Supporting Security Requirements for Resource Management in Cloud Computing

Ravi Jhawar, Vincenzo Piuri and Pierangela Samarati
Dipartimento di Informatica – Università degli Studi di Milano, 26013 Crema, Italy
Email: *firstname.lastname*@unimi.it

***Abstract*—We address the problem of guaranteeing security, with additional consideration on reliability and availability issues, in the management of resources in Cloud environments. We investigate and formulate different requirements that users or service providers may wish to specify. Our framework allows providers to impose restrictions on the allocations to be made to their hosts and users to express constraints on the placement of their virtual machines (VMs). User's placement constraints may impose restrictions in performing allocation to specific locations, within certain boundaries, or depending on some conditions (e.g., requiring a VM to be allocated to a different host wrt other VMs). Our approach for VM allocation goes beyond the classical (performance/cost-oriented) resource consumption to incorporate the security requirements specified by users and providers.**

***Index Terms*—Cloud security, Placement constraints, Reliability and availability, Resource management, VM allocation**

I. Introduction

Cloud computing provides an on-demand access to required amounts of computing resources to its users in the form of VM instances over the Internet. This computing paradigm is gaining an increasing popularity among users particularly to deploy applications that require different degrees of processing, memory and storage resources (e.g., a cpu-intensive or a memory-intensive VM instance).

When a user requests the service provider to allocate it a set of computing resources, typically the VM provisioning algorithm follows a heuristics-based approach to allocate VM instances on physical hosts and delivers them to the user. At present, most implementations either build their provisioning algorithms by focusing mainly on realizing the service with agility (hence not scaling well to the granularity of individual resource types on physical hosts) or guide their provisioning algorithms to meet the service provider's business objectives (e.g., utilize fewer number of physical hosts while allocating VM instances to save energy consumption costs). In this paper we identify that service providers may need to impose a set of additional conditions on the allocation algorithms, particularly in the way they access infrastructure resources, to maintain the security and the performance of their system. Similarly, users may need to impose a set of conditions on the placement and the relative placement of their VM instances to correctly satisfy the security, reliability and availability requirements of their applications.

The contributions of this paper are two-fold. First, we categorize and formalize several requirements that users and service providers may want to specify with respect to security, reliability and availability of the service. Second, we address the satisfaction of such requirements in the overall problem of resource allocation in infrastructure Clouds.

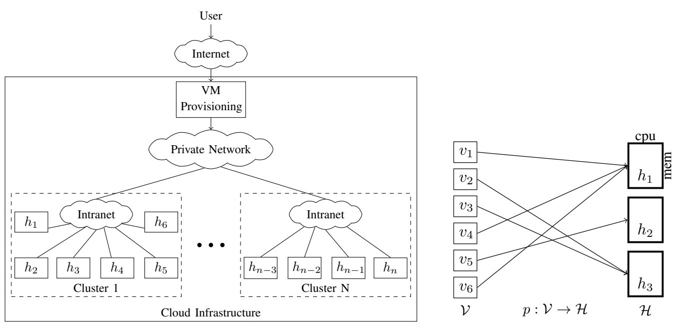
The paper is organized as follows. Section II provides the system overview, and Section III formalizes the VM allocation requirements in service provider's and user's perspectives. Section IV discusses our approach to VM provisioning. Section V summarizes the related work, and Section VI presents our conclusions.

II. System Overview

In Cloud computing, the service provider's system can be viewed as a large pool of interconnected physical hosts $\mathcal{H}$ that is partitioned into a set $\mathcal{C}$ of clusters. A cluster $C \in \mathcal{C}$ can be formed by grouping together all the hosts that have identical resource characteristics or administrative parameters (e.g., hosts that belong to the same network latency class or geographical location). Figure 1a illustrates an example of a Cloud infrastructure consisting of N clusters of interconnected hosts, and each cluster is connected through a network that is private to the service provider. We represent the resource characteristics of each physical host $h \in \mathcal{H}$ using a n-dimensional vector $\overrightarrow{h} = (h[d_1], h[d_2], \ldots, h[d_n])$, where each dimension $d_i \in \mathcal{D}$ represents the host's *capacity* corresponding to a distinct resource (e.g., CPU, memory, storage, network bandwidth). For the sake of simplicity, we consider that service provider denotes the resource capacity of hosts using normalized values (e.g., $\overrightarrow{h}$=(cpu,mem)=$(1, 1)$), and realizes a delivery scheme where computing resources are provisioned to the users in the form of an on-demand service over the Internet using VMs.

A user typically implements her applications and deploys them on the VM instances obtained from the service provider. The resource characteristics of each VM instance $v \in \mathcal{V}$ is also represented using a n-dimensional vector $\overrightarrow{v} = (v[d_1], v[d_2], \ldots, v[d_n])$, where each dimension corresponds to user's *requirements* for a specific computing resource. We assume that resource dimensions of VM instances are the same as that of the physical hosts, and service provider can translate user's requirements to normalized values. For example, a "small" instance in Amazon EC2 service may be translated to $\overrightarrow{v}$=(cpu,mem)=(0.4, 0.25) [1].

In the service provider's perspective, the task of resource provisioning involves allocation of VM instances of specified dimensions on physical hosts, and their delivery to the user. Resource provisioning can be characterized by a *mapping*



(a) Architecture of a Cloud infrastructure

(b) Mapping VMs on physical hosts

Fig. 1. Overview of the Cloud infrastructure and VM provisioning environment

function p:$\mathcal{V}$$\rightarrow$$\mathcal{H}$ that takes the set $\mathcal{V}$ of VM instances as input and maps each $v$$\in$$\mathcal{V}$ on the physical hosts $h$$\in$$\mathcal{H}$ as output. Figure 1b illustrates an example of mapping generated by p:$\{v_1, \ldots, v_6\}$$\rightarrow$$\{h_1, h_2, h_3\}$ where $p(v_1)$=$p(v_4)$=$p(v_6)$=h_1, $p(v_5)$=h_2 and $p(v_2)$=$p(v_3)$=h_3. The VMs and hosts are represented using rectangles to denote two resource dimensions (cpu and memory) by its sides. A physical host can accommodate more than one VM instance, but an individual VM can be allocated only on a single host. The mapping function is typically guided by the service provider to meet one (or both) of the following objectives:

- To reduce the energy consumption and operational costs, VM instances are consolidated on physical hosts to maximize the number of *free* hosts. For example, the mapping function p:$\{v_1, \ldots, v_6\}$$\rightarrow$$\{h_1, h_2, h_3\}$ is guided to achieve $p(v_1)$=$p(v_4)$=$p(v_6)$=h_1 and $p(v_2)$=$p(v_3)$=$p(v_5)$=h_3 so that the host h_2 remains unused. This allows the service provider to conserve the energy of running host h_2, and increase its service-response capability.
- To reduce the load variance of physical hosts across all the clusters in the Cloud to improve the overall performance and resilience of the system. For example, if we assume that $h_1$$\in$$C_1$, $h_2$$\in$$C_2$ and $h_3$$\in$$C_3$, the mapping function p:$\{v_1, \ldots, v_6\}$$\rightarrow$$\{h_1, h_2, h_3\}$ is guided to achieve $p(v_1)$=$p(v_4)$=h_1, $p(v_5)$=$p(v_6)$=h_2 and $p(v_2)$=$p(v_3)$=h_3 so that the VM instances (resource usage) are uniformly distributed across the clusters.

In this paper, we define a mapping function that considers the two objectives during resource provisioning and integrate it in a framework that identifies and categorizes further security, reliability and availability requirements which service providers and users may need to impose on the mapping function (see Section III). We integrate all the additional requirements within our resource provisioning algorithm and satisfy those requirements during VM allocation (see Section IV).

III. Mapping Constraints

We model the service providers and users requirements in the form of *placement constraints* and guide the mapping function to satisfy all the constraints. We distinguish three kinds of placement constraints:

- *global constraint* that applies to all the hosts and VM instances in the system at any given instant of time;
- *infrastructure-oriented constraints* that are specified by the service provider to maintain the security and quality of its service;
- *application-oriented constraints* that are specified by the users to increase the security of their applications.

For simplicity we consider constraints specified with respect to specific VM or host identifier but note that they can be specified with reference to their properties (e.g., all the hosts that belong to a cluster or VM instances owned by a user).

A. Global Constraint

The classical *resource capacity* constraint states that the amount of resources consumed by all the VM instances that are mapped on a single host cannot exceed the total capacity of the host in any dimension. Formally, for all the VM instances $v$$\in$$\mathcal{V}$ and hosts $h$$\in$$\mathcal{H}$ in the system, the mapping function p:$\mathcal{V}$$\rightarrow$$\mathcal{H}$ must satisfy

$$\forall h \in \mathcal{H}, d \in \mathcal{D}, \sum_{v \in \mathcal{V} | p(v)=h} v[d] \leq h[d] \tag{1}$$

This placement requirement is typically supported by all the solutions existing in the literature. However, several solutions do not consider that the amount of resources consumed by a VM when placed in isolation on a host and with other co-hosted VMs may not be the same. When multiple VMs are placed on a host, the hypervisor or host operating system may consume additional resources (e.g., CPU cycles or I/O bandwidth during resource scheduling), and VM instances may interfere with each other and consume higher amounts of shared resources (e.g., the L2 cache during context switching). Formally, if $v_j, v_k$$\in$$\mathcal{V}$$|$$p(v_j)$=$p(v_k)$=$h$, and utilize $v_j[d_i]$ and $v_k[d_i]$ amount of resources in the i^{th}-dimension when allocated individually, then $v_j[d_i]$ and $v_k[d_i]$ together may utilize a bit more than $(v_j[d_i]$+$v_k[d_i])$ resources from the host h (unless v_j and v_k uses the same VM image). To avoid performance degradation and inconsistent system state due to the above factors, we allow service providers to define an upper bound on the resource capacity of each host that can be used for VM provisioning. The remaining capacity is then used by the service provider for VM management. The resource capacity constraint demands that VM instances not be allocated on a host if its capacity in any dimension reaches the upper bound or $threshold$ value specified by the service provider, that is,

$$\forall h \in \mathcal{H}, d \in \mathcal{D}, \sum_{v \in \mathcal{V} | p(v)=h} v[d] \leq (h[d] * threshold[d]) \tag{2}$$

We note that the $threshold[d]$ value can be specified in terms of percentage or normalized value between $(0, 1)$.

Example 1. Suppose that the service provider specifies the upper bound on cpu and memory usage of host h_1 as 80% and 70% respectively for VM provisioning. Assuming that the

TABLE I
EXAMPLES OF THE CONSTRAINTS ON THE MAPPING FUNCTION

Perspective	Applied by	Constraints	Description
Global	Service Provider	resource capacity	Resources consumed by VM instances must be less than the upper bound ($threshold$) of host's capacity
Infrastructure oriented	Service Provider	forbid	Forbid a set of VM instances from being allocated on a specified host
		count	The number of VM instances deployed on a host must be less than a given value
Application oriented	User	restrict	Map a VM instance only on a specified set of physical hosts
		distribute	Allocate a specified pair of VM instances on different hosts
		latency	The network latency between the specified pair of VM instances must be less than a given value

resource capacity of host h_1 in each dimension is normalized to 1, our resource capacity constraint specifies that the mapping function must satisfy (see Figure 1b):

$$v_1[cpu]+v_4[cpu]+v_6[cpu]\leq 0.8 \quad \text{and}$$
$$v_1[mem]+v_4[mem]+v_6[mem]\leq 0.7$$

B. Infrastructure-oriented Constraints

A service provider may need to impose restrictions on the mapping function, involving a set of physical hosts in its infrastructure, to improve the security, operational performance and reliability of its service.

As typical needs, we include two representative infrastructure-oriented constraints in our allocation framework that can be adapted to satisfy such requirements: forbid and count.

Forbid: To improve security, a service provider may need to specify that a set of hosts in its infrastructure must execute only system-level services (e.g., the access control engine or reference monitor) and the mapping function must not allocate VM instances requested by the users on those hosts. To satisfy such requirements, we introduce the forbid constraint that prevents a VM instance v from being allocated on a physical host h. Formally, when a service provider defines a set $Forbid=\{(v_i, h_j)|v_i\in\mathcal{V}$ and $h_j\in\mathcal{H}\}$ specifying the VM instances $v_i\in\mathcal{V}$ that must be forbidden from being allocated on hosts $h_j\in\mathcal{H}$, our resource provisioning algorithm guides the mapping function $p:\mathcal{V}\rightarrow\mathcal{H}$ to satisfy the following condition:

$$\forall v \in \mathcal{V}, h \in \mathcal{H}, \quad (v,h) \in Forbid \implies p(v) \neq h \tag{3}$$

Count: As the number of co-hosted VM instances on a physical host increases, its performance degrades. For example, the performance of a storage disk decreases if the number of I/O intensive applications in the VM instances increases; similarly, the network traffic from a host, VM management costs, and cpu utilization costs gradually increases. To avoid such conditions, we introduce the count constraint that allows a service provider to limit the number of VM instances that can be allocated on a given host. Formally, when the service provider defines $count_h$, the maximum number of VM instances allowed on a host h, the mapping function $p:\mathcal{V}\rightarrow\mathcal{H}$ ensures that the following condition is satisfied:

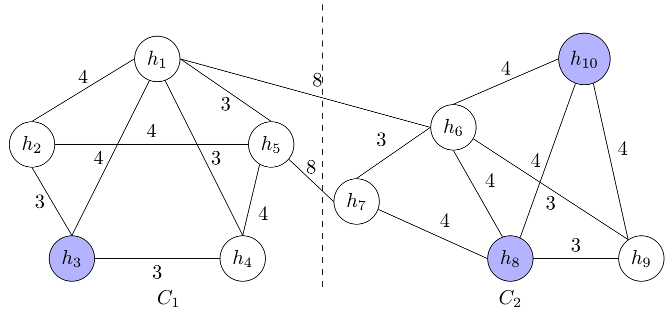


Fig. 2. An example of a Cloud infrastructure with network latency values for each network connection and service provider's constraints (the shaded nodes forbid allocation of user's VM instances)

$$\forall v \in \mathcal{V}, h \in \mathcal{H}, \quad |\{v \in \mathcal{V}|p(v) = h\}| \leq count_h \tag{4}$$

Example 2. Figure 2 illustrates an example of a Cloud infrastructure that consists of two clusters $\mathcal{C}=\{C_1,C_2\}$ and each cluster contains five physical hosts $C_1=\{h_1,\ldots,h_5\}$ and $C_2=\{h_6,\ldots,h_{10}\}$. Assume that the service provider *i)* runs security services on hosts $\{h_3,h_{10}\}$, *ii)* has taken host h_8 under maintenance due to a failure, *iii)* requires all the hosts in cluster C_1 to allocate not more than 3 VM instances each and utilize up to 80% of their aggregate cpu and 90% of memory capacity, and *iv)* requires hosts in cluster C_2 to accommodate a maximum of 2 VM instances each. If we assume that the service provider forbids all the VM instances from being allocated on physical hosts that are either used to run security services or that have undergone failures, the service provider can specify its additional requirements using the following constraints:

- $count_{h_1},\ldots,count_{h_5}\leq 3$
- $count_{h_6},\ldots,count_{h_{10}}\leq 2$
- $threshold_{h_1}[cpu]=\ldots=threshold_{h_5}[cpu]=0.8$
- $threshold_{h_1}[mem]=\ldots=threshold_{h_5}[mem]=0.9$
- $\forall v \in \mathcal{V}$: $\quad Forbid=\{(v,h_8),(v,h_3),(v,h_{10})\}$

C. Application-oriented Constraints

A user may need to impose a set of restrictions on the placement of her VM instances based on the security and privacy policies and the reliability mechanisms. For example, consider that a user employs a generic fault tolerance mechanism such as a replication technique to increase the reliability and availability of her application. The user may then need to impose a set of additional conditions on the system parameters and the relative placement of VM instances to successfully implement the fault tolerance mechanism.

As typical needs, we include three representative application-oriented constraints in our allocation framework that can be used to realize such conditions: restrict, distribute and latency.

Restrict: Based on the security and privacy policies [14], and mandatory government enforced obligations (e.g., EU Data Protection 95/46/EC Directive), a user may require that her VM instances be always located within a given community area (e.g., within EU countries). Similarly, to improve the application's performance, a user may require the mapping function to place her VM instances on hosts whose geographical location is closest to her customers. To satisfy such requirements, we introduce the restrict constraint that limits a VM instance $v \in \mathcal{V}$ on being allocated only on a specified group of physical hosts $H \subset \mathcal{H}$. When a user defines a set $Restr = \{(v_i, H_j) | v_i \in \mathcal{V} \text{ and } H_j \subset \mathcal{H}\}$, the mapping function $p: \mathcal{V} \to \mathcal{H}$ ensures the following condition:

$$\forall v_i \in \mathcal{V}, H_j \in 2^{H_j}, (v_i, H_j) \in Restr \implies p(v_i) \in H_j \quad (5)$$

Distribute: Any replication-based fault tolerance mechanism inherently requires that each replica be placed on different physical hosts to avoid single points of failure. For instance, if a user replicates her application on two VM instances, and if both the virtual machines are allocated on the same host, then a failure in the host results in a complete outage of the user's application. To avoid reaching such a state, we introduce the distribute constraint that allows a user to specify that two VM instances v_i and v_j be never located on the same host at the same time. When a user defines the set $Distr = \{(v_i, v_j) | v_i, v_j \in \mathcal{V}\}$ of pairs of VM instances that cannot be deployed on the same host, the mapping function $p: \mathcal{V} \to \mathcal{H}$ satisfies the following condition during resource provisioning:

$$\forall v_i, v_j \in \mathcal{V}, h \in \mathcal{H}, (v_i, v_j) \in Distr \implies p(v_i) \neq p(v_j) \quad (6)$$

Latency: The latency constraint enforces the mapping function to allocate two VM instances $v_i, v_j \in \mathcal{V}$ such that the network latency between them is less than a specified value T_{max}. When a user defines a set $MaxLatency = \{(v_i, v_j, T_{max} | v_i, v_j \in \mathcal{V})\}$ that specifies the acceptable network latency T_{max} between two VM instances v_i and v_j, the mapping function $p: \mathcal{V} \to \mathcal{H}$ ensures the following condition:

$$\begin{aligned} \forall v_i, v_j \in \mathcal{V}: \quad & (v_i, v_j, T_{max}) \in MaxLatency \\ & \implies latency(p(v_i), p(v_j)) \leq T_{max} \end{aligned} \quad (7)$$

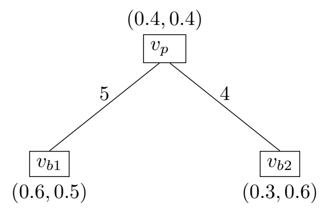


Fig. 3. An example of a user's request with resource requirements and network latency constraints

assuming that the network latency between two virtual machines is equal to the network latency between the physical hosts on which they are deployed. For instance, in a checkpoint-based reliability mechanism, the state of the backup VM instance must be frequently updated with that of the primary instance to maintain the system in a consistent state. This task involves high amounts of message exchanges, and hence an upper bound in the network delay is essential; otherwise, the wait-time of the primary instance during which the state transfer to the backup takes place may increase significantly and the overall availability of the application may be reduced.

Example 3. Figure 3 illustrates an example of a user's requirements for VM instances and associated allocation constraints. To increase the reliability and availability, consider that a user replicates the primary VM instance v_p of her application on two backup hosts v_{b1}, v_{b2}. To improve the security and to enforce government specified obligations the user may wish to impose a restriction on allocating v_p, v_{b1} and v_{b2} in cluster C_1, particularly v_p on one of the hosts $h_2 \ldots h_5$. To balance the application's performance and reliability, the user may require that the network latency limit T_{max} between v_p and v_{b1} is at most 5ms and between v_p and v_{b2} is at most 4ms. Furthermore, to avoid a single point of failure, the user may wish to ensure that v_p, v_{b1} and v_{b2} be deployed on different physical hosts. These requirements can be specified by the user as follows:

- $Restr = \{(v_p, \{h_2, \ldots, h_5\}), (v_{b1}, \{h_1, \ldots, h_5\}),$
  $(v_{b2}, \{h_1, \ldots, h_5\})\}$
- $MaxLatency = \{(v_p, v_{b1}, 5), (v_p, v_{b2}, 4)\}$
- $Distr = \{(v_p, v_{b1}), (v_p, v_{b2}), (v_{b1}, v_{b2})\}$

Table I provides a summary of the constraints in all three perspectives. To the best of our knowledge, none of the existing solutions allow both service providers and users to flexibly enforce their system's operational and VM placement requirements that support their security, reliability and availability goals during initial VM allocation.

IV. VM Provisioning Approach

We present an approach to VM provisioning in which each mapping of the function $p: \mathcal{V} \to \mathcal{H}$ is regulated to satisfy the security constraints specified by users and service providers, and system's objectives of reducing the load variance and energy consumption costs in the Cloud infrastructure. Given

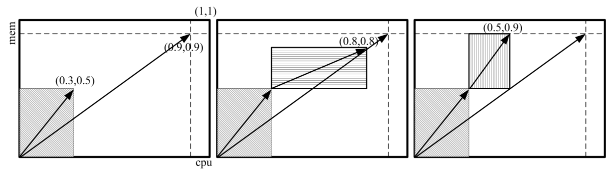


Fig. 4. An example of VM allocation on a host using vector dot-product method. Each side of a rectangle represents a resource type (cpu and memory). The VM instance v_0 represented with slanted pattern is already allocated. Allocation of v_1 and v_2 is represented with horizontal and vertical patterns

the NP-hardness of the VM provisioning problem, we propose a greedy heuristics-based approximation algorithm for its solution. At a high-level, for each user request, our algorithm analyzes the network to identify the clusters and physical hosts that can be used to perform VM provisioning, and for each physical host, it identifies the set of VM instances that can be allocated on that host.

To reduce the load variance between different clusters, our algorithm tries to allocate VM instances in the cluster that has highest amount of available resources. This heuristic is based on the observation that the load variance between clusters can be implicitly reduced by selecting the less-used cluster and utilizing its resources (by allocating VM instances requested by the user on its hosts). When selecting a cluster, resource availability of a cluster corresponding to a resource type may be greater than other clusters but smaller for other resource types (e.g., cpu and memory availability of clusters C_1, C_2 may be (0.6, 0.7) and (0.4, 0.8) respectively). In such cases, we determine the total amount of resources requested by the user in each dimension, and select the cluster whose availability is largest for the dimension that is most required by the user.

To reduce the energy consumption costs, our algorithm performs VM allocation in a host-centric manner, that is, once a cluster with maximum resource availability is selected, each host within the cluster is analyzed (e.g., by following identifier's order) to allocate as many VM instances on that host as possible. This heuristic allows our algorithm to improve the resource capacity usage of individual hosts and map all the VM instances on fewer number of physical hosts. In particular, similar to [13], [16], our algorithm leverages the vector dot-product method to analyze and map VM instances on a given host. We represent the VM instances and physical hosts as vectors (see Section II), and therefore, the vector dot-product value measures how a VM instance imposes itself on a given host based on the angle between optimum allocation (i.e., reaching point $(1, 1)$), vector magnitude, and present state of the host. Figure 4 illustrates an example of our method for a given host h and VM instances v_1 and v_2 where $\overrightarrow{v_1}{=}(0.5, 0.3)$ and $\overrightarrow{v_2}{=}(0.2, 0.4)$, their dot-product values are 0.166 and 0.129 respectively. By extracting the VM with larger dot-product value (v_1, which is cpu-intensive) the resource utilization of the host complements the already containing VMs (v_0, which is memory-intensive).

Figure 5 illustrates the algorithm that is executed each time a user requests the service provider to allocate her a set of

```
1:  INPUT   $\mathcal{H}$, $\mathcal{C}$, $\mathcal{V}$, $V$, Restr, Forbid, Count, Distr, MaxLatency
2:  OUTPUT  $p{:}V{\rightarrow}\mathcal{H}$
3:  MAIN
4:  /*Initialize the priority queue CL based on resource availability in each cluster*/
5:  CL:=Build_Priority_Queue($\mathcal{C}$) /*Analyze the clusters*/
6:  while $V{\neq}\phi$ do /*There are still VMs to be allocated*/
7:      /*Select the cluster C that has maximum resource availability*/
8:      C:= Extract_Max(CL)
9:      /*Consider each host in cluster C that does not violate the count constraint*/
10:     for each $h{\in}C \wedge h'[d{+}1]{<}count_h$ do /*$h'[d{+}1]$ is the usage counter*/
11:         $V'{:=}V$ /*Initialize the set $V'$ of VMs which can be allocated on host $h$*/
12:         /*Remove each VM that violates forbid/restrict constraints wrt host $h$*/
13:         for each $v{\in}V$ do
14:             if
15:                 $(v, h){\in}Forbid \vee$    /*If VM $v$ is forbidden from host $h$ or*/
16:                 $\{(v, H){\in}Restr{\wedge}h{\notin}H\}$ /*If $h$ is not in the restricted set of hosts*/
17:             then
18:                 $V'{:=}V'\setminus\{v\}$
19:         end for
20:         /*Initialize the set VMreq of VMs $v{\in}V'$ with their respective*/
21:         /*dot-product values computed wrt host $h$*/
22:         VMreq:=Build_Priority_Queue($h$,$V'$)
23:         while VMreq$\neq\phi$ do /*There are still VMs in VMreq*/
24:             /*Select the VM with largest resource needs*/
25:             $v$:=Extract_Max(VMreq)
26:             /*If the capacity and count constraints are satisfied, and*/
27:             /*If not in conflict with distribute constraint*/
28:             if
29:                 $\overrightarrow{v}{\leq}\overrightarrow{h'} \wedge$  /*If VM $v$ can be allocated in residual capacity of $h$ and*/
30:                 $\nexists(v, v_j){\in}Distr|p(v_j){=}h$  /*If distribute constraint is satisfied*/
31:             then
32:                 /*If $v$ is not related to $v_j{\in}V$ by latency constraints*/
33:                 if $\nexists(v, v_j, T_{max}){\in}MaxLatency$ then
34:                     /*Allocate $v$ on $h$ and update residual resource capacity value*/
35:                     $p(v){:=}h$
36:                     $\overrightarrow{h'}{:=}\ \overrightarrow{h'}{-}\overrightarrow{v}$
37:                     $V{:=}V\setminus\{v\}$
38:                 else
39:                     /*Initialize the set of VMs that must be allocated in the same all-*/
40:                     /*ocation cycle since they are linked by latency constraints to $v$*/
41:                     Reserve_list:=$\{(v,\ h)\}$
42:                     /*Find an allocation for $\forall v_j$ related to $v$ by latency constraints*/
43:                     if Forward_Allocate($v$, $h$) then
44:                         /*There is an allocation $\forall v_j$ related to $v$ by latency constraints*/
45:                         while Reserve_list$\neq\phi$ do
46:                             /*Allocate $v_j$ on host $h_j$*/
47:                             $p(v_j){:=}h_j$
48:                             $\overrightarrow{h'_j}{:=}\ \overrightarrow{h'_j}{-}\overrightarrow{v_j}$
49:                             Reserve_list:=Reserve_list$\setminus\{(v_j, h_j)\}$
50:                             $V{:=}V\setminus\{v_j\}$
51:                         end while
52:         end while
53:     end for
54: end while
```

Fig. 5. VM provisioning with Resource-capacity, Forbid, Restrict, Count, Distribute and Latency constraints

VM instances $V{=}\{v_1, \ldots, v_n\}$ where resource requirements for each $v_i{\in}V$ are specified in d-dimensions, and their security requirements are specified in the form of constraints (using $Restr$, $Distr$ and $MaxLatency$ sets). The algorithm takes the set V, set of all hosts $h{\in}\mathcal{H}$, parameters that group physical hosts into clusters $\mathcal{C}$ and snapshot of VM instances $\mathcal{V}$ already allocated in the Cloud as input. It also takes the sets that specify the security and performance requirements of users and service providers for all VM instances (including V) and hosts, and provisions the requested resources to the user, while satisfying all the constraints, as output.

The notion of selecting the least-used cluster to allocate $v_i{\in}V$ is realized by building the priority queue CL (line 5) based on resource availability in each cluster and extracting the

```
1:  **FORWARD_ALLOCATE**$(v, h)$
2:  /*Identify all $v_j$ directly/indirectly related to $v$ by latency constraints*/
3:  $V_l$:=**Build_Priority_Queue**($v_j | (v, v_j) \in MaxLatency \wedge v_j \notin$Reserve_list)
4:  while $V_l \neq \phi$ do /*There are VMs to be allocated*/
5:      /*Select the VM with the most stringent latency constraint*/
6:      $v_l :=$ **Extract_Min**($V_l$)
7:      /*Create the set K of hosts that qualify for $v_l$ wrt the latency value */
8:      /*forbid and restrict constraints*/
9:      K:=**Build_Priority_Queue**(
10:       $h_i |$latency$(h, h_i) \leq T_{max}(v, v_l) \wedge$ /*Hosts that satisfy latency constraint*/
11:       $(v, h_i) \notin Forbid \wedge$ /*Hosts that do not violate forbid constraint*/
12:       $h_i \in H | (v, H) \in Restr$) /*Hosts that satisfy the restrict constraint*/
13:     while K$\neq \phi$ do /*There are hosts to be considered*/
14:         /*Select a host $k$*/
15:         $k :=$ **Extract**(K)
16:         /*Verify the capacity, count and distribute constraints*/
17:         if
18:           $\overrightarrow{v} \leq \overrightarrow{k'}$ $\wedge$ /*If the capacity and count constraints are satisfied and*/
19:           $\nexists (v, v_j) \in Distr | p(v_j) = k$ /*If not in conflict with distribute constraint*/
20:         then
21:           /*Add $(v_l, k)$ to the set of VMs related by the latency constraint*/
22:           /*to $v$ which can be allocated*/
23:           Reserve_list:=Reserve_list $\cup \{(v_l, k)\}$
24:           /*Look for an allocation for VMs related by latency constraint to $v_l$*/
25:           if **Forward_Allocate**($v_l$, $k$) then
26:              K:=$\phi$ /*Allocation found - no other hosts need to be considered*/
27:           else
28:              /*Allocation not found - discard the tentative allocation for $v_l$*/
29:              Reserve_list:=Reserve_list$\setminus \{(v_l, k)\}$
30:         /*Else the host $k$ does not satisfy the capacity, count and distribute*/
31:         /*constraints and thus, another host in K should be considered for $v_l$*/
32:     /*Verify if an allocation has been found for $v_l$*/
33:     if $\nexists (v_l, k) \in$Reserve_list then
34:        /*No allocation found $\forall v_l$ directly/indirectly related to $v$ by*/
35:        /*latency constraints*/
36:        return 0
37:     end while
38: end while
39: /*All $v_i$ directly/indirectly related to $v$ with latency constraints can be*/
40: /*allocated satisfying all constraints*/
41: return 1
```

Fig. 6. The Forward_Allocate function of the VM provisioning algorithm

cluster C with maximum availability from CL (line 8). Similarly, the vector dot-product value of each $v_i \in V$ is calculated and stored in the VMreq priority queue (line 22). Entries from VMreq are then extracted in the decreasing order of the dot-product values (line 25) and analyzed for performing the final allocation. Note that in the absence of security constraints, the above two mechanisms: *i)* selecting the least-used cluster and *ii)* allocating VM instances on its hosts using the dot-product method is sufficient to perform VM provisioning and meeting service provider's objectives. We introduce various controls in this provisioning mechanism to ensure that all additional security and performance constraints specified by users and service providers are satisfied.

Based on the observation that forbid and restrict constraints define conditions on the association between VMs and physical hosts, we apply controls corresponding to these constraints mainly while building the VMreq priority queue (i.e., when analyzing the suitability of allocating $v_i \in V$ on host h). Note that only the VM instances that are extracted from VMreq are considered for allocation on any given host. Hence, we create a temporary set V' that contains all the VMs that must be allocated, discard all the VMs v from set V' (line 18):

- if an entry $(v, h) \in$Forbid exists (line 15), that is, discard all $v \in V'$ that are specified by the service provider in the *Forbid* set for host h.
- if a set of hosts H is specified in the *Restr* set for a VM $v \in V$ but the present host h does not belong to the set H (line 16), that is, discard v if $\exists (v, H) \in Restr \wedge h \notin H$.

and provide the set V' as input to build the priority queue VMreq. This control allows our algorithm to enforce the forbid and restrict constraints by ensuring that none of the VM instances that are in conflict with these constraints are allocated on host h.

The capacity and count constraints are confined to the resource usage of individual physical hosts. To ensure these constraints, we extend the d-dimensional vector representation of hosts and VM instances. In particular, as a control to the count constraint, we add dimension $d{+}1$ on each physical host and initialize its value to the number of VM instances that can be allocated on that host based on the value specified by the service provider i.e., $h[d{+}1] := count_h$. Similarly, the $[d+1]^{th}$ dimension of each VM is initialized to 1, and before allocating VM $v \in V$ on host h, the count control is enforced along with the capacity constraint (line 29, 36, 48). To enforce the capacity constraint, we maintain the residual resource capacity of each physical host $\overrightarrow{h'}$ that is initialized using the threshold values specified by the service provider. Before allocating a VM instance $v \in V$ on host h, the algorithm verifies if the resource requirements of the VM instance $\overrightarrow{v}$ is less than the residual capacity of that host $\overrightarrow{h'}$, that is, if $\overrightarrow{v} \leq \overrightarrow{h'}$ (covering all $d{+}1$ dimensions). If VM v is allocated on host h, the residual capacity of the host is updated as $\overrightarrow{h'} := \overrightarrow{h'} - \overrightarrow{v'}$ (line 36, 48).

To enforce the distribute constraint, we introduce a simple control that verifies if host h already contains a VM instance v_j for which the user has specified a condition $(v, v_j) \in Distr$ before allocating a VM v on that host (line 30). If $p(v_j) = h$ is found true, the algorithm skips that VM to the next host, hence satisfying the distribute constraint.

Lastly, to enforce the latency constraints, we introduce the notion of *forward allocate* and *reserve list*. When a VM v that satisfies all other constraints with respect to a host h is found, and if the VM v is related to other VMs v_j by the latency constraints, then v cannot be allocated until an allocation for all v_j is found. To this aim, our algorithm tentatively allocates the VM by saving the pair (v, h) in the Reserve_list (line 41) and calls the function Forward_Allocate to find an allocation for other VMs (line 43). The Forward_Allocate function first determines the set of all VMs that are related to v by the latency constraints and saves them in the priority queue V_l (line 3 in Forward_Allocate function). Each VM $v_l \in V_l$ is then extracted in the increasing order of T_{max} (line 6), and the set of hosts K that can be reached from h within the specified network threshold time, not conflicting the restrict and forbid constraints, is selected (lines 9-12). Each host $k \in K$ is considered and verified for capacity, count and distribute constraints (using similar controls as described above). When a host k is found for v_l that satisfies all the constraints and is not in turn associated with other VMs by latency constraints, the pair (v_l, k) is saved in the Reserve_list

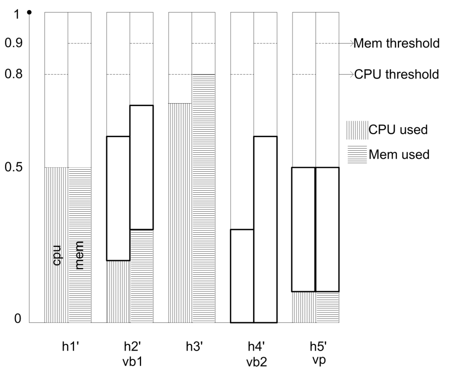


Fig. 7. Allocation of user requested VM instances on physical hosts based on our provisioning algorithm. VM instance v_p is allocated on host h_5, v_{b1} on host h_2 and v_{b2} on host h_4

(line 23). The Forward_Allocate function is recursively called until an allocation for all VMs is determined (line 25). If an allocation is not obtained, the entires from the Reserve_list are removed (line 29), and the function resumes from another host. The Forward_Allocate function returns 1 when an allocation for all VMs is found (line 41), otherwise it returns 0 (line 33-36) to the main function.

At present, our algorithm performs VM provisioning *only if all* the constraints are satisfied and allocation determined (line 43-50); no allocation is performed even if a single condition cannot be satisfied. However, we believe that our algorithm can be extended to perform partial allocations in a straightforward manner. Furthermore, based on the chosen heuristics, our algorithm is likely to find a solution with acceptable performance guarantees and ensure a solution if one exists. In the worst case, it analyzes all the hosts in the system to determine whether a solution exists or not.

Example 4. Consider the Cloud infrastructure and service provider's constraints described in Example 2, and user's request for VM instances with placement constraints described in Example 3. As an example, assume that VM instances are already allocated on the hosts in cluster C_1 and occupy resources as represented in Figure 7.

Table II illustrates the working of our VM provisioning algorithm. On receiving the user's request, a priority queue CL is first created by analyzing the resource availability in each cluster based on the assumed state of the infrastructure (see Figure 7), cluster C:$=C_1$ is extracted, and host h_1 is selected. The residual resource capacity in terms of cpu, memory and $count_{h_1}$ for host h_1 is $\overrightarrow{h_1'}=(0.3, 0.4, 2)$. The algorithm removes VM instances that are in conflict with h_1 based on restrict and forbid constraints, and therefore, VM v_p is not included in the VMreq priority queue. VMs v_{b1}, v_{b2} are considered one after another but since they do not satisfy the capacity constraint, the algorithm moves to the next host h_2. Residual resource capacity of host h_2 is determined and each VM $v \in V$ is analyzed based on the restrict and forbid constraints. Since none of the VM instances are in conflict with either constraints, they are included in the priority queue VMreq and vector dot-product values are calculated. VM v_{b1} is selected to be allocated on host h_2 and the capacity and distribute constraints are verified. Both the constraints are satisfied, but since v_{b1} is related to v_p by latency constraint, the pair (v_{b1}, h_2) is added to the Reserve_list and Forward_Allocate function is called. The Forward_Allocate function recognizes v_p to be the VM instance for which an allocation must be found, and generates the set of hosts $K=(h_2, h_3, h_4)$ that can be reached from h_2 in less than 5 latency units. Note that h_1 is not included due to the restrict constraint. Each host in K is analyzed to allocate v_p – host h_2 is in conflict with the distribute constraint since $(v_{b1},\ h_2)$ exists in the Reserve_list (i.e., tentatively allocated), and host h_3 forbids all the VMs to be allocated on it. Host h_5 satisfies the capacity and distribute constraints but VM v_p is related to latency constraint with v_{b2} (v_{b1} is not considered since it already exists in the Reserve_list). Similarly, (v_p, h_5) is added to the Reserve_list and Forward_Allocate identifies (v_{b2}, h_4) to be suitable for allocation. Since all the VMs are now tentatively allocated, satisfying all the constraints (including latency), VM instances are finally allocated.

V. Related Work

The problem of efficient resource management in Cloud computing has become a critical issue and is receiving an increasing attention. Existing solutions address resource management issues using two distinct processes: initial VM placement and run-time consolidation. In the first step, a process allocates VM instances on physical hosts and delivers them to the user. In the second step, another process continuously monitors the system and moves the VM instances to meet service provider's business goals. Several existing implementations such as the Eucalyptus Cloud manager [2] model the initial VM placement problem as a variant of the classical bin-packing problem and use simple heuristics to provision VM instances [15]. These solutions have limitations since bin-packing allows VMs to be placed besides or on top of each other on physical hosts (hence bin-packing based approaches implicitly violate our assumptions on resource usage model). As an alternative, the proposals introduced in [12], [13], [19] apply vector-packing model and leverage heuristics based on volume, norm-based greedy and dot-product mainly to perform run-time consolidation. In contrast to existing solutions, we merge the initial VM allocation and run-time consolidation processes into a single unified process and achieve a trade-off between initial placement and service provider's objectives.

An interesting line of research consists in developing resource management schemes that maximize service provider's profits (e.g., [4], [6]–[8], [10], [18], [20]). In particular, the solutions proposed in [4], [20] reduce the energy consumption costs of data centers by consolidating VM instances on minimum number of physical hosts. Similarly, the proposals in [10], [18] improve the performance and scalability of the system using efficient VM allocation and migration techniques, and the replica placement strategies in [6]–[8] allow delivery of fault tolerance as a service to user's applications. However,

TABLE II
AN EXAMPLE ILLUSTRATING THE WORKING OF THE VM PROVISIONING ALGORITHM

h	$\overrightarrow{h'}$	**Restrict**	**Forbid**	**Algorithm execution**	**Reserve_list**	**Allocation**
1	$(0.3, 0.4, 2)$	v_p	–	VMreq=$(v_{b1}, 0.22)(v_{b2}, 0.18)$		
				v_{b1}: no capacity (line 29)		
				v_{b2}: no capacity (line 29)		
2	$(0.6, 0.6, 2)$	–	–	VMreq=$(v_p, 0.13)(v_{b1}, 0.18)(v_{b2}, 0.16)$		
				v_{b1}: capacity ok (line 29)		
				distribute ok (line 30)		
				$(v_p, v_{b1}, 5)\in$*MaxLatency* found (line 33)	(v_{b1}, h_2)	
				Forward_Allocate(v_{b1}, h_2)		
				v_l=v_p		
				K=h_2, h_3, h_5		
				h_2: distribute constraint in conflict (line 19)		
				h_3: forbid constraint in conflict (line 11)		
				h_5: capacity ok (line 18)		
				distribute ok (line 19)		
				$(v_p, v_{b2}, 4)\in$*MaxLatency* (line 25, 3)	(v_{b1}, h_2), (v_p, h_5)	
				Forward_Allocate(v_p, h_5)		
				v_l=v_{b2}		
				K=h_1, h_2, h_4		
				h_1: no capacity (line 18)		
				h_2: no capacity (line 18)		
				distribute constraint in conflict (line 19)		
				h_4: capacity ok (line 18)		
				distribute ok (line 19)		
				latency ok (line 25)	(v_{b1}, h_2), (v_p, h_5), (v_{b2}, h_4)	$p(v_{b1})$:=h_2
						$p(v_p)$:=h_5
						$p(v_{b2})$:=h_4

as discussed in this paper, no solution categorizes and formalizes the security, reliability and performance requirements of both users and service providers, and provide an integrated and modular solution for VM allocation.

VI. CONCLUSIONS

We presented an approach to incorporate security constraints from users as well as service providers in managing resources in Cloud environments. We identified different kinds of requirements and presented a heuristics-based approach which takes them into account for allocating virtual machines to external hosts. Our work represents the first step towards a comprehensive inclusion of security requirements in the Cloud computing scenario and leaves space for immense future work, including consideration of a richer set of constraints and design of optimal algorithms for resource allocation.

ACKNOWLEDGMENTS

This work was supported by the Italian Ministry of Research within the PRIN 2008 project "PEPPER" (2008SY2PH4).

REFERENCES

[1] Amazon elastic compute cloud. http://aws.amazon.com/ec2/.
[2] Eucalyptus cloud manager. http://www.eucalyptus.com/.
[3] F. Hermenier, S. Demassey, and X. Lorca, "Bin repacking scheduling in virtualized datacenters," in *Proc of CP'11*, Perugia, Italy, Sep 2011.
[4] F. Hermenier, J. Lawall, J.-M. Menaud, and G. Muller, "Dynamic Consolidation of Highly Available Web Applications," INRIA, Tech. Rep. RR-7545, Feb 2011.
[5] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proc. of VEE'09*, Washington, DC, USA, Mar 2009, pp. 41–50.
[6] R. Jhawar and V. Piuri, "Fault Tolerance Management in IaaS Clouds," in *Proc. of ESTEL'12*, Rome, Italy, Oct 2012.
[7] R. Jhawar, V. Piuri, and M. Santambrogio, "A Comprehensive Conceptual System-Level Approach to Fault Tolerance in Cloud Computing," in *Proc. of IEEE SysCon'12*, Vancouver, Canada, Mar 2012, pp. 1–5.
[8] R. Jhawar, V. Piuri, and M. Santambrogio, "Fault tolerance management in cloud computing: A system-level perspective," *IEEE Systems Journal*, 2012 (to appear).
[9] F. Machida, M. Kawato, and Y. Maeno, "Redundant virtual machine placement for fault-tolerant consolidated server clusters," in *Proc. of NOMS'10*, Osaka, Japan, Apr 2010, pp. 32–39.
[10] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. of INFOCOM'10*, San Diego, California, USA, Mar 2010, pp. 1–9.
[11] K. Mills, J. Filliben, and C. Dabrowski, "Comparing VM-Placement Algorithms for On-Demand Clouds," in *Proc of CLOUD'11*, Washington, DC, USA, Jul 2011, pp. 91–98.
[12] M. Mishra and A. Sahoo, "On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach," in *Proc. of CLOUD'11*, Washington, DC, USA, Jul 2011.
[13] L. U. Rina Panigrahy, Kunal Talwar and U. Wieder, "Heuristics for vector bin packing," 2011, Microsoft Research, (unpublished).
[14] P. Samarati and S. De Capitani di Vimercati, "Data protection in outsourcing scenarios: Issues and directions," in *Proc. of ASIACCS'10*, Beijing, China, Apr 2010.
[15] S. S. Seiden, "On the online bin packing problem," *ACM Journal*, vol. 49, no. 5, Sep 2002.
[16] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *Proc. of SC'08*, Austin, TX, USA, Nov 2008, pp. 53:1–53:12.
[17] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *Journal of Parallel and Distributed Computing*, vol. 70, no. 9, Sep 2010.
[18] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proc. of WWW'07*, Alberta, Canada, May 2007, pp. 331–340.
[19] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Computer Networks*, vol. 53, no. 17, Dec 2009.
[20] J. Xu and J. A. B. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. of GREENCOM-CPSCOM'10*, Hangzhou, China, Dec 2010, pp. 179–188.