H-0299 (H1102-014) February 12, 2011
Computer Science

IBM Research Report

Policy-Driven Service Placement Optimization in Federated Cloud

David Breitgand
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

Alessandro Maraschini
Telespazio
Via Tiburtina
965 - 00156, Rome, Italy

Johan Tordsson
Department of Computing Science
Umea University
901 87 Umea, Sweden

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Policy-Driven Service Placement Optimization in Federated Clouds

David Breitgand
IBM Haifa Research Labs
Haifa University Campus,
Mount Carmel, Haifa, 31905, Israel
davidbr@il.ibm.com

Alessandro Maraschini
Telespazio
Via Tiburtina
965 - 00156, Rome, Italy
alessandro.maraschini@telespazio.com

Johan Tordsson
Department of Computing Science
Umeå University
901 87 Umeå, Sweden
tordsson@cs.umu.se

***Abstract*—Efficient provisioning of elastic services constitutes a significant management challenge for cloud computing providers. We consider a federated cloud paradigm, where one cloud can subcontract workloads to partnering clouds to meet peaks in demand without costly over-provisioning. We propose a model for service placement in federated clouds to maximize profit while protecting Quality of Service (QoS) as specified in the Service Level Agreements (SLA) of the workloads. Our contributions include an Integer Linear Program (ILP) formulation of the generalized federated placement problem and application of this problem to load balancing and consolidation within a cloud, as well as for cost minimization for remote placement in partnering clouds. We also provide a 2-approximation algorithm based on a greedy rounding of a Linear Program (LP) relaxation of the problem. We implement our proposed approach in the context of the RESERVOIR architecture.**

I. Introduction

Cloud computing facilitates thin provisioning through elastically matching variations in workloads by dynamically changing resource allotments to services. This way, non-functional Service Level Objectives (SLO) are maintained in spite of unpredictable workload spikes on the one hand, and, on the other hand, under-utilization is minimized when the load subsides. We consider a popular Infrastructure as a Service (IaaS) paradigm where service providers rent Virtual Execution Environments (VEE) on-demand from IaaS providers. These VEEs are used by the service providers to supply a functionality, a *service*, to the end-users. Usually, the IaaS provider is oblivious to the service semantics. From the IaaS service provider's vantage point, each service is a structured collection of VEEs. It is the responsibility of the service provider to dimension capacity of its service in terms of the number of VEE instances and instance sizes to achieve the desirable end-user QoS for the expected load levels.

An IaaS provider faces the problem of finding an optimal – according to some criterion set by the provider – mapping of the VEEs that comprise the services to physical hosts. If the VEEs of the service are successfully placed, the user-level QoS is protected. Obviously, achieving this goal at all times for the unlimited services population requires the IaaS provider to maintain significant server capacity. While this may be a possible, albeit not very cost-efficient, model for a handful of large providers, it is not a viable strategy for small and medium-sized ones. This problem is a key motivation for the RESERVOIR project [1], which is centered around the concept of organizing multiple data centers (IaaS providers) of varying sizes into a federation of clouds, where capacity can be shared by peer clouds, while each of these still preserves full business autonomy. Figure 1 depicts the RESERVOIR federation model. The architecture is discussed in greater details in Section IV. At this point in our presentation, it is important to stress that in this model, every IaaS cloud operates as an autonomous business that has its own administrative policy with respect to solving the VEE placement problem. As such, one cloud may place VEEs to minimize the number of powered up physical hosts aiming at maximal consolidation. Another may use a different placement policy, e.g., load balancing in a strive to equalize load variance across the active hosts.

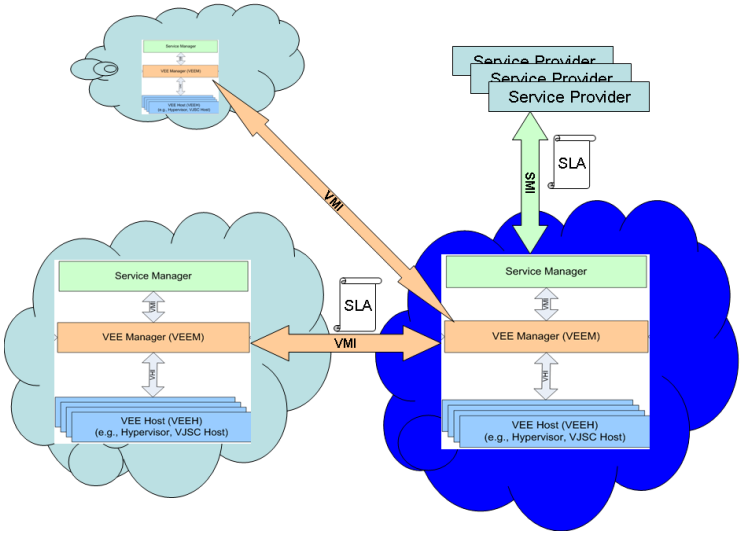

Fig. 1. The RESERVOIR Model

In RESERVOIR, the structure of an elastic service is declared in a service manifest along with cardinality constraints for the VEEs that make up the service. At run time, the structure of the service remains fixed, but the number of VEE

instances varies. Thus, at any given point in time, each service presents the IaaS provider with requirements expressed in terms of a number of VEE instances of different sizes.

RESERVOIR follows the approach of Amazon EC2 [2] by offering capacity using discrete virtual hardware configurations. We refer to each such configuration as *VEE type*. One important difference from the Amazon EC2 model is that in RESERVOIR, VEE types also include availability SLOs as part of the type definition. The charge-back is based on both the virtual capacity size and availability level.

Each successfully provisioned service (i.e., set of placed VEE instances) produces revenue for the IaaS provider on a per VEE and CPU hour basis. Providers that participate in the federation are interested in maximizing their own profit from placement of VEE instances. However, each provider optimizes this objective subject to its local placement policy. To facilitate capacity sharing, peer clouds enter *framework agreements*. These agreements specify the number of VEE instances of each type that on demand can be offered to a cloud from its peers, and at what cost. In addition, the framework agreement specifies availability SLOs and security levels for the VEE types offered. Table I provides an example framework agreement that illustrates how a cloud can place VEEs in three peer clouds.

TABLE I
FRAMEWORK AGREEMENT EXAMPLE WHERE THREE PEER CLOUDS IN TOTAL PROVIDE SEVEN VEE OFFERS.

ID	Type	Number	Site	Cost	QoS	Security
1	Large	4	UMU	0.15	Golden	High
2	Large	2	UMU	0.25	Platinum	High
3	Small	10	UMU	0.05	Golden	Medium
4	Small	10	IBM	0.02	Golden	Medium
5	Medium	5	IBM	0.10	Golden	High
6	Large	2	Tele	0.28	Platinum	High
7	Small	4	Tele	0.10	Platinum	High

In the context of our federation model, we define a novel Federated VEE Placement Optimization Meta-Problem (FVPM), where each cloud autonomously maximizes the utility of VEE placement using both the local capacity and remote resources available through framework agreements.

Different local placement policies can be incorporated into the framework provided by FVPM. We present Integer Linear Program (ILP) formulations for two local VEE placement optimization policies: *power conservation* and *load balancing*. The objective of the power conservation policy is to maximize consolidation of VEEs onto the physical hosts while maximizing utility of the placement. Load balancing aims at equalizing the load among physical machines with respect to a single, configurable, capacity dimension such as CPU utilization, memory usage, network bandwidth, etc., while at the maximizing utility of the placement. These two local placement optimization policies are augmented by a remote placement cost minimization policy. The latter policy is applied by each cloud to remotely place the VEEs that cannot be feasibly placed locally.

For small clouds with few VEEs and hosts, the FVPM may be solved exactly. However, for large-scale placement problems even the state of the art ILP solvers may experience difficulties. We show that under realistic assumptions our problem is amenable to a highly scalable 2-approximation heuristic. This heuristic uses LP relaxation to solve local part of the FVPM in each cloud and then applies a greedy LP rounding heuristic to the fractional solution. The rounding heuristic makes use of remote placement enabled by the framework agreements.

In summary, our contributions are as follows. We present an architecture for policy-driven placement of VEEs in a federation of clouds. We formulate a novel combinatorial optimization meta-problem, the FVPM. The problem is concretized in the form of policies for power saving and load balancing within a cloud, as well as cost minimization for placement in peer clouds. We also provide a highly scalable 2-approximation algorithm for the FVPM. Finally, we describe the integration of our proposed algorithms in the RESERVOIR architecture.

The rest of this paper is organized as follows. Section II describes our model for federated cloud computing and introduces the notation. Section III presents the problem and describes the algorithms used to solve it and provides their analysis. In Section IV we outline the architecture for implementing the proposed approach. Finally, some concluding remarks are found in Section VI.

II. MODEL

In this section we describe our model for federated cloud computing that forms the background for the studied placement problems. Our model comprises multiple clouds (IaaS providers) where each cloud consists of a set physical hosts used to provision VEEs. Each VEE has an availability SLO associated with it, which is part of a service SLA. Each VEE generates a certain revenue per time unit for the cloud provider according to the actual usage by the customer. Obviously, provisioning imposes variable production costs such as power consumption and SLA incompliance penalties unless availability SLOs of the service SLA are met. Also included in the model are costs associated with migration of a VEE from one physical host to another, potentially located in another cloud. The latter scenario is regulated through a framework agreement between the two clouds.

Formally, we consider a set of cloud providers $c_1, \ldots, c_n$ where each cloud c has m_c physical hosts. A physical host can be viewed as a d-dimensional knapsack of capacity $B = (b_1, \ldots, b_d)$ where each dimension represents a hardware capacity feature, e.g., number of CPUs, memory size, disk volume, or network bandwidth. We assume discrete time $t_1, t_2, \ldots$, where at each time instance t_k, each cloud provider c is given n_c d-dimensional items, VEEs, that have to be mapped on the physical hosts available to this provider locally, or remotely via the framework agreements with the peer

clouds[1].

All VEEs are offered with discrete virtual hardware configurations, i.e., there are a few pre-defined sizes of VEEs in terms of CPU, memory, disk, etc. Each VEE has an availability SLO associated with it. The availability SLO is a tuple of the following generic form: *(availability test, frequency, success ratio, billing period)*. An SLO that checks whether a service is online can look, e.g., as follows: *(ping host, 1 per minute, 98 %, 30 days)*[2].

If an availability SLO of VEE j is breached, the SLA where this SLO is one of the clauses mandates a penalty $fine(j)$ at the end of the billing period. Usually this penalty does not exceed the actual revenue generated by the VEE during this period. The customer gets compensated by the service credit for the next billing period. From the provider's vantage point the direct repercussion is a loss of revenue for the next billing period should the customer indeed use the service [2]–[4]. Indirect losses, for example, in terms of provider reputation, might be much more significant, but they are also more difficult to estimate and are omitted in this work. In our modeling we take a conservative approach and assume that if a SLO of the customer is breached in the current billing window, revenue is lost in the next window.

We assume that each cloud provider c has f_c framework agreements with its peer providers. Each framework agreement (corresponding to one line in Table I) mandates how many VEEs of a given type can be placed in a peer cloud under a given availability SLO. Table I shows a simple example of how a cloud can place VEEs in three peer clouds.

If placed at time t, each VEE j is expected to produce value $v_{j,c}$ for cloud provider c in the interval $(t, t+1)$, and 0 otherwise. The value is expressed in monetary terms as the price paid by the customers of this cloud provider for obtaining a VEE. We assume a fixed price model, i.e., the value produced by the placed VEE is time-independent and utilization independent. Each placement configuration exacts costs of provisioning on a cloud provider. The objective of each cloud provider c is to maximize its revenue, expressed by the utility function:

$$z = \sum_{t=1}^{T} \sum_{j=1}^{n} (v_j - p_j^t). \quad (1)$$

In Equation 1, p_j^t is the cost of placement that VEE j incurs on the cloud provider at time t [3]. The placement may be either local or remote. T is a billing period that comprises multiple time intervals.

It should be noted that the actual value v_j generated by VEE depends on whether the user actually boots VEE j after it is being placed by the provider. Obviously, it is possible to obtain 0 value from placing a VEE, if the user requests the VEE and then immediately shuts it down. The value obtained from placing a VEE on discrete time intervals is a binary random variable with value 1 if the user has powered up the VEE during this interval and 0 otherwise. Although observations of historical VEE behavior may be used to predict activity in future time intervals, we take a conservative approach and assume VEEs to be active after initial placement until they are decomissioned on request from the user.

It should be noted that non-placing a VEE at time t may result not only in losing revenue for the next time interval $(t, t+1)$, but also in penalties at the end of the billing period (as explained above) if non-placement at time t breaches availability SLO of the VEE in this billing period. The various aspects that affect the cost of placing a single VEE are modelled as follows:

- If VEE j is placed remotely at cloud c_2 using an appropriate framework agreement $f_{c1,c2}$ between clouds c_1 and c_2, c_1 pays a provisioning fee $cost_{c_2}^{f_{c1,c2}}(j_{c_1})$ to c_2.
- If at time $t-1$ VEE j is placed in cloud c_1 and at time t, VEE j is placed remotely at cloud c_2, then a long-haul migration penalty $long\_migr_{c1,c2}(j)$ is incurred on cloud c_1. This cost component is motivated by payments by cloud c to the Internet service provider and performance deterioration due to the long distance migration.
- If at time $t-1$ VEE j is placed at host h_1 in cloud c and at time t VEE j is placed at host $h_2 \neq h_1$ within the same cloud c, then a local migration penalty $local\_migr_{h_1,h_2}(j)$ is incurred on cloud provider c. This cost component is motivated by the fact that each in-band VEE migration competes for the same network resources as the regular services and thus potentially degrades their performance.
- If at time $t-1$, a host h in cloud c is not used to place any VEE and at time t host h is used to hold at least one VEE, cost $pow(h)$ of using the host h is incurred on the cloud provider c. The motivation for this cost is given by the operational expenses such as power, cooling, etc.

Definition 1. *The Federated VEE Placement Meta-Problem is to find a placement configuration that maximizes the utility function of Equation 1.*

FVPM is an optimization meta-problem in the sense that different concrete optimization policies can be plugged into this framework by each cloud c. A local placement optimization problem at any cloud has the generic form shown in Equation 1. However, a concrete problem for a given cloud is formulated subject to the *management policies* that are pursued by this cloud.

The term policy-driven management usually implies the

[1]We make an assumption that every VEE can be placed in any of the peer clouds having framework agreements, without sacrificing the QoS of the service to which this VEE belong.

[2]Depending on the availability test, a cloud provider may have to allocate different amount of resources to metering and monitoring. A more sophisticated type of availability test would be for example, logging into the system, executing some synthetic transactions and measuring their response time and throughput. The RESERVOIR model favors diversifying availability tests and embedding them within the standardized SLA offerings that are provided to the customer at different price levels.

[3]We omit the index c for simplicity of notation wherever this does not lead to ambiguity.

management approach based on executing the rules of the form *if(condition)-then(action)* that are defined to handle various situations. For example, a rule may be defined that if the average CPU utilization of a cluster of VEEs exceeds a predefined threshold, say, 30%, an additional VEE is added to the cluster. These simple policies serve as a basic building block for autonomic computing.

However, the overall optimality criteria of placement, are controlled by the management goals, which are defined at a higher level of abstraction than condition-action rules. Management goals, such as to maximize utility of placement while, e.g., conserving power, preferring local resources over remote ones, balancing workload, minimizing VEE migrations, etc. have complex logical structure and cannot trivially be expressed using condition-action rules. In our approach, specific management policies denote the high-level management goals. These placement policies influence the cost terms comprising p_j^t in the utility function of Equation 1 and introduce (or remove) placement constraints.

III. Placement Optimization Problems

In this section we formulate concrete placement optimization problems, to be used with the FVPM framework, as ILPs and present algorithms to solve them. The number of decision variables in these formulations can be quite large, however. For example, for 1000 physical hosts and 10000 VEEs, 10 million binary decision variables must be used. While some variables can be eliminated by preprocessing of the input to remove infeasible assignments, the problem may still have hundreds of thousands of binary decision variables, which makes an exact integer solution impractical even with the state-of-the-art solvers. To cope with the large scale problems, we perform LP relaxations and provide a simple 2-approximation greedy LP rounding heuristic in Section III-D. This heuristic makes use of the RESERVOIR federation model and remote placement cost minimization policy described in Section III-F.

We start with the most basic problem formulation that aims at minimizing provisioning costs due to penalties caused by violation of availability SLAs, the latter in term resulting from non-placement and migrations. We show that our basic problem is essentially a Generalized Assignment Problem (GAP) [5] with assignment restrictions. This basic formulation serves as a building block for the power conservation placement policy described in Section III-C and load balancing placement policy presented in Section III-E.

A. SLA Incompliance Penalties Minimization

We start with formulating the basic optimization problem, which is being invariant under any placement policy we consider. More specifically, under any high level placement policy we are interested to minimize the loss in value due to penalties and unnecessary migrations that may degrade user experience to the point of causing service unavailability. Let y_j^t denote a binary decision variable such that

$$y_j^t = \begin{cases} 1 & \text{if VEE } j \text{ is placed at time } t, \text{ and} \\ 0 & \text{if VEE } j \text{ is not placed at time } t. \end{cases} \quad (2)$$

As explained in Section II, each VEE has an availability SLO associated with it. This SLO defines percentage of non-placements that can be accumulated during the billing period T, before SLO incompliance is declared and $fine(j)(T)$ is exacted on the cloud provider. Notably, the fine does not exceed 100% of the value generated in this billing period. We term this percentage the *breach budget* of VEE j. Another notable side effect of relating service credits to the pay-as-you-go usage fees practiced by most cloud providers is that if the breach budget of a VEE is exceeded in the middle of the billing period, provisioning this VEE in the reminder of the billing period *increases* the fine that is to be exacted on the cloud provider by the end of the billing period, while not placing it would cap the losses by the already accumulated usage. Conversely, while maximal revenue produced by a placed VEE is determined by a CPU hour based pricing scheme and is time-independent, the profit of placing VEEs that do not exceed their breach budget increases with direct proportion to their actual usage and is captured by the time-dependent penalty $fine_j(t)$.

If at time t VEE j is not placed, the cloud provider loses the value $v(j)$ for duration $(t, t+1)$, but if the breach budget by the time $t+1$ is greater than 0 in spite of non-placement, no penalty is incurred on the cloud provider by the end of T. Otherwise, the cloud provider loses $v(j)$ as before and in addition pays $fine(j)(T)$ at the end of the billing period, which is at least $fine(j)(t)$, capturing actual usage of the VEE by up to time t.

A cloud provider aims to maximize the objective function of Equation 1. We define an indicator function $I_j(t)$ per VEE j as shown in Equation 3 and rewrite Equation 1 as shown in Equation 4 to account for SLA violation penalties due to non-placement of VEEs.

$$I_j(t) = \begin{cases} 0 & \text{not placing } j \text{ in } (t, t+1) \text{ within breach budget,} \\ 1 & \text{otherwise.} \end{cases} \quad (3)$$

$$z = \sum_{t=1}^{T} \sum_{j=1}^{n(t)} (y_j^t \cdot v(j) - (1 - y_j^t) \cdot fine(j)(t) \cdot I_j(t)) \quad (4)$$

This can be rewritten as:

$$z = \sum_{t=1}^{T} \sum_{j=1}^{n(t)} (y_j^t \cdot (v(j) + fine(j)(t) \cdot I_j(t)) - fine(j)(t) \cdot I_j(t)). \quad (5)$$

Since the second term is a sum of non-negative values, to maximize z, one has to maximize the first term. Thus, we can rewrite our problem simply as:

$$z = \sum_{t=1}^{T} \sum_{j=1}^{n(t)} (v_j + fine_j(t) \cdot I_j(t)) \cdot y_j(t). \quad (6)$$

.

This equation can be further rewritten as:

$$z = \sum_{t=1}^{T} \sum_{j=1}^{n(t)} y_j^t v_t(j), \quad (7)$$

where

$$v_t(j) = \begin{cases} v(j) & \text{if at time } t,\ j \\ & \text{is within breach budget,} \\ v(j) + fine(j)(t) & \text{otherwise.} \end{cases} \quad (8)$$

B. Migration Costs Minimization

Our next step in modeling the basic problem is to account for the costs due to unnecessary migrations of VEEs among the hosts. An exact modeling of the migration penalty is highly non-trivial. In an earlier contribution [6], we address modeling of the live migration costs due to performance degradation incurred on the VEEs by in-band migration. However, in this work we leave the problem of how migration penalty should be modeled out of the scope, assuming that migration penalties are known for each VEE j and each pair of hosts. Let $x_{i,j}^t$ be a binary decision variable such that:

$$x_{i,j}^t = \begin{cases} 1 & \text{if at time } t \text{ VEE } j \text{ is assigned to host } i, \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

Assigning the value of y_j^t of Equation 7 to 1 means that *at most* one $x_{i,j}^t$ is set to 1, while $y_j^t = 0$ means that all $x_{i,j}^t$ are zero. We rewrite the optimization problem of Equation 7 as follows:

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{i,j}^t \cdot v_t(j) - local\_migr_{i_{t-1}(j), i_t(j)}(j)). \quad (10)$$

We define:

$$i_t(j) = \begin{cases} i & \text{if at time } t \text{ VEE } j \\ & \text{is assigned to host } i, \\ undefined & \text{otherwise.} \end{cases} \quad (11)$$

Obviously, if a placement exists for a VEE at time $t-1$, the maximal value for this VEE (if it is placed in the next cycle, t) is attained if the VEE remains in place and no migration cost is incurred. Thus, the problem of Equation 10 can be rewritten as:

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j}^t \cdot v_t(i,j), \quad (12)$$

where:

$$v_t^{i,j} = \begin{cases} v_t(j) & \text{if } i_t(j) = i, \\ v_t(j) - local\_migr_{i_{t-1}(j), i_t(j)}(j) & \text{otherwise.} \end{cases} \quad (13)$$

To complete our problem formulation, we introduce a number of placement restrictions. Constraint 14 ensures that each VEE is assigned to at most one physical host in any feasible placement:

$$\forall i, j, t : \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j}^t \leq 1. \quad (14)$$

Constraint 15 ensures that the total capacity requirements of all VEEs assigned to a single host does not exceed capacity of this host:

$$\forall i, t : \sum_{j=1}^{n} x_{i,j}^t \cdot b_d(j) \leq b_d(i). \quad (15)$$

In addition to these, deployment constraints such as anti-collocation constraints can be specified. Let AAG_l denote sets of VEEs that cannot be placed on the same physical host. The the anti-collocation constraints can now be defined as follows:

$$\forall i, l, t : \sum_{j \in AAG_l} x_{i,j}^t \leq 1. \quad (16)$$

The objective function given by Equation 12 and constraints 14, 15, and 16 comprise our basic optimization problem.

Notably, the optimization problem that is greedily solved at each time instance t is the well studied Generalized Assignment Problem (GAP) [5] with assignment constraints. In absence of placement restrictions GAP is known to be APX hard [7]. In this context we consider both placement restrictions and extensions to GAP, such as cost of booting host for placement, to model variable production costs due to energy consumption. As discussed in Section V, these extensions make GAP considerably harder to approximate.

While we try to maximize the utility of placement at every time instance t, a greedy algorithm does not have to be optimal. An optimal algorithm, however, requires knowledge of the future VEE placement requests and usage patterns. This information is usually not available to cloud provider.

Small scale problems can be solved exactly. For large instances, an exact solution is not a practical option and approximations must be used. In Section III-D we present an approximation heuristic based on ILP relaxation and greedy LP rounding that – under reasonable assumptions about VEEs and hosts – produces a 2-approximation for local placement. The excess VEEs are placed in partner clouds under a remote placement revenue maximization policy via the RESERVOIR federation model as described in Section III-F.

C. Power Conservation Local Placement Policy

In the previous subsection we introduce the basic optimization problem that remains invariant under any placement policy. One specific placement policy that we consider, aims at saving power usage via workload consolidation on a minimal number of physical hosts. We call this policy **M**aximal **U**tility **M**aximal **C**onsolidation (MUM-C). With this policy,

the host usage penalty is factored into the objective function of Equation 12 as follows:

$$z_{mum-c} = \sum_{i=1}^{m}\sum_{j=1}^{n}(x_{i,j}^{t} \cdot v_t(i,j)) - \sum_{i=1}^{m} h_i^t \cdot pow(h). \quad (17)$$

Constraint 14 and Constraint 16 remain the same. Constraint 15 changes as follows:

$$\forall i,t : \sum_{j=1}^{n} x_{i,j}^{t} \cdot b_d(j) \leq b_d(i) \cdot h_i^t, \quad (18)$$

where:

$$h_i^t = \begin{cases} 1 & \text{if at time } t \text{ host } i \text{ is used for placing VEEs,} \\ 0 & \text{otherwise.} \end{cases} \quad (19)$$

Together the objective function of Equation 17 and Constraints 14, 16, and 18 form our local power conservation optimization problem. To allow for fast solutions, we relax Problem 18 by allowing values for $x_{i,j}^t$ to assume values in the interval $(0,1)$, rather than requiring them to be the binary decision variables. Once a fractional solution to the relaxed LP problem is obtained, we round it using the greedy heuristic described in Section III-D. After rounding, there may in each cloud exist VEEs that could not be placed locally. These overflow VEEs are placed in peer clouds using the minimal placement cost optimization algorithm described in Section III-F.

D. Greedy LP Rounding Heuristic

In this section we describe the LP rounding heuristic. First, we introduce a simplifying assumption, namely that every VEE can be placed on every host in absence of other VEEs. More formally, we assume that

$$\forall d,j,i : b_d(j) \leq b_d(i). \quad (20)$$

We order all fractionally placed VEEs in descending order of capacity required by the VEE and pass over this list, the $DC-list$, sequentially. At every step j we make a sequential pass over all the hosts and try to find a host i^* such that:

$$\sum_{k=1}^{n} v(j_k) \cdot x_{i^*,j_k} \leq v(j) \quad (21)$$

and for each capacity dimension d:

$$\sum_{k=1}^{n} b_d(j_k) \cdot x_{i^*,j_k} \geq b_d(j), \quad (22)$$

with $x_{i,j_k} \in (0,1)$. In other words, we are looking for a host i^* where the total value of the sum of the fractional assignments is *no larger* than that of j and total capacity taken up by the sum of the fractional assignment is *at least* that of j. If we find such a host i^*, we set $x_{i^*,j} = 1$ and $x_{i,j} = 0$ for all $i \neq i^*$. Otherwise, we drop j from the placement, i.e., set $x_{i,j} = 0$ for all i. If there exists an assignment of VEE $j^{'}$ to host i^*, i.e., $x_{i^*,j^{'}} = 1$, and there exists an anti-collocation placement constraint that prevents placing j and $j^{'}$ on the same host, we drop j from the current placement if $v(j,i^*) \leq v(j^{'},i^*)$ and $\forall d$, $b_d(j) \geq b_d(j^{'})$. Otherwise, we drop $j^{'}$ from the current placement, i.e., set $x_{i,j} = 0$ for each host i. VEEs that are dropped from placement at each step j are also dropped from the DC-list and are put on the pending remote placement list, the $PRP-list$.

Note that finding subsets satisfying Inequality 21 and Inequality 22 is equivalent to solving an optimization version of SUBSET-SUM decision problem [8]. SUBSET-SUM is an NP-hard problem, but since in our case all values and capacities are positive numbers, this problem admits a fully polynomial time approximation scheme (FPTAS). Furthermore, in the average case it might be expected that for any single host only a small fraction of all VEEs are assigned to this host and therefore the problem can be efficiently solved by dynamic programming or even by direct search.

The rounding algorithm terminates when the DC-list is empty. All VEEs omitted from the local placement are placed remotely solving the optimization problem of Subsection III-F exactly using ILP. In the worst case we have to place $\frac{n}{2}$ VEEs on c clouds. So, this still potentially a large problem, but its scale is smaller in the average case since $c \ll n$. It should also be noted that remote placement is an additional task that does not improve the theoretical bound on performance ratio given by Lemma 3. Indeed, it is easy to construct scenarios where placing each VEE remotely may produce only infinitesimally small amount of profit to the cloud provider. However, it is imporatant from a practical perspective. If the problem of remote placement becomes to large to be solved exactly, an LP relaxation of it can be solved, using greedy rounding heuristic similar to the described treating peer clouds as large knapsacks. Notably, any VEEs omitted from placement by the remote placement, are omitted permanently. Alternatively, one can order peer clouds in ascending order of cost and place VEEs greedily.

Lemma 2. *The LP rounding algorithm terminates.*

Proof:

The DC-list can contain no more than n VEEs. At each iteration at least one fractional decision variable gets rounded after inspecting at most $O(m)$ hosts at each step (m is the number of hosts). Therefore there cannot be more than n iterations, after which the algorithm terminates.

■

Lemma 3. *The greedy rounding heuristic is a 2-approximation.*

Proof: In the worst case, each decision variable in the LP solution is fractional. At every step j, the algorithm frees at least capacity taken up by VEE j by rounding the corresponding decision variables while loosing no more value than $v(j)$ in the relaxed LP solution. Since we process the DC-list in the descending order of capacity, if a VEE is dropped

from placement (i.e., all its decision variables are set to 0), we free at least the capacity needed to place at least one VEE that follows the omitted VEE in the DC-list provided that no anti-collocation placement restrictions exists for that VEE. Because of Assumption 20, each host created in our rounding process be utilized by at least one VEE. After $\frac{n}{2}$ iterations, there must exist integral solution for at least $\frac{n}{2}$ VEEs with the total value at least half of the value obtained in the relaxed LP solution. ■

E. Load Balancing Local Placement Policy

We next consider the VEE placement problem under the policy of load balancing. In general, since multiple capacity dimensions (e.g., CPU, memory, disk) with complex inter-dependencies are involved, it is not trivial to define load as a single dimensional metric. We thus consider a more simple variant of the load balancing policy. Suppose, an administrator selects a metric of interest, say memory or CPU and requests a feasible placement such that residual capacity with respect to the selected capacity dimension is as equalized as possible across the physical hosts. More formally, if capacity dimension b_d is chosen for load balancing, then $free_i = b_d(i) - \sum_{j=1}^{n} x_{i,j} \cdot b_d(j)$. VEE placement under the load balancing policy with respect to b_d is then to maximize the objective function of Equation 12 under constraints 14, 15, and 16 and to minimize z_{load}, where:

$$z_{load} = \sqrt{\sum_{i=1}^{m} (free_i - \mu_d)^2}, \tag{23}$$

with $\mu = \frac{1}{m} \sum_{i=1}^{m} free_i$.

Notably, Equation 23 is the standard deviation of the residual capacity, which is a non-linear function. To stay with the simple linear optimization, we use a heuristic to minimize Equation 23, by sequentially solving our basic optimization problem of Subsection III-B. Figure 2 presents the pseudo-code of the algorithm.

Essentially, this is a binary search algorithm. In this algorithm, successive attempts are being made to solve the placement problem by assuming less capacity than actually is available in each physical host. If a feasible solution is obtained, a further decrease in capacity is attempted until no feasible solution is found. At this point capacity is increased and the process is repeated until no further capacity modifications are possible.

F. Cost Minimization Remote Placement Policy

The cost minimization algorithm places VEEs at remote sites and is to large extent based on the information in the framework agreements. This algorithm minimizes the total cost for all remotely placed VEEs, while adhering to VEE constraints in terms of hardware configuration, availability, etc.

Similarly to the power saving and load balancing algorithms, we use a 0-1 ILP formulation for the cost minimization policy. The inputs to the algorithm are the n VEEs and the m different framework agreement offers $FA_1, FA_2, \ldots, FA_m$.

```
1.  Load_Balance(d, vees, hosts, ε) {
2.    //d is capacity dimension along which load balancing is required;
3.    //vees is the array of VEEs to be placed
4.    //hosts is the hosts array
5.    //ε stopping condition
6.    low_threshold = min_d(vees); //VEE with minimal value of
d
7.    util = Σ_{j=1}^{n} vees[i](d) / Σ_{i=1}^{m} hosts[i](d);
8.    //ratio of total capacity required to total capacity available
9.    //w.r.t. capacity dimension d
10.   weight = min{1, util};
11.   if (weight == 1) {//local capacity is fully utilized
12.     Solve Equation 18;
13.     Use Algorithm of Section III-F to place overflow VEEs;
14.   }
15.   lower = weight;
16.   upper = 1;
17.   while (|lower − upper| ≥ ε) { //main loop
18.     for each i, s.t. host[i](d) > lower_threshold(d),
19.     hosts[i](d) = hosts[i](d) · weight;
20.     Maximize Equation 12
21.     if (result contains unplaced (or fractionally placed) VEEs)
22.       lower = weight;
23.     else
24.       upper = weight;
25.     weight = (lower+upper)/2;
26.   //restore original values of capacity dimension d for all hosts
27.   } //end: while (main loop)
28. } //end: Load_Balance
```

Fig. 2. Pseudo-code for the threshold calculation algorithm.

The latter each corresponds to one specific offer, as defined by one line in Table I. Now, let $x_{ij} = 1$ if VEE j is placed remotely under FA_i, and 0 otherwise. The total cost of the remote placement (our objective function to minimize) now becomes:

$$z = \sum_{i=1}^{m} (cost_i * \sum_{j=1}^{n} x_{ij}), \tag{24}$$

where $cost_i$ is the hourly cost for a VEE under framework agreement FA_i.

All solutions that minimize Equation 24 must adhere to the following constraints:

- *Framework Agreement granularity constraints* - The number of VEEs placed under each framework agreement cannot exceed the number of VEEs offered. In other words, for each framework agreement k, $\sum_{j=1}^{n} x_{kj} \leq number_k$ where $number_k$ is the number of VEEs offered, as illustrated in Table I.
- *Aptness constraints* - a VEE may only be placed under a framework agreement that fulfills the requirements of the VEE in terms on VEE type, QoS, and security level. This is encoded as follows. For each framework agreement i and VEE j, add an additional constraint $x_{ij} = 0$ unless framework agreement i has the exact type (i.e., the same hardware capacity) that VEE j requests and the framework agreement offers levels of QoS and security at least as high as the VEE requests.

- *Completeness constraints* - this constraint expresses that each VEE must be placed exactly once. We encode this as: for each VEE l, $\sum_{i=1}^{m} x_{il} = 1$.

IV. Architecture

This section describes how our proposed algorithms for VEE placement are incorporated in the RESERVOIR architecture, which is outlined in Figure 3. The *Service Manager (SM)* works at the level of abstraction of a service. The term service is herein used to describe a collection of VEEs that provide a functionality towards end-users. The overall configuration of a service, including number and types of VEEs, VEE affinity and anti-affinity constraints, SLAs, Key Performance Indicators (KPIs) and elasticity rules, is specified in a *service manifest*. Upon deployment of a new service, the SM requests creation of new VEEs. Based on the service layout defined in this manifest, the SM requests the creation of VEEs from the *Virtual Execution Environment Manager (VEEM)*. When an elasticity rule is triggered in the SM (based on matching of KPIs against monitoring data), additional VEEs may be requested to be created or existing ones may be shut down. Notably, we here only consider the case of horizontal elasticity, i.e., creation of additional VEEs, and not vertical elasticity, i.e., allocation of more physical resources to existing VEEs.

One prominent feature in the RESERVOIR architecture is that the actual physical layout of the VEEs of a service is unknown to the SM. Management of the VMs of the provisioned services and the physical hosts these VM execute in is the responsibility of the VEEM. The *Policy Engine* component of the VEEM decides the placement of VEEs onto physical hosts. The *VEEM Core* is responsible for the enactment of the placement, i.e., creation, deletion, and migration of VEEs, as well as for other tasks in the VEE life cycle, e.g., contextualization. These management actions are performed by the VEEM Core via two interfaces, the *Virtual Host Interface* (VHI) for interaction with local hosts and *Virtual Manager Interface* (VMI) the for remote clouds. The framework agreement between two sites defines the exact terms for placement of VEEs at remote clouds. This agreement specifies the offered number of VEEs, provisioning costs, SLA levels, etc. as exemplified in Table I. The *Virtual Execution Environment Host (VEEH)* layer is responsible for the low-level management of VEEs on physical hosts, and thus creates an abstraction over virtualization platforms such as KVM and Xen, but can also accommodate other types of VEEs such as virtual Java service containers.

As this work focuses on placement issues, our attention in on the Policy Engine architecture. Figure 4 illustrates the internal structure of the Policy Engine and also outlines its main operations. The Policy Engine performs regular *optimization cycles*, where the current state of the local cloud is assessed and the distribution of VEEs over hosts is adjusted accordingly. Such a cycle also includes placement of new VEEs, that are created either upon admission of an additional service, triggering of an elasticity rule for an existing service,

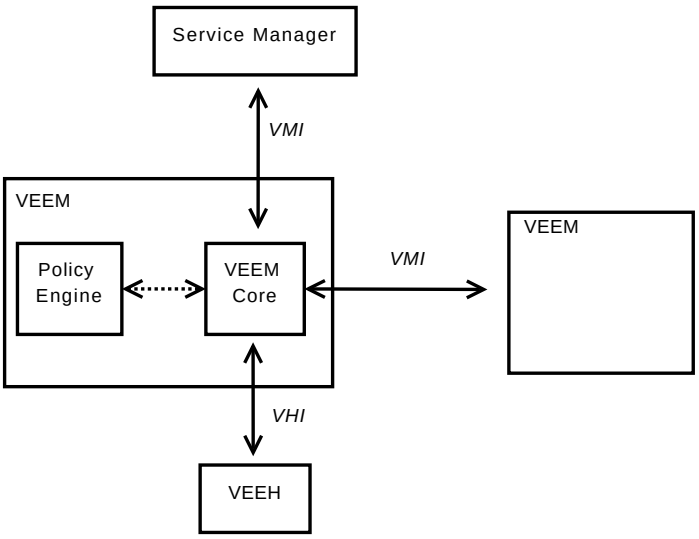

Fig. 3. Outline of the RESERVOIR architecture that illustrates the relationship between the Policy Engine and the related components.

or an incoming request for remote capacity from a partnering cloud. Each optimization cycle includes the following steps:

1) Collection of state information from the VEEM database and creation of the *Object Model* instance, containing information about the available hosts, the VEEs and their runtime state, as well as information about capacity available (and currently used) in other data centers.
2) Gathering of placement information from the *Policy Repository*: placement algorithms to be used, order to apply these algorithms in, and policies configuration parameters. Of particular interest here is the usage of policy chains, which enable multiple algorithms to be applied in sequence. This enables efficient implementation of the combination of local placement followed by remote placement, further as was discussed in Section III.
3) Solving the placement problem: in this phase the placement algorithms gathered at previous step are executed in the needed order and the optimized allocation is determined for VEEs to hosts/remote data centers, with respect to the Object Model, deployment constraints, available capacity, and optimization criteria.
4) Enacting the placement decision: the resulting placement decisions of previous step form a list of tuples that map VEEs to hosts (for local placement) or framework agreements (for remote placement). These are later converted into placement and/or migration instructions and sent to the VEEM Core, the component responsible for performing the actual VEE deployment (or migration) operations.

The components in the Policy Engine and their responsibilities are as follows.

- The *Lifecycle Manager* component is responsible for managing and coordinating all other Policy Engine com-

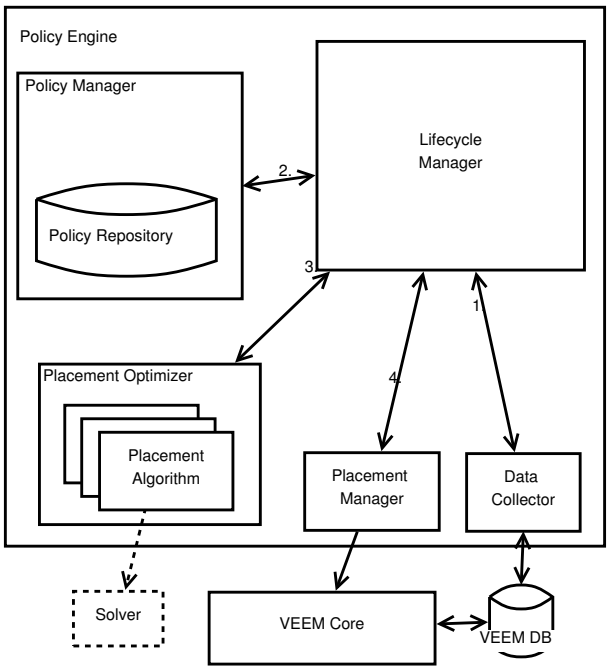

Fig. 4. The main components of the policy engine and their interactions.

ponents. It periodically triggers the execution of a new optimization cycle.

- The *Data Collector* is responsible for populating the Policy Engine object model by connecting to and reading from the VEEM Database.
- The *Policy Manager* component handles customization of placement policies. It internally manages a database of policies allowing dynamic loading, removing, and updating of new policies from external libraries (jar files).
- The *Placement Optimizer* computes the optimal allocation of VEEs over hosts and/or other data centers. An optimizer algorithm can be implemented directly in Java, but does more commonly interact with a solver software to calculate the optimal placement.
- The *Placement Manager* translates the placement decisions into operations (deploy/migrate) and performs the needed requests to the VEEM core.

In all, these components provide a flexible and configurable framework for VEE placement optimization that allows us to use the herein proposed algorithms in a real-world toolkit for federated cloud computing.

V. Related Work

Theoretical results related to this work include a contribution by Chekuri and Khanna [7], who observed that the work of Shmoys and Tardos [5] implies a 2-approximation algorithm for GAP, which is known to be APX-hard. Subsequently, Fleischer et al. [9] design an $\mathrm{e}/(\mathrm{e}-1)$-approximation algorithm . This approximation ratio was improved by Feige and Vondrák [10] to $\mathrm{e}/(\mathrm{e}-1)+\epsilon$ for some constant $\epsilon > 0$.

Another related field of research is the application of combinatorial optimization approaches to placement of virtual machines across multiple clouds. Examples of contributions to this area include Chaisiri et al., who propose a method to minimize the cost of renting virtual machines from public cloud providers, where stochastic integer programming is used to handle uncertainty in future prices and resource demands [11]. Van den Bossche et al. [12] study how to minimize cost of external provisioning in a hybrid cloud scenario, where partial workloads are outsourced from an internal cloud to public providers. Their work focus on deadline-constrained and non-migratable workloads, where memory, CPU, and networking are taken into account in a binary integer programming problem formulation. Van den Bossche et al. also provide some experimental insight into the tractability of their problem formulations.

Tordsson et al. [13] propose cloud brokering mechanism where the virtual machines of a service are deployed across multiple clouds to maximize performance, while considering various constraints in terms of budget, service configuration, load balance, etc. They formulate the cloud brokering problems an an integer programming one, and demonstrate the feasibility of the proposed approach by deploying and benchmarking some sample service configurations.

Breitgand and Epstein [14] consider the problem of multi-VM workload placement subject to set constraints that require to place full configurations to obtain maximum benefit. Breitgand and Epstein also show that this problem cannot be approximated to a constant factor and propose a column-generation based approach to solve the large scale problems exactly while trading precision for time.

VI. Concluding Remarks

We address the management challenge of efficient provisioning of elastic cloud services with a federated approach, where cloud providers can subcontract workloads among each other to meet peaks in demand without costly overprovisioning. Our proposed placement algorithms aim at maximizing provider profit while protecting QoS. These algorithms, cast as ILP formulations, can be used to optimize power saving or load balancing internally in a cloud, as well as to minimize the cost for outsourcing workloads to external partners. We show that under realistic assumptions, our problems are amenable to a highly scalable 2-approximation. To further strengthen the feasibility of our approach, we demonstrate how our placement algorithms are integrated into the RESERVOIR toolkit for federated cloud computing.

Acknowledgment

We thank all members of the RESERVOIR project whose contributions enabled this work. Partial financial is provided by the European Commission's Seventh Framework Programme ([FP7/2001-2013]) under grant agreement no. 215605 (RESERVOIR).

REFERENCES

[1] B. Rochwerger and D. Breitgand and E. Levy and A. Galis and K. Nagin and I. Llorente and R. Montero and Y. Wolfsthal and E. Elmroth and J. Caceres and M. Ben-Yehuda and 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] , "Amazon EC2," www.amazon.com/ec2.

[3] , "Rackspace," http://www.rackspacecloud.com.

[4] , "Gogrid," http://www.gogrid.com.

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

[6] D. Breitgand, G. Kutiel, and D. Raz, "Cost-aware live migration of services in the cloud," in *SYSTOR*, 2010.

[7] 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.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[9] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, "Tight approximation algorithms for maximum general assignment problems," in *Proc. 21th ACM-SIAM Symp. on Discrete Algorithms*, 2006, pp. 611–620.

[10] 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.

[11] S. Chaisiri, B. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*. IEEE, 2010, pp. 103–110.

[12] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads," in *IEEE 3rd International Conference on Cloud Computing (CLOUD)*.

[13] J. Tordsson, R. Montero, R. Vozmediano, and I. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," 2010, submitted for journal publication.

[14] D. Breitgand and A. Epstein, "SLA-aware Placement of Multi-Virtual Machine Elastic Services in Compute Clouds," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM'11)*, Dublin, Ireland, May 2011, Special Track on Management of Cloud Services and Infrastructures.