonstrated that the tradeoff between solution optimality and timeliness can be efficiently managed. We argue that the operational costs related to SLA provisioning and the network overhead incurred by live VM migrations can be modeled within the same framework. We evaluated our approach using simulations under the reasonable simplifying assumptions. As demonstrated by our results, the column generation method is capable of obtaining close to optimal solutions while controlling the tradeoff between the optimality gap of a solution and its timeliness.

One future work direction that we explore is how to integrate the operational costs such as energy into the proposed framework. One promising approach is extending the multi-unit combinatorial auction modeling by adding the "opening cost" of bins.

References

[1] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The RESERVOIR Model and Architecture for Open Federated Cloud Computing," *IBM Journal of Research and Development*, vol. 53, no. 4, 2009.

[2] "Right Scale," http://www.rightscale.com/m/.

[3] "The RESERVOIR project home page," http://www.reservoir-fp7.eu, 2008.

[4] B. Urgaonkar, A. L. Rosenberg, and P. J. Shenoy, "Application placement on a cluster of servers," *Int. J. Found. Comput. Sci.*, vol. 18, no. 5, pp. 1023–1041, 2007.

[5] S. Mehta and A. Neogi, "Recon: A tool to recommend dynamic server consolidation in multi-cluster data centers," in *IEEE Network Operations and Management Symposium (NOMS 2008)*, Salvador, Bahia, Brasil, Apr 2008, pp. 363–370.

[6] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-Box and Gray-Box Strategies for Virtual Machine Migration," in *USENIX Symposium on Networked System Design and Implementation (NSDI'07)*, Cambridge, MA, USA, Apr 2007.

[7] S. Khuller, J. Li, and B. Saha, "Energy efficient scheduling via partial shutdown," in *Proc. 21th ACM-SIAM Symp. on Discrete Algorithms*, 2010, pp. 1360–1372.

[8] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM*, 2010.

[9] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM J. Comput.*, vol. 35, no. 3, pp. 713–728, 2005.

[10] D. Shmoys and E. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming A*, vol. 62, pp. 461–474, 1993.

[11] U. Feige and J. Vondrák, "Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e," in *Proc. 47th IEEE Symp. on Found. of Comp. Science*, 2006, pp. 667–676.

[12] D. J. Lehmann, L. O'Callaghan, and Y. Shoham, "Truth revelation in approximately efficient combinatorial auctions," *J. ACM*, vol. 49, no. 5, pp. 577–602, 2002.

[13] S. Dobzinski, N. Nisan, and M. Schapira, "Approximation algorithms for combinatorial auctions with complement-free bidders," *Math. Oper. Res.*, vol. 35, no. 1, pp. 1–13, 2010.

[14] B. Lehmann, D. J. Lehmann, and N. Nisan, "Combinatorial auctions with decreasing marginal utilities," in *ACM Conference on Electronic Commerce*, 2001, pp. 18–28.

[15] P. Briest, P. Krysta, and B. Vöcking, "Approximation techniques for utilitarian mechanism design," in *STOC*, 2005, pp. 39–48.

[16] A. Srinivasan, "Improved approximation guarantees for packing and covering integer programs," *SIAM J. Comput.*, vol. 29, no. 2, pp. 648–670, 1999.

[17] P. Raghavan, "Probabilistic construction of deterministic algorithms: Approximating packing integer programs," *J. Comput. Syst. Sci.*, vol. 37, no. 2, pp. 130–143, 1988.

[18] A. Srinivasan, "An extension of the lovász local lemma, and its applications to integer programming," in *SODA '96: Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, 1996, pp. 6–15.

[19] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting stock problem-Part II," *Oper. Res.*, vol. 11, pp. 863–888, 1963.

[20] M. Desrochers, J. Desrosiers, and M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows," *Oper. Res.*, vol. 40, no. 2, pp. 342–354, 1992.

[21] M. Desrochers and F. Soumis, "A column generation approach to urban transit crew scheduling," *Transportation Sci.*, vol. 23, pp. 1–13, 1989.

[22] P. Värbrand, D. Yuan, and P. Björklund, "Resource Optimization of Spatial TDMA in Ad Hoc Radio Networks: A Column Generation Approach," in *INFOCOM*, 2003, pp. 818–824.

[23] S. de Vries and R. V. Vohra, "Combinatorial auctions: A survey," *INFORMS Journal on Computing*, vol. 15, no. 3, pp. 284–309, 2003.

[24] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, *Algorithmic Game Theory*. New York, NY, USA: Cambridge University Press, 2007.

[25] H. N. Gabow, "Data structures for weighted matching and nearest common ancestors with linking," in *SODA*, 1990, pp. 434–443.

[26] C. Barnhart, E. L. Johnson, G. L. Nemhauser, and M. W. P. Savelsbergh, "Branch-and-price: Column generation for solving huge integer programs," *Oper. Res.*, vol. 46, no. 3, pp. 316–329, 1998.

[27] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, *CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience*. New York, USA: Wiley Press, 2010.

[28] Amazon Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2.

[29] IBM ILOG CPLEX 12.1, http://www-01.ibm.com/software/integration/-optimization/cplex-optimizer.

[30] D. Breitgand and A. Epstein, SLA-aware Placement of Multi-Virtual Machine Elastic Services in Compute Clouds. IBM Technical Report, H-0287, 2010.