Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-aware Virtual Machine Placement

Deepal Jayasinghe, Calton Pu
CERCS, Georgia Institute of Technology
266 Ferst Drive, Atlanta, GA 30332-0765, USA.
{deepal, calton}@cc.gatech.edu

Tamar Eilam, Malgorzata Steinder, Ian Whalley, Ed Snible
IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532, USA.
{eilamt, steinder, inw, snible}@us.ibm.com

***Abstract*—The increasing popularity of modern virtualization-based datacenters continues to motivate both industry and academia to provide answers to a large variety of new and challenging questions. In this paper we aim to answer focusing on one such question: how to improve performance and availability of services hosted on IaaS clouds. Our system, structural constraint-aware virtual machine placement (SCAVP), supports three types of constraints: demand, communication and availability. We formulate SCAVP as an optimization problem and show its hardness. We design a hierarchical placement approach with four approximation algorithms that efficiently solves the SCAVP problem for large problem sizes. We provide a formal model for the application (to better understand structural constraints) and the datacenter (to effectively capture capabilities), and use the two models as inputs to the placement problem. We evaluate SCAVP in a simulated environment to illustrate the efficiency and importance of the proposed approach.**

***Keywords*-Availability; Clouds; Datacenter; IaaS; Optimization; Performance; Structural Constraints; VM Placement**

I. Introduction

Virtual machine (VM) placement has become a crucial issue in modern virtualization-based datacenters and has attracted significant attention recently [1]–[6], [9], [10], [12], [15], [22], [23]. However, due to associated complexity, most existing techniques focus on the placement of individual VMs and ignore the structural information of the application [2]–[4]. Yet, many services deployed in a cloud consist of several VMs, which may be clustered for availability or form different tiers of a multi-tier application. The performance and availability of such applications greatly depend on how its components (that is to say, the VMs) are placed relative to one another in the datacenter.

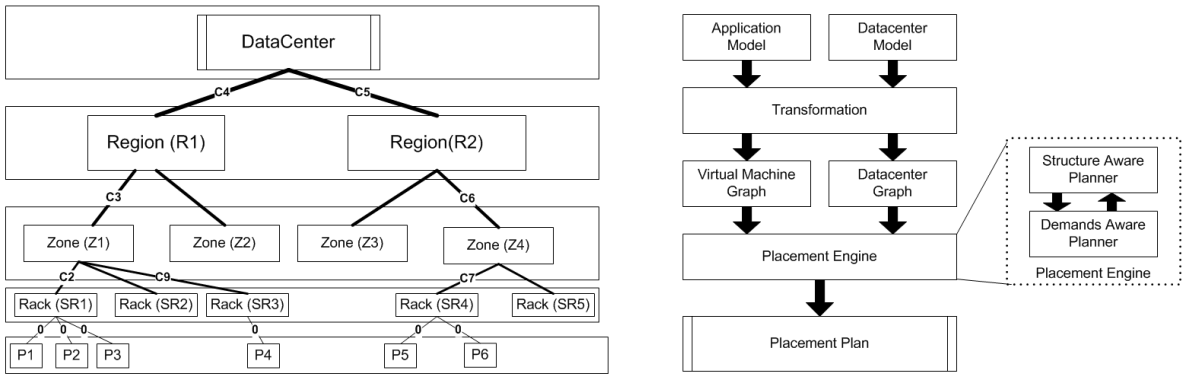
In this paper we present structural constraint-aware VM placement (SCAVP) to improve the performance and availability of services deployed in Infrastructure as a Service (IaaS) clouds. In SCAVP, we consider an application (we use 'application' and 'service' interchangeably) as a cluster of ***related*** VMs, rather than a set of individual VMs. By augmenting the placement system with awareness of availability and communication requirements (such systems are typically already aware of demand requirements), we can improve the overall performance and availability of the application.

When deploying an application, SCAVP attempts to satisfy three types of constraints: demand, availability, and communication. Demand satisfaction means providing resources (e.g., # of CPU cores, memory) to the service sufficient for it to meet its service level agreement (SLA). Observing availability constraints permit the overall availability of the application to be improved, perhaps by deploying VMs across different isolation levels of the datacenter. Finally, communication constraints permit us to minimize overall communication costs by assigning VMs with large mutual communication costs to PMs in close proximity to one another. This communication cost minimization has significant economic benefits on modern commercial clouds. For example, in Amazon EC2, the cost of data transfer to and from the Internet is \$0.08-\$0.15 per GB, whereas data transfer within the region costs only \$0.01 per GB. Thus, placing VMs intelligently so as to reduce communication costs can affect not only performance, but also the pecuniary cost of running the system.

We formulate SCAVP as an optimization problem and show hardness; more concretely, we show that SCAVP consists of three different optimization problems: a) VM clustering to optimize communication cost, b) VM clusters to server rack mapping, and c) VMs to PMs mapping. We propose a hierarchical placement approach that efficiently solves the VM placement problem for large problem sizes. Methodologically, we use Vespa [1]—an existing conventional placement approach—and extend it to support structural information of the application. To better analyze the structural information, we formally model the application, and to effectively capture capabilities we provide a formal model for the datacenter. We then use the two models as inputs to the placement problem. Finally, we evaluate SCAVP in a simulated environment to illustrate the efficiency and importance of the proposed approach.

The remainder of this paper is structured as follows. In Section II we formally model the datacenter and application and formalize the placement problem. Our hierarchical placement approach is discussed in Section III, and in Section IV we present our algorithms and discuss the complexity. Section V provides our experiment results. Related work is summarized in Section VI, and finally we conclude in Section VII.

978-0-7695-4462-5/11 \$26.00 © 2011 IEEE
DOI 10.1109/SCC.2011.28



(a) Hierarchical datacenter model

(b) Hierarchical placement process

Fig. 1. System architecture - Structural constraint-aware VM placement

II. Problem Formulation

The core of the VM placement problem is to create a placement plan which dictates where VMs should be deployed (that is to say, to which physical machines (PMs)) in order to satisfy various constraints. For brevity, we simply refer to "VM placement" as "placement" in the rest of the paper. The inputs to a placement problem are an application (i.e., a set of related VMs) and a datacenter (i.e., a set of PMs).

Placement can be further subdivided into 'initial VM placement' and 'runtime VM placement'. The initial VM placement problem is to find the most suitable PMs to deploy the application (i.e., VMs) for the very first time; in contrast, the runtime VM placement problem is to manage the system after initial deployment, including migrating VMs from one PM to another in order to adjust system utilization, improve performance, and reduce the power cost (amongst other possible goals). Without loss of generality, in this paper we focus on creating an efficient initial placement plan by satisfying the following three structural constraints:

Demand constraint
: is defined as the lower bound of resource allocations that each VM requires for the service to meet its SLA. Each VM is given a set of projected resource demands (e.g., CPU, memory, #cores); the placement plan should provide the requested demands.

Availability constraint
: is to improve the overall availability of the service. The availability of a service is expressed as a combination of anti-collocation/collocation constraints; the placement plan MUST respect these constraints and produce a plan with maximum service availability.

Communication constraint
: is a user-assigned value that represents the communication requirement between two VMs such that the application functions. The value is higher when the communication between the pair of VMs requires more bandwidth or lower latency. Our objective is to minimize the communication cost while satisfying both demand and availability constraints.

A. Datacenter Modeling

We model the datacenter as a tree structure with arbitrary depth, and we use both physical network topology (e.g., switches, routers) and logical groupings of the datacenter (e.g., power zones, security zones, user defined zones) as key parameters. A typical datacenter can be modeled using three or four levels—an example is shown in Fig 1(a).

The bottom-most level in the tree is the *Server racks* (e.g., SR1, SR2); we then group a set of server racks into a *Zone* (e.g., Z1, Z2). A zone can be a security zone, a power zone or a zone defined through some other means. The communication cost between two PMs in the same rack is zero while the communication cost between two PMs on two different zones is non zero, and is the same for any two PMs in respectively the same two zones. Similarly we group a set of zones to a *Region* (e.g., R1, R2); regions may, for example, be geographically distributed and more isolated from one another and therefore incur significantly higher communication costs.

As stated before we assume the communication cost between any two PMs in the same rack as zero, and that communication costs increase as we move up the hierarchy (i.e., `0 < C2 < C3 < C4` for Fig 1(a)). The communication cost between two PMs in two server racks can be calculated either by counting the switches between the racks or by measuring the end-to-end communication cost (e.g., latency or bandwidth) between the racks. For example, in Fig 1(a) the communication cost between *P1* and *P4* becomes `C2 + C9`. Next, we formalize different entities in the datacenter model:

Definition 1. *A physical machine (PM) exists only at the base of the datacenter hierarchy and is given by* $P = (N, \vec{PC}, SR)$*, where* N *is the name (an arbitrary value unique within* SR*),* $\vec{PC}$ *is a capability vector, and* SR *is the parent server rack.*

Definition 2. *A capability is a property of a PM and is given by PC = (N, V), where* N *is the name (e.g., CPU speed, memory) and* V *is the value (e.g., 1.7 GHz, 4GB).*

Definition 3. *Capacity* S *of an entity* E *is defined as the maximum number of VMs that can be placed on* E*, where* E *is any intermediate level in the datacenter hierarchy. Note, At*

any level, S *is calculated using the aggregated capacity of* E*'s children. Additionally we defined average capacity* $\sum S_i/n$ *as the average number of VMs that can be placed on a server rack, where* S_i *is the capacity of* ith *server rack and* n *is the total number of server racks.*

Definition 4. *The* i^{th} *intermediate level of a datacenter is given by* L_i *= (N,* C_i*,* L_{i+1}*, S), where* N *is the name,* C_i *is the communication cost to parent node,* L_{i+1} *is the set of children nodes, and* S *is the capacity at level* i*.*

Definition 5. *Server Rack (*SR*) is a special instance of intermediate level (bottom most), and here* L_{i+1} *is the set of all PMs* $\{P\}$ *in* SR *and* S *is the total number of VMs that* SR *can support. Moreover,* $\forall p \in \{P\}$ *is homogeneous with respect to* P*'s type (e.g., operating system) and communication cost between any two PMs in* SR *is zero.*

B. Application Modeling

The main objectives of application modeling are to effectively convert domain specific concepts (e.g., high availability clusters, primary-backup clusters) into structural constraints among application components and to permit a better understanding of resource demands. A typical application consists of sets of servers (e.g., Web servers, Application Servers), each of which has constraints and demands. In virtualization, common practice is to package application components as VMs and place VMs on PMs. In practice, more than one application components can be packaged into one VMs, for example if an application consists of *x* components packaged on *y* VMs, then *y* can be smaller than *x*. Nevertheless, from our modeling point of view, we will not know about individual application components, rather we only consider about the VMs, which will be associated with the aggregated demand and constraints of application components. Next we formally define different entities used in the application model.

Definition 6. *A virtual machine (VM) is given by* V *= (N,* $\vec{R}$*), where* N *is the name and* $\vec{R}$ *is the set of projected resource demands. Furthermore,* $\vec{R}$ *must be a subset of the physical machine's capability vector* PC *(defined in Def 1, 2).*

Definition 7. *A constraint* SC *is referred as a link between a pair of VMs, and the link must be associated with at least one constraint. A constraint can be either an availability constraint or a communication constraint.*

Definition 8. *A communication constraint is given by CC = (S, T, C), where* C *is the communication cost between source VM* S *and target VM* T*. Notice that when the cost is zero, the VMs are considered as non-communicating.*

Definition 9. *Availability constraints are a tuple A = (S, T, L, B), where* S *is the source VM,* T *is the target VM,* L *is a level in the datacenter hierarchy (*$L \in \cup L_i$*), and* $B \in \{colloc, anticolloc\}$ *is the constraint type.* $B = colloc$ *(i.e., collocation) implies that* S *and* T *must be placed on the same* L *level node, and* $B = anticolloc$ *(i.e., anti-collocation) implies* S *and* T *must be placed on two different* L *level nodes. Note, collocation and anti-collocation have the following properties:*

- Collocation is transitive: if $A \rightarrow B$, and $B \rightarrow C$, then $A \rightarrow C$
- Anti-collocation is not transitive: if $A \rightarrow B$, and $B \rightarrow C$, this does not imply $A \rightarrow C$

Definition 10. *An application is given by an un-directed graph A = (*$\{V\}$*,* $\{SC\}$*), where* $\{V\}$ *is a set of VMs and* $\{SC\}$ *is set of structural constraints that connects VMs with each other.*

Problem Statement: For a given arbitrary application *A=(*$\{V\}$*,* $\{SC\}$*)* and a datacenter *D*, find A's VM placement plan on *D* whereby the VMs in $\{V\}$ are assigned to PMs in *D*. The created placement plan should provide projected resource demands ($\sum \vec{R}$), satisfy structural constraints ($\sum SC$), and minimize the total communication cost ($\sum CC$).

III. Placement Process

We have designed a hierarchical placement approach to find the placement plan for a given application on a given datacenter, and our approach is illustrated in Fig 1(b). Firstly, we model both the datacenter and the application, as described above. Secondly, the application model is transformed into a VM graph and the datacenter specification is transformed to a datacenter graph. Finally, these two graphs are provided to the placement engine as input parameters.

The placement engine is the core of SCAVP and consists of two key sub-planners: the *Structure-Aware Planner (SAP)*, and the *Demands-Aware Planner (DAP)*. The *SAP* handles the structural constraints of the application and the *DAP* handles the resource constraints.

The *SAP* uses the structural constraints of the application and produces a candidate placement plan, which the *DAP* then refines. Methodologically, SCAVP uses a *divide-and-conquer* approach to create the placement plan. In the *divide* stage, the application graph is split into a set of smaller VM groups, and the group size is determined by the average capacity of a server rack. In this process, only the structural constraints are used: the demand constraints are ignored. Next, for each VM group, the *SAP* finds a candidate server rack and refines it using the *DAP*. In this process the *DAP* only uses demand constraints and ignores the other two constraints. *DAP* can safely ignore other constraints since each instance of the algorithm is applied to one VM group and one server rack, and *SAP* guarantees that there are no anti-collocation constraints between VMs in one VM group.

The *DAP* is a conventional *bin-packing* placement engine [1]. It uses individual and aggregated demand constraints of VMs and capabilities of PMs and determines whether the given sets of VMs can be placed on the given sets of PMs to provide the projected demands. The *SAP* is notified when the set of PMs cannot provide the projected demand. This causes the *SAP* to find another candidate server rack (i.e., new sets of PMs) and refine it with the *DAP*. This process continues until it finds a suitable server rack for each VM group.

Finally, in the *conquer* stage, the *SAP* combines each VM group to server rack mapping and produces a complete plan. Deployment toolkits can use the produced plan to deploy VMs to the actual datacenter. The overall process of our approach is outlined below and illustrated in Fig 1(b).

1) Divide the complex VM graph into a set of smaller VM groups, and the upper bound of the VM group size (i.e., number of VMs in the group) is determined by the average capacity of a server rack (Def 3). The created groups should maximize intra-communication cost and minimize inter-communication cost while satisfying the availability constraints.
2) Find the suitable server rack for each VM group, such that the mapping minimizes the total communication cost and guarantees the satisfaction of anti-collocation and collocation constraints at each level of the datacenter hierarchy.
3) Refine the VM groups to server rack assignment using the *DAP*, and find the most suitable PM for each VM.

Our placement problem can be classified as an optimization problem, which in fact consists of three *NP* problems: 1) communication aware VM clustering (graph cuts *NP*-hard), 2) VM group to server rack mapping (graph matching *NP*-complete), and 3) VM to PM mapping (bin-packing *NP*-hard). In addition, finding the problem solvability is also an *NP*-hard problem. We find the solvability of the problem using availability constraints, more specifically, by using anti-collocation constraints and transforming it to a graph coloring problem.

IV. Algorithms and Complexity

The algorithms presented here can be used for a datacenter with an arbitrary number of levels; however, for simplicity we discuss our algorithms by using the example datacenter model shown in Fig 1(a).

A. Constraint-aware VM Grouping (bottom-up)

The intuition of constraint-aware grouping is to respect the structural constraints of the application and to create a set of small groups both to minimize the overall communication cost and to enforce availability constraints. SCAVP employs a bottom-up approach, in which it uses the VM graph to build a logical hierarchy similar to that of datacenter: for example, if the datacenter has three intermediate levels then the created VM hierarchy also has three levels.

SCAVP uses the *MinMaxVMGrouping* algorithm to create the VM hierarchy. It starts with the VM level and creates a set of VM groups (VMGs) to enforce the structural constraints at bottom-most level (i.e., server rack). The maximum number of VMs in a VM group is determined by the average capacity of a server rack. Then, it uses previously created VMGs and creates a new set of VMGs for the next level in the hierarchy (i.e., zone according to Fig 1(a)). Likewise it recursively uses the algorithm and continues the process until it creates a hierarchy similar to the input datacenter.

For each level (including VM) it transforms communication aware grouping to a *K-Min-Max-cut* problem [21]. In *K-Min-Max-cut*, the requirement is to cut the graph into a set of *K* cluster so that the inter-cluster cost is minimized and intra-cluster cost is maximized. In SCAVP, the requirement is also to create a set of VM groups (no constraints on *K*) so that communication cost between any two groups is minimized. Moreover, *K-Min-Max-cut* is an *NP*-hard problem; hence, only approximate algorithms are possible. We use a greedy algorithm with a heuristic: the heuristic is to merge two VMs with highest communication cost together to form a group (unless constraints are violated).

1) Formal Problem: Assume V=$\{v_1, v_2, \cdots, v_n\}$ is a set of VMs, and let $l(u, v)$ denote the communication cost between VM *v* and VM *u*, and assume *s* as the capacity. If the number of groups created is k, then,

$$\lceil n/s \rceil \leq k < n$$

Total communication cost within a group (where $\cup$ is the set of all VMs in the group)

$$l(v, \cup) = \sum_{u \in \cup} l(v, u)$$

$$Total\ com\ cost\ (when\ k = n) = \sum_{i<j} l(v_i, u_j)$$

$$Total\ com\ cost\ (when\ k \neq n) = \sum_{i<j} l(v_i, u_j) - \sum_{j=0}^{k} l(v_j, U_j)$$

$\cup U_j = V$ and $u_i \cap u_j = \emptyset\ \forall i \neq j$. The minimum communication cost is achieved when there is only one group and the cost is maximized when there are n groups. Thus, the problem is to find the correct balance while satisfying the constraints.

2) Algorithm: Here we discuss the *MinMaxVMGrouping* algorithm for a single step (say level *l*) (SCAVP recursively uses the algorithm for all the levels in the hierarchy). Firstly, SCAVP checks the collocation constraints for level *l* and creates a new set of VMGs to satisfy the constraints. For example, if two VMs (VMGs) are needed to collocate at the server rack level, then a new VMG is created for the two VMs and any others that have transitive collocation requirements with them. This process continues until it processes the collocation constraints of all the VMs.

Secondly, SCAVP creates an $n * n$ communication matrix (n = # VMG at level *l*) to represent the communication requirement between different VMGs. Next, it picks the element with the highest value from the matrix and merges the two corresponding VMGs together (unless constraints are violated). Then, it updates the communication matrix to reflect the changes and picks new element with the highest value and continues the process until it merges all the possible VMGs. Our algorithm is illustrated in *MinMaxVMGrouping*.

3) Optimization: MinMaxVMGrouping is a greedy algorithm and produces locally optimized results. Thus, further optimization can be used to reduce the total communication cost. We have designed another greedy algorithm *Optimize-TotalComcost* which further minimizes the communication cost of produced results. We use a simple heuristic: a VM is chosen from a VMG, and the total communication cost for that VM to and from other VMGs is calculated. If we

Algorithm: MinMaxVMGrouping

Input: $V = (V_1, V_2, \cdots, V_n)$, #S
Output: $G = [G_i, G_2 \cdots G_k]$, where k $\leq$ n
begin
  $G = doCollocationAwareGrouping(V)$
  $G = [G_1, G_2, \cdots, G_m]$, where $\bigcup G_i = V$, $G_i \subseteq V$ &
  $G_i \cap G_j = \emptyset \ \forall i \neq j$
  $|C|_{nn} = buildCommunicationMatrix(G)$
  Let current = 0;
  while *true* **do**
    $e = pickTheNextMaxEdge(C, current)$
    $current \leftarrow e.getValue$
    if $current = 0$ **then**
      break
    Let G_i, G_j are the two corresponding vertexes of e.
    if $G_i.size + G_j.size < \#S$ **then**
      $G_i \leftarrow G_i + G_j$
      $G \leftarrow G \setminus G_j$
      $C = updateCommunicationMatrix(G)$

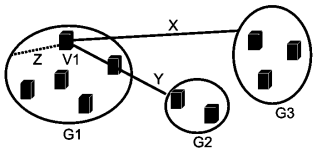


Fig. 2. Intra and Inter communication cost for a VM *V1*

find a VM that has a larger inter-communication cost (say VMG1) than intra-communication cost, then we move the VM to VMG1 only if the future total cost is less than the current total cost. For example, as shown in Fig 2, value `Z` represents the communication cost to V2 from all the VMs in G1, `X` represents the total cost from G3, and `Y` represent the total cost from G2. If `Z` is less than `X`, it means there is a higher communication cost for V2 from G3 than G1; hence, our algorithm tries to move V2 into G3 if the projected total communication cost is less than the current communication cost. The algorithm is outlined in *OptimizeTotalComcost*.

Algorithm: OptimizeTotalComcost

Input: $B = [VMG_i \cdots VMG_k]$, $V = (V_1, V_2, \cdots, V_n)$
Output: $B = [VMG_i \cdots VMG_k]$ where k < n
begin
  while *change* **do**
    change = FALSE
    $currentTolCost = calcualteCurrentComCost()$
    for *each* v_i *in* V **do**
      Let v_i resides in VMG_c
      C_i = CalCostForEachGroup(v_i)
      $C_i = \{c_1, c_2 \cdots, c_k\}$, where c_j =
      $cost(v_i, VMG_j) | 1 \leq j \leq k$
      Let c_intra as the communication cost from VMG_c
      $c_{max} = max\{c_i | 1 \leq i \leq k\}$
      Let VMG_{max} is the group with c_{max}
      if *c_intra* $< c_{max}$ **then**
        $futureCost = costAfterMove()$
        if *futureCost* $<$ *currentTolCost* **then**
          if *satisfy_group_constraints* **then**
            Move v_i to VMG_{max}
            change = TRUE break

B. VMGs to Server Racks Assignment (top-down)

The process of assigning VMGs to racks proceeds in a top-down fashion: it starts with the top-most level and tries to find the most suitable node from the datacenter ('region', in the terminology of Fig 1(a)) to place the largest node at (0^{th}) level from the VM hierarchy. Next, it moves one level down, and tries to find the most appropriate zone (from the selected region) to place the largest zone from the VM group hierarchy. If the datacenter hierarchy has more levels, then this process continues until it reaches the server rack level.

At the server rack level, the problem is to assign a small set of VMGs to a small set of server racks. Here, SCAVP creates a VMG graph (*VG*) by using aggregated communication requirements. In *VG*, VMGs are represented as vertices and communication requirements are represented as weighted edges. Similarly by using server racks data, it produces a server rack graph (*RG*) to represent the communication cost among all the server racks in the selected set. In *RG*, each vertex represents a server rack and edges represent communication costs. This converts the assignment problem into a weighted graph matching problem, which is said to be a *NP*-complete.

We have designed an approximation algorithm *AssignRacks* with a heuristic to solve the assignment problem. In our heuristic, a VMG pair with largest communication requirement be assigned to a server rack pair with the lowest communication cost. More concretely, we extended [27] to capture the complexity of our problem (e.g., structural constraints) and to create more deterministic results. In each phase we generate a set of candidate server racks (one centered at the server rack is being analyzed currently), then choose one from this set in a greedy manner.

The *AssignRacks* algorithm finds VMG to server rack mappings. It works as follows: for each VMG, it finds a candidate server rack and then refines it using *DAP*. If *DAP* can place the VMs on the candidate server rack to meet the requested demands, then algorithm moves to then next VMG, otherwise it tries to find another candidate and refines it with *DAP*. Once each VMG is assigned to a server rack the algorithm moves to the next zone. Failure to find an assignment to at least one VMG causes it to try another zone from the same region and repeat the above procedure. Unless it finds a suitable zone from the selected region, it tries another region from the datacenter and refines the assignment using *AssignRacks*.

C. VM to PM Mapping (bin-packing)

In our system, we use Vespa [1] as the *Demands-Aware Planner*. For the purposes of this work, we can regard Vespa as a conventional placement engine which considers VM placement as a bin-packing problem and provides a placement plan that maximizes the demand satisfaction. Hence, the refining process is also an *NP*-complete problem. SCAVP uses Vespa to process the demand constraints and finds the VM to PM mapping. More concretely, *AssignRacks* finds candidate server racks for a VM group and refines it with Vespa. In that process, Vespa tries to place the set of VMs onto PMs in the

Algorithm: AssignRacks

Input: $VG = \{G_1, \cdots, G_k\}$, $RG = \{R_1, \cdots, R_r\}$ where k $\leq$ r
Output: $|P|_{kr}$
begin
  Let M is completed set of VMGs, and R is completed set of racks.
  $M \leftarrow \emptyset$, $R \leftarrow \emptyset$
  while $|M| < n$ **do**
    e_{max} = pickMaxEdge($VG \backslash$)
    $\{E\}$ = picMinEdges($RG \setminus R$)
    $e_r = pickAnEdge(E)$
    Let e_{max} *as the cost between* G_i *and* G_j
    Let e_r *as the cost between* R_x *and* R_y
    if G_i *and* G_j *can be placed on* R_x *and* R_y **then**
      Let $TV \leftarrow \{G_i, G_j\}$, and $TR \leftarrow \{R_x, R_y\}$
      while *hasmore* **do**
        $e_{max} = getMaxEdge(TV \rightarrow VG \setminus (M \cup TV))$
        if $e_{max} = NULL$ **then**
          hasmore = FALSE
        else
          $e_r = pickAMaxEdge(TR \rightarrow RG \setminus (R \cup TR))$
          $N(e_{max}) = (v, v_k)$, where $v \in TV$
          $N(e_r) = (r, r_k)$, where $r \in TR$
          $addMapping(v_k \rightarrow r_k, P)$
          $M \leftarrow M \cup v_k$, $R \leftarrow R \cup r_k$
          $TV \leftarrow TV \cup v_k$, $TR \leftarrow TR \cup r_k$

given so that PMs can provide the projected demand. If the server rack can provide the projected demand then it produces the VM to PM mapping, else it returns `false` causing our algorithm to find another candidate server rack. When Vespa returns `true`, it also provides detailed a description with $VM \rightarrow PM$ mappings and individual demand satisfactions (e.g., CPU, Memory, #cores).

V. Evaluation

We have created a SCAVP prototype in Java, there we have implemented all the mentioned algorithms and integrated with Vespa. In this sections we analyze SCAVP in a simulated environment and provide experiment results to illustrate the efficiency of our approach. To the best of our knowledge no other existing tools handle the structural constraints supported in SCAVP. Thus, our comparative analysis is focused on finding the improvements of SCAVP with conventional placement approach Vespa.

We simulated four different application architectures each having different resource, communication, and availability requirements. We selected these four types to cover a broad range of commercial applications. For each application type we evaluated the placement efficiency by varying the application size from 20 VMs to 100 VMs (e.g., 20, 30, ... ,100), and for each configuration, we performed 400 experiments and present here the averaged results.

- 3-tier: Modeled after the traditional three tier web serving architecture. We assume all the servers in a given tier communicate with all the other servers in the same tier with similar communication costs. The communication cost between the web tier and application tier is less than that of the cost between the application and database tier. All the servers in any given tier are assumed to be homogeneous.
- 4-tier: The difference between 3-tier and 4-tier is the existence of a cache tier. We assume higher communication requirements among the serves in the cache tier compared to other tiers.
- Map-reduce architecture: All servers communicate with all the other servers with same communication requirements. All the servers are assumed to be homogeneous.
- Server mesh: An arbitrary application, where a set of heterogeneous application components are connected using heterogeneous communication requirements.

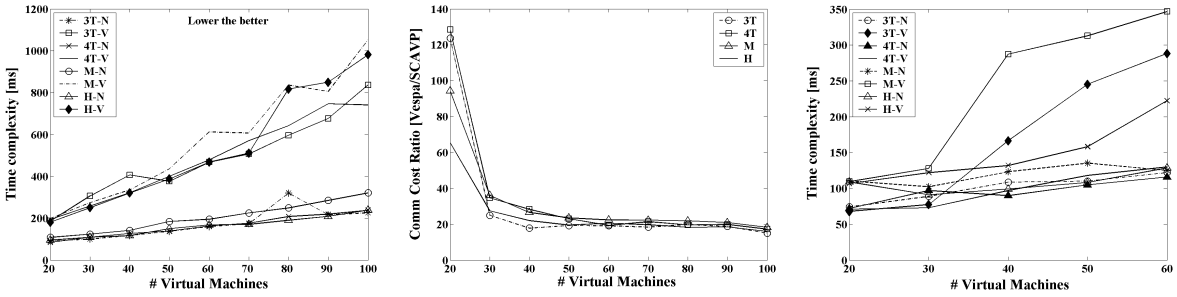
We simulated two types of datacenters: a small datacenter consisting of up to 60 PMs and a large datacenter consisting of over 200 PMs. Furthermore, we simulated each server rack to have at most 16 PMs, and each PM in a given server rack is homogeneous with respect to its types. The datacenter was simulated to have heterogeneous communication cost among different server pairs with the objective of representing an actual datacenter. To represent the running status of an actual datacenter each PM was simulated with different capabilities (resource utilization levels). Moreover, we simulated two and tree regions for the small and larger datacenter respectively.

Notation: We use a notation #A-#T to represent each concrete configuration. #A denotes the application architecture (*3T* for 3-tier, *4T* for 4-tiers, *M* for server mesh, and *H* for map-reduce). #T denotes the placement engine *N* for SCAVP and *V* for Vespa. For example *3T-N* represents 3-tier application evaluated using SCAVP.

A. Evaluation on Large datacenter

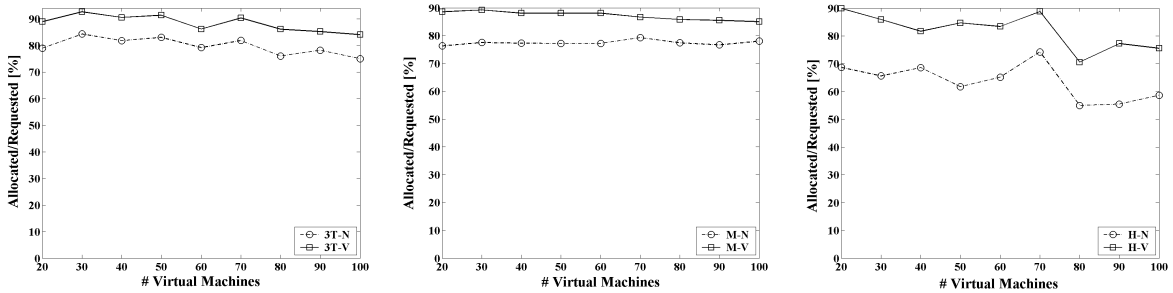
First, we evaluated the time complexity of two approaches by analyzing the time taken to produce the placement plan. Our results showed that SCAVP performed much better when compared to Vespa alone. More importantly, as the size of the application increases, Vespa takes significantly longer than SCAVP. This is expected: as Vespa uses a bin-packing approach, when the problem size increases, the complexity increases exponentially. With SCAVP, the problem is subdivided, and overall complexity is reduced. Our experiment results are illustrated in Fig 3(a).

We then measured the improvement in communication cost when SCAVP is used instead of Vespa alone. Of course, Vespa does not consider communication cost, and so an improvement is expected: Vespa tries to find the most suitable PMs that can provide the requested resources without considering their location. As a result, the PMs that are chosen can be spread across the datacenter. In contrast, SCAVP focuses on communication cost minimization; thus, it produces a placement with less communication cost. For each application size, we calculated the total communication cost produced by the two approaches. Then we calculated the ratio to find the improvements (Vespa's cost/SCAVP's cost), and our results are shown in Fig 3(b). As shown in the figure, by confirming our assumptions SCAVP produces a placement plan with significantly less total communication cost. More specifically, the plan produced by SCAVP



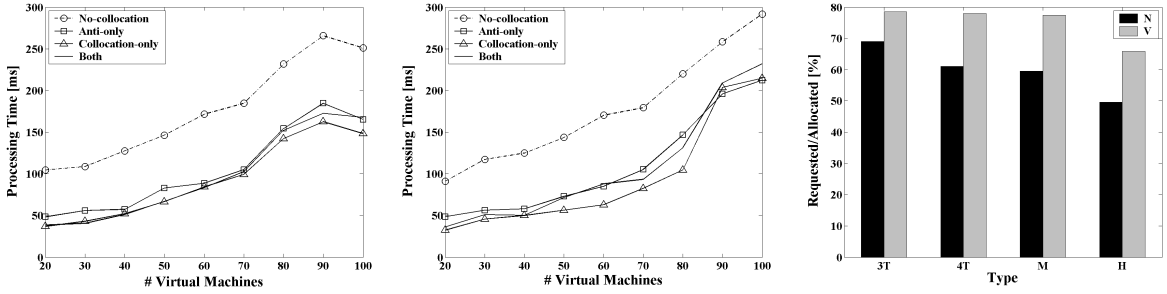
(a) Time complexity on large datacenter (b) Com-cost improvement on large datacenter (c) Time complexity on small datacenter

Fig. 3. Comparison of time complexity and efficiency of created placement plan



(a) 3-tier system (b) Server mesh system (c) Map-reduce system

Fig. 4. Percentage demand satisfaction on large datacenter (Requested resource demand/ Allocated resources)



(a) 3-tier systems on large datacenter (b) 4-tier systems on large datacenter (c) Demand satisfaction (%)

Fig. 5. a), b) Time complexity with allocation constraints; c) Average demand satisfaction (%) for four application types on small datacenter

has 20 times less communication cost to Vespa.

Next, we measured the demand satisfaction when SCAVP is used and compared it with Vespa alone. Here we expect Vespa alone to achieve a higher demand satisfaction (at the cost of no network consideration and greater runtime). Vespa finds the placement plan to maximize demand satisfaction across all the VMs; in contrast, SCAVP tries to maximize demand within a small group (datacenter vs. a server rack). Our experiment results justified our assumptions, where Vespa performed better compared to SCAVP. Our results are illustrated in Fig 4(a), (b) and (c). As shown in the figure Vespa provides average of 90% demand satisfaction while SCAVP gives 80%.

We then evaluated the effect of availability constraints by measuring the time complexity. We analyzed it in four different ways: 1) collocation only, 2) anti-collocation only, 3) both collocation and anti-collocation, and 4) no collocation nor anti-collocation. Existence of structural constraints helps to reduce search speech significantly (e.g., collocation between two VM enforces to consider two VMs as one VM), hence application with availability constraints produced the placement plan much faster compared to that of with no constraints. Our experiment results are shown in Fig 5(a) and (b), and as illustrated in the figure, when the application has either type of constraints, then time complexity reduces by around 30% (30%—50%).

B. Evaluation for Small datacenter

We extended our analysis to smaller datacenters with the intuition of finding the significance of SCAVP when the problem size is smaller. We first measured the time complexity for the two approaches and our results are illustrated in Fig 3(c). As shown in the figure, even for smaller problem sizes SCAVP performs better compared than Vespa alone. As shown in the figure, for smaller datacenters, the difference in time complexities not that significant: Vespa can find a placement plan very quickly. Nevertheless, the generated plan has similar communication gains compared to that of large datacenters. We also measured the percentage demand satisfaction, due to the negligible variances among four configurations (i.e., for

VM sizes 20, 30, 40, 50) have presented the average demand satisfaction across four configurations in Fig 5(c).

VI. Related Work

In modern virtualization-based datacenters, the problem of VM placement has become a crucial issue and attracted significant attention recently [1]–[8], [12], [18], [25]. To achieve high levels of efficiency in cloud computing requires robust strategies for VM placement. We have previously presented a bin-packing based approach [1] which maximizes the resource satisfaction on a given datacenter. A static heuristic-based vector bin-packing algorithm to minimize the number of required servers is discussed in [10]. Grit et al. has presented a broker-based scheme architecture and algorithm for assigning VMs to physical machines [9]. Kimbrel et al. proposed an approach to minimize the number of VM migrations while reallocating resources [11]. The approach presented here supports both resource and structural constraints of the application.

Power optimization has also become a crucial factor in VM placement and greatly motivated the research community. For example, the authors of [22] focus mainly on power optimization. [23] proposes a linear power model for full-system power analysis and modeling, while [24] presents a power-aware placement algorithm based on a power model and a migration cost model. A similar approach is proposed in [26], which discussed how to maximize the revenues of cloud providers by trimming down their electricity costs.

Placement approaches focus on profit maximization have received significant attention too, for example in [12], the authors describe how to place VMs across a number of datacenters to increase profit, and [13] presents an example of an economic model of a datacenter. [15] proposes a graph based solution for semi-automated service creation, which expresses the mapping between a business support system and an operations support system.

Duong et al. present a placement approach to improve the performance on highly interactive distributed virtual environments, which has some similarities with our communication optimization approach [14]. The closest approach to ours is [25], which discusses network-aware placement; however, we go beyond and also present a hierarchical approach to support structural information of the application.

VII. Conclusion

We proposed a structural constraint-aware virtual machine placement as an effective way to improve the performance and availability of services hosted on modern datacenters, particularly on IaaS clouds. Careful VM placement can reduce communication costs significantly and improve overall system availability. We formalized the datacenter and application, and used two models as inputs to the placement problem. We formulated VM placement as an optimization problem and show NP hardness, and we propose a hierarchical approach composed of four approximation algorithms to solve the placement problem for larger problem sizes. Finally, we illustrated the significance of proposed system on a simulated environment.

References

[1] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise datacenters. In WWW 2007, Banff, Alberta, Canada.

[2] VMware Capacity Planner. http://www.vmware.com/products/capacityplanner/.

[3] IBM WebSphere CloudBurst Appliance. http://www-01.ibm.com/software/webservers/cloudburst.

[4] Novell PlateSpin Recon. http://www.novell.com/prodcuts/recon/.

[5] C. Hyser, B. McKee, R. Gardner, and B J. Watson. Autonomic Virtual Machine Placement in the Data Center. In *Tech Report: HPL-2007-189*.

[6] N. Bobroff, A. Kochut, and K. Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *Integrated Network Management, 2007. IM '07*.

[7] Machida, F., Kawato, M., and Maeno. Y Redundant virtual machine placement for fault-tolerant consolidated server clusters. In *NOMS 2010*.

[8] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. Performance and Availability Aware Regeneration for Cloud Based Multitier Applications. In *DSN 2010*, Chicago, Illinois, USA.

[9] L. Grit, D. Irwin, A. Yumerefendi. and J. Chase. Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration. In *VTDC* November 2006.

[10] J. Shahabuddin, A. Chrungoo, V. Gupta, S. Juneja, S. Kapoor, and A. Kumar. Stream-Packing: Resource Allocation in Web Server Farms with a QoS Guarantee. Lecture Notes in Computer Science, 2001.

[11] T. Kimbrel, M. Steinder, M. Svirdenko, and A. Tantawi. Dynamic Application Placement under Service and Memory Constraints. In *Workshop on Efficient and Experimental Algorithms, 2005*.

[12] S. Chaisiri, B. Lee, and D. Niyato. Optimal Virtual Machine Placement across Multiple Cloud Providers. In *APSCC 2009*.

[13] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centres. In *Proc. ACM Symposium on Operating Systems Principles, pp. 103.116, '01*.

[14] D. Ta, S. Zhou, W. Cai, and X. Tang. Network-Aware Server Placement for Highly Interactive Distributed Virtual Environments. In *IEEE/ACM DS-RT 2008*.

[15] A. Sailer, M. Head, A. Kochut, and H. Shaikh. Graph-based Cloud Service Placement. In *SCC2010*.

[16] Iganacio M. Liorente. Cloud Computing for on-Demand Resource Provisioning. In *7th NRENs and Grids Workshop, 2008*.

[17] Y. Chen, T. Wo, and J. Li. An Efficient Resource Management System for On-line Virtual Cluster Provision. In *CLOUD 2009*.

[18] H. Nguyen and F. Tran. Autonomic virtual resource management for service hosting platforms. In *CLOUD 2009*.

[19] Y. Zhang, G. Huang, X. Liu, and H. Mei. Integrating Resource Consumption and Allocation for Infrastructure Resources on-Demand. In *CLOUD 2010*.

[20] F. Chang, J. Ren, and R. Viswanathan. Optimal Resource Allocation in Clouds. In *CLOUD 2010*.

[21] Saran, H. and Vazirani. V. Finding k-cuts within twice the optimal. Proc. 32nd Ann. IEEE Symp. on Foundations of Comput. Sci., IEEE Computer Society, 743-751.

[22] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong. EnaCloud: An Energy-saving Application Live Placement Approach for Cloud Computing Environments. In *CLOUD 2010*.

[23] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In 2nd WS Modeling, Benchmarking & Simul., pages 158–168, Boston, MA, June 2006.

[24] A. Verma, P. Ahuja, and A. Neogi. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In *Middleware 2008*.

[25] X. Meng, V. Pappas, and Li. Zhang. Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In *INFOCOM 2010*.

[26] T. Henzinger, A. V. Singh, V. Singh, T. Wies, and D. Zufferey. FlexPRICE: Flexible Provisioning of Resources in a Cloud Environment. In *CLOUD 2010*.

[27] Pettie S. and Sanders, P. A simpler linear time $2/3 - \varepsilon$ approximation for maximum weight matching. *Inf. Proc.Lett* 91, 271-276.