RC25548 (WAT1504-054) July 22, 2015
Computer Science

IBM Research Report

Investigating the Impact of Biasing on the Quality of Optimized Application Placement in the Cloud

Asser N. Tantawi
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598 USA

Research Division
Almaden – Austin – Beijing – Brazil – Cambridge – Dublin – Haifa – India – Kenya – Melbourne – T.J. Watson – Tokyo – Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Many reports are available at http://domino.watson.ibm.com/library/CyberDig.nsf/home.

Investigating the impact of biasing on the quality of optimized application placement in the cloud

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

ABSTRACT

We consider a cloud environment, consisting of physical entities, subjected to user application requests, consisting of logical entities with relationship constraints among them, such as location constraints. We are concerned with the application placement problem, which is a mapping of logical to physical entities that satisfies the constraints and optimizes an objective function, which combines system and user performance. The typical problem size, nature of relationship constraints, complexity and adaptability requirement of the objective function, as well as solution timing budget make traditional techniques for solving this combinatorial optimization problem infeasible.

In this paper we describe an efficient technique that is based on random search methods and uses biased statistical sampling methods. In particular, the proposed technique utilizes (1) importance sampling as a mechanism for characterizing the optimal solution through marginal distributions, (2) independent sampling via a modified Gibbs sampler with intra-sample dependency, and (3) a jumping distribution that uses conditionals derived from the relationship constraints given in the user request and cloud system topology, and the importance sampling marginal distributions as posterior distributions. We demonstrate the feasibility of our methodology using several large-size simulation experiments. We note that the magnitude of biasing has an important impact on the quality of placement. Thus, we investigate the tradeoff between biasing and optimality of placement solutions.

1. INTRODUCTION

Cloud services have progressed in recent years from provisioning single Virtual Machines (VM) in the physical cloud infrastructure to virtual platforms and virtual applications, which have become the new cloud workload. Open source programmable interfaces have been developed to interact with the cloud management system and define a software environment describing the cloud infrastructure which consists of compute, storage and network nodes. The user specifies a workload consisting of logical entities, such as VMs, data volumes, communication links, and services, and their needs of the underlying physical resources. Moreover, the user specifies requirements on the provisioned topology of such logical entities. Examples of such requirements include physical proximity of the logical entities, availability/reliability concerns, preferred hosting requirements, licensing and cost issues, and migration requirements. The degree with which such requirements are satisfied during provisioning is a user measure of Quality-of Service. On the other hand, the cloud service provider attempts to maximize the use of the physical resources in a way that provides best performance to users, e.g. load balanced resources. When a user request arrives to the cloud management system, the placement engine decides on a mapping of the logical entities in the request to the physical entities in the cloud system, given its current state, in a way to optimize a given objective function which combines user and provider objectives [3]. This placement optimization problem is quite challenging for several reasons. The size of the problem is quite large, having thousands of physical entities, hundreds of logical entities and hundreds of constraints in a request. Allowing the remapping of existing allocations makes the problem even larger. Typically, the objective function is constructed from policies that users and providers specify. Such policies are not necessarily well-behaved, in the mathematical sense, and subject to change and evolution. Hence, the optimization approach cannot assume and/or exploit properties of the objective function. Needless to say that the placement decision is expected to be fast, i.e. sub-second and not seconds or minutes, in order to cope with the cloud workload traffic and the potential need to redistribute resources through migration of logical entities in the cloud.

We briefly summarize prior work in this area as follows. The original cloud placement problem involved only the placement of one VM in the cloud infrastructure. This gives rise to a bin packing problem with multiple dimensions, where a dimension represents a resource type. Then, multiple VMs were considered along with their communication needs, resulting in the so-called traffic-aware VM placement problem which is naturally quadratic [11, 4]. The inclusion of other objectives resulted in a multi-objective VM placement problem [15]. Extending the modeling of the physical entities in the cloud from resources with given capacities, models were developed to capture performance as a function of load [10]. Special hierarchical structures of clouds were exploited to devise heuristic placement algorithms [1]. Various aspects of cloud resource management were considered [5]. As for the optimization technique itself that is used to solve the cloud placement problem posed as a combinatorial optimization problem, we find a variety of techniques, ranging from heuristic-based, simulation, to evolutionary algorithms [8].

Recently, a technique for attempting to decrease the size of the problem has been proposed [7]. Further, a more promising technique which uses biased sampling along with cross-entropy [12] was introduced [14]. In this paper, we provide a probabilistic framework and statistical sampling method for the technique outlined in [14]. The problem is stated as a search problem [13] and a general random search method [9, 2] is sought. An independent Metropolis-Hastings sampling is performed. In particular, the Gibbs sampling [6] technique is modified and restricted to intra-sample dependency. The constraints specified in the user request are used to construct the conditional probabilities and the importance sampling marginal distributions are employed as posterior distributions.

The paper is organized as follows. The cloud placement problem and the motivation for biasing the solution are discussed in Section 2. The problem statement and related definitions are provided in Section 3. The solution approach and algorithm are described in Section 4. A particular choice of biasing functions is given in Section 5. In Section 6 we present simulation results demonstrating the efficiency of our algorithm. Further, we investigate the sensitivity of the placement solution to the values of bias factors. Section 7 concludes the paper and suggests further work.

2. CLOUD PLACEMENT

2.1 Placement Engine

Basically, a cloud provider possesses the cloud infrastructure which consists of physical entities (PE) such as Physical Machines (PM), storage devices, and communication networks. Such an infrastructure is subjected to a stream of requests from cloud users, where each request represents an application (also referred to as pattern and/or workload) that the cloud user wants deployed in the cloud. The request specifies the logical entities (LE) of the application, such as Virtual Machines (VM), data volumes, and virtual networks. Further, the request includes constraints related to the deployment of the applications. Some constraints relate to individual LE, such as specific properties and/or resource demands. Other constraints relate to pairs of LEs, such as communication needs between them. Further, other constraints relate to a group of homogeneous LEs, such as collocation (or anti-collocation) of member LEs of the group at some level in the physical topology of the cloud. (Also known as affinity and anti-affinity constraints, respectively.)

Given such an environment, the result of each request is a placement, i.e. a mapping of LEs in the request to PEs in the cloud. Typically, there are objectives for such a placement dictated by policies addressing both the cloud provider and the cloud user. Hence, with each request, we are faced with an assignment optimization problem. As depicted in Figure 1, the placement algorithm is embedded in a placement engine. The placement engine receives requests for pattern deployment. The observer component monitors the cloud system and provides periodic updates to the state of the cloud model utilized by the placement engine. The various policies are provided through a management layer to the placement engine, yielding the objective function of the optimization problem. And, once the placement problem is solved, the resulting optimal assignment is conveyed as the

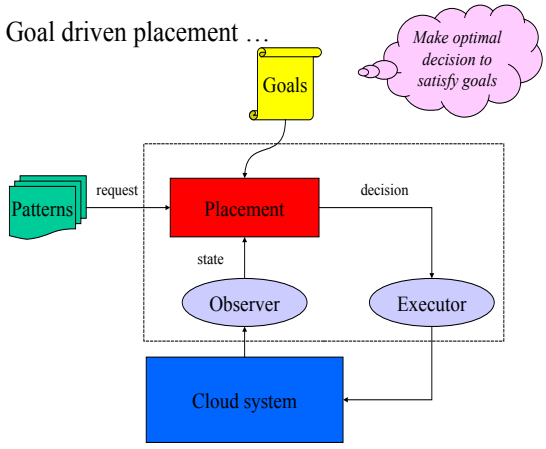


Figure 1: Cloud placement engine.

decision of the placement engine to the Executor component, which in turn realizes the change in the cloud infrastructure.

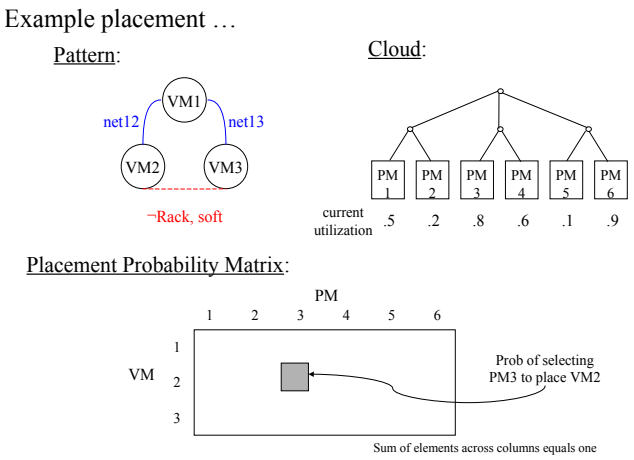
It is worth noting that a straightforward extension of this application addresses the dynamic version of the placement problem, namely, the periodic migration of LEs as the performance of the cloud and/or applications deteriorate. In such a case, we are faced, again, with an assignment problem, albeit the size of the request is much larger, since it involves all migrating LEs, and the objective function includes an additional cost function related to migration.

2.2 Motivation for Biasing

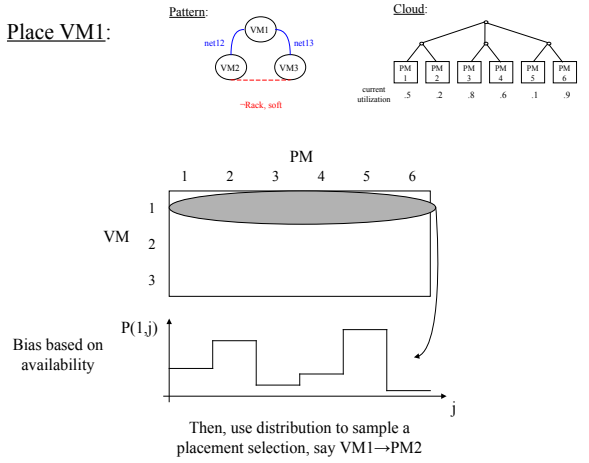
Given the large size of the placement problem, the optimization algorithm searches for good solutions in a combinatorially explosive state space. A promising approach is to bias the search towards a subset of good solutions. In this section, we describe the mechanics and intuition behind the use of biasing through examples.

Consider the simple example illustrated in Figure 2. The cloud infrastructure consists of six homogeneous PMs, PM1 through PM6, having a single resource type, CPU cores. The current CPU utilization of the PMs is shown, where PM6 is highly utilized (0.9) and PM5 is least utilized (0.1). The cloud topology is a tree of height 2, where the leaves at level 0 represent the PMs, the nodes at level 1 represent the racks, and the root at level 2 represents the entire cloud system. As shown, there are 3 racks in the system, each housing 2 PMs. Assume that this tree containment hierarchy describes the communication network connectivity as well. In other words, the PMs communicate through a tree network which matches the cloud hierarchy.

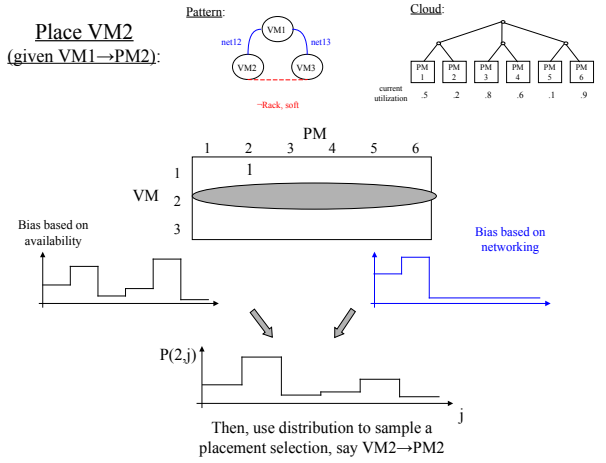
Consider a request for a pattern (application) which consists of 3 homogeneous VMs, VM1 through VM3. As illustrated in Figure 2(a), VM1 and VM2 have communication need with bandwidth net2, and VM1 and VM3 have communication need with bandwidth net3. Further, VM2 and VM3 are to be placed on different racks. The later location constraint is specified as soft, as opposed to a hard constraint. Thus, closeness to achieving such a constraint is a component of



(a) Cloud and pattern request configuration.



(b) Step 0: Place VM1.



(c) Step 1: Place VM2, given VM1 is placed on PM2.

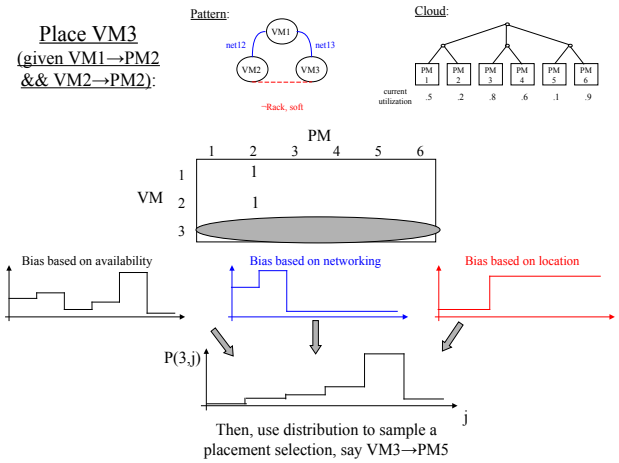
Figure 2: Example of biasing (part I)

the overall objective function. Let **P** be the placement probability matrix, where the rows (columns) represent the VMs (PMs), respectively, as depicted pictorially in Figure 2(a). Assume that the initial values of **P** are proportional to PM availability, i.e. all rows are equal and each element is proportional to (1-utilization), s.t. each row sums up to one.

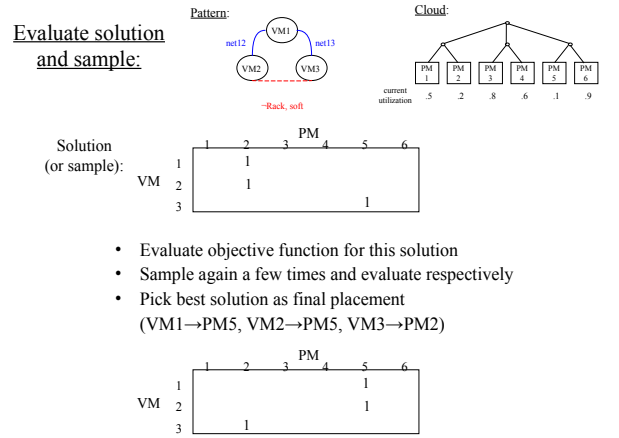
Assume that the provider objective is to balance load and the user objective is, in addition to satisfying the resource and networking requirement, to minimize communication paths length and satisfy the soft location constraint as best as possible.

The VMs in the pattern are initially ordered according to their placement complexity. In other words, a VM with higher resource demands comes first. Further, a VM with more constraints comes first. We use a heuristic function of resource needs and constraints to order the VMs. Assume that the resulting order is [VM1, VM2, VM3].

In this example we employ three biasers: resource usage, networking, and location. In step 0, as illustrated in Figure 2(b), VM1 is placed by sampling from the probability



(d) Step 2: Place VM3, given VM1 and VM2 placed on PM2.



(e) Step 3: Evaluate solution.

Figure 3: Example of biasing (part II)

distribution given by the first row, $\mathbf{P}_1$, which favors PMs with higher resource availability. Let's assume the outcome of the draw was that PM2 hosts VM1.

In step 1, as illustrated in Figure 2(c), VM2 is to be placed given the fact that VM1 is already placed on PM2. Using the distribution given by the second row, $\mathbf{P}_2$, without any biasing for the conditional knowledge of VM1 placement, would not likely to result in a placement for VM2 that would yield a small value for the objective function. Hence, we need to bias $\mathbf{P}_2$ to account for the fact that VM1 is placed on PM2. More precisely, the resource usage has changed and the communication need net2 between VM1 and VM2 induce two biasing functions as shown in Figure 2(c). Both functions are distributions over the PMs, the first reflecting resource availability and the second favoring PMs that are closer (in terms of path lengths, or may use other measures such as path congestion) to PM2. The two biasing distributions are combined through a weighted product with the original marginal distribution $\mathbf{P}_2$ to yield a new distribution, from which a sample PM is drawn for VM2. Let's assume that PM2 was chosen to host VM2.

In step 2, as illustrated in Figure 3(d), VM3 is to be placed given the fact that both VM1 and VM2 are already placed on PM2. In a similar fashion to the placement of VM2 we create usage and networking biasing distributions. However, due to the additional location constraint between VM2 and VM3, we create an additional location biasing distribution. The location constraint favors for VM3 a PM in a different rack than where VM2 is placed. Thus, as illustrated in Figure 3(d), the location biasing function is high for PM3 through PM6, whereas it is low for PM1 and PM2. If the location constraint was hard instead, the low value would have been set to zero. In a similar fashion, we compute a new distribution from $\mathbf{P}_3$ and the three biasing distributions, from which a sample PM is drawn for VM3. Let's assume that PM5 was chosen to host VM3.

In step 3, as illustrated in Figure 3(e), all VMs in the pattern are placed and we have a sample solution described by the deterministic assignment probability matrix shown in the figure. The best and final placement was the assignment $\{VM1 \rightarrow PM5, VM2 \rightarrow PM5, VM3 \rightarrow PM2\}$. The objective function is evaluated for such a solution, and the process is repeated K times. When done the solutions are ordered according to their corresponding objective values. Then, a fraction of the best solutions, say ρK where ρ is a small fraction such as 0.1 or 0.05, is chosen as a set $\mathcal{A}$ of important samples. A corresponding new stochastic matrix $\mathbf{P}'$ is calculated and is used as generator of samples in the subsequent iteration as will be described in Section 4.

3. PROBLEM STATEMENT

We use the following notation. For vectors and matrices, we use boldface capital letters, e.g. $\mathbf{V}$ and $\mathbf{M}$. A corresponding small letter denotes an element in the vector or matrix, e.g. v and m. A subscripted small letter represents a particular element given by the value of the subscript, e.g. v_i and $m_{i,j}$. For convenience, a subscripted capital letter representing a matrix denotes a vector row in the matrix, where the row number is given by the value of the subscript, e.g. $\mathbf{M}_i$. (We will not need to denote vector columns in matrices.) The 1-norm of a vector is denoted by $\|\mathbf{V}\|$, which is the sum of the absolute values of its elements. For sets, we use a calligraphic capital letter, e.g. $\mathcal{S}$. A normal capital letter is an integer, and its corresponding small letter takes values in the enumeration from one to the value of the capital letter, e.g. I and $i = 1, 2, \cdots, I$.

Define the sets $\mathcal{M} = \{1, 2, \cdots, M\}$ and $\mathcal{N} = \{1, 2, \cdots, N\}$, where $M, N \geq 1$. Let $\mathbf{X} = [x_1\ x_2\ \cdots\ x_M]$ be a vector representing variables taking values in $\mathcal{N}$, i.e. $x_m \in \mathcal{N}$, $m \in \mathcal{M}$. We refer to a particular valued vector $\mathbf{A} = [a_1\ a_2\ \cdots\ a_M]$, where $a_m \in \mathcal{N}$, $m \in \mathcal{M}$, as an assignment to $\mathbf{X}$.

Define a scalar objective function $f(\mathbf{X})$ with range $\mathbb{R}$, the set of real numbers. The unconstrained state space for variable $\mathbf{X}$ is the cartesian power $\mathcal{N}^M$. Let $\mathcal{S}$ denote a constrained, nonempty state space, $\mathcal{S} \subseteq \mathcal{N}^M$. The optimization problem is stated as,

$$\min_{\mathbf{X}} f(\mathbf{X}), \quad \mathbf{X} \in \mathcal{S}. \tag{1}$$

We use the notation $\mathbf{X}_{<m}$ to denote the variable vector $\mathbf{X}$ excluding the elements $\{x_m, x_{m+1}, \cdots, x_M\}$, where $m = 2, \cdots, M$ and $M \geq 2$.

In relation to the cloud placement (assignment) problem, we have N physical entities, M logical entities in the user request, $\mathbf{X}$ is a variable mapping logical to physical entities, $\mathbf{A}$ is a particular mapping (solution to the problem), and S the set of possible solutions given the requirement constraints specified in the user request. The objective function $f(\mathbf{X})$ combines user and provider objectives. We do not make assumptions about $f(\mathbf{X})$ other than it could be numerically evaluated given $\mathbf{X}$ and the current state of the system.

Define $\mathcal{A}$ as an arbitrary, nonempty subset of $\mathcal{S}$, i.e. $\emptyset \neq \mathcal{A} \subseteq \mathcal{S}$. Let $\mathcal{A}$ contains $L \geq 1$ unique assignments, i.e. $\mathcal{A} = \{\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_L\}$, where $\mathbf{A}_l = [a_{l1}\ a_{l2}\ \cdots\ a_{lM}]$, $l \in \mathcal{L}$, is an assignment vector. If $L = 1$ then $\mathcal{A}$ represents the set consisting of a unique solution $\mathbf{A}_1$.

Define an $M \times N$ (row) stochastic matrix $\mathbf{P}$, i.e. element $p_{m,n} \in [0, 1]$, $\|\mathbf{P}_m\| = 1$, $m \in \mathcal{M}$, and $n \in \mathcal{N}$. We refer to a stochastic matrix $\mathbf{P}$ as *deterministic* if the elements are such that $p_{m,n} \in \{0, 1\}$.

Given a set of assignments $\mathcal{A}$, define the *generator* $\mathbf{G}(\mathcal{A})$, as a generator of assignments using stochastic matrix $\mathbf{P}$, such that $p_{m,n}$ is the probability that $a_{lm} = n$ over $l \in \mathcal{L}$. Hence, $\mathbf{P}_m$ represents the marginal probability distribution of the m^{th} element of the assignments in $\mathcal{A}$.

4. SOLUTION

Since the objective function $f(\mathbf{X})$ in Equation 1 could be quite general, we develop a solution approach that does not rely on properties, such as convexity, nor devise heuristics implied by its shape. Rather, we consider the optimization problem as a general search problem [13] for an optimal $\mathbf{X}^*$ with minimum $f(\mathbf{X}^*)$ in the solution space S.

4.1 Random Search

A generalized random search method [2] consists of the following steps. (A stopping criterion is expected but not spec-

ified.)

1. Select a starting point $\mathbf{X}(0) \in \mathcal{S}$ and an initial estimate of the optimal solution $\mathbf{X}^*(0) \in \mathcal{S}$. Let $k = 0$.
2. Generate a candidate solution $\mathbf{X}'(k) \in \mathcal{H}(k)$, where $\mathcal{H}(k)$ defines a neighborhood of solutions around $\mathbf{X}(k)$ s.t. $\mathcal{H}(k) \subset \mathcal{S} \setminus \mathbf{X}(k)$.
3. Determine the next point $\mathbf{X}(k+1) \in \{\mathbf{X}(k), \mathbf{X}'(k)\}$, using $f(\mathbf{X}(k))$ and $f(\mathbf{X}'(k))$.
4. Obtain a new estimate of the optimal solution $\mathbf{X}^*(k+1)$. Let $k = k+1$ and go to step 2.

There are several choices in this general search method: (1) defining the neighborhood $\mathcal{H}(k)$; (2) generating a candidate solution $\mathbf{X}'(k)$; (3) determining a next point $\mathbf{X}(k+1)$; and (4) estimating an optimal solution $\mathbf{X}^*(k)$.

Examples of optimization algorithms that follow this random search paradigm are simulated annealing (SA) and evolutionary computation, such as genetic algorithms. In contrast, we introduce a new class of random search algorithms which uses importance sampling and biasing. For short, we refer to this algorithm as Biased Sampling Algorithm (**BSA**). In particular, our solution approach makes the following choices: (1) the neighborhood $\mathcal{H}(k)$ is characterized by the marginal probability distributions of the m^{th} element in $\mathbf{X}$; (2) candidate solution $\mathbf{X}'(k)$ is generated using a modified Gibbs sampling as described below; (3) the next point $\mathbf{X}(k+1)$ is the generated point $\mathbf{X}'(k)$; (Hence, BSA is different than SA as it does not attempt to walk from point to point in the search space.) (4) an estimate of optimal solution $\mathbf{X}^*(k)$ is generated using importance sampling technique [12]. The optimality of BSA has been demonstrated on small-sized problems [14]. The main idea of importance sampling 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 *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.

4.2 BSA Approach

A straightforward implementation of the random search algorithm to the cloud placement problem has been shown to be impractical [14]. Rather than the single loop outlined in the random search algorithm above, we use two loops: an outer loop and an inner loop. The outer loop is related to the estimation of the optimal solution. It uses the importance sampling technique, applied to all sample points generated in the inner loop, to create a generator of points in the subsequent execution of the outer loop. In particular, we use a generator $\mathbf{G}(\mathcal{A})$, where $\mathcal{A}$ is the set of important samples generated in the inner loop. And, the inner loop relates to generating candidate points using the generator provided by the outer loop. Hence, the neighborhood $\mathcal{H}(k)$ is defined by the outer loop generator, candidate solution points in the inner loop are sampled using the generator provided by the outer loop. And, the outer loop generator is computed using the importance sampling technique applied to the generated candidate solution points in the prior execution of the outer loop. A high level description of the BSA algorithm follows.

1. Iteration $\iota = 0$. Initialize generator $\mathbf{G}_\iota$.
2. While stopping criterion not met. (Outer loop)
   (a) For $k = 1, 2, \cdots, K$. (Inner loop)
      i. Generate $\mathbf{X}(k)$ through sampling using generator $\mathbf{G}_\iota$, and applying biasing (as described in Section 4.3).
   (b) $\iota = \iota + 1$. Create set $\mathcal{A}$ including the $L \ll K$ best points $\mathbf{X}(k)$, i.e. with minimum $f(\mathbf{X}(k))$. Create a new generator $\mathbf{G}_\iota(\mathcal{A})$.

After the stopping criterion is met, we use the sample with the minimum $f()$, throughout, as the solution to the problem.

Let $\mathbb{B} = \{\mathbf{B}(r);\ r \in \mathcal{R}\}$ be a family of R stochastic matrices, each of size $M \times N$, where $\mathcal{R} = \{1, 2, \cdots, R\}$, $R \geq 1$. We refer to $\mathbb{B}$ as a set of biasing matrices. For R biasing criteria, $\mathbf{B}(r)$ represents the basing matrix for criterion $r, r \in \mathcal{R}$. Each criterion represents a type of requirement constraint in the user request, e.g. communication, location, target preference, license usage, and cost constraints. Define a weight vector $\mathbf{W}$ of length R, where element $w_r \geq 0$ is a weight associated with $\mathbf{B}(r)$, $r \in \mathcal{R}$.

4.3 BSA Algorithm

An outline of the BSA placement algorithm follows.

1. $\iota = 0$. Initially, set the stochastic matrix $\mathbf{P}(0)$ proportional to resource availability, i.e. $p_{m,n}(0) = 1 - u(PM_n)$, $m \in \mathcal{M}$, and $n \in \mathcal{N}$, where $u()$ represents the utilization of the bottleneck resource or a measure of utilization of the multiple resources on a PM. (This particular choice helps achieve load balancing. Other expressions may be needed for different objectives.)
2. Use $\mathbf{P}(\iota)$ to generate K independent samples (solutions).
   (a) A sample $\mathbf{X}$ is constructed incrementally, one element at a time. After $(m-1)$ elements are generated, where $m = 2, \cdots, M$, we have $\mathbf{X}_{<m}$. The element x_m is generated given $\mathbf{X}_{<m}$. In other words, the set $\mathbb{B}$ are filled in as conditional probabilities given $\mathbf{X}_{<m}$. Hence, we generate the elements as per the Gibbs sampling method, except that we remove the dependency on the previous sample. This yields independent samples, rather than a Markov Chain Monte Carlo sequence. (In general, the chain resulting from independent samples behaves well if the jumping distribution has a heavier tail than the posterior marginal distributions.)
   (b) We evaluate the one-step jumping stochastic matrix $\mathbf{P}'(\iota)$ as follows. Let $\mathbf{B}$ denote the weighted product of $\mathbf{B}(r)$, given by

$$\mathbf{B} = \underset{r \in \mathcal{R}}{\circ} \mathbf{B}(r)^{w_r}, \tag{2}$$

   where the symbol $\circ$ represents the Hadamart element wise product of matrices, and the exponent

w_r applies to all elements of matrix $\mathbf{B}(r)$. Then, we write

$$\mathbf{P}'(\iota) = \mathbf{diag}(\mathbf{C}) \ (\mathbf{P}(\iota) \circ \mathbf{B}), \quad (3)$$

where $\mathbf{C}$ is a normalization constant vector of length M to make $\mathbf{P}'(\iota)$ stochastic.

3. Order the K generated samples w.r.t. $f()$ and select the best top portion of the samples. Represent each selected sample by its corresponding *deterministic* stochastic matrix, add all such matrices elementwise and normalize to generate a new stochastic matrix $\mathbf{P}(\iota+1)$. The latter is a characterization of the top generated samples. In such a matrix, $\mathbf{P}_m(\iota+1)$ represents the marginal probability distribution of the m^{th} element in the optimal solution.
4. $\iota = \iota + 1$. Go to step 2 until a stopping criterion is satisfied.

5. FUNCTIONAL DEFINITIONS

5.1 Biasing Functions

In this section we describe the biasing $\mathbb{B}$ that we use in our implementation of the BSA algorithm. Obviously, the functions are parameterized and the effect of such parameters on the quality of the placement results is studied in Section 6. We consider two biasing functions: usage and location. Other functions follow a similar mechanism for their creation.

Without loss of generality, we consider only PMs as PEs and VMs as LEs. Let $\mathcal{PM}$ denote the set of N physical machines in the cloud, $N = |\mathcal{PM}|$. We will refer to an element in the set as $pm_n, n = 1, 2, \cdots, N$. Each PM provides a set of resources, $\mathcal{E}$, consisting of resources $r_e, e = 1, 2, \cdots, E$. Examples of such resources are CPU, memory, and disk storage. The total capacity of resource r_e on pm_n is denoted by $c_{e,n}$. The utilization of such a resource is denoted by $u_{e,n} \in [0, 1]$.

We assume that the cloud system forms a hierarchical tree with height L, where the leaves are the PMs and an intermediate node represents a zone of availability. Levels are defined as follows. A node at level l, $l = 0, \cdots, L$, represents a leaf if $l = 0$ and the root if $l = L$. For convenience we define $g_n(l), n = 1, 2, \cdots, N$, and $l = 0, \cdots, L$ as the set of PMs such that for $pm_{n'} \in g_n(l)$ we have pm_n and $pm_{n'}$ with the lowest common ancestor at level l.

A VM is characterized by a set of resource demands, one per resource type in the set $\mathcal{E}$. We refer to the PM which hosts vm_m as $pm(vm_m)$. The resource demand of vm_m for resource r_e is denoted by $d_{e,m}$. Assuming that over-utilization is not allowed, then before placing vm_m on $pm(vm_m)$, it must be that

$$d_{e,m} \leq (1 - u_{e,pm(vm_m)}) c_{e,pm(vm_m)},$$

for all $e = 1, 2, \cdots, E$. A pattern is a collection of M VMs that make up a deployable application unit given by the set $\mathcal{VM} = \{vm_1, vm_2, \cdots, vm_M\}$.

Location constraints are expressed as follows. Let $\mathcal{S} \subset \mathcal{VM} \times \mathcal{VM}$ be a set of distinct pairs of VMs in the pattern. A pair $(vm_m, vm_{m'}) \in \mathcal{S}$ has a location constraint specified with a desired l and an achieved level given by $v_{pm(vm_m),pm(vm'_m)}$. This requirement is satisfied if $pm(vm_{m'}) \in g_k(l)$, where $pm(vm_m) = pm_k$ for some $l, 0 \leq l \leq L$. As a simple extension, one may specify a range of levels $[l_{inf}, l_{sup}]$, where $0 \leq l_{inf} \leq l_{sup} \leq L$, instead of the fixed value l. In this case, it is straightforward to handle a collocation constraint at level l as a range $[0, l]$ and anti-collocation constraint at level l as a range $[l+1, L]$. Further, location constraints are specified as hard or soft.

We place VMs in the pattern in a sequential manner, without backtracking, i.e. once $vm_{m'}, m' = 1, 2, \cdots, m-1$, are placed, the choice for placement is only left for $vm_m, \cdots, vm_M$. Once $vm_{m'}$ is placed on say $pm_k = pm(vm_{m'})$, we examine any location constraint with $vm_m, m = m'+1, \cdots, M$ in a look-ahead fashion. More precisely, we apply biasing functions for the choice of placement of vm_m given the placement of $vm_{m'}, m' = 1, 2, \cdots, m-1$. We consider two biasing functions: usage and location, abbreviated as *usg* and *loc*, respectively.

5.1.1 Usage Biasing

Let $\mathbf{B}(usg)$ be the biasing probability matrix for usage biasing and $bias(usg)_{m,n}$ be the bias of placing vm_m on pm_n. The probability distribution given by $\mathbf{B}(usg)_m$ is the vector resulting from normalizing $\mathbf{bias}(usg)_{m,n}, n = 1, 2, \cdots, N$. The value of $bias(usg)_{m,n}$ is calculated as follows. Assuming an objective of load balancing, biasing should be towards a PM with higher resource availability. (Other objectives may be handled according to the objectives.) Without loss of generality, assume that resource r_1 is the resource of concern (i.e. the bottleneck resource). Let $u'_{1,n}$ be the current utilization of $r_{1,n}$, i.e. before considering placing vm_m on pm_n. And, let $U(m, n)$ be the respective utilization after placing vm_m on pm_n, i.e.

$$U(m, n) = u'_{1,n} + d_{1,m}/c_{1,n},$$

The value of $\mathbf{bias}(usg)_{m,n}, n = 1, 2, \cdots, N$ should be a non-increasing function of resource utilization. We use a simple function given by

$$\mathbf{bias}(usg)_{m,n} = \begin{cases} (1 - u_{1,n})^{\beta_{usg}}, & U(m,n) \leq 1, \\ 0, & U(m,n) > 1, \end{cases} \quad (4)$$

where $\beta_{usg} \geq 0$ is a parameter which we refer to as the *usage bias factor*. A value of $\beta_{usg} = 0$ corresponds to no biasing at all, and the higher the value of β_{usg} the more biasing is applied.

5.1.2 Location Biasing

Let vm_m and $vm_{m'}$ have a location constraint at level l. Then, we need to bias $p_{m,n}$ positively towards $pm_n \in g_k(l)$, where $pm_k = pm(vm_{m'})$, 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 $pm_n \in g_k(l-1) \cup g_k(l+1)$, $pm_n \in g_k(l-2) \cup g_k(l+2)$, and so on. That is if the constraint is soft on both sides of the desired location level. Otherwise, it would consider only the higher levels.

Let $\mathbf{B}(loc)$ be the biasing probability matrix for location biasing and $bias(loc)_{m,n}$ be the bias of placing vm_m on pm_n. The probability distribution given by $\mathbf{B}(loc)_m$ is the vector resulting from normalizing $\mathbf{bias}(loc)_{m,n}, n = 1, 2, \cdots, N$.

The value of $bias(loc)_{m,n}$ is calculated as follows. Let $v(m, m', n)$ be the level of the lowest common ancestor of $pm(vm_{m'})$ and pm_n, where $(vm_m, vm_{m'}) \in \mathcal{S}$ and $m' = 1, 2, \cdots, m-1$. Let $dev(l, l') \in [0, 1]$ be a measure of deviation for a location constraint with desired level l and achieved level l'. Define the deviation measure as

$$dev(l, l') = \frac{|l - l'|}{L},$$

where $l, l' = 0, 1, \cdots, L$. Thus, a deviation of zero (one) corresponds to a best (worst) case for the constraint. The value of $\mathbf{bias}(loc)_{m,n}$ should be a non-increasing function in the amount of deviation. We use a simple power function given by

$$\mathbf{bias}(loc)_{m,n} = \beta_{loc}^{\tau \sum_{m'}(1-2\, dev(l, v(m,m',n)))}, \qquad (5)$$

where the sum over m' covers $(vm_m, vm_{m'}) \in \mathcal{S}$ and $m' = 1, 2, \cdots, m-1$, and $\beta_{loc} \geq 0$ and $\tau > 0$ are parameters. The bias value ranges from a maximum bias of β_{loc}^{τ} and a minimum bias of $\beta_{loc}^{-\tau}$. We arbitrarily choose $\tau = 3$. However, the choice of β_{loc}, which we refer to as the *location bias factor*, is crucial in determining the amount of bias applied when searching for a "good" solution. A value of $\beta_{loc} = 0$ corresponds to no biasing at all, and the higher the value of β_{loc} the more biasing is applied. In general, in the case of conflicting constraints in the placement of a pattern, it is not desirable to apply a lot of bias for each constraint. This may force the search for solution towards the infeasible region. The investigation of the impact of the value of the bias factor is studied in Section 6.

5.2 Objective Function

The objective function $f(\mathbf{X})$ is a weighted sum of system (provider) objective and pattern (user) objective, as a result of placement $\mathbf{X}$, given by

$$f(\mathbf{X}) = \frac{w_{sys} f_{sys}(\mathbf{X}) + w_{pat} f_{pat}(\mathbf{X})}{w_{sys} + w_{pat}}. \qquad (6)$$

The system objective is taken to be the standard deviation of the utilization of the prime resource across the cloud system, i.e. the objective is to balance the load across the cloud. The pattern objective captures the deviation from the desired location constraints specified in the pattern request. For all m and m' s.t. $(vm_m, vm_{m'}) \in \mathcal{S}$, let $desired(m, m')$ be the desired level of the pair-wise constraint and $achieved(m, m') = v_{pm(vm_m), pm(vm'_m)}$ be the achieved level. Then, we write

$$f_{pat}(\mathbf{X}) = \frac{1}{|\mathcal{S}|} \sum_{m,m'} dev(desired(m, m'), achieved(m, m')), \qquad (7)$$

$\forall m, m'$ s.t. $(vm_m, vm_{m'}) \in \mathcal{S}$.

6. EXPERIMENTAL RESULTS

We briefly describe the setup and present the performance of our placement algorithm as well as an investigation of the impact of the choice of bias factors on the quality of placement solutions. The BSA algorithm is coded in C and runs on a MacBook Pro with 2.4 GHs Intel Core 2 Duo and 4GB RAM, running Mac OS X 10.9.5 with optimized code. The number of samples generated per BSA iteration is $K = 20$ and a fraction $\rho = 0.1$ of those is used as important samples. The stopping criterion is a relative improvement in the objective function of less than 0.001, or a maximum number of iterations set at 10. The weights in the objective function are set to $w_{sys} = w_{pat} = 1$. Further, $R = 2$ biasing distributions are used, one for usage and the other for location. The biasing weight vector $\mathbf{W}$ is $\mathbf{1}$.

6.1 Setup

We consider a cloud system which consists of 1024 PMs, each with CPU resource capacity of 8 units. The cloud system has a balanced tree topology of height 4 with widths 2, 4, 8, and 16, respectively from the root downward. In other words, the cloud system consists of 2 data centers, where each data center has 4 zones, each zone consists of 8 racks, and each rack holds 16 PMs. The system is simulated, starting from an empty system, and subjected to a load of 0.8. This is done through simulating a stream of Poisson requests, each with a uniformly distributed lifetime, spanning an average lifetime, such that the average CPU utilization across the cloud system reaches 0.8 in steady state, given that no request was dropped due to placement failure. After the initial warm-up period, the average utilization varied in the range [0.64, 1.00], with an average of 0.81 and standard deviation of 0.07.

We consider several pattern configurations in three experiments to demonstrate the efficiency of the BSA algorithm and investigate the sensitivity of the bias factors.

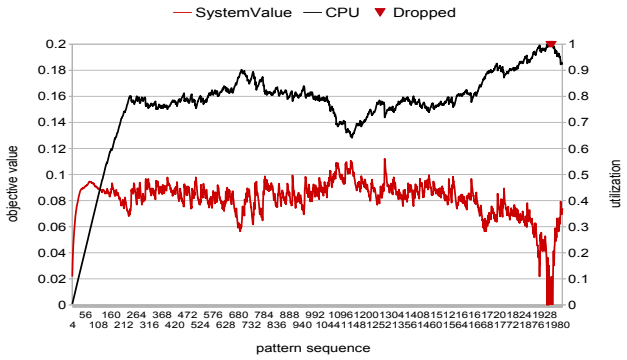
6.2 Experiment 1

The pattern in this experiment consists of 32 independent homogeneous VMs, each with demand 1 CPU units. Hence, there are no location constraints among the VMs. Starting from an idle system, steady state (average utilization of 0.8) was reached after 234 request arrivals. An additional 1,766 requests were simulated in steady state. Only 7 of those requests were dropped only due to lack of resources, i.e. there were no placement failures resulting from our BSA algorithm. Therefore, the probability of request drop was 0.004. All statistics are based on the successful 1,759 requests that were admitted and placed into the system.

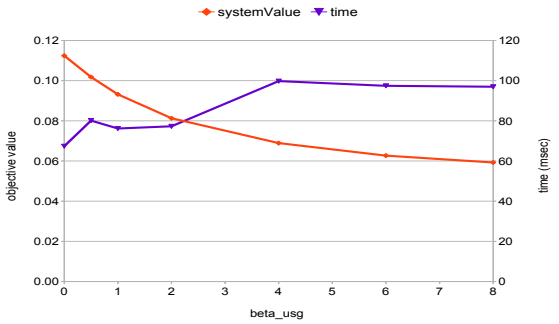
In this case, due to independence, we have $f_{pat}(\mathbf{X}) = 0$, and therefore we get from Equation 7 that $f(\mathbf{X}) = f_{sys}(\mathbf{X})$, which is the standard deviation of the CPU utilization across the cloud system.

In Figure 4(a), we use $\beta_{usg} = 2$ where the x-axis shows the sequence of pattern arrival requests and the (left) y-axis shows the value of the objective function, the standard deviation of CPU utilization in this case, and the (right) y-axis shows the average utilization of the cloud system, i.e. the load at the time. At each point on the x-axis, i.e. request arrival, the placement problem is solved for the incoming request through invoking the BSA algorithm. Note that as the utilization increases, the corresponding standard deviation decreases simply due to the shrinking of resource availability range. At the steady state utilization of 0.8, the objective was kept around 0.08.

In Figure 4(b), we vary the *usage bias factor* β_{usg} in the range $[0, 8]$. The values plotted for the average objective function and algorithm execution time are averaged over the

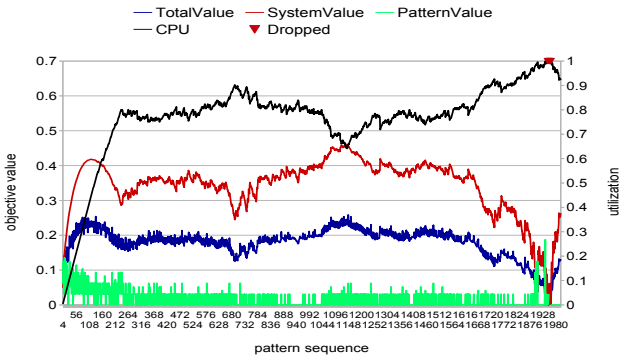


(a) Time series. ($\beta_{usg} = 2$)

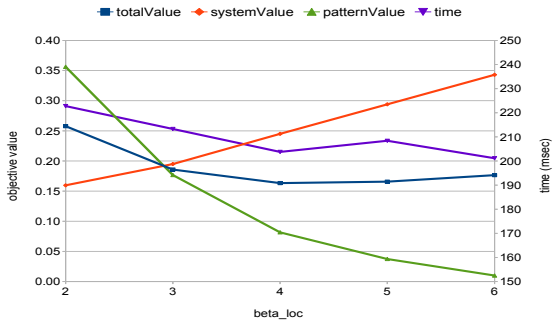


(b) Effect of β_{usg}.

Figure 4: Experiment 1.



(a) Time series ($\beta_{loc} = 6$).



(b) Effect of β_{loc}.

Figure 5: Experiment 2. ($\beta_{usg} = 2$)

steady state period of the simulation. Overall, we note that the average execution time is 100 msec, or less. Small values of β_{usg} result in less iterations in the algorithm since the stopping criterion kicks in earlier, hence the smaller average execution time of about 80 msec. As for the value of the objective function, we note that it improves, i.e. decreases, as β_{usg} increases. The rate of improvement decreases from an initial 0.0212, going from 0 to 0.5, to an insignificant 0.0017, going from 6 to 8. We use a default value of $\beta_{usg} = 2$.

6.3 Experiment 2

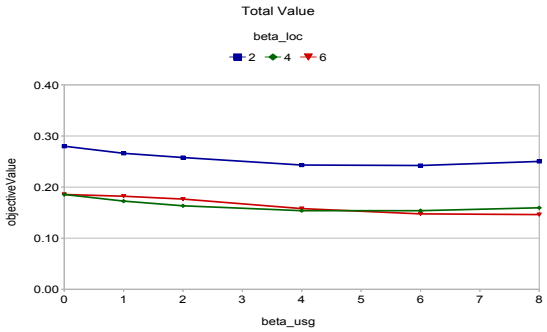
In this experiment we consider location constraints. In particular, a pattern in this experiment consists of 4 groups of homogeneous VMs, each with demand 2 CPU units, denoted by group1, group2, group3, and group4. Each group consists of 4 VMs. The location constraints are set to model a need for hard availability requirements among the groups, yet with soft closeness of VMs within each group (to minimize communication overhead). The inter-group location constraints are all hard and specified as follows: anti collocation at the data center level between group1 and group2, as well as between group3 and group4; and anti-collocation at the zone level between group2 and group3, as well as between group4 and group1. The intra-group location constraints are all soft and specified as collocation at the PM level. As in experiment 1, steady state was reached after 234 arrivals, with additional 1,766 requests in steady state. Only 7 of those requests were dropped only due to lack of resources.

In Figure 5 we keep $\beta_{usg} = 2$. In Figure 5(a) we show the time series of one experiment where we set $\beta_{loc} = 6$. The total value of the objective function is a weighted sum of system value and pattern value. As before, the system value is the standard deviation of CPU utilization. The pattern value is the deviation of the intra-group location constraints from the desired level of same PM. Given that each group has a size 4, then the total number of edges in a group is 6, hence a total number of edges of 24 over all 4 groups. In case only one VM in only one group is placed in a different PM than VMs of its group, but in the same rack, 3 edges to the other 3 VMs in the group would deviate by 1 level. Hence, in this case the pattern value, as given in Equation 7, would be $3/(24 * 4) = 0.031$. This is the most frequent non-zero value of the pattern value illustrated in Figure 5(a). We also see roughly multiples of such a value, especially during the warm up period. Though not visible due to the thickness of the pattern value curve, the pattern value was zero with probability 0.71 during steady state. In other words, all soft constraints in the pattern were fully satisfied.

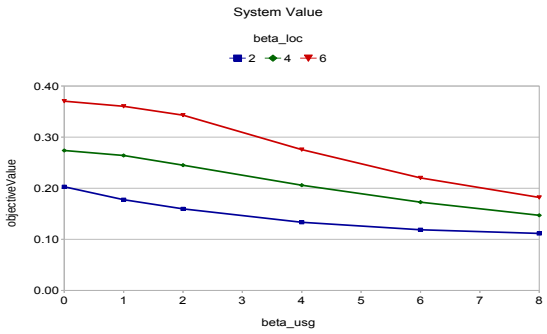
β_{loc}	Total	System	Pattern	Satisfied
2	0.2578	0.1596	0.3563	0.0000
3	0.1858	0.1950	0.1767	0.0006
4	0.1634	0.2451	0.0816	0.0404
5	0.1657	0.2940	0.0374	0.2297
6	0.1765	0.3430	0.0100	0.7089
hard	0.1874	0.3748	0.0000	1.0000

Table 1: Experiment 2: Effect of β_{loc}.

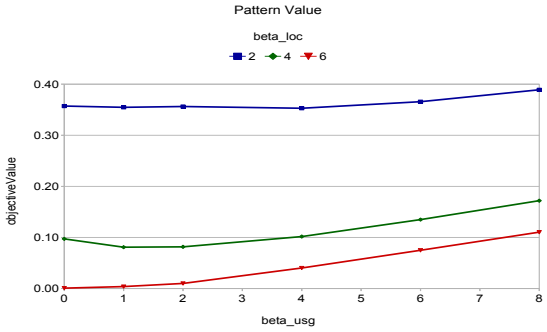
In Figure 5(b) we vary the *location bias factor* β_{loc} in the range $[2, 6]$. The corresponding values are provided in Table 1, where in addition we provide in the last column the probability that all soft constraints in the pattern are satisfied. Further, the last row provides values when the intra-group location constraints are made hard. This corresponds to the best pattern value and is provided for reference. As illustrated in Figure 5(b), the algorithm execution time is fairly short at slightly above 200 msec. As β_{loc} increases, the solution favors achieving a lower patent value and, at the same time, a higher system value. The total value exhibit a convex behavior with a minimum at about $\beta_{loc} = 4$.



(a) Total value.



(b) System value.



(b) Pattern value.

Figure 6: Experiment 2.

Since we have two bias factors in this experiment we illustrate the effect of both on the total objective value, the system value, and the pattern value in Figures 6(a)-(c), respectively, where we vary the *usage bias factor* β_{usg} on the x-axis and plot values for the *location bias factor* $\beta_{loc} = 2, 4,$ and 6. Clearly, the case $\beta_{loc} = 2$ results in fairly large pattern values. The cases for $\beta_{loc} = 4$ and $\beta_{loc} = 6$ result in opposite behavior, as far as the system value and pattern value are concerned. However, the combined total value seems fairly flat, suggesting the insensitivity of β_{usg} in those cases.

6.4 Experiment 3

The objective of this experiment is to consider a special configuration where the optimal placement of a pattern is known a priori as placing the pattern entirely in one PM. We vary the usage bias factor to investigate how close can we get to that solution. The cloud system is made 4 times smaller, i.e. 256 PMs, each with CPU resource capacity of 8 units. The tree topology of height 3 with widths 2, 8, and 16, respectively from the root downward. In other words, the cloud system consists of 2 data centers, where each data center consists of 8 racks, and each rack holds 16 PMs. As the above experiments, the system is simulated, starting from an empty system, and subjected to a load of 0.8, with Poisson arrivals and uniform life times.

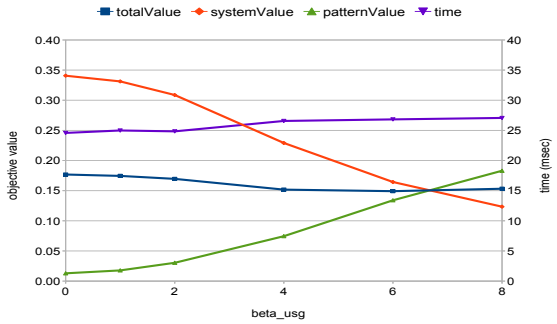


Figure 7: Experiment 3: Effect of β_{usg}. ($\beta_{loc} = 2$)

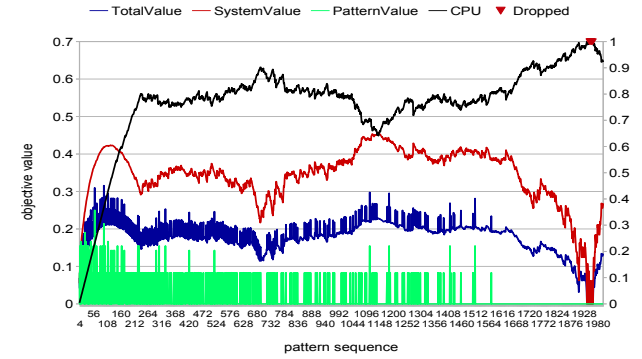
The pattern consists of a single group of 8 homogeneous VMs, each with a CPU demand of 1 unit. All VMs are required to be located in the same PM through an intra-group soft location constraint at the PM level. As in the above experiments, steady state is reached after 234 arrivals, with 1,766 requests in steady state and 7 dropped requests due to lack of resources.

β_{usg}	Total	System	Pattern	Satisfied
0	0.1768	0.3407	0.0129	0.8488
1	0.1745	0.3313	0.0178	0.7965
2	0.1695	0.3087	0.0304	0.6748
4	0.1518	0.2291	0.0745	0.3559
6	0.1492	0.1643	0.1340	0.1285
8	0.1533	0.1235	0.1831	0.0648
hard	0.1874	0.3748	0.0000	1.0000

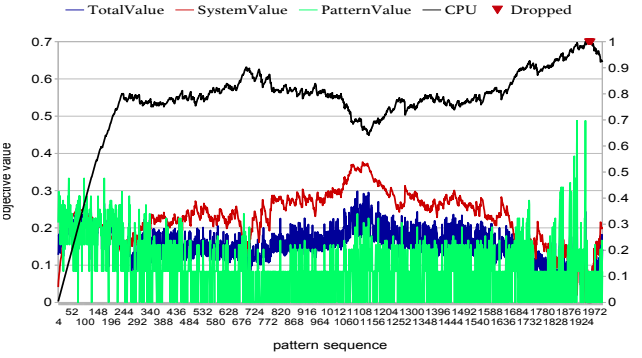
Table 2: Experiment 3: Effect of β_{usb}.

In Figure 7 we vary the *usage bias factor* β_{usg} in the range $[0, 8]$. The corresponding values are provided in Table 2, where in addition we provide in the last column the probability that all soft constraints in the pattern are satisfied. Further, the last row provides values when the intra-group location constraints are made hard. This corresponds to the best pattern value and is provided for reference. As illustrated in the figure, the algorithm execution time for this

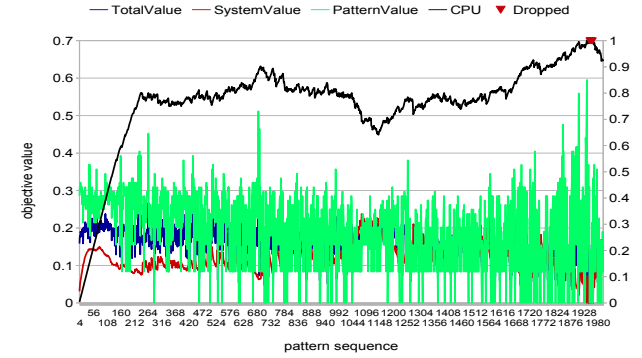
small pattern and 256 PMs is fairly swift at about 25 msec. As β_{usg} increases, the system value decreases and the pattern value increases, resulting in a fairly constant total value around 0.17.



(a) $\beta_{usg} = 0$.



(b) $\beta_{usg} = 4$.



(c) $\beta_{usg} = 8$.

Figure 8: Experiment 3: Time series. ($\beta_{loc} = 2$)

In Figures 8(a)-(c), we fix the *location bias factor* $\beta_{loc} = 2$ and show the time series corresponding to three experiments for the *usage bias factor* $\beta_{usg} = 0, 4,$ and 8, respectively. Note the continual decrease in system values across the three experiments. But, more importantly, note the behavior of the pattern value. For $\beta_{usg} = 0$ in Figure 8(a), the pattern value is fairly small and, surprisingly, most patterns have their location constraints satisfied at high load. A simple calculation suggests that one VM in the pattern being placed in a different PM than the rest, yet in the same rack, result in a pattern value of $7/(24 * 3) = 0.0833$. For $\beta_{usg} = 4$ in Figure 8(b), the pattern value increases resulting in only about a third of the patters having their location constrains satisfied. Finally, for $\beta_{usg} = 8$ in Figure 8(c), the pattern value becomes increasingly high with only 6 percent of the patterns satisfied. However, the system value drops to an average 0.12.

7. CONCLUSION

We presented an approach for solving a cloud placement optimization problem. The approach is based on a general random search technique where we employ importance sampling to construct the marginal distribution of the solution, and a modified Gibbs sampling technique restricted to intra-sample dependency using the marginal distributions as posteriors. This dependency results into biasing the sampling process. We investigated the sensitivity of the solution to the choose of biasing factors. Several remaining work is underway. We mention, for example, convergence properties of the algorithm, criterion for selecting parameters of the algorithm, comparison of the algorithm to other techniques.

8. ACKNOWLEDGMENTS

The author is thankful to Malgorzata Steinder for many interesting and fruitful discussions.

9. REFERENCES

[1] A. Aldhalaan and D. A. Menasce. Autonomic allocation of communicating virtual machines in hierarchical cloud data centers. In *Proceedings of the 2014 IEEE International Conference on Cloud and Autonomic Computing*, CAC '14, London, UK, September 8-12 2014. IEEE, IEEE Computer Society.

[2] S. Andradóttir. Accelerating the convergence of random search methods for discrete stochastic optimization. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 9(4):349–380, 1999.

[3] W. Arnold, D. Arroyo, W. Segmuller, M. Spreitzer, M. Steinder, and A. Tantawi. Workload orchestration and optimization for software defined environments. *IBM Journal of Research and Development*, 58(2):1–12, March 2014.

[4] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera. A stable network-aware vm placement for cloud systems. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 498–506, Washington, DC, USA, 2012. IEEE Computer Society.

[5] M. Cardosa and A. Chandra. Resource bundles: Using aggregation for statistical large-scale resource discovery and management. *Parallel and Distributed Systems, IEEE Transactions on*, 21(8):1089–1102, Aug 2010.

[6] G. Casella and E. I. George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[7] I. Giurgiu, C. Castillo, A. Tantawi, and M. Steinder. Enabling efficient placement of virtual infrastructures in the cloud. In *Middleware 2012*, pages 332–353. Springer, 2012.

[8] B. Jennings and R. Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, pages 1–53, 2014.

[9] T. L. Lai. Sequential analysis: some classical problems and new challenges. *Statistica Sinica*, 11(2):303–350, 2001.

[10] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. Performance model driven qos guarantees and optimization in clouds. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 15–22, Washington, DC, USA, 2009. IEEE Computer Society.

[11] 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.

[12] 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.

[13] J. C. Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 65. John Wiley & Sons, 2005.

[14] A. Tantawi. A scalable algorithm for placement of virtual clusters in large data centers. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 3–10, Aug 2012.

[15] J. Xu and J. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 179–188, Dec 2010.