A Scalable Algorithm for Placement of Virtual Clusters in Large Data Centers

Asser N. Tantawi
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
tantawi@us.ibm.com

***Abstract*—We consider the problem of placing virtual clusters, each consisting of a set of heterogeneous virtual machines (VM) with some interrelationships due to communication needs and other dependability-induced constraints, onto physical machines (PM) in a large data center. The placement of such constrained, networked virtual clusters, including compute, storage, and networking resources is challenging. The size of the problem forces one to resort to approximate and heuristics-based optimization techniques. We introduce a statistical approach based on importance sampling (also known as cross-entropy) to solve this placement problem. A straightforward implementation of such a technique proves inefficient. We considerably enhance the method by biasing the sampling process to incorporate communication needs and other constraints of requests to yield an efficient algorithm that is linear in the size of the data center. We investigate the quality of the results of using our algorithm on a simulated system, where we study the effects of various parameters on the solution and performance of the algorithm.**

***Keywords*-application placement; cloud management; virtual clusters; importance sampling; combinatorial optimization; cross-entropy;**

I. Introduction

As cloud computing matures, the demand for virtual resources is changing from units of virtual machines (VM) to a collection of heterogeneous VMs with communication demand among them as well as availability constraints. Such a collection forms a virtual cluster where, once deployed in the physical infrastructure, the requester user launches a distributed application where components of the applications run on different reliable VMs with defined communication needs. The problem for the cloud provider is to efficiently place such a virtual cluster in the cloud on physical machines (PM), in such a way that the availability constraints are satisfied, the virtual cluster experiences good performance, and the rejection rate is kept at minimum. Further, the time to obtain a placement decision should be reasonably small and it should scale with the size of the cloud. A less constrained problem than the one considered in this paper was shown to be NP-Hard [1]. General techniques for solving bin packing and/or mixed integer programming problems may be applicable, but are usually inefficient since heuristics specially tailored to the problem at hand should result in more efficient solutions. Recently, there were research attempts to address those issues, though with some limitations. In [1], the authors present heuristic placement algorithms based on graph decomposition, but only in the case where one VM is placed on a PM. In [2], a heuristic algorithm based on clustering is presented and is shown to work when placing in an empty system.

A similar problem exists in the area of virtual networking. Notably, a divide-and-conquer approach is presented in [3] where a collection of connected physical resources is first identified as the target for placement, instead of the entire system.

Looking at the mapping of a virtual cluster to PMs as a mapping of a graph representing the interconnected VMs in the virtual cluster to a graph representing the interconnected PMs through some communication network is not new. In the area of parallel processing and grid computing, a similar problem exists where a task graph is to be mapped onto a system graph. An approximate algorithm based on ordering the nodes in the task graph in such a way that communication overhead is minimized is presented in [4], [5]. Further, the cross entropy technique [6] which is analogous to importance sampling has also been considered [7]. The only drawback is that one has to analyze a large number of samples, hence hindering the possibility of dealing with a large system.

In this paper we are concerned with the problem of placing patterns of VMs with some networking demands and availability constraints onto a large-sized physical infrastructure. We introduce a placement algorithm that is based on importance sampling, where we bias the sampling to accommodate the problem constraints and the communication demand. This biasing technique is shown to lead to efficient placement solutions and to exhibit a linear complexity in the system size.

The paper is organized as follows. We describe the physical and virtual components of the cloud, along with communication and availability definitions in section II. The performance measures and objectives are introduced in section III. In section IV we state the placement optimization problem. Our placement algorithm is described in section V. Simulation results are presented in section VI.

II. System description

We consider a cloud system which consists of a set of physical machines (PM) that are connected through switches (SW) via links (LK). A PM hosts virtual machines (VM). A collection of VMs make up a virtual cluster, or pattern, which is the unit of deployment in the cloud. We proceed to describe each of the above entities and their relationships in more detail

A. Physical machines

Let $\mathcal{PM}$ denote the set of n_{pm} physical machines in the cloud, $n_{pm} = |\mathcal{PM}|$. We will refer to an element in the set as $pm_i, i = 1, \cdots, n_{pm}$. Each PM provides a set of resources, $\mathcal{R}$, consisting of resources $r_k, k = 1, \cdots, n_r$. Examples of such resources are CPU, memory, and disk storage. The total capacity of resource r_k on pm_i is denoted by $c_{i,k}$. The demand (also referred to as usage) of such a resource is denoted by $d_{i,k}$, $d_{i,k} \leq c_{i,k}$.

B. Physical network

A PM is connected to one or more switches via links. Switches are interconnected via links. This interconnection network forms a graph where the vertices are PMs and SWs, and the edges are LKs. Let $\mathcal{SW}$ denote the set of switches. A particular element in $\mathcal{SW}$ is referred to as $sw_i, i = 1, \cdots, n_{sw}$, where n_{sw} is the number of switches, $n_{sw} = |\mathcal{SW}|$. Furthermore, let $\mathcal{LK}$ denote the set of links. And, we refer to an element in $\mathcal{LK}$ as $lk_i, i = 1, \cdots, n_{lk}$, where n_{lk} is the number of links, $n_{lk} = |\mathcal{LK}|$. Each link provides bandwidth for communication. The total bandwidth capacity of lk_i is denoted by b_i. The demand (also referred to as usage) of such a link is denoted by a_i, $a_i \leq b_i$. We assume that the switches are fast and that communication delay is solely due to link congestion. A path $h_{i,j}$ between pm_i and pm_j consists of an ordered set of links $\{lk_{\pi_{i,j}(1)}, lk_{\pi_{i,j}(2)}, \cdots\}$, with path length denoted by $\eta_{i,j} = |h_{i,j}|$. For convenience we define $w_i(l), i = 1, 2, \cdots, n_{pm}$, and $l = 0, \cdots, L$ as the set of PMs such that for $pm_j \in w_i(l)$ we have $\eta_{i,j} = l$.

C. System availability

We imagine that a data center is partitioned into a hierarchy of availability zones, where PMs in the same zone have similar availability characteristics. In such a case, one may model this hierarchy as a tree, where the leaves are the PMs and an intermediate node represents a zone of availability. Let $v_{i,j}$ be the probability that at least one of pm_i or pm_j is available, i.e. one minus the probability that both pm_i and pm_j are down. (Hence, $v_{i,j} = v_{j,i}$ and $v_{i,i}$ is the probability that pm_i is available.) As such, we associate an availability level, V_l, $l = 0, \cdots, L$, for a node at level l in the tree, where $l = 0$ represents the leaves and $l = L$ represents the root of the tree with height L. We assume that $V_0 < V_1 < \cdots < V_L$, since two PMs in $distant$ availability zones have less common components and hence higher chance of having at least one of them available. Using this tree model, two PMs pm_i and pm_j with the lowest common ancestor at level l have $v_{i,j} = V_l$. (Clearly, $v_{i,i} = V_0$). For convenience we define $g_i(l), i = 1, 2, \cdots, n_{pm}$, and $l = 0, \cdots, L$ as the set of PMs such that for $pm_j \in g_i(l)$ we have $v_{i,j} = V_l$.

D. Virtual machines

Each VM is characterized by a set of resource demands, one per resource type in the set $\mathcal{R}$. These resource demands are taken into consideration when placing a particular VM onto a PM, making sure that there is enough available resource capacity on the PM to satisfy the resource demand of the VM. We refer to the PM which hosts vm_i as $pm(vm_i)$. Furthermore, a pair of VMs, say vm_i and vm_j may have bandwidth demand which we denote by $\lambda_{i,j}$. Again, such demands need be satisfied by all the links along the path connecting $pm(vm_i)$ and $pm(vm_j)$.

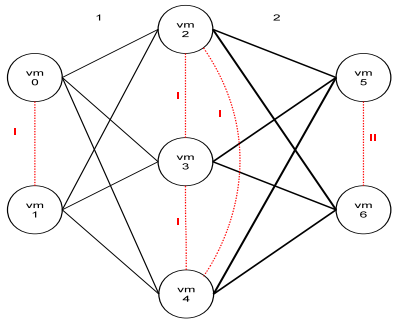


Fig. 1. An example multi-tier pattern.

E. Virtual clusters (Patterns)

A virtual cluster is a collection of VMs that make up a deployable unit which we refer to as *pattern*. Hence, a pattern is defined as a set of VMs, along with the resource demands of the VMs and the pairwise bandwidth demand among the VMs in the pattern. In particular, let's consider pattern $\mathfrak{p}$. Denote the number of VMs in $\mathfrak{p}$ by $n_{vm}(\mathfrak{p})$. They form the set $\mathcal{VM}(\mathfrak{p}) = \{vm_1(\mathfrak{p}), vm_2(\mathfrak{p}), \cdots, vm_{n_{vm}(\mathfrak{p})}(\mathfrak{p})\}$. The communication bandwidth demand of $\mathfrak{p}$ may be represented by a matrix $[\lambda_{i,j}]$, where $1 \leq i, j \leq n_{vm}(\mathfrak{p})$. We assume that this matrix is symmetrical with zero diagonal, i.e. the bandwidth requirements among VMs in a pattern may be represented by an undirected graph where the nodes represent the VMs, the edges represent pairwise communications, and the weight of an edge represents the amount of bandwidth demand between a pair of distinct VMs.

For pattern $\mathfrak{p}$, we express availability requirements as follows. Let $\mathcal{S}(\mathfrak{p}) \subset \mathcal{VM}(\mathfrak{p}) \times \mathcal{VM}(\mathfrak{p})$ be a set of distinct pairs of VMs in $\mathfrak{p}$. A pair $(vm_i, vm_j) \in \mathcal{S}(\mathfrak{p})$ has an availability constraint specified as $v_{pm(vm_i),pm(vm_j)} = \alpha$. This availability requirement is satisfied if $\alpha \leq V_l$ and $pm(vm_j) \in g_k(l)$, where $pm(vm_i) = pm_k$ for some $l, 0 \leq l \leq L$.

We denote by $\pi(\mathfrak{p})$ a particular placement of pattern $\mathfrak{p}$. In other words, $\pi(\mathfrak{p})$ maps each VM in$\mathcal{VM}(\mathfrak{p})$ to a PM in such a way that (1) the resource requirement of the VM is satisfied by this PM, (2) the communication demand between this VM and other VMs in $\mathfrak{p}$ is satisfied by the links of the communication network, and (3) the pairwise availability requirements are satisfied. We write such a placement as $\pi(\mathfrak{p}) = \{(vm_i, pm_m) \mid pm(vm_i) = pm_m, \forall vm_i \in \mathcal{VM}(\mathfrak{p})\}$.

F. An example

Consider pattern $\mathfrak{p}$, depicted in Figure 1, which consists of 7 VMs, $vm0$ through $vm6$, arranged in three tiers $\{vm0, vm1\}$, $\{vm2, vm3, vm4\}$, and $\{vm5, vm6\}$, respectively. Communication demands are shown as solid edges. The communication

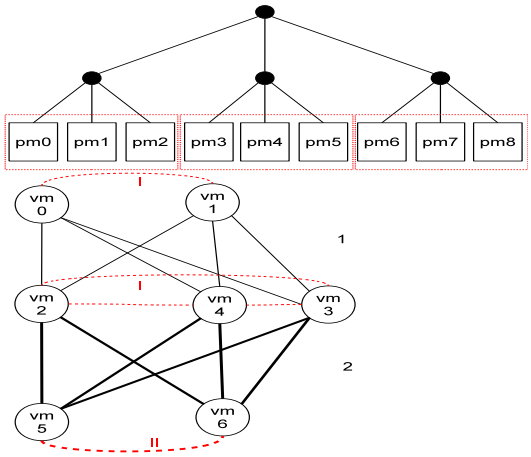


Fig. 2. Placement of the multi-tier pattern.

demand between VMs in tier 1 and VMs in tier 2 is 1 unit per each pair of VMs. Further, the communication demand between tier 2 and tier 3 is 2 units per each pair of VMs. Availability constraints, depicted as dashed lines, are specified among VMs in the same tier. For a given constraint, its availability α is represented as a Roman value of the corresponding level $l, 0 \leq l \leq L$, where $\alpha \leq V_l$.

Consider a system with 9 PMs and a tree network of degree 3, hence a 2-level tree, as shown in Figure 2. The PMs are partitioned into three blade centers: $\{pm0, pm1, pm2\}$, $\{pm3, pm4, pm5\}$, and $\{pm6, pm7, pm8\}$, respectively. Thus, $vm0$ and $vm1$ need to be placed in two different PMs. Similarly, $vm2$, $vm3$, and $vm4$ need to be placed on separate PMs. However, $vm5$ and $vm6$ need to be placed in two different PMs in different blade centers. A valid placement of this pattern is depicted in Figure 2, where $\pi(\mathfrak{p}) = \{(vm0, pm0), (vm1, pm3), (vm2, pm0), (vm3, pm5), (vm4, pm3), (vm5, pm0), (vm6, pm3)\}$. This placement results in a certain network load. For example, the core link connecting blade center 1 and blade center 2 will carry the sum of network demand between $\{(vm0, vm3), (vm0, vm4), (vm1, vm2)\}$, each of 1 unit, and between $\{(vm2, vm6), (vm3, vm5), (vm4, vm5)\}$, each of 2 units, resulting in a total of 9 units.

III. Performance Evaluation

A. System performance

System performance is characterized by resource utilization and delay measures. We define the utilization, $\rho_{i,k}$, of resource r_k on pm_i as $\rho_{i,k} = u_{i,k}/c_{i,k}$. Further, let ρ_k be the random variable representing the utilization of resource r_k among all PMs. Moreover, let ρ denote the vector of ρ_k.

For link lk_i in the interconnection network, we define the utilization, ν_i, as $\nu_i = a_i/b_i$. Further, let ν be the random variable representing the utilization of links in the network. We divide links into broadly two types: edge links and core links. We define edge links to be the links directly connected to a PM, whereas all other links are core links. The utilization of edge links and core links are denoted by ν_{edge} and ν_{core}, respectively. The state of the cloud is represented by $C = \{\rho, \nu\}$.

The network delay between pm_i and pm_j is the sum of the link delays along the path $h_{i,j}$. We are not concerned about absolute delay, rather delay factor as provided by a series of $M/M/1$ queues, each representing a link along the path. Denoting the delay factor between pm_i and pm_j by $T_{i,j}$, we write

$$T_{i,j} = \sum_{k=1}^{\eta_{i,j}} \frac{1}{1 - \nu_{h_{i,j}(k)}}.$$

The above expression gives a nominal value of delay assuming a unit service time. In comparing delay among pairs of PMs, we use a delay index metric denoted by δ. The delay index $\delta_{i,j}$ between pm_i and pm_j along path $h_{i,j}$ is given by

$$\delta_{i,j} = 1 - \frac{\eta_{i,j}}{T_{i,j}},$$

which is a value in $[0, 1]$, where $\delta_{i,j} = 0$ represents no delay and $\delta_{i,j} = 1$ represents infinite delay. A higher value of $\delta_{i,j}$ signifies more relative network congestion along the path $h_{i,j}$.

Thus far, we have defined performance measures such as resource utilization, $\rho_{i,k}$, link utilization, ν_i, path bottleneck utilization, $\gamma_{i,j}$, path delay factor, $T_{i,j}$, path length, $\eta_{i,j}$, and path delay index, $\delta_{i,j}$. Such measures were defined on a resource, link, or path between a pair of PMs. For a given pattern, we provide measures defined on the pattern.

B. Pattern performance

We express the performance of a pattern by taking the normalized weighted sum of a particular performance measure of the pattern. For example, we define a weighted path length (distance) for pattern $\mathfrak{p}$, denoted by $\eta(\mathfrak{p})$, as

$$\eta(\mathfrak{p}) = \frac{\sum_{vm_i, vm_j} \lambda_{i,j} * \eta_{pm(vm_i), pm(vm_j)}}{\sum_{vm_i, vm_j} \lambda_{i,j}},$$

where vm_i and vm_j go over the set $\mathcal{VM}(\mathfrak{p})$. Similarly, we define the weighted delay index for pattern $\mathfrak{p}$, denoted by $\delta(\mathfrak{p})$, as

$$\delta(\mathfrak{p}) = \frac{\sum_{vm_i, vm_j} \lambda_{i,j} * \delta_{pm(vm_i), pm(vm_j)}}{\sum_{vm_i, vm_j} \lambda_{i,j}}.$$

C. Performance objective

The overall performance objective comprises two components: (1) system performance expressed as the average and skew of the utilization of the various system resources and (2) pattern performance expressed as the average and skew of pattern related measures such as communication path length, communication delay, and deviation from availability requirements. Now, we define a combined overall performance objective for the cloud using the above-mentioned performance metrics. As for the first component, we consider PM and network link resources. Thus, system performance is characterized by $\rho_k, k = 1, 2, \cdots, n_r$, ν_{edge} and ν_{core}. For the second component, we have $\eta(\mathfrak{p})$ and $\delta(\mathfrak{p})$, representing

representing the weighted path length and the weighted delay index of pattern $\mathfrak{p}$, respectively.

In general, let X be the random variable representing a performance metric. We denote the average and standard deviation of X by $\mu(X)$ and $\sigma(X)$, respectively.

The objective function is defined as

$$\begin{aligned} F(\pi(\mathfrak{p})|C) = & \textstyle\sum_{k=1}^{n_r} \omega 2_{res_k}\, \sigma(\rho_k) \\ & + \omega 1_{edge}\, \mu(\nu_{edge}) + \omega 2_{edge}\, \sigma(\nu_{edge}) \\ & + \omega 1_{core}\, \mu(\nu_{core}) + \omega 2_{core}\, \sigma(\nu_{core}) \\ & + \omega_{path}\, \eta(\mathfrak{p}) + \omega_{delay}\, \delta(\mathfrak{p}), \end{aligned}$$

where $\omega i_{res_k}, \omega i_{edge}, \omega i_{core}, i = 1, 2$, are weights for the average and standard deviation of the utilization of PM resources, edge and core links, respectively, and ω_{path} and ω_{delay} are weights for the pattern weighted path length and delay index, respectively. As far as the PM resources are concerned, we care about the imbalance through the standard deviation, whereas the average is not included in the objective function since the pattern to be placed imposes some given demand independently ($\omega 1_{res_k} = 0$). As for the network, we seek to lower both the average amount of traffic as well as any imbalance among the links. It is desirable to have $\omega 1_{edge} > \omega 1_{core}$ while $\omega 2_{edge} < \omega 2_{core}$. This is due to the more damaging impact from the imbalance in the core network than the edge links.

IV. Optimization problem

Given a cloud in state C, we are concerned with the placement $\pi(\mathfrak{p})$ of pattern $\mathfrak{p}$ so as to minimize the objective function $F(\pi(\mathfrak{p})|C)$. In particular, consider an arrival process of patterns which are to be placed in the cloud. A placed (deployed) pattern remains in the cloud for some lifetime after which the pattern departs and releases all of its resources. In this paper we focus on the handling of an arriving pattern rather than studying queueing and occupancy characteristics. As such, we are dealing with a loss system where an arriving pattern request $\mathfrak{p}$ may be lost in case the placement algorithm fails to find a mapping $\pi(\mathfrak{p})$ to place the pattern. In other words, we state our optimization as follows.

$$Given\ C, find\ \pi(\mathfrak{p}) \mid min\{F(\pi(\mathfrak{p})|C)\}.$$

An optimal placement algorithm attempts to find PMs in the cloud that have enough capacity to host the VMs in the pattern, while making sure that there is enough bandwidth in the network to accommodate inter-VM bandwidth requirements, as well as satisfying any pairwise VM availability constraints. Such a choice of placement would have to minimize the above mentioned objective function.

As stated above, finding an optimal solution is NP-hard. Several approaches are possible. A basic approach may be to attempt to only satisfy the constraints while neglecting the optimization part. This may lead to an inefficient state of the cloud where patterns may be dropped even though they could have been accepted with some better placement (or rearrangement) of already deployed patterns. Heuristic-based approaches would try to incorporate the objectives in the way a solution is sought while searching the solution space. The difficulty with such an approach is that the heuristic procedure would have to change as the objective function is altered. Our approach is to find a close to optimal solution by statistically sampling mapping solutions, then using importance sampling techniques (cross-entropy) to refine the sampling process and get closer to sampling near an optimal solution. A straightforward implementation of such a technique may prove inefficient due to the potentially large number of samples that one has to consider. Alternatively, we use the communication and availability constraints to bias the sampling as we build a sample of a mapping for a pattern.

V. Placement algorithm

A. Importance sampling

An overview of the cross-entropy method, also known as importance sampling, may be found in [6]. The main idea is that in order to find an optimal solution to a combinatorial (maximization) problem, one generates many samples of solutions using a parametrized probability distribution. The samples (solutions) are ordered in their attained values of the objective function. Then, the top small fraction of samples, in other words the important samples, are used to adjust the values of the parameters of the generating probability distribution so as to skew the generation process to yield samples with large objective values. The method iterates a few times until a good solution is obtained.

In our case, a sample is analogous to a mapping solution of VMs in a given pattern to PMs in the cloud. Of course we only consider valid samples, in which individual VM resource demands as well as pairwise VM communication demands and availability constraints are met. We need to solve this placement problem at the time a pattern $\mathfrak{p}$ arrives, given the current state of the cloud C. For simplicity, since in this section we refer to a particular pattern $\mathfrak{p}$, we will omit any variable related to the pattern. We define the parameters for the sample generation process as a matrix $\mathbf{P} = [p_{i,j}]$ of probabilities, where $i = 1, 2, \cdots, n_{vm}$, is the i^{th} VM in the pattern according to some order discussed below, and $j = 1, 2, \cdots, n_{pm}$, is pm_j in the cloud. Thus, $p_{i,j}$ represents the probability of assigning vm_i in the pattern to pm_j. Starting from some initial $\mathbf{P}$, which may be based on resource availability as discussed below, the algorithm proceeds to modify $\mathbf{P}$ until a solution is reached, represented by a dominant (close to 1) entry in each row of $\mathbf{P}$ and all other entries are negligibly small (close to 0). In every iteration of the importance sampling algorithm, the entries in $\mathbf{P}$ that correspond to a large objective value are reinforced and amplified, at the cost of other solutions that are away from optimality. A straightforward implementation of the importance sampling method to our placement mapping problem would be terribly inefficient (as illustrated in section VI), since typically thousands of samples need to be generated at each iteration. This is a very costly proposition for a cloud-sized problem. Therefore, one needs to bias the values

of $\mathbf{P}$ while building a solution so as to accelerate arriving at a solution to the problem

B. Sample biasing

As we decide on the placement of a pattern we sequentially consider the VMs in the pattern without backtracking, i.e. once $vm_i, i = 1, 2, \cdots, n_{vm}$ is placed, the choice for placement is only left for $vm_{i+1}, \cdots, vm_{n_{vm}}$. Thus, the order of VMs in a pattern plays a role during placement. We propose to satisfy the availability constraints before satisfying communication need. Hence, we place the VMs with availability constraints ahead of the remaining VMs in the order. Further, we propose to satisfy the higher communication needs than the lower ones. Hence, we order the VMs according their pairwise communication need.

1) Initial biasing: The initial setting of $\mathbf{P}$ should reflect the state C. A simple choice that is only based on PM resources is $p_{i,j} \propto 1/\rho_{j,k}$, where k is the bottleneck resource.

2) Availability constraint biasing: Once vm_i is placed on say $pm_i = pm(vm_i)$, we examine any availability constraint with $vm_m, m = i+1, \cdots, n_{vm}$ in a look-ahead fashion. More precisely, let's say that vm_i and vm_m have an availability constraint of level l. Then, we need to bias $p_{m,j}$ positively towards $pm_j \in g_i(l)$ and negatively to all other PMs. In case the constraint is hard then the negative bias should make the corresponding entries zeros. Otherwise, the negative biasing becomes more negative for $pm_j \in g_i(l-1) \cup g_i(l+1)$, $pm_j \in g_i(l-2) \cup g_i(l+2)$, and so on. That is if the constraint is soft on both sides of the desired availability level. Otherwise, it would consider only the higher availability levels.

The way biasing is done is through multiplying $p_{m,j}$ by a factor $f_{m,j}$ and normalize the $p_{m,\cdot}$ after all biasing factors are applied. In our simulation experiments we set $f_{m,j} = 10^d$, where $d \in [3, -3]$, a range that is divided for deviation values $0, 1, \cdots, L$, corresponding to cases $l, l-1, \cdots, l-L$, respectively.

3) Communication biasing: In a similar way to biasing the probabilities to reflect availability constraints, we bias them depending on the number of hops between a given PM and the other PMs in the cloud. A measure that is based on path congestion, rather than number of hops is also possible but it is more complex to partition the PMs based on congestion with respect to a given PM. Once vm_i is placed on say $pm_i = pm(vm_i)$, consider the communication demand $\lambda_{i,m}$ between vm_i and vm_m, $m > i$. In order to keep vm_m placed close to pm_i we positively bias $p_{m,j}$ towards $pm_j \in w_i(0)$, i.e. pm_i, then less positively towards $pm_j \in w_i(1)$, and so on until we reach a most negative bias towards $pm_j \in w_i(L)$, the farthest away PMs from pm_i. Similar to availability constraint biasing, we multiply $p_{m,j}$ by a factor $f_{m,j}$ and normalize $p_{m,\cdot}$. In our simulation experiments we set $f_{m,j} = 10^d \lambda_{i,m}$, where $d \in [3, -3]$, a range that is divided for distance values $0, 1, \cdots, L$.

C. Complexity

The complexity of biasing depends on the size of matrix $\mathbf{P}$ which is $n_{vm} \times n_{pm}$. Since we place one VM at a time, and once placed we consider networking demand and availability constraints of all remaining (unplaced) VMs in the pattern, and accordingly we modify entries in the whole row, then we end up with a complexity of $O(n_{pm} n_{vm}^2)$.

VI. Results

A. Description of setup

We consider a cloud with a base configuration consisting of 256 PMs, each with a CPU capacity of 64 cores. The PMs are connected by a tree network with degree of 16, hence a two-level tree. The bottom level consists of 256 edge links, each with capacity 256 units, and the top level consists of 16 core links each with capacity 1024 units. We consider multi-tier patterns similar to the one shown in Figure 1. In particular, we assume a generic pattern consisting of {1, 2, 1} VMs arranged in 3 tiers, respectively, totaling 4 VMs. Larger patterns are generated by scaling this generic pattern by multiplying the number of VMs by a factor. For example, a factor of 4 results in patterns of size {4, 8, 4} VMs in the 3 tiers, respectively, totaling 16 VMs. The scale factor for a pattern is uniformly distributed between 1 and 4, hence an average number of 10 VMs. The CPU demand of a VM is {2, 4, 8} cores for VMs in the 3 tiers, respectively. The inter-VM communication bandwidth requirement is 1 unit for all pairs of VMs between tier 1 and tier 2, and 2 units between tier 2 and tier 3. As described earlier, our availability model is hierarchical and overlays the communication tree topology. In other words, each PM forms an availability zone of level 0, each group of 16 PMs that are connected to a common switch forms an availability zone of level 1, and the group of the latter groups forms a zone of level 2. Availability constraints in a pattern involve all pairs of VMs in the same tier, at the same level. Different tiers of the same pattern may require different levels. The zone level corresponding to a tier is either 1 or 2, with probability 0.8 and 0.2, respectively. This is a hard constraint.

We simulate the above described system and workload, starting from an empty system, leading up to a loading of 80% average PM CPU utilization, then having Poisson pattern arrivals and exponentially distributed pattern lifetime maintaining the 80% average loading figure. The placement algorithm is configured to generate 20 samples per iteration with the top 0.1 fraction used to generate the importance sampling of the subsequent iteration. The stopping criterion is obtaining the same value of the objective function in two consecutive iterations, or reaching a maximum of 10 iterations.

We use the following weights in the objective function: $\omega 2_{res_0} = 2, \omega 1_{edge} = 4, \omega 1_{core} = 2, \omega 2_{edge} = 1, \omega 2_{core} = 2$, $\omega_{path} = 3$, and $\omega_{delay} = 0$. We are mostly concerned with the amount of traffic on edge links, hence we wanted to force placement of VMs in the same pattern on the same PM as much as possible. As discussed earlier, we care about the imbalance in the core network than the edge network. The imbalance in CPU loading among PMs is relevant, but not as important as decreasing network traffic. Lastly, for pattern

performance measures, we consider the path length to be more relevant than the delay index.

The algorithm is coded in Java and runs on a MacBook Pro with 2.4 GHs Intel Core 2 Duo and 4GB RAM, running Mac OS X 10.5 and JVM 1.6.0. The code is not optimized and could be easily made faster, but our purpose here is purely comparative. Note that our algorithm is easily parallelizable since several samples may be tried independently in parallel in case the number of PMs is an order of magnitude higher.

B. Optimality verification

In order to compare the quality of our algorithm to optimal placement we consider a numerical integer programming optimization package, namely IBM ILOG [8]. However, due to the quadratic nature of the optimization problem and its complexity, we consider a small system consisting of 16 host PMs, connected via a tree network of degree 4. We consider the generic pattern described above, consisting of 4 VMs in three tiers. We set a time limit of 10 seconds for numerically solving the optimization problem and producing a pattern placement solution mapping. The results of using the IBM ILOG package are contrasted to that of our algorithm. The comparison is summarized in Table I. As shown, the performance of our algorithm is quite comparable to optimal for this manageable system, except that it is orders of magnitude faster.

C. Unbiased importance sampling

A straight implementation of importance sampling resulted in having to generate a large number of samples, most of which failed to satisfy the constrains. For the system and pattern configurations described above in Section VI-A, we turned off biasing in our algorithm and allowed a maximum of 1000 trials to generate a valid sample. All other parameters remained the same. The comparison between the base algorithm (with biasing) and a plain implementation of importance sampling, or cross-entropy, without biasing is summarized in Table II. As shown, a large fraction, about two-thirds, of the requests were rejected due to failing to generate a feasible placement. Consequently, the utilization of the PMs and the network links dropped substantially. As a result the pattern delay index measure was lower, 0.13 compared to 0.28, due to the drop in link utilization. The pattern path length was significantly higher, 2.99 compared to 1.99, highlighting the fact that the algorithm without biasing failed to place the VMs in a given pattern close to each other. The placement time for patterns that were placed was about three times longer than that of our algorithm.

	Our Algorithm	Optimal (ILOG)
PM utilization	0.77	0.77
Edge link utilization	0.51	0.52
Core link utilization	0.20	0.20
Pattern path length	1.21	1.22
Pattern delay index	0.26	0.28
Pattern rejection prob.	0.02	0.02
Placement time *(msec)*	**3**	**10,329**

TABLE I
COMPARISON OF OUR ALGORITHM TO OPTIMAL.

	Our Algorithm (with biasing)	Plain Importance Sampling (without biasing)
PM utilization	0.80	0.22
Edge link utilization	0.35	0.06
Core link utilization	0.44	0.17
Pattern path length	1.99	2.99
Pattern delay index	0.28	0.13
Pattern rejection prob	**0.0005**	**0.6382**
Placement time *(msec)*	115	354

TABLE II
THE IMPACT OF BIASING ON PERFORMANCE.

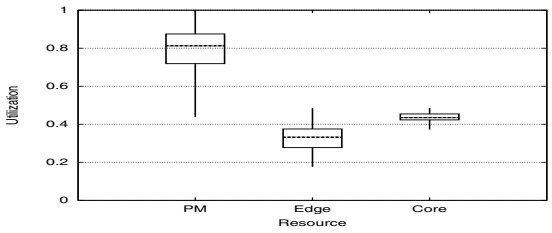


Fig. 3. PM and network utilization.

D. Biased importance sampling: Base case

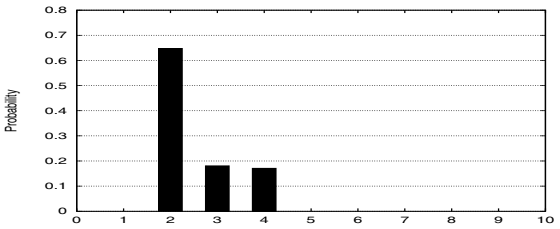
A good placement algorithm would attempt to keep the highly communicating VMs in a pattern close to each other as much as possible. Also, it would try to keep the utilization of the core links balanced. Our algorithm achieved those objectives as illustrated. In Figure 3 we show the distribution of PM CPU utilization as well as link utilization as Box-Whisker diagrams. For the CPU utilization we obtained a relatively well-balanced system around a median of 81%. The link utilization was kept low, with the median utilization for edge links and core links at about 33% and 43%, respectively. The traffic through the core links was inevitable in order to accommodate the availability constraint of level 2. Noticeable though is the extremely low skew in the utilization of the 16 core links in the closed range [37%, 48%].

The histograms of the pattern performance measures are exhibited in Figure 4. The average weighted path length was 1.96 with a stdDev of 0.81. The weighted delay index measure for patterns was 0.26 on average with a stdDev of 0.08. This means that the effect of link congestion was small.

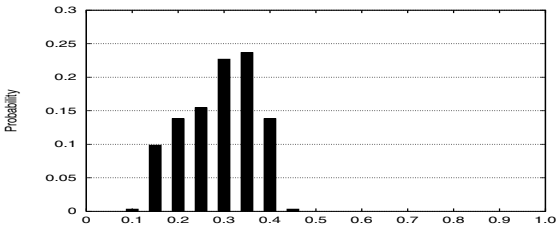
The performance of the placement algorithm is exhibited in Figure 5. The average number of trials (samples) per pattern placed was about 55 with a stdDev of 19. The placing time for a pattern was 106 msec on average with a stdDev of 108 msec. The maximum placing time experienced in this case was 471 msec.

E. Effect of pattern size

Now, we investigate the effect of the pattern size. We increase the pattern scaling factor from 1 to 6, where as described earlier a generic pattern consists of 4 VMs arranged as $\{1, 2, 1\}$ in 3 tiers. The scaling factor multiplies the number of VMs in the pattern, keeping the same ratio among the 3

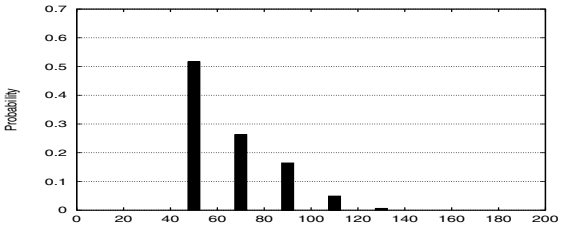


(a) Pattern weighted path length.

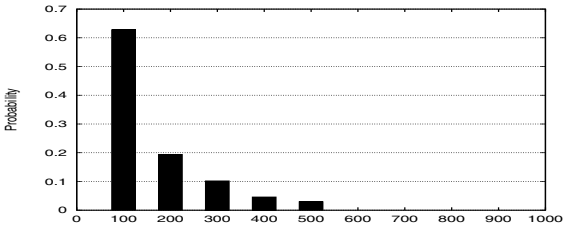


(b) Pattern delay index.

Fig. 4. Pattern performance.



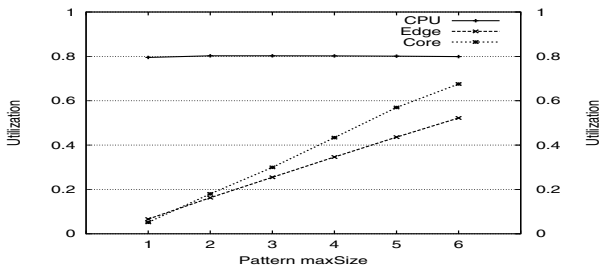
(a) Number of trials.



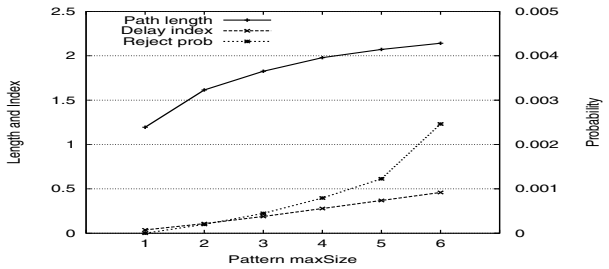
(b) Pattern placement time (msec).

Fig. 5. Placement performance.

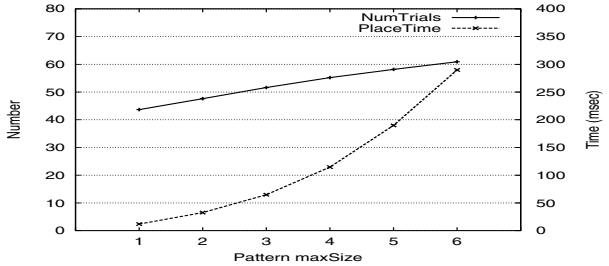
tiers. Given a scaling factor, patterns are generated using a uniform distribution from 1 to the value of the scaling factor. The results are depicted in Figure 6. The load was kept at 80% PM CPU utilization. Edge and core link utilization grew almost linearly with the pattern size. Note that the placement algorithm managed to keep the traffic split between the edge links and core links the same, independent of the pattern size. As for pattern performance measures, the weighted delay index grew linearly, whereas the weighted path length was concave, approaching a limit. This is due to the structure and location constraints of the pattern. There were more rejections as the pattern size increased, though with a maximum of 0.25%. Since the biasing of samples is $O(n^2)$ in the size of the pattern, we see that the placing time of a pattern grew quadratically, but still less than 300 msec on average.



(a) PM and network utilization.



(b) Pattern performance.



(c) Placement performance.

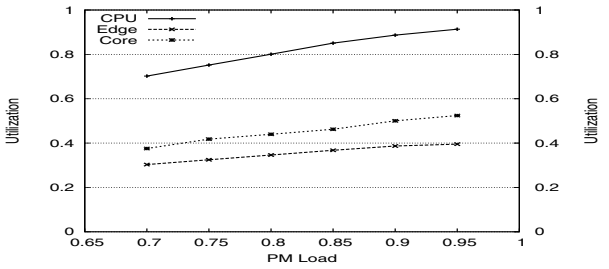
Fig. 6. Effect of maximum pattern size.

F. Effect of load

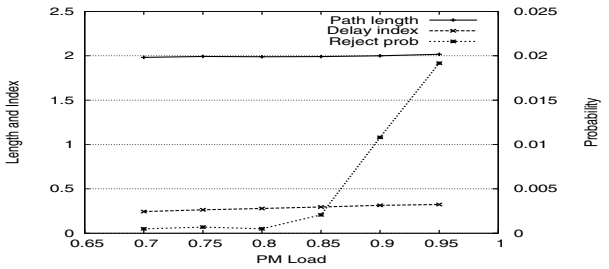
We varied the PM CPU utilization from 70% to 95% with increments of 5%, as depicted in Figure 7. We clearly see a linear increase in link utilization, indicating that the placement algorithm performed well even at high utilization. Further, the pattern performance measures, weighted path length and weighted delay index, also grew linearly and very slightly. Another indication that the placement algorithm managed not to create a skew in network utilization and hence congestion. The rejection probability of patterns started to increase at 95% loading to a significant 1.9%, but quite expected at that saturation level. Surprisingly, the placing time for a pattern and the number of trials (samples) remained fairly constant and independent of the loading factor.

G. Effect of number of hosts

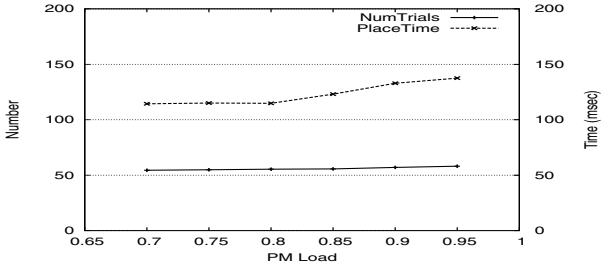
In order to investigate the scalability of our placement algorithm we varied the number of PMs from 128 to 1024 in powers of 2, as illustrated in Figure 8. The load factor for the PM CPU utilization was kept at 80%. As the system size increased, the sample space increased exponentially due to the combinatorial effect. However, our placement algorithm managed to keep the network link utilization fairly constant, with the core link utilization only increasing from an average



(a) PM and network utilization.

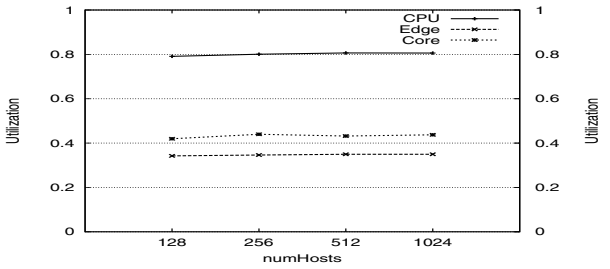


(b) Pattern performance.

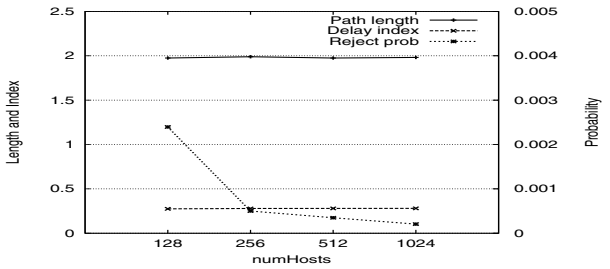


(c) Placement performance.

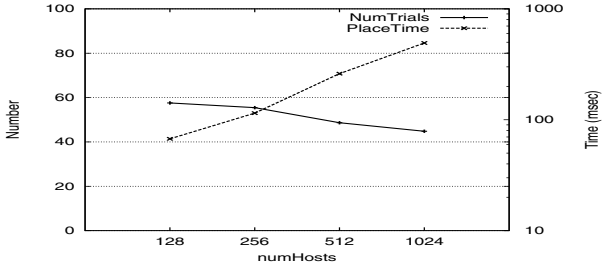
Fig. 7. Effect of load.



(a) PM and network utilization.



(b) Pattern performance.



(c) Placement performance.

Fig. 8. Effect of number of hosts.

of 42% to 44%. The pattern performance measures, weighted path length and weighted delay index, also were fairly constant, with an increase from 1.97 to 1.98, and from 0.27 to 0.28, respectively. Placement time grew linearly as discussed earlier, with the number of trials slowly decreasing. Further, pattern rejections decreased with the increase in system size, which is an anticipated behavior of loss systems.

VII. Conclusion and future work

We demonstrated a method for biasing samples when performing an importance sampling approach to solving a large-scale optimization problem arising from placing virtual clusters in compute clouds. The performance of our algorithm grows linearly in the number of PMs in the cloud and is shown to take about 500 msec in the case of 1024 PMs.

Several issues need further investigation. The algorithm is quadratic in the size of the pattern. This begs the question of whether the ordering of VMs in a pattern could help reduce the complexity to a linear one by applying the biasing on a fixed number of VMs, instead of all remaining VMs in the pattern. Also, when placing a pattern, we were concerned with the performance that the pattern would experience given the current state of the system. One should also be concerned with the impact of placing a pattern, not only on system resource utilization, but on the performance experienced by the already placed patterns.

References

[1] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1 –9.

[2] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement," in *Services Computing (SCC), 2011 IEEE International Conference on*, July 2011, pp. 72 –79.

[3] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *INFOCOM 2006 Proceedings IEEE*, April 2006, pp. 1 –12.

[4] K. Taura and A. Chien, "A heuristic algorithm for mapping communicating tasks on heterogeneous resources," in *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*, 2000, pp. 102 –115.

[5] C.-W. Yeh, C.-K. Cheng, and T.-T. Lin, "Circuit clustering using a stochastic flow injection method," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 14, no. 2, pp. 154 –162, February 1995.

[6] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and Computing in Applied Probability*, vol. 1, pp. 127–190, 1999.

[7] S. Sanyal and S. Das, "Match : Mapping data-parallel tasks on a heterogeneous computing platform using the cross-entropy heuristic," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, April 2005.

[8] "Ilog." www.ibm.com/software/websphere/products/optimization/