Provisioning of Requests for Virtual Machine Sets with Placement Constraints in IaaS Clouds

Lei Shi, Bernard Butler, Dmitri Botvich and Brendan Jennings
Telecommunications Software & Systems Group (TSSG),
Waterford Institute of Technology, Ireland
Email: {lshi, bbutler, dbotvich}@tssg.org, bjennings@ieee.org

***Abstract*—We study the problem of optimising the provisioning of collections of virtual machines (VMs) having different placement constraints (e.g., security and anti-collocation) and characteristics (e.g., memory and disk capacity), given a set of physical machines (PMs) with known specifications, in order to achieve the objective of maximising an IaaS cloud provider's revenue. We propose two approaches. The first is based on the formulation of the problem as an integer linear programming (ILP) problem, the solution to which provides an optimal VM placement. The second approach is a heuristic based on classifying the requests into different categories and satisfying the constraints in a particular order using a first fit decreasing (FFD) algorithm for multi-dimensional vector bin packing problem. Given a model of VM placement constraints, offered resources and requests with multiple VM types, both approaches devise a placement plan in a way that maximizes revenue, having due regard both to customer requirements and PM capacities. We evaluate the relative performance of the solutions by means of numerical experiments. The results suggest the optimal solution is not practical for medium to large problems, but it is encouraging that the placement plans of the heuristic are close to those of the optimal solution for smaller problem sizes. We use the heuristic to generate results for large scale placement problems; experiments suggest that it is practical in terms of its runtime efficiency and can provide an effective means of online VM-to-PM mapping.**

I. Introduction

Fostered by innovations in virtualization and distributed computing, and the increased availability of high-speed internet connectivity, cloud computing is becoming a general utility providing on-demand computing resource leases as transparent services to users. In a cloud data center multiple requests for sets of VMs need to be placed onto the PMs. Each request has a list of the number of different VM types, possibly along with one or more placement constraints. Requests for placement of VM sets vary depending on the cloud user and the applications they will use those VMs to realize. For example, requests for VM sets supporting services requiring high-level fault tolerance typically are associated with an anti-collocation placement constraint [1]; requests for data safety-sensitive services are often associated with hardware-based security related constraints [2]; requests for VM sets exploiting content-based memory page sharing are associated with collocation placement constraints [3]; requests for service realized by a number of interdependent VMs require full deployment [1]. Additionally, constraints can be set for performance optimisation; for example, Sudevalayam and Kulkarni [4] suggest affinity-aware placement in order to improve response/transfer times and reduce the load on network resources and CPU utilisation.

In most cases VM provisioning requests can be satisfied, however there are circumstances where insufficient resources are available. Given that different VM requests are typically associated with different revenue levels for the cloud operator, in those circumstances where not all requests can be satisfied it is reasonable to assume that the cloud provider will seek to maximize the total revenue of the VMs assigned to the PMs, whilst ensuring that the placement is feasible in respect of the PM capacity and placement constraints. On the other hand, when ample resources are available, the cloud provider is likely to maximize its revenue by seeking to consolidate VMs on to as few PMs as possible, so that PMs can be put into power consumption minimising modes—thereby reducing energy costs.

The majority of the published literature on VM provisioning in IaaS clouds focuses on particular aspects of placement optimisation, including: PM consolidation [5]–[7], migration costs [8], [9] and placement restrictions [1], [10], [11]. However, revenue maximisation, PM consolidation, multiple requests that each contains a list of the number of different VM types, and mixture of different placement constraints associated with these requests are rarely considered together. The solutions presented in this paper address these aspects together, taking into account a rich set of placement constraints for VM sets. Our solutions address the problem of predicting the optimized allocation of VM requests, which enables the cloud provider to have a better understanding of revenue and operational considerations *before* negotiating SLAs with groups of customers.

Given a typical mix of customer demands for virtualized computing resources, we propose two VM placement approaches. The first is based on the formulation of the problem as an Integer Linear Programming (ILP) problem. Solution of this ILP model produces optimal placements for VM requests having different placement constraints. The second approach is based on classifying the requests for VM placements into different categories and satisfying the constraints in a particular order using a first fit decreasing (FFD) algorithm for the multi-dimensional vector bin packing problem; this is thus a heuristic that will produce generally sub-optimal VM placement plans. In the paper the relative performance of the approaches is evaluated by means of numerical experiments.

The paper is organized as follows. §II presents a brief background of the VM placement problem and provides motivation for our work. §III specifies the problem we address and presents the ILP formulation. In §IV, we develop a heuristic to provide placement plans for problem sizes where optimal

978-3-901882-50-0

solution of the ILP problem in a reasonable timeframe would be infeasible. Subsequently, in §V, we present experiments that evaluate the VM placement plans achieved by our approaches for various VM requests settings. Also presented is our interpretation of the results with evidence to support this interpretation. Finally, we conclude the paper in §VI.

II. Related Work

Placement of VMs onto PMs is closely related to the vector bin packing problem, which can be used to model static resource allocation problems where the resources used by each item are additive in each dimension. The optimal placement plan is to pack the items (VMs) into a minimum number of bins (PMs) in such a way that the vector sum of the items received by any bin does not exceed its resource (typically CPU and memory capacity) limit. The vector bin packing problem and its variants are NP-hard problems [12], which means there is no polynomial time algorithms unless $P = NP$. Thus, heuristic algorithms are commonly used in practical applications. Most heuristics are based on greedy algorithms using simple rules, such as first fit and best fit.

Panigrahy *et al.* [13] systematically studied variants of the FFD algorithm and proposed a new geometric heuristic algorithm, which is scalable for larger data centers without a dramatic decrease in performance. This new heuristic algorithm runs nearly as fast as FFD for reasonable values of n and d, where n represents items and d is the length of the vector of resource types. Mills *et al.* [14] compared 18 algorithms for the VM placement in on-demand infrastructure clouds. These studies only consider the capacity constraints associated with the vector packing problem, but not integrate the incompatible constraints of the VMs into modelling the resource allocation.

Urgaonkar *et al.* [11] consider the problem of maximising the number of placed applications, where each application comprises a set of application components that should be placed in full on a cluster of PMs. They also considered an anti-collocation placement constraint since some applications might want their components to be placed on different nodes for fault tolerance. They demonstrate this placement problem with the full deployment and anti-collocation placement constraints might not have a Polynomial Time Approximation Scheme (PTAS) and propose efficient approximation algorithms. However, their work does not consider that the revenue from application components can vary due to diversified VM sizing and pricing, in which case maximising the *number* of placed applications does not guarantee that the cloud provider obtains maximum revenue.

When two applications cannot be assigned to the same target PM, item-item incompatibility constraints occur. Bin-item incompatibility constraints arise when a given application cannot be moved to a particular PM. For consolidating PMs with these constraint types, Gupta *et al.* [15] propose a placement solution with a two-stage heuristic algorithm. Tang *et al.* [16] present a online algorithm which dynamically deploys application instances to the PMs to adapt to the change of resource requirements. These works are based on the application rather than VM for the placement plan.

Xu *et al.* [17] formulate VM placement as a multi-objective combinatorial optimisation problem aiming to simultaneously optimize possibly conflicting objectives. The objectives include maximising resource usage, balancing thermal distribution, and minimising power consumption. In [18], Jiang *et al.* consider VM consolidation placement problem with the constraint of minimising the traffic cost in the data center. In [19] and [20], the VM placement problem is formulated as a stochastic bin packing problem characterized by physical resource oversubscription, using statistical multiplexing in order to improve consolidation of VMs. However, neither of [19], [20] take into account the characteristics of user requests and their consideration in terms of maximising revenue generated for the cloud provider.

Similar to us, Breitgand and Epstein [1] consider maximising the IaaS provider revenue from service provisioning obtained from the placed VMs, while respecting placement constraints and resource capacity constraints. A direct integer programming formulation is proposed to obtain the exact solutions and a column generation heuristic method is presented to obtain near optimal solutions for large data centers. However, [1] (and similarly, [11]) consider only full deployment and anti-collocation constraints for the VM instance sets comprising. Other placement constraints, notably the hardware-based security placement constraint, requests with multiple VM types and mixture of request constraints we discuss below, are not considered.

III. Integer Linear Programming Formulation

We consider the problem of placing requests for sets of VMs on PMs subject to one of a defined number of per-request constraints selected by the requester (cloud user). Let us assume that we have I available PMs and we wish to satisfy J requests for placement of VM sets. Let $i = 1, \ldots, I$ denote an arbitrary PM and let $\mathbf{v_j}$, where $j = 1, \ldots, J$, denote an arbitrary VM set placement request. Each request, $\mathbf{v_j}$, is a vector of k' VM specifications: $\mathbf{v_j} = \{v_j^1, v_j^2, \ldots, v_j^{k'}\}$, where $k' = 1, \ldots, K_{max}$ and K_{max} denotes the maximum number of VMs that a cloud user is allowed to request in a VM set. Let v_j^k, where $k = 1, \ldots, k'$, denote an arbitrary VM specification in $\mathbf{v_j}$.

We assume that all requests $\mathbf{v_j}$ can have associated with them one of the VM placement constraints listed below. If no placement constraint is specified then we term the request as being *Unconstrained*. We employ the following placement constraints:

- The *Full Deployment (FULL)* constraint means that all VMs $v_j^k \in \mathbf{v_j}$ must be placed, or none at all;
- The *Anti-colocation (ANTI)* constraint requires that, for fault tolerence purposes, all $v_j^k \in \mathbf{v_j}$ must be placed on different PMs;
- The *Security (SEC)* constraint requires that a PM i can only be assigned VMs v_j^k from the same request $\mathbf{v_j}$ and not be assigned any VMs from other requests.

Additionally, certain constraints can be applied together, specifically: *Security and Full Deployment (SEC_FULL)* and *Anti-colocation and Full Deployment (ANTI_FULL)*. We view the anti-colocation and security constraints as being mutually exclusive in practice—the only feasible solution for a request

including both is to have a single VM on a single PM. Whilst it is conceivable that a cloud user may make such a request we believe it would be very rare, so we do not consider this constraint combination. For a typical IaaS cloud we envisage that the cloud user mix would be such that requests associated with a mix of placement constraints (or none at all) can be expected.

Let us assume that each PM i has associated with it a D-dimensional vector $\mathbf{c_i} = \{c_i^1, c_i^2, \ldots, c_i^D\}$ representing the capacities of the PM's D resource types (CPU capacity, memory, network bandwidth, etc.) that can be assigned to VMs. Let $d = 1, \ldots, D$ denote an arbitrary PM resource type. We assume that VMs are assigned a fixed amount of such resources when placed on a PM, notwithstanding the potential for hypervisors to dynamically share unallocated resources between VMs [21], [22]. Let each $v_j^k \in \mathbf{v_j}$ have an associated D-dimensional resource requirement vector $\mathbf{u_j^k} = \{u_j^{k1}, u_j^{k1}, \ldots, u_j^{kD}\}$ specifying its resource requirements and have an associated revenue for the cloud provider of $r(v_j^k)$ per hour once it is placed on a PM. Let $x_i(v_j^k)$ indicate VM placement, such that $x_i(v_j^k) = 1$ if v_j^k is assigned to PM i and $x_i(v_j^k) = 0$ otherwise.

Given the above, the objective of our ILP formulation for the VM to PM placement problem is to maximize the revenue associated with the placement of VMs on PMs, subject to resource capacity constraints and to placement constraints (specified below):

$$max \quad \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{|\mathbf{v_j}|} x_i(v_j^k)\cdot r(v_j^k)$$

$$s.t. \quad \forall i \in \{1,\ldots,I\}, \forall j \in \{1,\ldots,J\}, \forall k \in \{1,\ldots,|\mathbf{v_j}|\}:$$
$$x_i(v_j^k) \in \{0,1\}$$

The resource capacity constraint is specified as follows:

$$\forall d \in \{1,\ldots,D\}, \forall i \in \{1,\ldots,I\}:$$
$$\sum_{j=1}^{J}\sum_{k=1}^{|\mathbf{v_j}|} x_i(v_j^k)\cdot u_j^{kd} \leq c_i^d$$

We now specify the three placement constraints, starting with *Full Deployment (FULL)*. This constraint ensures that all VM specified in a request must be placed; if this is not possible then none of them will be placed. Let $\mathbf{v_F} \subseteq \{\mathbf{v_j}\}$ denote the subset of the J requests to which the FULL placement constraint applies. Let y_{v_j} be an indicator variable assuming 1 if the whole request v_j is included into the placement and 0 otherwise. Assigning the value of y_{v_j} to 1 means all v_j^k can be found on one of the PMs; otherwise y_{v_j} is 0, ensuring none of VMs in request v_j is placed. The constraints can be expressed as:

$$\forall \mathbf{v_j} \in \mathbf{v_F}, \forall k \in \{1,\cdots,|v_j|\}:$$
$$y_{v_j} = \sum_{i=1}^{I} x_i(v_j^k)$$

$$y_{v_j} \in \{0,1\}$$

The *Anti-colocation (ANTI)* placement constraint ensures that the VMs for a given request cannot be placed on PMs that have other VMs from the same request placed on them. In the event of a PM failure this will mean that a degree of fault tolerance is introduced, wherein VMs placed on other PMs can handle the application workloads for the failed VM. Let $\mathbf{v_A} \subseteq \{\mathbf{v_j}\}$ denote the subset of the J requests to which the ANTI placement constraint applies. The constraint can then be expressed as:

$$\forall \mathbf{v_j} \in \mathbf{v_A}, \forall i \in \{1,\ldots,I\}:$$
$$\sum_{k=1}^{|\mathbf{v_j}|} x_i(v_j^k) \leq 1$$

Requests can also be associated with the *Security (SEC)* constraint which ensures that VMs belonging to the same request must be placed on the same PM. Furthermore, even if that PM is not fully utilized, its remaining capacity cannot be used for placement of VMs from other requests. By calculating the sum of VMs on a given PM for two different VM requests (the result will be one if the sum of VMs for a specific VM request is equal or greater than one, otherwise zero), and limiting their summation to one, the security placement constraint can be satisfied on this PM. Let $\mathbf{v_S} \subseteq \{\mathbf{v_j}\}$ denote the subset of the J requests to which the Security placement constraint applies. Let $z_i(\mathbf{v_j})$ be an indicator variable, assuming the value 1 if any VMs from request $\mathbf{v_j}$ are placed on PM i, and 0 otherwise. The constraint can then be expressed as:

$$\forall \mathbf{v_j} \in \mathbf{v_S}, \forall i \in \{1,\ldots,I\}:$$
$$z_i(\mathbf{v_j}) = \begin{cases} 1 & \text{if } \sum_{k=1}^{|\mathbf{v_j}|} x_i(v_j^k) > 0 \\ 0 & \text{if } \sum_{k=1}^{|\mathbf{v_j}|} x_i(v_j^k) = 0 \end{cases}$$
$$z_i(\mathbf{v_j}) + \sum_{j'=1, j' \neq j}^{J} z_i(\mathbf{v_{j'}}) \leq 1$$

For requests with SEC_FULL constrains, the mathematical formulation is the formulation combination of SEC and FULL requests; for requests with ANTI_FULL constrains, the formulation is that of ANTI and FULL requests, respectively.

IV. Heuristic Solution

For our problem with placement constraints, although the ILP formulation yields an optimal solution, it is not appropriate for runtime placement of newly arrived VM requests. It might take several hours (using commercial optimisation software without parameter configurations), or even days, to produce a placement solution even for a relatively small size problem. Therefore, a fast, even if sub-optimal, approach to compute the

placement is required for online placement of newly arriving requests for provisioning of VM sets.

We divide the placement constraints to three categories. The first two categories are requests with SEC and SEC_FULL constraints respectively. The intuition behind is that security constraints prohibit the requests from sharing PMs with other requests. The third category contains ANTI_FULL, ANTI, FULL and unconstrained requests, which can co-exist with each other. Given the constraint types, the requests in this category are sorted in the order of ANTI_FULL, ANTI, FULL and unconstrained, so that the ordering scheme deals with the stricter conditions first, with unconstrained requests last, as they are the easiest to satisfy.

The heuristic algorithm is outlined in Alg. 1. The inputs to the algorithm are the set of PMs i and $\mathbf{v_S} \subseteq \{\mathbf{v_j}\}$, $\mathbf{v_{SF}} \subseteq \{\mathbf{v_j}\}$, $\mathbf{v_{AF}} \subseteq \{\mathbf{v_j}\}$, $\mathbf{v_A} \subseteq \{\mathbf{v_j}\}$, $\mathbf{v_F} \subseteq \{\mathbf{v_j}\}$, denoting the subset of the J requests to which the SEC, SEC_FULL, ANTI_FULL, ANTI, FULL respectively apply, together with $\mathbf{v_U} \subseteq \{\mathbf{v_j}\}$, denoting the subset of unconstrained requests. We initially sort the requests in each of these subsets in descending order of the number of VMs specified in each request (line 1).

For each category of requests, different allocation strategies are used to generate PM groups, where a PM group is the smallest set of PMs necessary to place interdependent VMs. We denote a PM group as p and use $P = \{p\}$ to denote a set of PM groups. In the algorithm we initially set $P = \emptyset$ (line 2) and use it to accumulate PM groups for the final placement plan. The size of the PM group depends on the request's constraint (if any). For example, for requests with SEC constraints the size of PM group is 1, because a minimum of 1 PM is needed to satisfy this constraint. In contrast, for requests with the ANTI_FULL constraint the size of the PM group is the number of VMs specified in the request, as all must be placed on different PMs. For SEC and SEC_FULL requests, we execute the FFD algorithm to obtain the allocation mapping from VMs to the PMs for each request (lines 4-13). For each request, we start with new empty PMs for allocating the VMs in this request. The time complexity of FFD algorithm is $O(|\mathbf{v_j}| \log |\mathbf{v_j}|)$ [23], where $|\mathbf{v_j}|$ is the number of VMs to be allocated in this request.

Requests with ANTI_FULL, ANTI, and FULL placement constraints, as well as unconstrained requests, can co-exist on PMs, so we process them together (lines 14-33). In the algorithm we use $\mathbf{M} \in \mathbb{N}^{I \times n}$ to denote an I by n matrix, where I is the number of PMs and $n = |\mathbf{v_A} \cup \mathbf{v_{AF}}|$ is the number of requests with ANTI and ANTI_FULL placement constraints. If a VM belonging to a request with ANTI or ANTI_FULL placement constraints is allocated to a PM, then the corresponding element of $\mathbf{M}$ is set to 1, which indicates that the PM is not available for placement of any other VMs for the same request. All VMs in each request are sorted in decreasing order based on the length of the VM resource vector, where the resource vector is the vector sum of the normalized resource requirements of the VM. For each request, all the VMs specified in the request are checked to see if they can be allocated to available PMs, which are ordered in decent order in terms of residual resource capacity. If the placement constraint of a request is ANTI_FULL or FULL and the request is not fully allocated, then $\mathbf{M}$ and residual PM capacities are reset to the previous values; otherwise we calculate the residual capacities of PMs, rank them in descending order and process the next request.

Algorithm 1 Heuristic for VM Placement with Constraints

Input: $\{i\}$, $\mathbf{v_S}$, $\mathbf{v_{SF}}$, $\mathbf{v_{AF}}$, $\mathbf{v_A}$, $\mathbf{v_F}$, $\mathbf{v_U}$
Output: placement plan

1: Sort $\mathbf{v_S}$, $\mathbf{v_{SF}}$, $\mathbf{v_{AF}}$, $\mathbf{v_A}$, $\mathbf{v_F}$, and $\mathbf{v_U}$ in descending order of number of VM specifications included in the requests.
2: Set $P = \emptyset$
3: ▷ Process SEC requests
4: **for** each request $\mathbf{v_j} \in \mathbf{v_S}$ **do**
5: Run FFD algorithm to obtain the allocation
6: Set $p = \{i'\}$ where $i' \in \{i\}$ is any PM to which VMs have been allocated
7: Set $P = P \cup p$
8: **end for**
9: ▷ Process SEC_FULL requests
9: **for** each request $\mathbf{v_j} \in \mathbf{v_{SF}}$ **do**
10: Run FFD algorithm to obtain the allocation
11: Set $p = \{i''\}$ where $i'' \in \{i\}$ is the PM Group to which all VMs in request $\mathbf{v_j}$ have been allocated
12: Set $P = P \cup p$
13: **end for**
14: ▷ Process all other requests
14: $\mathbf{M} = \emptyset$
15: **for** each request $\mathbf{v_j} \in \mathbf{v_A} \cup \mathbf{v_{AF}} \cup \mathbf{v_F}$ **do**
16: **for** $k = (1, \ldots, |\mathbf{v_j}|)$ **do**
17: **for** $i = (1, \ldots, |\{i\}|)$ **do**
18: **if** v_j^k fits in PM i and $\mathbf{v_j} \in \mathbf{v_{AF}} \cup \mathbf{v_A}$ and $\mathbf{M}(v_j^k, i) = 0$ **then**
19: Allocate VM v_j^k to PM i
20: Set $\mathbf{M}(v_j^k, i) = 1$
21: Continue
22: **else if** v_j^k fits in PM i and $\mathbf{v_j} \in \mathbf{v_F} \cup \mathbf{v_U}$ **then**
23: Allocate VM v_j^k to PM i
24: **end if**
25: **end for**
26: **end for**
27: **if** $\mathbf{v_j} \in \mathbf{v_{AF}} \cup \mathbf{v_F}$ and $\mathbf{v_j}$ is not fully allocated **then**
28: Reset $\mathbf{M}$, the allocation mapping and residual PM capacities.
29: **else**
30: Recalculate the PM residual capacities
31: Rank PMs in descending order of residual capacities
32: **end if**
33: **end for**
34: Construct a spanning forest based on the allocation mapping using the Reverse-Delete Algorithm to obtain PM groups p' from unconnected spanning trees
35: Set $P = P \cup p'$ ▷ Compute placement plan
36: Sort P in descending order of revenue generated by VMs allocated to each $p \in P$
37: Given the PMs $\{i\}$, run the FFD algorithm on the sorted P to obtain the final VM to PM placement plan

In lines 34-35, the Reverse-Delete Algorithm [24] is used to find the collection of interrelated PMs because of the full placement constraints and merge them into P. This algorithm is originally designed to find the minimum spanning tree,

TABLE I: Experimental Test Specifications

Test Number	VM Range	Requests Range	Constraint
1	1-6	1-3	FULL
	7–16	4-6	SEC
	17–26	7-10	ANTI
	27–37	11-14	SEC_FULL
	38–43	15-17	ANTI_FULL
	44–102	18-33	Unconstrained
2	1-6	1-2	FULL
	7–13	3-4	SEC
	14–17	5-6	ANTI
	18–23	7-8	SEC_FULL
	24–27	9-10	ANTI_FULL
	28–31	11-12	Unconstrained
3	1-8	1-3	FULL
	9–21	4-6	SEC
	22–31	7-9	ANTI
	32–42	10-12	SEC_FULL
	43–54	13-15	ANTI_FULL
	55–65	16-18	Unconstrained
4	1-1276	1-41	mixed
5	1-1638	1-51	mixed
6	1-1472	1-45	mixed
7	1-1775	1-55	mixed
8	1-1651	1-52	mixed

which is the subgraph of the original graph connecting all the vertexes originally connected with minimum cost, for each disconnected part of the graph. By applying the definition of spanning tree to disconnected graph, we obtain the spanning forest, which in our case is the collection of interrelated PMs because of the full placement constraints. The runtime of the algorithm is $O(|E|logI(loglogI)^3)$ [25], where $|E|$ is the number of edges connecting the interdependent PMs because of full placement constraints and I is the number of PMs.

Finally, we rank the PM groups obtained from all three categories according to the revenue generated by the groups. Given the available number of PMs the cloud, we run FFD algorithm on the PM groups and devise the placement plan in lines 36-37.

V. Performance Evaluation

In this section we report on numerical experiments used to assess the performance of the ILP and heuristic solutions. We investigate eight different test cases that illustrate the effectiveness and scalability of the two approaches.

A. Implementation and Test Cases

The ILP formulation is programmed in AMPL [26] language and is inputted to the optimisation software Gurobi [27] for solution. We also developed a problem instance generator that focuses on those features of the placement problem that are most important for revenue modelling. The heuristic algorithm is implemented using Matlab.

Table I summarizes the eight test cases we used to investigate the usefulness and performance of the ILP approach and heuristic algorithm. For all experiments, the specification of the PMs is homogeneous: 24 processors (each equivalent to 1 Amazon ECU) with each PM having 32GB of memory.

TABLE II: Amazon EC2 VM types (from [2])

VM Type	*Processors* (ECU)	*Memory* (GB)	*Price* ($)
Standard Large	4	7.5	0.34
Standard XLarge	8	15	0.68
Micro	2	0.6	0.02
iMem XLarge	6.5	17.1	0.50
HiCPU Medium	5	1.7	0.17
HiCPU XLarge	20	7	0.68

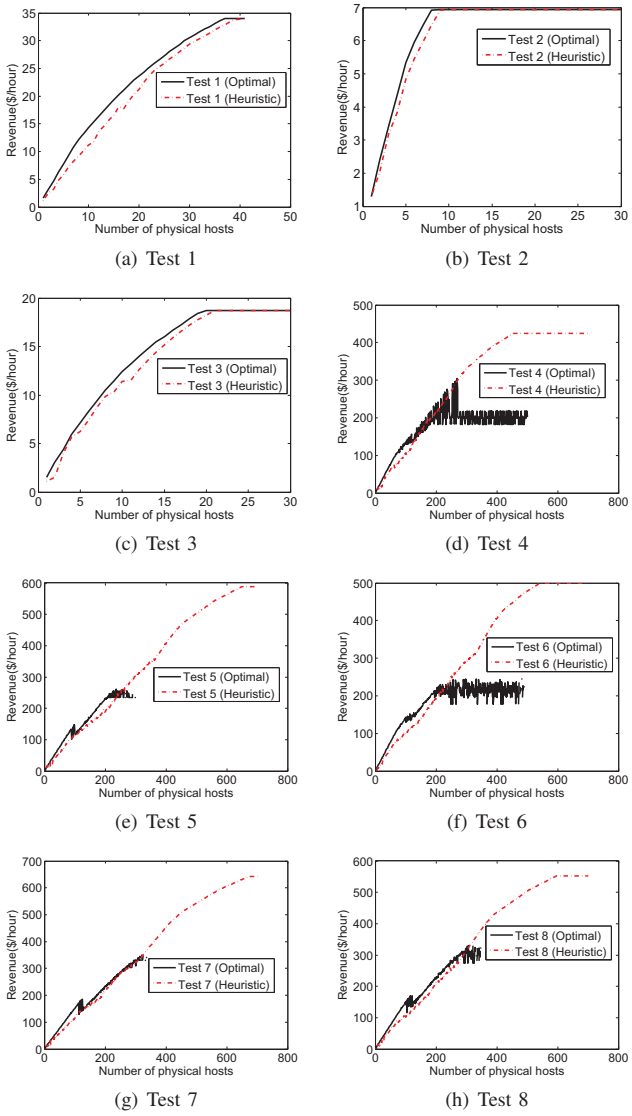
All VMs are drawn from Amazon EC2 VM instance types, as outlined in Table II. Note that each VM type requires different resources such as number of processors or memory capacity. The price per hour depends on the resources that are packaged up and offered by each VM type. The experiments form two main groups. In experiments 1-3, VM requests and the multiple VM types within each single request are generated by a combinatorial auction algorithm CATS [28]. Test 1 has the most number of requests and Test 3 has an intermediate number of requests. In Tests 4-8, instead of using the combinatorial auction algorithm, the VM ranges and types are randomly assigned to each request. For all experiments, the placement constraints are randomly generated.

The runtime of each numerical experiment depends on the problem size, as quantified by number of requests, request placement constraints, number of VMs to be placed and number of VM types, as well as number of PMs available. Generally, the experiment "sizes" of Tests 1-3 were chosen in such a way that experiment run-times for ILP approach were considered acceptable. However, there were some runs requires excessive time, we limit the run time of each placement for Tests 1-3 to 10 minutes. For Tests 4-8, the test "size" has reasonable number of VM requests for a medium size cloud data center. We target the scalability aspects of the placement problem and increase the number of VMs tested up to 1775. Due to the increased complexity of the VM placement program with the large scale input, we limit the run time of each placement for Tests 4-8 to 5 minutes for ILP approach.

B. Analysis of Results

Fig. 1 shows predicted revenue as a function of PM set size. For Tests 1-3, as expected, the revenue saturates at a constant level. The other main feature is the linear increase to the saturation level. The slope depends on the number of VMs and the weighted price of those VMs; the placement constraints seem to have little effect. The revenue generated by the heuristic algorithm is very close to the optimal value by the ILP approach. However, due to the computation time limit on large scale requests input, for Tests 4-8 the ILP approach can only produce a feasible but not optimal solution; in some cases, it even can not produce a feasible solution. The heuristic algorithm can always generate a feasible solution in few seconds, which shows the scalability of the algorithm for the medium or large data centers.

The VM placements are illustrated visually in Fig. 2, which shows which VMs are placed for each of a range of PM set sizes in increasing order. For clarity, we show the results of test conditions 1 and 7 only. The results of the other tests exhibit similar features. Reading from left to right, more VMs are



(a) Test 1 (b) Test 2

(c) Test 3 (d) Test 4

(e) Test 5 (f) Test 6

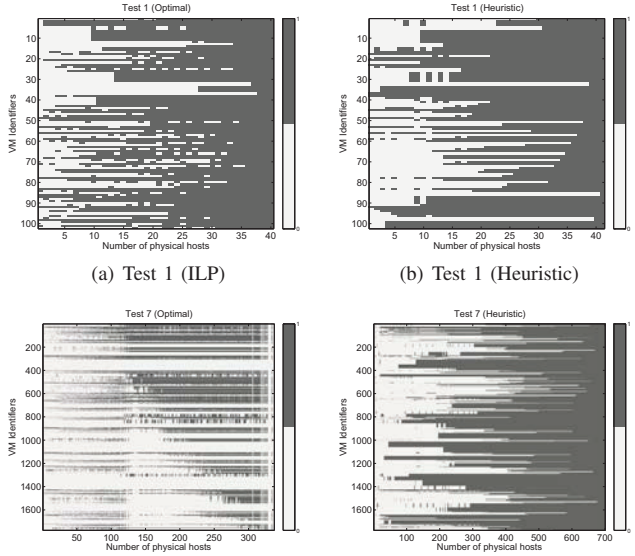
(g) Test 7 (h) Test 8

Fig. 1: Revenue accruing from placement of VMs on PMs for different experiments for the ILP and heuristic approaches. The heuristic is seen to closely approximate the revenue generated from optimal placement for small problem sizes. For medium to large scale problems the ILP approach typically cannot produce an optimal solution within our computation time limits.

placed as the number of PMs increases, as might be expected. However, there is a point at which, for a given level of VM demand, all VM requirements can be satisfied. After that point, additional PMs are not utilized and so the revenue generated does not change. In such cases any unused PMs can be placed in a power saving mode in order to reduce energy costs.

VI. Conclusion and Future Work

We present a ILP model and a heuristic algorithm for VM placement problem that enables cloud service providers to perform numerical simulations of different VM demand



(a) Test 1 (ILP) (b) Test 1 (Heuristic)

(c) Test 7 (ILP) (d) Test 7 (Heuristic)

Fig. 2: Mappings of VMs to PMs for different experiments for both approaches. In the figures 1 (darkly shaded) means that the VM in question is placed on a PM, whereas a 0 (lightly shaded) means that it has not been placed. We see that as the number of available PMs increases both approaches succeed in satisfying a larger proportion of the VM requests.

scenarios. The scenarios can include complex placement constraints and VM requests with multiple VM types. One main contribution is a comprehensive ILP model dealing with a wide variety of placement constraints and VM types. The model has been used in numerical experiments to address scenarios in which possibly conflicting placement constraints are specified independently by customers. The other contribution is the heuristic algorithm which derives placement plans comparative to those produced via the ILP model, but which is scalable to large data centers and suitable for use in an online VM placement process. The results show that having a variety of VM types and placement constraints can have a dramatic effect on VM placements, but interestingly, relatively little effect on predicted revenue. That is, such complicated scenarios present operational difficulties, but are not critical when the cloud provider prepares a business model.

There are several interesting future research directions indicated by our work. First, it would be interesting to apply the method to more general VM allocation scenarios that include (once-off) *reallocation costs* in addition to (ongoing) *allocation revenue*. That is, the objective changes to maximizing the profit at each step, rather than the revenue. Second, if reallocation costs are included in the objective function, it is also possible to consider trade-offs between obtaining the optimal allocation at a given instant and obtaining a temporally sub-optimal solution that increases profit in the long run.

Acknowledgement

This work has been funded by Science Foundation Ireland, via the "FAME" Strategic Research Cluster.

REFERENCES

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

[2] Amazon, "Amazon EC2 Dedicated Instances." [Online]. Available: http://aws.amazon.com/dedicated-instances/

[3] T. Wood, G. Tarasuk-Levin, P. J. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory buddies: exploiting page sharing for smart colocation in virtualized data centers," in *VEE*, 2009, pp. 31–40.

[4] S. Sudevalayam and P. Kulkarni, "Affinity-Aware Modeling of CPU Usage for Provisioning Virtualized Applications," *2011 IEEE 4th International Conference on Cloud Computing*, pp. 139–146, Jul. 2011.

[5] S. Mehta and A. Neogi, "Recon: A tool to recommend dynamic server consolidation in multi-cluster data centers," in *NOMS*, 2008, pp. 363–370.

[6] S. Khuller, J. Li, and B. Saha, "Energy efficient scheduling via partial shutdown," in *SODA*, 2010, pp. 1360–1372.

[7] S.Lee, R.Panigrahy, V.Prabhakaran, V.Ramasubrahmanian, K.Talwar, L.Uyeda, and U.Wieder, "Validating heuristics for Virtual Machine Consolidation," Microsoft Research, Tech. Rep. MSR-TR-2011-9, January 2011.

[8] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Computer Networks*, vol. 53, no. 17, pp. 2923–2938, 2009.

[9] A. Murtazaev and S. Oh, "Sercon: Server consolidation algorithm using live migration of virtual machines for green computing," *IETE Tech Rev*, vol. 28, pp. 212–31, 2011.

[10] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM*, 2010, pp. 1154–1162.

[11] B. Urgaonkar, A. Rosenberg, and P. Shenoy, "Application placement on a cluster of servers," *International Journal of Foundations of Computer Science*, vol. 18, no. 5, pp. 1023–1041, 2007.

[12] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "Approximation algorithms for np-hard problems," D. S. Hochbaum, Ed. Boston, MA, USA: PWS Publishing Co., 1997, ch. Approximation algorithms for bin packing: a survey, pp. 46–93.

[13] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," Microsoft Research, Tech. Rep., 2011.

[14] K. Mills, J. Filliben, and C. Dabrowski, "Comparing VM-Placement Algorithms for On-Demand Clouds," *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pp. 91–98, Nov. 2011.

[15] R. Gupta, S. K. Bose, S. Sundarrajan, M. Chebiyam, and A. Chakrabarti, "A Two Stage Heuristic Algorithm for Solving the Server Consolidation Problem with Item-Item and Bin-Item Incompatibility Constraints," *2008 IEEE International Conference on Services Computing*, pp. 39–46, Jul. 2008.

[16] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," *Proceedings of the 16th international conference on World Wide Web - WWW '07*, p. 331, 2007.

[17] J. Xu and J. a. B. Fortes, "Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments," *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pp. 179–188, Dec. 2010.

[18] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *INFOCOM*, 2012, pp. 2876–2880.

[19] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective VM sizing in virtualized data centers," *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pp. 594–601, May 2011.

[20] D. Breitgand and A. Epstein, "Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds," *2012 Proceedings IEEE INFOCOM*, pp. 2861–2865, Mar. 2012.

[21] M. Cardosa, M. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on*, Jun. 2009, pp. 327 –334.

[22] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "VMware distributed resource management: Design, implementation, and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, pp. 45–64, Mar. 2012.

[23] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*. Wiley-Interscience, Dec. 1999.

[24] J. M. Kleinberg and É. Tardos, *Algorithm design*. Addison-Wesley, 2006.

[25] M. Thorup, "Near-optimal fully-dynamic graph connectivity," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, ser. STOC '00. New York, NY, USA: ACM, 2000, pp. 343–350.

[26] AMPL. [Online]. Available: http://www.ampl.com/

[27] GUROBI. [Online]. Available: http://www.gurobi.com/

[28] K. Leyton-Brown, M. Pearson, and Y. Shoham, "Towards a universal test suite for combinatorial auction algorithms," in *ACM Conference on Electronic Commerce*, 2000, pp. 66–76.