

Autonomic Allocation of Communicating Virtual Machines in Hierarchical Cloud Data Centers

Arwa Aldhalaan  
Volgenau School of Engineering  
George Mason University  
Fairfax, VA 22030, USA  
Email: aaldhala@masonlive.gmu.edu

Daniel A. Menascé  
Computer Science Department  
George Mason University  
Fairfax, VA 22030, USA  
Email: menasce@gmu.edu

Abstract—Cloud providers are typically hierarchically organized into interconnected data centers, each with a collection of racks of servers organized into clusters. The communication cost between two servers is a function of their relative location in the cloud infrastructure. Cloud consumers submit allocation requests for virtual machines, of different types and capacities, and provide an indication of the communication strength between all pairs of requested virtual machines. There is therefore a need for autonomic provisioning of virtual machines in a cloud environment. This paper formalizes the problem of finding an optimal allocation for the requested virtual machines that maximizes the cloud provider’s revenue, which depends on how close the requested machines are allocated. This paper presents efficient heuristic algorithms for this NP-hard problem. Experiments show the heuristics to significantly outperform an allocation strategy that is oblivious to the communication strength between virtual machines. The proposed heuristics were also shown to generate between 80% and 90% of the optimal revenue.

Keywords—cloud computing; virtual machines; virtual machine placement; autonomic computing.

I. INTRODUCTION

Autonomic computing deals with the design of self-optimizing, self-configuring, self-healing, and self-protecting computer systems [18]. One of the main applications of autonomic computing is on very large and complex computer systems (e.g., cloud computing infrastructures) for which it is virtually impossible for human beings to make low level decisions at run-time. In autonomic computing, human beings establish high level goals that are used by autonomic controllers that follow the MAPE-K loop [18] to (1) analyze data obtained through monitors, (2) plan steps for optimization, configuration, failure recovery, and/or protection against security attacks, and (3) execute these plans.

In [6], two authors of this paper presented an autonomic algorithm for the provision of virtual machines into servers of a cloud provider. The goal of that paper was to maximize the revenue (a function of the availability provided to each consumer) of the cloud provider subject to availability and capacity constraints.

In this paper we consider a different kind of problem and a different organization for the infrastructure of a cloud provider. In [6], the organization of the servers was considered to be flat. Here, we consider a hierarchical organization consisting of several interconnected data centers. Each data center has a collection of servers organized into clusters, which have several racks of servers. The communication cost between two servers varies significantly depending on their relative location in the infrastructure.

A cloud provider typically has a hierarchically organized networking infrastructure. The servers of a rack are connected through a local switch, which has some ports used for uplinks to a cluster-level switch that provides connectivity across all racks of a cluster. Cluster-level switches have uplinks to a data center switch, which allows servers from across different clusters of a data center to communicate. The data center switch also has uplinks to other data center switches so that servers can communicate across data centers. The communication latency increases and the bandwidth decreases as we move from same server, to different server but same rack, to different rack but same cluster, to different cluster but same data center, and to different data center [4].

Using the latency and bandwidth values in the example in [4], one can estimate that a 1-MB message would take 0.05 msec to be transmitted within a server (e.g., between two virtual machines on the same server), 10 msec between servers of the same rack, and 100 msec between servers in different racks of the same cluster. Thus, it is imperative that virtual machines that communicate among themselves be allocated as close as possible to each other.

There is a growing number of applications that require close communication between the virtual machines used to support the application. MapReduce [8] is an example of that; map tasks must communicate with reduce tasks to complete a job. Thus, when allocating virtual machines from a cloud provider, one should try to optimize the locality of communication. In addition, consumers of cloud provider services generally do not receive performance guarantees because cloud providers determine the allocation of virtual
Cloud consumers submit requests to the CP to allocate data center resources.
The contributions of this paper are: (1) A pricing model for cloud resource usage based on how close, communication-wise, VMs are allocated by a cloud provider. This pricing model provides incentives to the cloud provider to reduce performance uncertainties. (2) The formalization of the problem of finding an optimal allocation for the requested virtual machines that maximizes the cloud provider’s revenue, which depends on how close the requested machines are allocated. (3) An efficient heuristic algorithm to solve this NP-hard problem; the heuristic is shown experimentally to perform significantly better than an allocation strategy that is oblivious to the communication strength between virtual machines. In particular, the proposed heuristic was shown to generate between 80% and 90% of the optimal revenue. Because of the efficiency of the heuristic, it can be used to solve the VM placement problem in an online manner as new requests arrive from consumers.

The rest of this paper is organized as follows. Section II presents the assumptions and notation used throughout the paper. Section III discusses the two revenue models used in the paper: a linear revenue and an exponential revenue model. The next section formalizes the optimization model considered here. Section V presents the heuristic algorithm used to find a near optimal solution to the optimization problem. The next section provides experimental results that compare the revenue obtained through the heuristic, with that of an allocation method that is oblivious to communications costs, and to an upper bound of the optimal solution. Section VII briefly discusses some related work. Finally, section VIII presents some concluding remarks.

II. Problem Assumptions and Notation

Figure 1 illustrates the hierarchical infrastructure for a cloud provider considered in this paper. This infrastructure consists of various interconnected data centers typically situated in different geographical regions to improve business continuity in the face of natural disasters as well as for reducing response time by increasing proximity to a widespread set of consumers. Each data center has a number of servers organized in clusters (aka arrays) of racks, with each rack containing several servers.

The following are our assumptions regarding cloud consumers and providers:

- A Cloud Provider (CP) offers Infrastructure as a Service (IaaS) services to consumers, who can request a certain number of virtual machines (VM) of different types and capacities.
- The cloud provider has a number of data centers (DC). Each data center has a number of clusters of racks and each rack has a number of servers.
- There are several categories of cloud consumers who pay different amounts to obtain a better allocation for their virtual machines, i.e., an allocation that is more consistent with the communication needs of the consumer’s requested virtual machines.

- Cloud consumers submit requests to the CP to allocate a certain number of VMs. Each request indicates the type of each requested machine and also provides a communication strength index between each pair of requested VMs. This index, a number between 0 and 1, indicates the intensity of the communication between the various VMs. A value of 1 represents maximum communication coupling and 0 represents no communication between the requested VMs.
- The CP charges a fee from the consumer that depends on how well the CP is able to allocate the requested VMs so that VMs that have higher communication strength indices are allocated as close as possible to each other.
- The CP wants to maximize its revenue.

The following notation is used throughout the paper.

- $D$: number of data centers in the cloud provider infrastructure ($1 \leq d \leq D, d \in \mathbb{N}$).
- $C(d)$: number of clusters of racks in data center $d$ ($1 \leq c(d) \leq C(d), c(d) \in \mathbb{N}$)
- $R(c,d)$: number of racks in cluster $c(d)$ of data center $d$ ($1 \leq r(c,d) \leq R(c,d), r(c,d) \in \mathbb{N}$)
- $S(r,c,d)$: number of servers in rack $r(c,d)$ of cluster $c(d)$ of data center $d$. ($1 \leq s(r,c,d) \leq S(r,c,d), s(r,c,d) \in \mathbb{N}$)
- $N$: number of VM types offered by the CP ($1 \leq t \leq N, t \in \mathbb{N}$).
- $K$: number of VMs requested by a cloud consumer in each allocation request.
- $type(i)$: type of VM $i$.
- $P$: number of CP consumer categories. $1 \leq p \leq P, p \in \mathbb{N}$
The document contains a mathematical formulation related to cloud computing and revenue allocation. The main entities and notations used are:

- $\tilde{\mu} = (\mu_1, \ldots, \mu_K)$: vector of virtual machines requested by a cloud consumer. $\mu_k$ ($k = 1, \ldots, K$) is the type of the $k$-th virtual machine requested by a consumer in its allocation request. $1 \leq \mu_k \leq N$, $\mu_k \in \mathbb{N}$
- $C$: $K \times K$ communication strength matrix such that $0 \leq C[i, j] \leq 1$, $C[i, j] = C[j, i]$, $C[i, i] = 1$, $\forall i, j \in \{1, \ldots, K\}$. We discuss later how the values in the matrix $C$ can be estimated.
- $\beta = (\tilde{\mu}, C)$: allocation request coming from a cloud consumer. A request consists of the vector of VMs requested (including their types) and the communication strength matrix for the requested VMs.
- $X = \{x_{t,s(r,c,d)}\}$: CP state, which indicates the allocation of VMs to the servers of the CP infrastructure.
- $C_{s(r,c,d)}$: nominal capacity of server $s(r,c,d)$ measured in compute units.
- $d_t$: capacity needed to instantiate and operate a VM of type $t$ on a server measured in compute units.
- $A_i = (s, r, c, d)$: allocation of VM $i$ on server $s$ of rack $r$ of cluster $c$ of data center $d$.
- $x^i_{s,r,c,d}$: decision variable for the optimization problem. This value takes the value 1 if there is an allocation $A_i = (s, r, c, d)$ and 0 otherwise.
- $n_{t,s(r,c,d)} \in \mathbb{N}$: number of VMs of type $t$ allocated to server $s(r,c,d)$. Note that $n_{t,s(r,c,d)} = \sum_{i \text{ s.t. } \text{type}(i)=t} x^i_{s,r,c,d}$.
- $c_h \in [0, C_h]$ current available capacity of server $h$. The following capacity constraint must be satisfied: $c_h = C_h - \sum_{t=1}^{N} n_{t,h} \cdot d_t$, $\forall h$
- $r_{i,j}^p$: revenue obtained by the CP when allocating a pair of VMs $i$ and $j$ to consumers of category $p$ ($p = 1, \ldots, P$) depends on the type of the co-locations $A_i$ and $A_j$ according to Table I. A co-location indicates the relative proximity of two VMs in terms of their position in the hierarchical CP infrastructure.

### Table I

<table>
<thead>
<tr>
<th>Co-location Type ($\alpha$)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Same server</td>
<td>$A_i = A_j = (s, r, c, d)$</td>
</tr>
<tr>
<td>2 Different servers of the same rack</td>
<td>$A_i = (s_i, r, c, d), A_j = (s_j, r, c, d)</td>
</tr>
<tr>
<td>3 Different racks, same cluster</td>
<td>$A_i = (s_i, r_i, c, d), A_j = (s_j, r_j, c, d)</td>
</tr>
<tr>
<td>4 Different clusters, same data center</td>
<td>$A_i = (s_i, r_i, c_i, d_i), A_j = (s_j, r_j, c_j, d_j)</td>
</tr>
<tr>
<td>5 Different data centers</td>
<td>$A_i = (s_i, r_i, c_i, d_i), A_j = (s_j, r_j, c_j, d_j)</td>
</tr>
</tbody>
</table>

We first consider a revenue function $r_{i,j}^p(\alpha)$ that represents a linear decrease in revenue as the co-location type ($\alpha$) goes from 1 to 5. So,

$$r_{i,j}^p(\alpha) = \left[ \frac{r_{\text{min}}^p - r_{\text{max}}^p}{4} \cdot \alpha + \frac{5}{4} \cdot \frac{r_{\text{max}}^p - r_{\text{min}}^p}{4} \right] \times C[i, j]$$ (2)
where $\alpha$ is the co-location type (see column 1 of Table I) and $r_{\alpha,x}^{\max}$, $r_{\alpha,x}^{\min}$, and $C[i,j]$ are as defined above. Note that $r_{i,j}^{P}(1) = r_{\alpha,x}^{\max}$ and $r_{i,j}^{P}(5) = r_{\alpha,x}^{\min}$ in Eq. (2).

Another revenue model also considered here is one in which the revenue decreases exponentially with the co-location type $\alpha$. Thus,

$$r_{i,j}^{P}(\alpha) = \left(\frac{r_{\alpha,x}^{\max}}{r_{\alpha,x}^{\min}}\right)^{\frac{1}{4}} \times e^{\ln\left(\frac{r_{\alpha,x}^{\max}/r_{\alpha,x}^{\min}}{\alpha}\right) \times C[i,j]}. \quad (3)$$

As with Eq. (2), $r_{i,j}^{P}(1) = r_{\alpha,x}^{\max}$ and $r_{i,j}^{P}(5) = r_{\alpha,x}^{\min}$ in Eq. (3).

Note that the revenue function $r_{i,j}^{P}(\alpha)$ is a function of the allocations $A_i$ and $A_j$. Thus, $r_{i,j}^{P}(\alpha) = r_{i,j}^{P}(A_i, A_j)$.

IV. Optimization Problem

The optimization problem to be solved can now be expressed as:

$$\max \ R = \sum_{p=1}^{P} \sum_{A_i, A_j, i<j, i,j \in [1, \ldots, K]} r_{i,j}^{P}(A_i, A_j) \quad (4)$$

s.t.

(Server capacity constraint)

$$c_h = C_h - \sum_{t=1}^{N} n_{t,h} \cdot d_t \quad \forall h$$ \quad (5)

Note that, according to Eq. (1), $n_{t,h}$ depends on the VM allocations.

This optimization problem is NP-hard. The number of possible allocations is of the order of $H^K$ where $H$ is the total number of servers in the cloud infrastructure and $K$ is the number of VMs requested by the cloud consumer. Large data centers and cloud infrastructures (e.g., Google, Amazon, and Microsoft) have of the order of millions of servers. Thus, a request for 10 virtual machines would generate of the order of $10^{160}$ possible allocations considering one million servers.

For this reason, an efficient heuristic is required to find a near-optimal solution to this optimization problem.

V. Heuristic Algorithms

We first describe the Basic VM Allocation Heuristic (BVAH), which does not deallocate any already allocated VM in order to find a near-optimal placement for the requested VMs. Then, we describe the Advanced VM Allocation Heuristic (AVAH) that uses BVAH and considers the possibility of deallocating some of the most recently allocated VMs, allocating the VMs in the new request, and reallocating the deallocated VMs. Finally, in this section we also discuss an allocation strategy that we call NoComm, which allocates the VMs in a way equivalent to BVAH and AVAH but does not take into account the values of the communication strength index.

A. Basic VM Allocation Heuristic (BVAH)

At a high-level, the basic heuristic algorithm shown below builds a labeled undirected graph in which the vertices are the VMs requested and the edges represent the existence of a non-zero communication strength value between the vertices at the end points of the edge. The labels in the graph are the values of the communication strength indices (step 1).

We then build a maximum spanning tree for the graph (step 2). The rationale behind this step is to cover all VMs and obtain a maximum sum of the communication strengths, avoiding connections with low communication strength. The edges of the maximum spanning tree are sorted in step 3 in descending order of communication strength and stored in a list $L$.

Then, in the loop between steps 5 and 10, each edge $(v, w)$ in the list $L$ is examined from the highest to the lowest communication strength edge. If VMs $v$ and $w$ were already allocated then the algorithm considers the next edge in the list (step 6). If VM $v$ has been allocated but VM $w$ has not, then the algorithm invokes the AllocateCloseTo $(w, v)$ procedure that allocates VM $w$ as close as possible to VM $v$ (step 7). More about this procedure later. Step 8 is similar to step 7. In this case, VM $v$ is allocated as close as possible to the already allocated VM $w$. Finally, in step 9, VMs $v$ and $w$ were not allocated yet. In this case, the CoAllocate procedure is used to allocate these two VMs as close as possible to each other.

- **Step 1**: Build the undirected graph $G = (V, E, L)$ where the set of nodes $V$ corresponds to each of the VMs requested by a consumer request $\beta = (\bar{\mu}, C)$, the set of edges $E = \{(v, w) | v, w \in V; C[v, w] \in (0, 1)\}$, and $L$ is a set of edge labels such that the label of edge $(v, w)$ is $C[v, w]$.

- **Step 2**: Build the maximum spanning tree, $T$, for the graph $G$. This can be done using any of the existing algorithms for finding the minimum spanning tree (e.g., the Kruskal algorithm [21]) by using the negative values of the edge labels in $L$. The number of edges in $T$ is $K - 1$ where $K$ is the number of nodes in the tree (equal to the number of VMs requested in $\beta$).

- **Step 3**: Sort the edges of the tree $T$ in descending weight order. Let this list be denoted as $L$ and its elements denoted as $L(1), \ldots, L(K - 1)$. So, $L(1)$ corresponds to the edge in $T$ with the highest communication strength. Note that the edges in $L$ are not necessarily the $K - 1$ edges in $G$ with the largest weight.

- **Step 4**: [Allocation loop] $k \leftarrow 1$.

- **Step 5**: If $k = K$ then Stop else let $L(k) = (v, w)$.

- **Step 6**: If VMs $v$ and $w$ have already been allocated Go to Step 10.

- **Step 7**: If VM $v$ has been allocated but VM $w$ has not been allocated then AllocateCloseTo $(w, v)$; Go to Step 9.
10. **Step 8:** If VM \( w \) has been allocated but VM \( v \) has not been allocated then AllocateCloseTo \((v, w)\); Go to Step 10.

- **Step 9:** If VMs \( v \) and \( w \) have not been allocated then CoAllocate \((w, v)\).
- **Step 10:** \( k \leftarrow k + 1 \); Go to Step 5.

We illustrate the operation of the algorithm through a small example. Consider the undirected graph of Fig. 2(a) and the corresponding maximum spanning tree in Fig. 2(b).

![Figure 2](image)

**Figure 2.** Example of the Operation of the BVAH Algorithm.

Then, the list \( \mathcal{L} \) is \( \{(4,5), (1,2), (1,6), (1,3), (3,4)\} \). According to the algorithm above, the following sequence of allocation calls will take place: (a) CoAllocate \((4,5)\); (b) CoAllocate \((1,2)\); (c) AllocateCloseTo \((6,1)\); and (d) AllocateCloseTo \((3,1)\).

Consider now that the CP’s infrastructure has a single data center with two clusters with two racks each and two servers per rack. Assume for simplicity that all requested VMs have unit capacity. Table II illustrates the state of the CP as allocations (a)-(d) above take place. The first row after the headers in the table shows the available capacity of the CP before any VMs are allocated. For example, server \( S1 \) of rack 1 of cluster 1 has available capacity equal to 1 unit and server \( S1 \) of rack 2 of the same cluster has an available capacity of 2 units. The following four pairs of rows correspond to the four allocations (a)-(d). The first row of each pair shows where the VMs are allocated and the second row shows the remaining capacity after the allocation.

During allocation (a), VMs 4 and 5 are co-allocated at the same server (server \( S1 \) of rack 2 of cluster 1). After this allocation, the remaining capacity of this server becomes equal to zero. Allocation (b) requires that VMs 1 and 2 be allocated as close as possible. The only possibility is for these VMs to be allocated on servers \( S1 \) and \( S2 \) of the same rack (rack 2) of cluster 2. Next, allocation (c) requires VM 6 to be allocated as close as possible to VM 1, which is allocated at rack 2 of cluster 2. The closest available server on that cluster is server \( S2 \) on rack 1. Finally, allocation (d) requires that VM 3 be allocated as close as possible to VM 1, which is allocated in cluster 2. It turns out that at this point there is no available capacity in cluster 2. Thus, VM 3 is allocated in cluster 1.

_**Table II** Allocation Example._

<table>
<thead>
<tr>
<th>Data Center</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rack 1</td>
<td>Rack 2</td>
<td>Rack 1</td>
</tr>
<tr>
<td>( S1 )</td>
<td>( S2 )</td>
<td>( S1 )</td>
</tr>
<tr>
<td>-</td>
<td>4,5</td>
<td>1,1</td>
</tr>
<tr>
<td>(a)</td>
<td>1,0</td>
<td>0,1</td>
</tr>
<tr>
<td>(b)</td>
<td>4,5</td>
<td>1,2</td>
</tr>
<tr>
<td>(c)</td>
<td>4,5</td>
<td>6,2</td>
</tr>
<tr>
<td>(d)</td>
<td>3,4,5</td>
<td>6,1,2</td>
</tr>
</tbody>
</table>

We now explain the algorithm used by procedures AllocateCloseTo and CoAllocate. Algorithm 1 shows the pseudocode for the CoAllocate algorithm. It compares the remaining capacity at each server with the required capacity by a VM and attempts to place VMs \( v \) and \( w \) on the same server (line 3), same rack (line 6), same cluster (line 9), same data center (line 12). If everything fails, VMs \( v \) and \( w \) are allocated on different data centers if there is available capacity.

Algorithm 2 shows the process used to allocate VM \( v \) as close as possible to an already allocated VM \( w \). The algorithm first attempts to allocate VM \( v \) at the server where VM \( w \) is allocated (lines 4-5). If this fails, it attempts to allocate VM \( v \) on the same rack as \( w \) (lines 7-8). If not possible, it attempts the allocation in the same cluster (lines 10-11). If this attempt is not successful, an allocation on the same data center is tried (lines 13-14). If that also fails, the algorithm attempts to allocate VM \( v \) at any other data center.

An efficient implementation of algorithms 1 and 2 would use a \( B^* \)-tree [7] in which the keys (stored in ascending order) are the result of concatenating the data center id, cluster id, rack id, and server id. The value associated with a key is the remaining capacity of the server at the location indicated by the key. In a \( B^* \)-tree, all leaf nodes are typically linked so that one can traverse just the leaves if necessary. A leaf node contains several key-value pairs. By traversing the leaf nodes from left to right one could more efficiently find available capacity on the same server, same rack, same cluster, and same data center. One can also obtain, as required by algorithm 2, with a few steps the available capacity at a given server (even when the CP has millions of servers).

**B. Advanced VM Allocation Heuristic (AVAH)**

This advanced allocation procedure is performed when a new allocation request \( \beta \) arrives. Consider the set \( D \)
Algorithm 1 CoAllocate Algorithm
1: CoAllocate (VM v, VM w);
2: /* allocate VMs v and w as close as possible of each other */
3: if \( \exists \) server \( s \) such that \( c_v \geq d_v + d_w \) then
4: allocate \( v \) and \( w \) on \( s \)
5: else
6: if \( \exists \) a rack with servers \( s_1 \) and \( s_2 \) s.t. \( c_{s_1} \geq d_v \) and \( c_{s_2} \geq d_w \) then
7: allocate \( v \) on \( s_1 \) and \( w \) on \( s_2 \)
8: else
9: if \( \exists \) a datacenter with servers \( s_1 \) and \( s_2 \) s.t. \( c_{s_1} \geq d_v \) and \( c_{s_2} \geq d_w \) then
10: allocate \( v \) on \( s_1 \) and \( w \) on \( s_2 \)
11: else
12: if \( \exists \) datacenters \( d_1, d_2 \) with servers \( s_1 \) and \( s_2 \) s.t. \( c_{s_1} \geq d_v \) and \( c_{s_2} \geq d_w \) then
13: allocate \( v \) on \( s_1 \) and \( w \) on \( s_2 \)
14: else
15: no allocation possible
16: end if
17: end if
18: end if
19: end if
20: end if
21: end if
22: end if
23: end if

Algorithm 2 AllocateCloseTo Algorithm
1: AllocateCloseTo (VM v, VM w);
2: /* allocate VM v as close as possible to VM w */
3: let \( A_w = (s, r, c, d) \)
4: if \( c_v \geq d_v \) then
5: allocate \( v \) on \( s \)
6: else
7: if \( \exists \) a server \( s_1 \) on rack \( r \) s.t. \( c_{s_1} \geq d_v \) then
8: allocate \( v \) on \( s_1 \)
9: else
10: if \( \exists \) a server \( s_1 \) on data center \( d \) s.t. \( c_{s_1} \geq d_v \) then
11: allocate \( v \) on \( s_1 \)
12: else
13: no allocation possible
14: end if
15: end if
16: if \( \exists \) a server \( s_1 \) on data center \( d_1 \) \( (d_1 \neq d) \) s.t. \( c_{s_1} \geq d_v \) then
17: allocate \( v \) on \( s_1 \)
18: else
19: no allocation possible
20: end if
21: end if
22: end if
23: end if
24: end if

consisting of the \( M \) most recent allocation requests and the corresponding revenues obtained by these allocations. When a new VM allocation request \( \beta \) arrives, the advanced heuristic performs the following steps, which implements a sort of greedy hill climbing:

- **Step 1**: Allocate the VMs in \( \beta \) using the Basic VM Allocation Heuristic (BV AH) described in the previous subsection and compute the total revenue \( R \) obtained by this allocation.

- **Step 2**: For each request \( \beta_d \in \mathcal{D} \) do
  - **Step 2.1**: Let \( R_d \) be the revenue originally generated by request \( \beta_d \). Deallocate the VMs in \( \beta_d \).
  - **Step 2.2**: Allocate the VMs in the request \( \beta \) using BV AH and obtain the revenue \( R^{\text{new}} \) generated by this allocation.
  - **Step 2.3**: Allocate the VMs in the request \( \beta_d \) using BV AH and obtain the revenue \( R^{\text{new}}_d \) generated by this allocation.
  - **Step 2.4**: If \( (R^{\text{new}}_d + R^{\text{new}}) > (R_d + R) \) then go to Step 3.
  - **Step 2.5**: Deallocate \( \beta_d \); Deallocate \( \beta \); Allocate \( \beta_d \) and \( \beta \) in this order using BV AH.

- **Step 3**: Update \( \mathcal{D} \) by removing the least recent request and adding the request \( \beta \) to \( \mathcal{D} \).

The AVAH allocation procedure mitigates the problem of cloud infrastructure fragmentation because it deallocates the VMs requested in the \( M \) most recent requests and reallocates them using BV AH.

C. No Communication (NoComm) Allocation Strategies

For purposes of comparison, we consider two variants of the BV AH and AV AH VM allocation strategies. Both are first fit strategies based on ignoring the values of the communication strength index for allocation purposes only. These strategies can be easily implemented by making \( C[i,j] = 1 \) \( \forall i, j \). Clearly, the revenue \( r^p_{i,j} \) should be still calculated using the original values of the communication strength indices in the matrix \( C \).

We call these modified BV AH and AV AH strategies B-NoComm and A-NoComm, respectively. They consider the fact that the CP infrastructure is organized hierarchically and has different communication costs depending on the relative location of VMs in the cloud and also consider the capacity constraints.
VI. EXPERIMENTAL RESULTS

We implemented the various strategies in MatLab. Table III shows the parameters used in the experiments. We consider 1,600 servers organized in two data centers, each with ten clusters with four racks each. There are 20 servers per rack. There are three types of VMs ($N = 3$) offered by the CP and their capacities are 1, 2, and 4 compute units, respectively. Each request requires 10 VMs and each server has a capacity equal to 10 compute units. There are three categories of consumers ($P = 3$).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>2</td>
</tr>
<tr>
<td>$C(d)$</td>
<td>10 for all data centers</td>
</tr>
<tr>
<td>$R(c, d)$</td>
<td>4 for all clusters</td>
</tr>
<tr>
<td>$S(r, c, d)$</td>
<td>20 for all racks</td>
</tr>
<tr>
<td>$N$</td>
<td>3</td>
</tr>
<tr>
<td>$P$</td>
<td>3</td>
</tr>
<tr>
<td>$d_t$</td>
<td>1, 2, and 4</td>
</tr>
<tr>
<td>$K$</td>
<td>10</td>
</tr>
<tr>
<td>$C_k(r, c, d)$</td>
<td>10 for all servers</td>
</tr>
<tr>
<td>$M$</td>
<td>20</td>
</tr>
<tr>
<td>$r^\text{max}$</td>
<td>5.5, 6.7, and 7.9 for $p = 1, 2, 3$</td>
</tr>
<tr>
<td>$r^\text{min}$</td>
<td>1, 2, and 3</td>
</tr>
</tbody>
</table>

Table III
PARAMETER VALUES FOR THE EXPERIMENTS.

We randomly generated 30 workloads composed of 600 requests each for a total of 18,000 requests. In each workload, the communication strength matrix C, the virtual machine types, and customer class for each request were randomly generated. We then used the same 30 workloads to compare all VM allocation strategies described above. We computed the average per-request revenue and the cumulative revenue along with their 95% confidence intervals for all 30 workloads and all requests.

We plotted the CP’s normalized capacity utilization $\rho$, defined below.

$$\rho = \frac{\sum_{h=1}^{H} \sum_{t} n_{t, h} \times d_t}{\sum_{h=1}^{H} C_h}.$$  \hspace{1cm} (6)

Note that the numerator in Eq. (6) is the total allocated capacity, expressed as the summation over all $H$ servers and over all VM types of the number $n_{t, h}$ of VMs of type $t$ allocated at server $h$ multiplied by the required capacity $d_t$ of a VM of type $t$. The denominator of this equation is simply the total capacity available in all servers.

The workload for the experiments was generated in a way that maintains $\rho$ below a target value of around 0.9.

We also defined a very-easy-to-compute upper bound (which we call UpperOPT) for the optimal per-request revenue by considering that there is no space constraint on the servers. That implies that $\alpha = 1$ in the revenue equations (2) and (3).

Figure 3 shows the variation of the normalized capacity utilization $\rho$ as the 600 requests are processed. The figure shows that the maximum value of $\rho$ is very close to 90% but slightly below. The figure also shows that when 300 requests were processed the normalized allocated capacity is slightly below 45%.

Figure 4 shows a comparison between AV AH (top) and BVA H (bottom) allocation strategies for the linear revenue function. The graphs show that both strategies are identical until the 20-th request because the value of $M$ (number of requests that are eligible for deallocation in Step 2.1 of the AV AH strategy) is equal to 20 in our experiments. After that point, AV AH shows a marked advantage over BVA H because AV AH is capable of improving the revenue by switching the order of allocations between a current request and a previously allocated request if this improves the revenue. The average revenue for BVA H is $40.7 \pm 0.14$ at the 95% confidence level and for AV AH is $47.1 \pm 0.18$. Thus, AV AH provides a 17% higher average revenue than BVA H and performs better than BVA H at the 95% confidence level.

Figure 5 shows a comparison between AV AH (top) and A-NoComm (bottom) when the revenue function is linear. When the allocation strategy does not consider the influence of the communication strength between VMs an inferior allocation in terms of revenue is obtained. In fact, the average revenue for the AV AH case is $47.1 \pm 0.18$ and that of A-NoComm is $31.1 \pm 0.12$. Therefore, AV AH provides a 51% better revenue and is better than A-NoComm at the 95% confidence level.

A similar graph to that in Fig. 5 is shown in Fig. 6 for the exponential revenue function. In this case, AV AH provides an average revenue of $41.8 \pm 0.12$ while A-NoComm provides an average revenue of $30.9 \pm 0.12$. Thus, AV AH generates 36% more revenue than A-NoComm and is better than A-NoComm at the 95% confidence level.
Figure 7 compares the average revenue of BVAH (top) and B-NoComm (bottom) for a linear revenue function. The figure clearly shows that BVAH is superior than B-NoComm. In fact, the average revenue generated by BVAH is $40.7 \pm 0.14$ and by B-NoComm is $32.5 \pm 0.13$. Thus, the average revenue generated by BVAH is 25% superior than that of B-NoComm and BVAH is better than B-NoComm at the 95% confidence level.

Figure 8 is similar to Fig. 7 except that in this case the exponential revenue function is used. Similarly to the AVAH case, there is marked separation between the two strategies. BVAH’s average revenue is $40.6 \pm 0.11$ while B-NoComm is $26.2 \pm 0.11$. Thus, BVAH generates 55% more revenue than B-NoComm and is better than B-NoComm at the 95% confidence level.

Table IV shows a summary of the results as well as the UpperOPT results that show that BVAH achieves more than 93% (for linear and exponential revenue functions) and AVAH more than 89% (for linear function) and 80% (for the exponential function) of the upper bound for the optimal solution.

VII. RELATED WORK

The problem of VM placement is usually formulated as an optimization problem and different solution techniques were proposed with different goals and constraints, from response time to availability and power saving [17]. Frameworks for an exact solution were presented in [14], [19], [25], [28] and heuristics can be found in [16], [13], [9], [10]. In [11], auto-scaling is used to minimize cost while meeting service level obligations. A unified method that considers policies to place application replicas and distribution of client requests among them was discussed in [24]. In [28], a bin-packing formulation that maximizes resource satisfaction in a datacenter is proposed. The minimization of VM migrations during reallocation of resources was considered in [19] and [20] proposes simple models to predict the effect of storage contention when multiple VMs are consolidated on the same server. Principles for monitoring and analyzing systems used for managing large-scale data centers are described in [22].

Heuristics are presented in [5] for dynamic VM consolidation with the goal of minimizing energy consumption while meeting CPU performance SLAs. The work in [6] considered the maximization of revenue given availability constraints. Heuristics to minimize power consumption are proposed in [13], [27]. Rodero et al. presented energy-efficient application-aware online provisioning of virtualized clouds and data centers [26].
Resource allocation policies for cloud virtualized environments that identify performance and energy trade-offs and provide a priori availability guarantees for cloud end-users are presented in [1]. In [10], the authors propose to use horizontal and vertical scaling of VMs to optimize resource usage and the reconfiguration cost incurred due to scaling. The authors assume that SLAs are violated when the CPU utilization exceed a specified threshold.

All above mentioned solutions do not consider the problem of maximizing the revenue of a cloud service provider given communication patterns among VMs in a hierarchically organized cloud infrastructure. There are however a few notable exceptions. One is the work of Hu et al. [15], which shows how to collect information about intra-ensemble VM interactions for co-locating communicating VMs in order to reduce the consumption of bi-section bandwidth. The work of Ballani et al. [2] uses simulation to argue for a pricing model that is location independent to cloud consumers. Their work proposes dynamic resource provisioning, which essentially charges consumers based on the dominant resource, i.e., the greater of the occupancy and network price. The work in [2] does not discuss any heuristic for VM allocation based on VM communication patterns. In [3], Ballani et al. present the design of virtual network abstractions that capture the trade-off between the performance guarantees offered to cloud consumers, their costs and the provider revenue. Their work describes Oktopus, a system that implements their abstractions.

The authors in [12] address the problem of VM placement in a large scale data center with compute, network, and availability constraints as one integrated solution. Their technique is based on cold spots which is a collection of compute nodes that provide high availability. They then cluster connecting VMs to allocate them in the cloud using a graph-based search algorithm. Our technique uses a heuristic algorithm that builds a maximum spanning tree to determine the placement order of VMs based on their communication strength in order to place them in close proximity using hill climbing with the goal of maximizing the revenue.

In [30], the author provides an algorithm for VM placement that takes a collection of VMs called pattern and finds a near optimal deployment in the cloud that satisfies availability and capacity constraints. Their algorithm is based on the importance sampling technique. Also, the authors in [29] used the statistical sampling method (cross-entropy) to solve a VM placement problem where VMs are clustered based on their communication needs. They consider communication and availability constraints. Our paper introduces four heuristic algorithms for allocating communicating VMs in a hierarchal data center with the goal of maximizing the revenue subject to capacity constraints.

The authors in [23] use traffic-aware placement of VMs to improve network scalability. Their heuristic algorithm
partitions VMs and hosts into clusters, then places the VM clusters in close proximity with the goal of minimizing network traffic. Their input includes a traffic matrix and a cost matrix among host machines. Our work uses a communication matrix as input to determine the communication strength between VMs, and then uses a heuristic algorithm to solve the placement problem with the goal of maximizing the revenue.

VIII. CONCLUDING REMARKS

This paper considered the problem of optimal allocation of VMs in a hierarchically organized CP infrastructure. Servers are organized into racks, which are organized into clusters, which are organized into data centers. The communication cost increases as we move from same server, to different server but same rack, to different rack but same cluster, to different cluster but same data center, and to different data center. The VM allocation problem deals with maximizing the CP’s revenue, which is a function of the relative placement of servers and of the degree of communication intensity between VMs. The results of the paper show that consumers can benefit if they are able to provide accurate information about their application needs and cloud providers can increase their revenue if they place requested VMs as close as possible based on communication needs.

A basic heuristic (BVAH) and an advanced heuristic (AVAH) were presented to find a near optimal solution to this NP-hard problem. We also presented versions of BVAH and AVAH, called B-NoComm and A-NoComm, which ignore the communication strength among VMs when allocating VMs. The experiments showed that AVAH is better than BVAH and that BVAH (AVAH) is better than B-NoComm (A-NoComm) at the 95% confidence level. These experiments also showed that BVAH and AVAH generate between 80% and 90% of the upper bound of the optimal revenue.

REFERENCES


