**RC25248 (W1112-050) December 16, 2011**
**Computer Science**

# IBM Research Report

# Optimized Cloud Placement of Virtual Clusters Using Biased Importance Sampling

**Asser N. Tantawi**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

# Optimized cloud placement of virtual clusters using biased importance sampling

Asser N. Tantawi
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
tantawi@us.ibm.com

## ABSTRACT

In cloud computing, a cloud provider is faced with the problem of allocating physical resources to an incoming stream of requests for virtual resources in such a way to satisfy requirements and maximize resource usage while providing good performance. Traditionally, when a request is for a virtual machine (VM), the problem of placing it on a particular physical machine (PM) may be expressed as a bin packing or an online, stochastic knapsack problem. The typically large size of such a problem forces one to resort to approximate and heuristics-based optimization techniques. Now, the trend is for a request to involve more than just a single VM. In particular, a request consists of a set of heterogeneous VMs with some interrelationships due to communication needs and other dependability-induced constraints. The placement of such constrained, networked virtual clusters in the physical infrastructure, including compute, storage, and networking resources is much more challenging. Several research avenues are and have been explored to deal with such a problem, albeit mostly in other areas such as mapping task graphs in parallel systems and provisioning of virtual private networks. The sheer size of the cloud incites one to investigate statistical approaches to solving this cloud placement problem.

In this paper, we provide an algorithm for the placement of constrained, networked virtual clusters in the cloud, that is based on importance sampling (also known as cross-entropy). 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 cloud. We investigate the quality of the results of using our algorithm on a simulated cloud, where we study the effects of the various parameters on the solution and performance of the algorithm.

## 1. 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 come up with 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 [12]. 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 [12], the authors present heuristic placement algorithms based on graph decomposition, but only in the case where one VM is placed on a PM. In [9], a heuristic algorithm based on clustering is presented and is shown to work when placing in an empty system.

A similar problem has existed in the area of virtual networking. Notably, a divide-and-conquer approach is presented in [25], where a collection of connected physical resources is first identified as the target for placement, instead of the entire system. The problem of identifying the best cluster for placement has been studied in [11]. An alternate separation technique is described in [24]. A practical testbed implementation is discussed in [14]. In [2], a cloud networking platform is described. And, various models of communication among VMs are investigated in [13].

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 existed 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 [20, 23]. A heuristic algorithm which investigates the traversing of a graph and identifying potential neighborhoods for placement is described in [1]. Several heuristic algorithms for this graph mapping problem are contrasted in [3]. A graph matching technique has also been tried [7, 19]. Other approaches are presented in [6, 10]. A similar problem related to code placement has also been considered [18, 22].

Biologically-inspired algorithms and genetics methods have been investigated and suggested to solve this graph mapping problem, [4] and [5], respectively. Further, the cross entropy technique [15, 16] which is analogous to importance sampling has also been considered [17, 8]. 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 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 2. The performance measures and objectives are introduced in section 3. In section 4 we state the placement optimization problem. Our placement algorithm is described in section 5. Simulation results are presented in section 6.

## 2. 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

### 2.1 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 $d_{i,k}$, $d_{i,k} \leq c_{i,k}$.

### 2.2 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 network congestion occurs due to link utilization only. In other words, 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 $\eta_{i,j} = l$.

### 2.3 System availability

We model the availability of the cloud as follows. Focussing on a pair of PMs, say $pm_i$ and $pm_j$, $i, j = 1, \cdots, n_{pm}$. 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.) The matrix $[v_{i,j}]$ may be obtained through a comprehensive availability analysis which considers units such power supplies, cooling units, security hazards, in addition to electronic, cables, and storage units and their interdependencies as far as failure and repair characteristics are considered. Such an analysis is beyond the scope of this paper. As an alternative to this analysis, it may suffice to partition a data center 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. 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 $L$. It follows 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 $v_{i,j} = V_l$. In the general case where the matrix $[v_{i,j}]$ does not partition the PMs into a hierarchy, we define $G_i(v), i = 1, 2, \cdots, n_{pm}$, and $0 \leq v \leq 1$ as the set of PMs such that for $pm_j \in G_i(v)$ we have $v_{i,j} \leq v$.

### 2.4 Virtual machines

Each VM is characterized by a set 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 $pm(vm_j)$.

### 2.5 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 $\mathcal{S}(\mathfrak{p}) \subseteq \mathcal{VM}(\mathfrak{p})$ be a subset of VMs in $\mathfrak{p}$ of size $n$. An availability requirement is specified as $k$-out-of-$n$, $1 \leq k \leq n$, VMs in $\mathcal{S}(\mathfrak{p})$ have to be be simultaneously available with some probability $\beta$, written as $A(\mathcal{S}(\mathfrak{p}), k) = \beta$. (Without loss of generality, we assume throughout that whenever the values of availability are discrete, as in the tree model described above, the availability requirement is that of equality, whereas if the values are continuous, as in the general case, the availability requirement is that of superiority (larger than)). The VMs in $\mathcal{S}(\mathfrak{p})$ may be each providing a given service and a minimum $k$ of them need to be available with high probability. Multiple non-overlapping subsets in a pattern may be defined as $A(\mathcal{S}_m(\mathfrak{p}), k_m) = \beta_m, m = 1, 2, \cdots, M$. The requirement $A(\mathcal{S}(\mathfrak{p}), k) = \beta$ may be translated to a set of requirements $v_{pm(vm_i),pm(vm_j)} = \alpha$, where $vm_i, vm_j \in \mathcal{S}(\mathfrak{p})$. The value of $\alpha$ is calculated using $\beta$ through a straightforward $k$-out-of-$n$ analysis. Note that the homogeneous requirement of $\alpha$ is sufficient, but not necessary. One might choose to have two VMs in $\mathcal{S}(\mathfrak{p})$ be placed on a pair of highly available PMs, whereas the rest of the VMs are placed on less available PMs. However, for simplicity we assume a homogeneous requirement.

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 $\pi(\mathfrak{p}) = \{(vm_i, pm_m) \mid pm(vm_i) = pm_m, \forall vm_i \in \mathcal{VM}(\mathfrak{p})\}$.

## 3. PERFORMANCE EVALUATION

### 3.1 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 $\rho_k$ be the random variable representing the utilization of resource $r_k$ among all PMs. Moreover, let $\rho$ denote the vector of $\rho_k$.

For link $lk_i$ in the interconnection network, we define the utilization, $\nu_i$, as $\nu_i = a_i/b_i$. Further, let $\nu$ 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 $\nu_{edge}$ and $\nu_{core}$, respectively. (Other, more detailed classification of links are possible, such as the level of a link with respect to a PM in a hierarchical network.) The state of the cloud is represented by $C = \{\rho, \nu\}$.

The utilization of a path consists of the set of utilization of the links constituting the path. The path bottleneck utilization is defined as the utilization of the most utilized link in the path, which we denote by $\gamma_{i,j}$. Hence, we have

$$\gamma_{i,j} = max\{\nu_{h_{i,j}(1)}, \nu_{h_{i,j}(2)}, \cdots, \nu_{h_{i,j}(\eta_{i,j})}\}.$$

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 $\delta$. 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, $\nu_i$, 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.

### 3.2 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

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

Further, let $\eta$ and $\delta$ denote the random variable representing the weighted path length and the weighted delay index among all patterns in the system.

### 3.3 Performance objective

The question here is: when deciding on the placement of a pattern, $\mathfrak{p}$, what should the performance objective be? 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 $\rho_k, k = 1, 2, \cdots, n_r$, $\nu_{edge}$ and $\nu_{core}$. 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 $\omega_{path}$ and $\omega_{delay}$ 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.

Note that, in addition to system performance measures, the objective function includes performance measures of the pattern to be placed. As such our objective function targets both social optimization, reflected in the health of the system, including PMs and network links, as well as individual optimization, reflected in the specific performance that a given pattern is experiencing.

## 4. 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, i.e. minimize the imbalance (skew) of resource usage, the amount of network traffic, and path lengths and network congestion for the pattern. Note that other objective functions are possible. For example, one may be concerned with rejection probability over a time horizon, or some overall performance of all patterns already placed in the cloud. These and others may be subject of future research.

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. In the next sections we describe our method and then provide simulation results.

## 5. PLACEMENT ALGORITHM

### 5.1 Importance sampling

We provide a brief overview of the cross-entropy method, also known as importance sampling, in appendix A. 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. (Note that the importance sampling algorithm is stated to solve a maximization problem, whereas we deal with an equivalent minimization problem.) A straightforward implementation of the importance sampling method to our placement mapping problem would be terribly inefficient (as illustrated in section 6), 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

## 5.2 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. Other ordering criteria are also possible. A natural one is where VMs with high communication need are placed close to each other in the order. Such approach was taken in [20] using a clustering technique described in [23].

### *5.2.1 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. It may also depend on the communication bandwidth available to $pm_j$ through the utilization of say link $lk_j$ connected to $pm_j$, as $p_{i,j} \propto 1/(\rho_{j,k} + \nu_j)$. Or, it may include the utilization of further hops, but it makes the computation more complex. The choice of the initial $p_{i,j}$ may also depend on the resource need of $vm_i$ to mimic best-fit or first-fit strategies. In our simulation experiments we use the simple criterion $p_{i,j} \propto 1/\rho_{j,k}$.

### *5.2.2 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.

### *5.2.3 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 $0, 1, \cdots, L$.

## 6. RESULTS

## 6.1 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 equal to 16, hence a two-level tree. The bottom level consists of 256 edge links, each with capacity 128 units, and the top level consists of 16 core links each with capacity 512 units. There is a predefined set of VM sizes. The choices of VM CPU capacities are {1, 2, 4, 8, 16} cores. And, the choices of inter-VM communication bandwidth capacities are {1, 2, 4, 8, 16} units, with no specific relation to the number of cores of the two communicating VMs. We assume that the choice mix of VM sizes are inversely proportional to the size, hence the average VM demand is $5 \times 16/31 = 2.58$ CPU cores. The number of VMs in a pattern is uniformly distributed between 2 and 14, hence an average of 8 VMs. The particular VMs that make up a pattern are not necessarily homogeneous, rather they are generated using the above mix. VMs that communicate within a pattern form a graph model where a node corresponds to a VM and the existence of an edge signifies communication needs between a pair of VMs. Rather than assuming a random graph model, we use a "small-world" graph which exhibits more structured connectivity [21]. In such a graph model, nodes are laid down on a ring and each node is originally connected to its $K$ adjacent nodes from both sides. Then, each edge is rewired with probability $p$ by selecting one endpoint of the edge and replacing it by a different node at random without self-cycling or duplicating edges. We use $K = 1$ and $p = 0.5$.

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. This hierarchy may correspond to 16 PMs on a blade center, and 16 blade centers are housed in a rack. Two PMs in the same blade center have lower availability level than two PMs in different blade centers in the rack.

We assume that for patterns of more than 4 VMs, one-third (rounded to nearest integer) of the VMs have an availability level requirement among them of 1 or 2, with probability 0.6 and 0.4, respectively. This is a hard constraint, so a pattern is dropped if the constraint cannot be satisfied by the placement algorithm.

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.

We use the following weights in the objective function: $\omega2_{res_0} = 2$, $\omega1_{edge} = 4$, $\omega1_{core} = 2$, $\omega2_{edge} = 1$, $\omega2_{core} = 2$, $\omega_{path} = 3$, and $\omega_{delay} = 0$. We are mostly concerned about 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, which is a function of link utilization which already appears in the equation somewhere else.

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. A word on the algorithm execution time is in order. The placing time of a pattern depends on the arrival rate of patterns and their lifetime. Let's do some back-of-the-envelope calculation. Consider a configuration of 1000 PMs running at 80% with a workload similar to the one described above. Given an average of 2 cores per VM and an average of 8 VMs per pattern, a PM with capacity 64 cores could host a maximum of $64/(2\times8) = 4$ patterns. Hence, the whole data center would have an average of $1000 \times 0.8 \times 4 = 3,200$ concurrently deployed patterns. If the average lifetime of a pattern is 8 hours, then the arrival rate of patterns is $3,200/(8 \times 3,600) = 0.11$ requests/sec, i.e. the average time between two consecutive requests is about 9 sec. Therefore, practically speaking 1 sec or less placing time for one pattern is quite reasonable for this configuration. 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. Also, other solutions such as decomposition of the cloud into separately managed zones are possible.

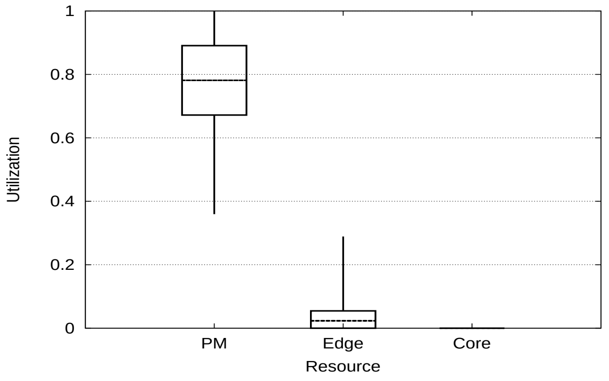
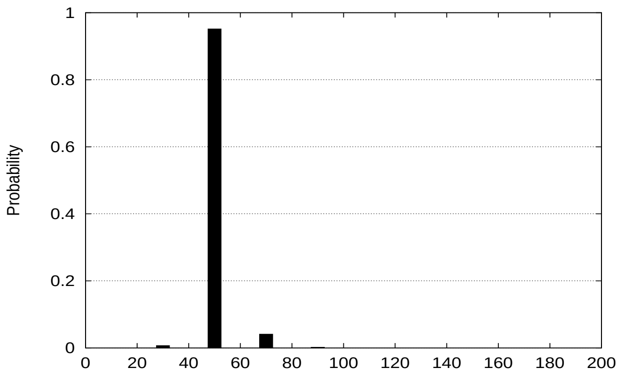
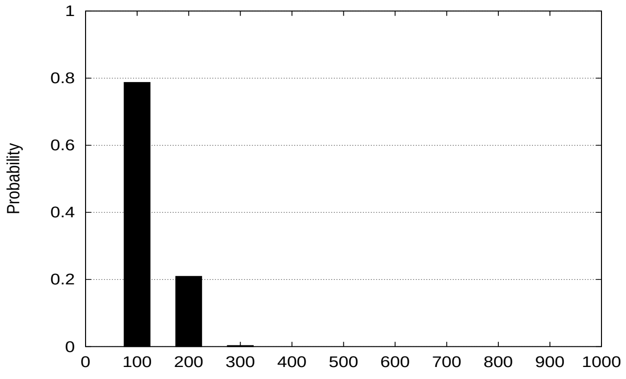


**Figure 1: PM and network utilization (unconstrained).**



(a) Number of trials.



(b) Pattern placement time (msec).

**Figure 2: Placement performance (unconstrained).**

## 6.2 Unbiased importance sampling

A straight implementation of importance sampling resulted in having to generate a large number of samples. We considered two scenarios: 1,000 and 10,000 samples per iteration. In the case of no availability constraints, the resulting utilization of the edge links were 38% and 28% in the two scenarios, respectively. And, the utilization of the core links were 64% and 28%, respectively. Consequently, patterns experienced an average weighted path length of 1.29 and 0.70, and an average weighted delay index of 0.22 and 0.07, respectively. The average total number of trials (samples) in order to place one pattern was 6,048 and 83,041, respectively. Further, the average placing time of a pattern was 388 msec and 5,406 msec, respectively. As demonstrated next, using our biasing algorithm yields much better performance, resulting in almost no network traffic for the base case, at a much lower cost.

## 6.3 Base: biased, unconstrained

We first consider the case where there are no availability constraints. As depicted in the box-and-whisker diagrams shown in Figure 1, we note that the distribution of PM utilization is centered close to the 80% target, at around 78% with half of the PMs in the range [67%,89%], with a reasonable spread where there is very few PMs below 50% and a few reaching 100%. A perfect load balancing is not desirable since one would want to have some PMs with low to moderate utilization to accommodate a potential future request for a large-sized VM. The amount of network traffic is negligible (about 2% average), hence communicating VMs were mostly placed on the same PM. Consequently, the weighted path length and weighted delay index for deployed patterns were almost zero (not shown).

In Figure 2 we plot the histograms of the number of trials (samples) considered for placing a pattern and the real time it took the placement algorithm to place a pattern. The average placing time of a pattern in this case was 57 msec with a stdDev of 47 msec.

## 6.4 Base: biased, constrained

Now, let us consider the availability constraints described above. Such constraints would force the placement of a third of the PMs in the pattern in different availability zones, hence increasing the amount of communication traffic that flows over the network. 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 the figures. In Figure 3 we show the distribution of PM CPU utilization as well as link utilization. For the CPU utilization we were able to obtain similar results to the unconstrained case. The link utilization was kept low, with the average utilization for edge links and core links at about 22% and 35%, respectively. The traffic through the core links was inevitable in order to accommodate the availability constraint of level 2, i.e. some VMs in a pattern had to be placed on different blade centers, hence incurring cross blade center communication. Noticeable though is the extremely low skew in the utilization of the 16 core links (standard deviation was 0.03%).

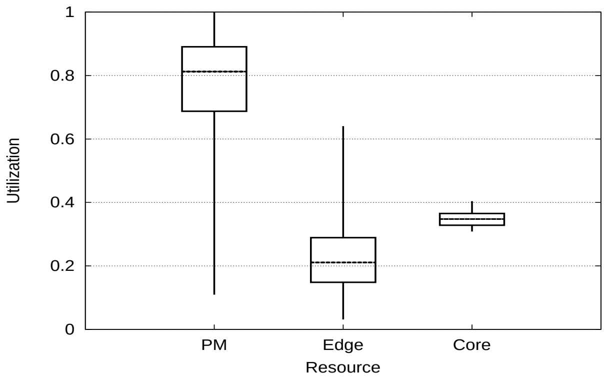
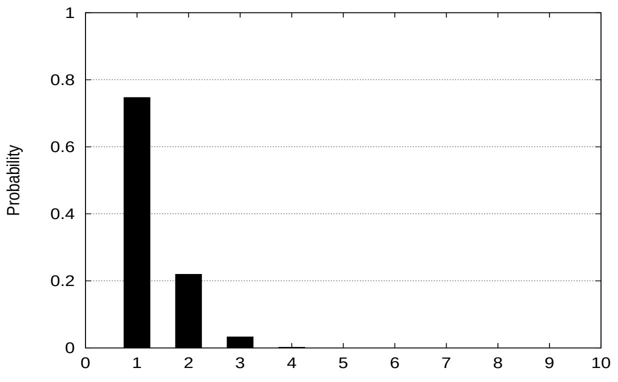
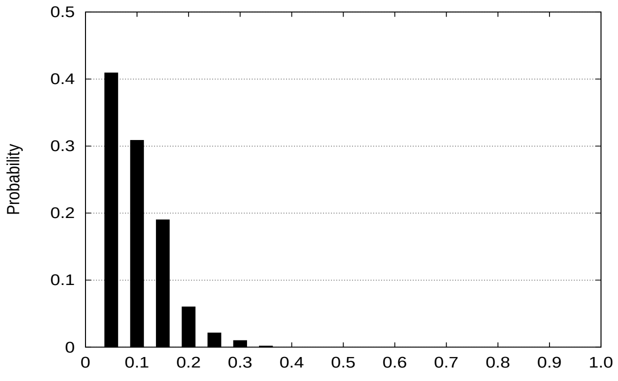


**Figure 3: PM and network utilization (constrained).**



(a) Pattern weighted path length.



(b) Pattern delay index.

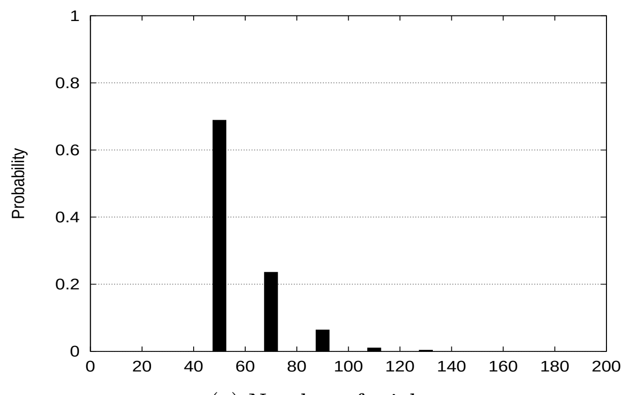
**Figure 4: Pattern performance (constrained).**

The pattern performance measures are exhibited in Figure 4. The average weighted path length was 0.67 with a stdDev of 0.59. In other words, two communicating VMs in a pattern had an average of 2/3 of a link between them. Note that 2 PMs in the same blade center are 2 links apart, and 2 PMs on different blade centers are 4 links apart. The weighted delay index measure for patterns was 0.07 on average with an equal stdDev of 0.06. This means that the effect of link congestion was minimum.

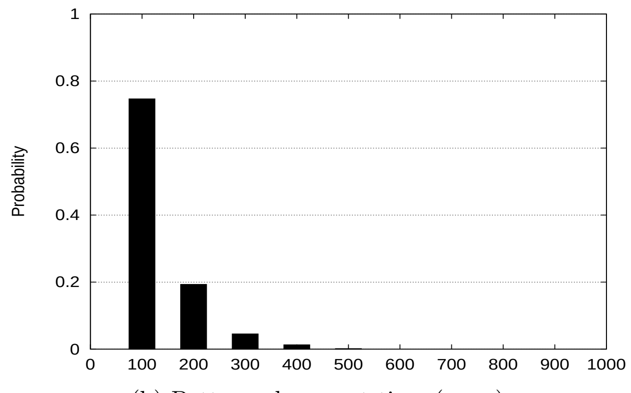
The performance of the placement algorithm is exhibited in Figure 5. The average number of trials (samples) per pattern placement was about 48 with a stdDev of 14. The placing time for a pattern was 71 msec on average with a stdDev of 72 msec. The maximum placing time experienced in this case was 428 msec, i.e. quite reasonable given the discussion above.

## 6.5 Impact of biasing

In order to illustrate the impact of biasing the sampling process during pattern placement, we consider a small example where we have only 9 PMs with a tree network of degree 3, hence a 2-level tree. In other words, we have three blade centers containing the following PMs {pm0, pm1, pm2}, {pm3, pm4, pm5}, and {pm6, pm7, pm8}, respectively. All other parameters are as presented above in the base case. We trace the placement of particular pattern, pat28, with connectivity graph depicted in Figure 6. As shown, the pattern consists of 9 VMs, vm252 through vm260, with communication de-



(a) Number of trials.



(b) Pattern placement time (msec).

**Figure 5: Placement performance (constrained).**

mands as per the weights on the edges. Three VMs {vm252, vm253, vm254} have pairwise availability constraints of level 2 (shown as dashed lines with a Roman II), i.e. each has to be placed in separate blade center.

The sequence of steps in placing this pattern is illustrated in Table 1. With PMs as columns and VMs in the pattern as rows, entries are the probabilities of placing a particular VM on a particular PM. At step 0, the probabilities are set for the various PMs inversely proportional to their availabilities and independent of the VMs. Here, the VMs are considered in their numerical order. At step 1, vm252 is equaly likely to be placed on pm0, pm3, or pm4. Then, vm253 is equaly likely to be placed on pm0, pm1, or pm3. Accordingly, vm254 favors the third blade center in order to be distant from vm252 and vm253, choosing pm6 and pm8 with probabilities 2/3 and 1/3, respectively. Starting from vm255 through vm260, the communication needs causes a VM to be close to the ones that it communicates with. The probabilities keep getting refined until step 4 we see the final feasible placement assignment, where {vm252, vm256, vm257} are assigned to pm3, {vm253, vm255, vm258, vm259, vm260} are assigned to pm1, and {vm254} is assigned to pm6. The placement of the pattern is depicted in Figure 7. Note that {vm252, vm253, vm254} are each placed in a separate blade center. Also note that vm256 was assigned the same PM as vm252, with which it has a high communication demand of 4 units. Further vm257 and vm256 are co-assigned the same PM since vm257 communicates only with vm256. All

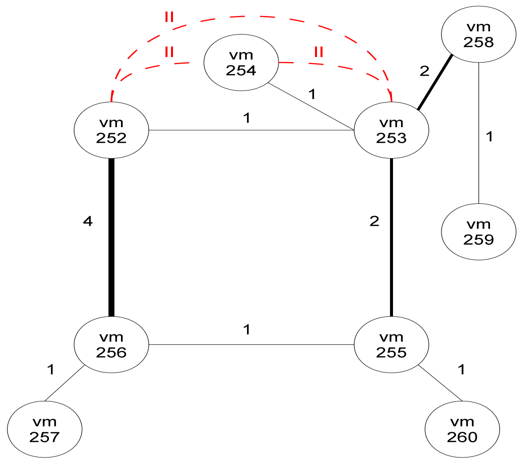


**Figure 6: Pattern pat28 description.**

the remaining VMs were assigned the same PM as vm253, with which there is significant communication demand.

## 6.6 Effect of pattern size

Now, we investigate the effect of the pattern size. We increase the maximum number of VMs in a pattern from 10 to 31, in steps of 3. The results are depicted in Figure 8. The load was kept at 80% PM CPU utilization. Due to our 'small-world' graph model of the communication demand in a pattern, the number of edges is proportional to the number of nodes in the graph, and hence the increase in communication demand is linear in the pattern size as shown in the figure. The $R^2$ value of the linear fit was 0.98 to 0.99 for edge and core utilization, respectively. 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. The pattern performance measures, weighted path length and weighted delay index, also grew linearly with the pattern size with $R^2$ equal to 0.98 and 0.99, respectively. 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 ($R^2 = 1.00$), but still less than a second on average (maximum 420 msec at max pattern size of 31). The number of trials, however, grew linearly with $R^2 = 0.99$. In all cases, there was hardly any pattern rejections (less than 0.0005).

## 6.7 Effect of load

We varied the PM CPU utilization from 70% to 95% with increments of 5%, as depicted in Figure 9. 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 1.7%, but quite ex-

| pm | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Step 0 | | | | | | | | | |
| vm252 | .15 | .16 | .08 | .10 | .20 | .13 | .11 | .02 | .05 |
| vm253 | .15 | .16 | .08 | .10 | .20 | .13 | .11 | .02 | .05 |
| vm254 | .15 | .16 | .08 | .10 | .20 | .13 | .11 | .02 | .05 |
| vm255 | .15 | .16 | .08 | .10 | .20 | .13 | .11 | .02 | .05 |
| vm256 | .15 | .16 | .08 | .10 | .20 | .13 | .11 | .02 | .05 |
| vm257 | .15 | .16 | .08 | .10 | .20 | .13 | .11 | .02 | .05 |
| vm258 | .15 | .16 | .08 | .10 | .20 | .13 | .11 | .02 | .05 |
| vm259 | .15 | .16 | .08 | .10 | .20 | .13 | .11 | .02 | .05 |
| vm260 | .15 | .16 | .08 | .10 | .20 | .13 | .11 | .02 | .05 |
| Step 1 | | | | | | | | | |
| vm252 | .33 | 0 | 0 | .33 | .33 | 0 | 0 | 0 | 0 |
| vm253 | .33 | .33 | 0 | .33 | 0 | 0 | 0 | 0 | 0 |
| vm254 | 0 | 0 | 0 | 0 | 0 | 0 | .67 | 0 | .33 |
| vm255 | .33 | .33 | 0 | .33 | 0 | 0 | 0 | 0 | 0 |
| vm256 | .33 | 0 | 0 | .33 | .33 | 0 | 0 | 0 | 0 |
| vm257 | .67 | 0 | 0 | .33 | 0 | 0 | 0 | 0 | 0 |
| vm258 | .33 | .33 | 0 | .33 | 0 | 0 | 0 | 0 | 0 |
| vm259 | .33 | .33 | 0 | .33 | 0 | 0 | 0 | 0 | 0 |
| vm260 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Step 2 | | | | | | | | | |
| vm252 | 0 | 0 | 0 | .33 | .67 | 0 | 0 | 0 | 0 |
| vm253 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm254 | 0 | 0 | 0 | 0 | 0 | 0 | .67 | 0 | .33 |
| vm255 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm256 | 0 | 0 | 0 | .33 | .67 | 0 | 0 | 0 | 0 |
| vm257 | .33 | 0 | 0 | .67 | 0 | 0 | 0 | 0 | 0 |
| vm258 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm259 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm260 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Step 3 | | | | | | | | | |
| vm252 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| vm253 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm254 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| vm255 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm256 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| vm257 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| vm258 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm259 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm260 | .67 | .33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Step 4 | | | | | | | | | |
| vm252 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| vm253 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm254 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| vm255 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm256 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| vm257 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| vm258 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm259 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| vm260 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 1: Pattern pat28 placement biasing steps.**

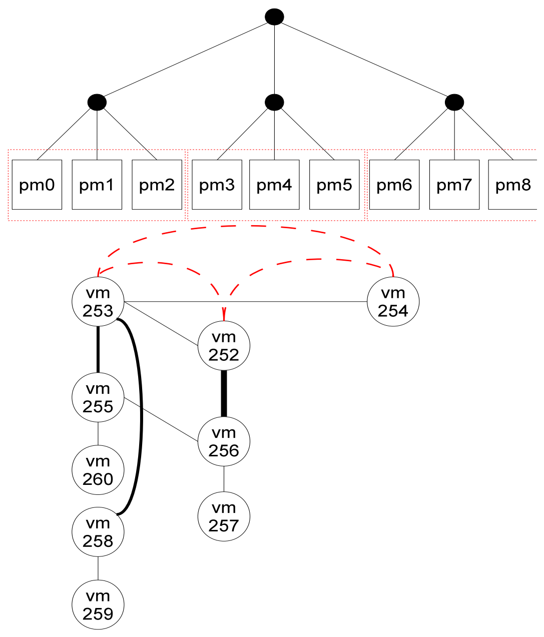


**Figure 7: Placement of pattern pat28.**

pected 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. Hence, the placement algorithm did not have to work any harder to place patterns at high loading.

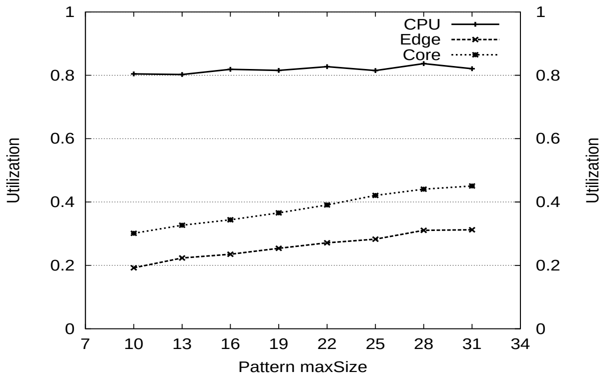
## 6.8 Effect of number of hosts

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 10. 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 increase the network link utilization only slightly by about 1% for every doubling of the system size, while keeping a linear placing time ($R^2 = 0.76$). The pattern performance measures, weighted path length and weighted delay index, also grew by about 2% and 1%, respectively, for every doubling of the system size. The utilization and its derivative measures kept constant in the system size. The placement algorithm exhibits a linear behavior in the system size. Further, there were no pattern rejections experienced even at the large system size of 1024 PMs

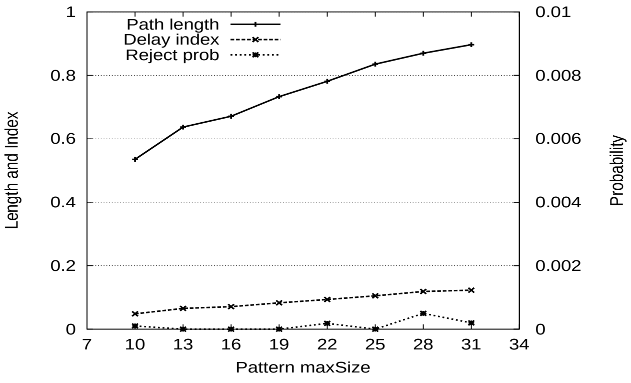
## 7. CONCLUSION AND FUTURE WORK

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 275 msec in the case of 1024 PMs.

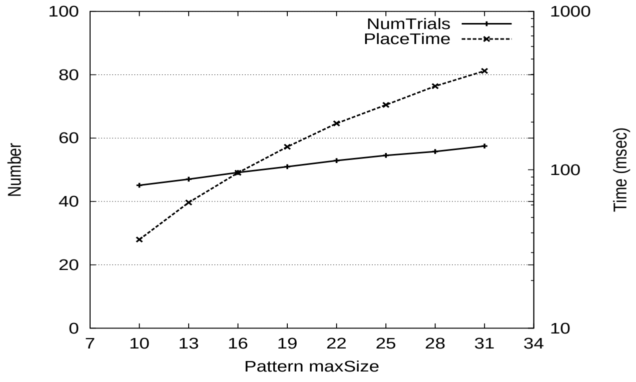
Several issues need further investigation. The algorithm is quadratic in the size of the pattern. This begs the question of whether the ordering of VMs in a pattern could help reduce



(a) PM and network utilization.

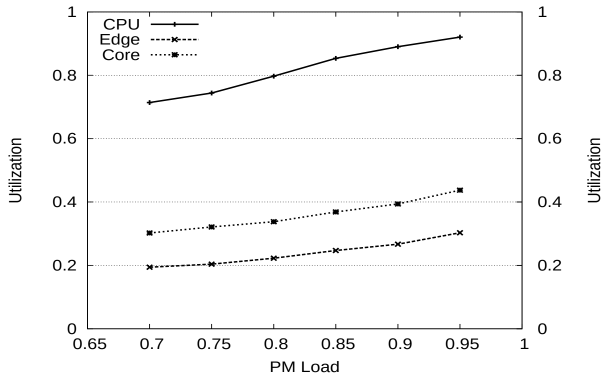


(b) Pattern performance.

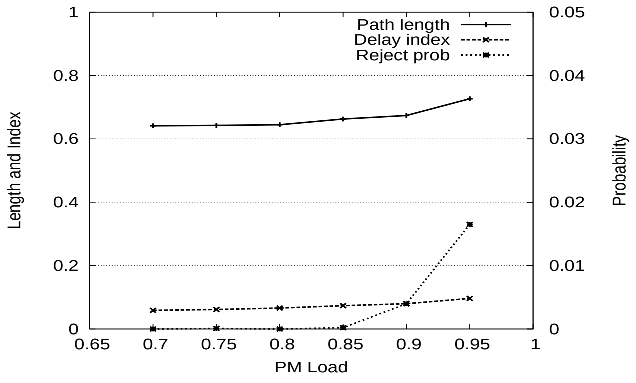


(c) Placement performance.

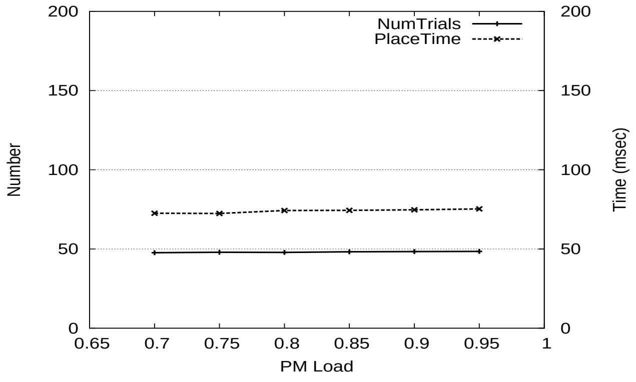
**Figure 8: Effect of maximum pattern size.**



(a) PM and network utilization.

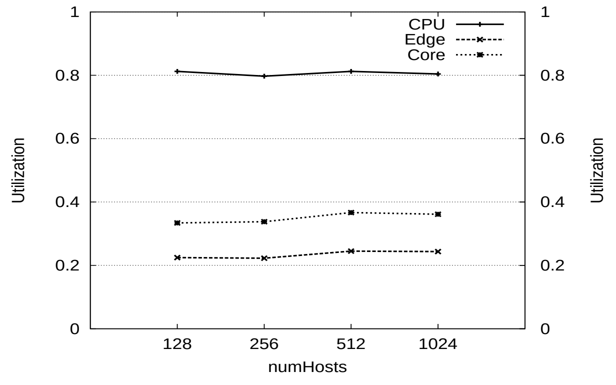


(b) Pattern performance.

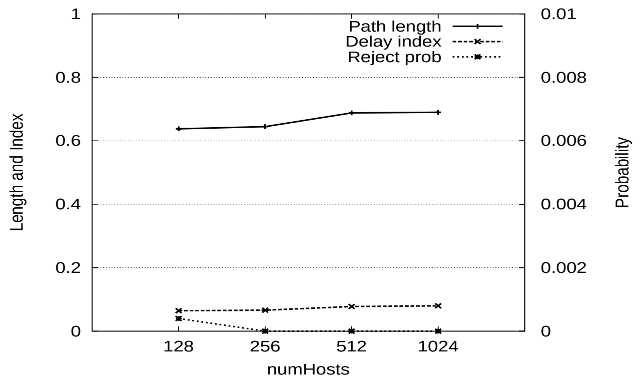


(c) Placement performance.

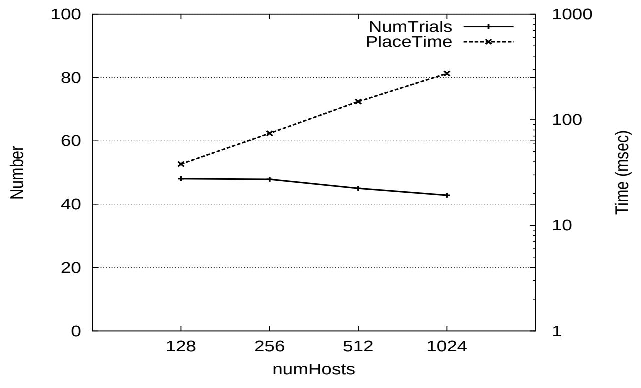
**Figure 9: Effect of load.**



(a) PM and network utilization.



(b) Pattern performance.



(c) Placement performance.

**Figure 10: Effect of number of hosts.**

the complexity to a linear one by applying the biasing on a fixed number of VMs, instead of all remaining VMs in the pattern.

When placing a pattern, we were concerned about 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.

## 8. ACKNOWLEDGMENTS

The author is thankful to Claris Castilo, Ioana Giurgiu, Wolfgang Segmuller, Mike Spreitzer, and Malgorzata Steinder for many fruitful discussions.

## 9. REFERENCES

[1] T. Agarwal, A. Sharma, A. Laxmikant, and L. Kale. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, April 2006.

[2] T. Benson, A. Akella, A. Shaikh, and S. Sahu. Cloudnaas: a cloud networking platform for enterprise applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 8:1–8:13, New York, NY, USA, 2011. ACM.

[3] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837, 2001.

[4] M. Britton, V. Shum, L. Sacks, and H. Haddadi. A biologically-inspired approach to designing wireless sensor networks. In *Wireless Sensor Networks, 2005. Proceedings of the Second European Workshop on*, pages 256 – 266, Jan-Feb 2005.

[5] H. Esbensen and P. Mazumder. Saga : a unification of the genetic algorithm with simulated annealing and its application to macro-cell placement. In *VLSI Design, 1994., Proceedings of the Seventh International Conference on*, pages 211 –214, January 1994.

[6] M. Eshaghian and Y. Wu. Mapping heterogeneous task graphs onto heterogeneous system graphs. In *Heterogeneous Computing Workshop, 1997. (HCW '97) Proceedings., Sixth*, pages 147 –160, April 1997.

[7] M. Gori, M. Maggini, and L. Sarti. Exact and approximate graph matching using random walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1100–1111, 2005.

[8] K.-P. Hui, N. Bean, M. Kraetzl, and D. Kroese. The cross-entropy method for network reliability estimation. *Annals of Operations Research*, 134:101–118, 2005.

[9] 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*, pages 72 –79, July 2011.

[10] N. Koziris, M. Romesis, P. Tsanakas, and G. Papakonstantinou. An efficient algorithm for the physical mapping of clustered task graphs onto multiprocessor architectures. In *Parallel and Distributed Processing, 2000. Proceedings. 8th Euromicro Workshop on*, pages 406 –413, 2000.

[11] K. Macropol and A. Singh. Scalable discovery of best clusters on large graphs. *Proc. VLDB Endow.*, 3:693–702, September 2010.

[12] 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*, pages 1 –9, March 2010.

[13] E. Oki and A. Iwaki. Performance comparisons of optimal routing by pipe, hose, and intermediate models. In *Sarnoff Symposium, 2009. SARNOFF '09. IEEE*, pages 1 –5, April 2009.

[14] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *SIGCOMM Comput. Commun. Rev.*, 33:65–81, April 2003.

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

[16] R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Information in Sciences and Statistics Series. Springer-Verlag New York, LLC, 2004.

[17] 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.

[18] V. Sarkar. *Partitioning and scheduling parallel programs for execution on multiprocessors*. PhD thesis, Stanford, CA, USA, 1987.

[19] C.-C. Shen and W.-H. Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *Computers, IEEE Transactions on*, C-34(3):197 –203, March 1985.

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

[21] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440 – 442, June 1998.

[22] T. Yang and A. Gerasoulis. Pyrros: static task scheduling and code generation for message passing multiprocessors. In *Proceedings of the 6th international conference on Supercomputing*, ICS '92, pages 428–437, New York, NY, 1992. ACM.

[23] 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*, 14(2):154 –162, February 1995.

[24] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.*, 38:17–29, March 2008.

[25] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1 –12, April 2006.

## APPENDIX

## A. CROSS-ENTROPY FOR COMBINATORIAL OPTIMIZATION

We provide a brief overview of the cross-entropy method for combinatorial optimization [15, 16]. Consider the maximization problem of a real-valued objective function $S$ on a finite set of states $\mathcal{X}$, whose solution $\mathbf{x}^*$ defined by

$$S(\mathbf{x}^*) = \gamma^* = \max_{\mathbf{x}\in\mathcal{X}} S(\mathbf{x}). \tag{1}$$

Let $\mathcal{V}$ be a set of vectors of real-valued parameters and define $\{f(\cdot;\mathbf{v}), \mathbf{v} \in \mathcal{V}\}$ as a family of discrete pdfs on $\mathcal{X}$ parametrized by $\mathbf{v}$. Define the associated stochastic problem to optimization problem 1 as

$$\ell(\gamma) = \mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma) = \sum_{\mathbf{x}} I_{\{S(\mathbf{x})\geq\gamma\}} f(\mathbf{x};\mathbf{u}) = \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X})\geq\gamma\}},$$

where $\mathbf{u} \in \mathcal{V}$, $\mathbb{P}_{\mathbf{u}}$ is a probability measure, $\mathbb{E}_{\mathbf{u}}$ denotes expectation, and $\{I_{\{S(\mathbf{x})\geq\gamma\}}\}$ is a collection of indicator functions on $\mathcal{X}$ for various values $\gamma \in \mathbb{R}$. The problem at hand is to be able to estimate parameter values $\mathbf{v}$, use them to generate samples using $f(\cdot;\mathbf{v})$ in such a way to mostly generate the solution $\mathbf{x}^*$ and $\gamma$ is close to $\gamma^*$. The method of minimizing cross-entropy yields an equation for the parameter as

$$\mathbf{v}^* = \operatorname*{argmax}_{\mathbf{v}} \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X})\geq\gamma\}}\ lnf(\mathbf{X};\mathbf{v}).$$

The method provides an iterative algorithm which uses the likelihood ratio estimator to estimate such parameter values leading to an optimal solution. At iteration $t, t = 1, 2, \cdots, T$, of the algorithm, we get estimates for $\gamma$ and $\mathbf{v}$ as $\hat{\gamma}_t$ and $\hat{\mathbf{v}}_t$, respectively, in such a way that $\hat{\gamma}_T$ is close to the optimal $\gamma^*$ and $\hat{\mathbf{v}}_T$ leads to the generation of the optimal solution $\mathbf{x}^*$ given by equation 1. Let $\hat{\mathbf{v}}_{j,t}$ be the $j^{th}$ element of $\hat{\mathbf{v}}_t$, then it is computed at iteration $t$, assuming that we generated $n$ samples $\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_n$ from the density $f(\cdot;\hat{\mathbf{v}}_{t-1})$ as

$$\hat{\mathbf{v}}_{j,t} = \frac{\sum_{i=1}^{n} I_{\{S(\mathbf{X}_i)\geq\hat{\gamma}_t\}}\ I_{\{\mathbf{X}_i\in\mathcal{X}_j\}}}{\sum_{i=1}^{n} I_{\{S(\mathbf{X}_i)\geq\hat{\gamma}_t\}}}, \tag{2}$$

where $\mathcal{X}_j$ is the set of solutions which result with parameter value $\hat{\mathbf{v}}_{j,t}$.

The cross-entropy optimization algorithm follows. It has two parameters: $n$, the number of samples which is typically in the hundreds or thousands depending on the size of the problem, and $\rho$, the fraction of samples which may lead to the optimal solution, typically set around 0.01.

1. Initialization. Set $t = 0$ and some initial value $\hat{\mathbf{v}}_0$.
2. Iteration step. Set $t = t + 1$.
   (a) Generate samples. Generate $\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_n$ from the density $f(\cdot;\hat{\mathbf{v}}_{t-1})$ and compute the $(1-\rho)$-quantile, $\hat{\gamma}_t$.

(b) Adjust parameters. Using generated samples compute $\hat{\mathbf{v}}_t$ using equation 2.

(c) Stopping condition. Check stopping criterion, e.g. minimal change in $\hat{\gamma}_t$.

3. Optimal solution.$\gamma^* = \hat{\gamma}_t$.