Solution Biasing for Optimized Cloud Workload Placement

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

***Abstract*—We consider the cloud workload placement problem, which is a mapping of logical to physical entities, that satisfies some constraints and optimizes an objective function. We describe an efficient solution 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. In a case where an optimal solution may be obtained, we show that our method is within 20% of optimality. Since the magnitude of biasing impacts the quality of placement, we investigate the tradeoff between biasing and optimality of placement solutions.**

I. Introduction

We are concerned with the cloud workload placement problem, which is a mapping of logical entities (LE), such as Virtual Machines (VM), data volumes, communication links, and containers, along with topological requirements, such as physical proximity of the logical entities, availability/reliability concerns, and preferred hosting, onto physical entities (PE), such as Physical Machines (PM), storage devices, and communication networks, in such a way as to optimize an objective function, combining user and provider objectives [1].

An original, single VM, placement problem was formulated as a bin packing problem. Then, multiple VMs were considered along with their communication needs, resulting in a quadratic, traffic-aware VM placement problem [2]. Special hierarchical structures of clouds were exploited to devise heuristic placement algorithms [3]. As for the optimization technique itself, we find a variety of techniques, ranging from heuristic-based, simulation, to evolutionary algorithms [4]. The placement problem had been extended to a multiple cloud arrangement [5]. Recently, a technique which uses biased sampling was introduced [6], where the motivation for biasing is explained [7] and applied to hierarchical clouds [8].

In this paper, we provide a probabilistic framework and statistical sampling method for the technique outlined in [6]. The problem is stated as a search problem [9] and a general random search method [10] is sought. An independent Metropolis-Hastings sampling is performed. In particular, the Gibbs sampling [11] technique is modified and restricted to intra-sample dependency. The problem constraints are used to construct the conditional probabilities and the importance sampling marginal distributions are employed as posterior distributions.

II. Problem Statement

The cloud infrastructure is subjected to a stream of requests from cloud users, where each request represents an application/service that the user wants deployed in the cloud. The request specifies the LEs of the application and constraints related to the deployment of the applications. Some constraints relate to individual LE, such as properties and/or resource demands. Other constraints relate to pairs of LEs, such as communication needs. 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.

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. $\mathbf{M}_i$. 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 $i = 1, 2, \cdots, I$, respectively

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 $\mathbf{X}$. 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 $M \geq 2$.

Define a scalar objective function $f(\mathbf{X}; \Theta)$ with range $\mathbb{R}$, the set of real numbers, and a parameter set Θ. 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, $\min_{\mathbf{X}} f(\mathbf{X}; \Theta),\ \mathbf{X} \in \mathcal{S}$.

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), and S the set of possible solutions given the requirement constraints specified in the user request. The parameter set Θ represents the current state of the cloud system. For brevity, and without loss of generality, we will omit Θ and write the objective function as $f(\mathbf{X})$. It 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, Θ. We restate the optimization problem as,

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

Define $\mathcal{A}$ as an arbitrary, nonempty subset of $\mathcal{S}$, i.e. $\emptyset \neq \mathcal{A} \subseteq \mathcal{S}$, with size $T \geq 1$, i.e. $\mathcal{A} = \{\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_T\}$, where $\mathbf{A}_t = [a_{t1}\ a_{t2}\ \cdots\ a_{tM}]$, $t \in \mathcal{T}$, is an assignment vector. 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 $n \in \mathcal{N}$. In the special case where the elements are such that, $p_{m,n} \in \{0, 1\}$, we refer to the stochastic matrix $\mathbf{P}$ as *deterministic*. Given a set of assignments $\mathcal{A}$, define the *generator* $\mathbf{G}(\mathcal{A})$, as a probabilistic generator of assignments using stochastic matrix $\mathbf{P}$, such that $p_{m,n}$ is the probability that $a_{tm} = n$ over $t \in \mathcal{T}$. Hence, $\mathbf{P}_m$ represents the marginal probability distribution of the m^{th} element of the assignments in $\mathcal{A}$.

III. 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 [9] for an optimal $\mathbf{X}^*$ with minimum $f(\mathbf{X}^*)$ in the solution space S. A generalized random search method [10] consists of the following steps.

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.

Obviously, one needs a stopping criterion for the above algorithm. Examples are limiting the number of iterations and reaching diminishing return. 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 random search 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. The main idea of importance sampling is that in order to find an optimal solution to a combinatorial problem, one generates many samples of solutions using a parametrized probability distribution. The samples are ordered in their attained values of the objective function. Then, a small fraction, ρ, best or *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 better samples. After a few iterations, a good solution is obtained.

A straightforward implementation of the random search algorithm to the cloud placement problem, without any biasing, has been shown to be impractical [6]. 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 below).
   b) $\iota = \iota + 1$. Create set $\mathcal{A}$ including the $T \ll K$ 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. Time complexity is $O(NM^2)$.

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, and cost constraints. Define a weight vector $\mathbf{W}$ of length R, where element $w_r \geq 0$ is a weight associated with $\mathbf{B}(r)$, $r \in \mathcal{R}$.

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.

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

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

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

IV. Functional definitions

In this section we describe the biasing $\mathbb{B}$ that we use in our implementation of the BSA algorithm. Biasing helps in generating "good" solutions through applying functions that "push" the solution towards one that satisfies the constraints and optimizes the objective function. Biasing functions are designed to have parameters, also known as bias factors. Setting the values for such parameters is critical to the quality of the placement results as demonstrated in Section V.

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 $r_e, e = 1, 2, \cdots, E$, e.g. 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 $[l+1, L]$. Location constraints may be 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.

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. 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

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

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, and the higher the value of β_{usg} the more biasing is applied.

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 $dev(l, l') = |l - l'|/L$, 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

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

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 $\tau = 3$. We refer to β_{loc} as the *location bias factor*. A value of $\beta_{loc} = 0$ corresponds to no biasing, 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 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

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

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

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

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

V. 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 functions 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 $\mathbf{1}$.

We consider a cloud system which consists of 1024 PMs, each with CPU resource capacity of 8 (core) 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 an offered 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 ramp-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.

1) Experiment 1: The pattern in this experiment consists of 32 independent homogeneous VMs, each with demand 1 CPU unit. 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.

We use $\beta_{usg} = 2$. In Figure 1, 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 Table I, we vary β_{usg} in the range $[0, 8]$. The values for the average objective function and algorithm execution time are averaged over the 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,

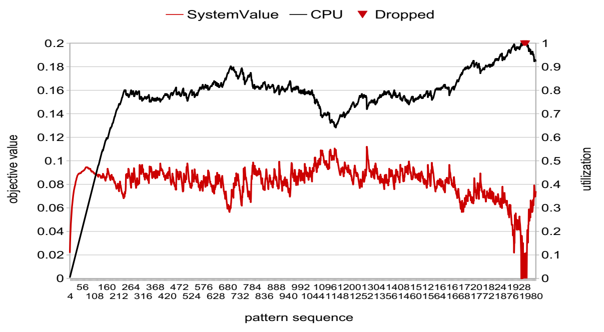

Fig. 1. Experiment 1: Time series ($\beta_{usg} = 2$).

i.e. decreases, as β_{usg} increases. The rate of improvement decreases from an initial 0.0192, going from 0 to 0.5, to an insignificant 0.0017, going from 6 to 8. We use a default value of $\beta_{usg} = 2$.

β_{usg}	0	1	2	4	6	8
System	0.1124	0.0932	0.0813	0.0689	0.0627	0.0593
Time	67.4	76.2	77.3	99.7	97.4	96.9

TABLE I
EXPERIMENT 1: EFFECT OF β_{usg}.

2) Optimality Test: In Experiment 1, the problem is unconstrained and all PMs and VMs are homogeneous. Hence, balancing the load on the single resource, the CPU, i.e. minimizing the standard deviation of the CPU utilization among all PMs, may be achieved through implementing a very simple best-fit policy. In such a policy, a VM is placed in the PM with the smallest CPU utilization, with ties broken by random selection. The result of this policy for Experiment 1 is depicted in Figure 2. The system value is the value of the standard deviation of CPU utilization among all PMs. A value of zero is achieved only if all PMs have the same CPU utilization. This event occurs when the 32 VM patterns evenly divide the 1,024 PMs. From the figure we note that such an event happened during the ramp-up phase quite regularly as no pattern departed to leave some fragmentation. Later on, this event happened only a few times due to the random nature of arrival and departure processes.

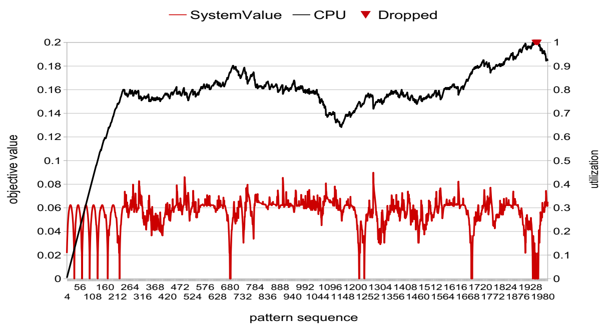

Fig. 2. Experiment 1: Optimal placement.

In any case, the average standard deviation value during steady state was 0.0569 with stdev of 0.0118. The average pattern placement time was 8 msec. Comparing these values to running our algorithm with $\beta_{usg} = 2$ and same code base, which yielded an average of 0.0681 with stdev of 0.0119, we find that our algorithm had a relative difference of 0.1968, i.e. within 20% of optimum. The similarity in the stdev values suggests that our algorithm is as tight as the optimal one.

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.

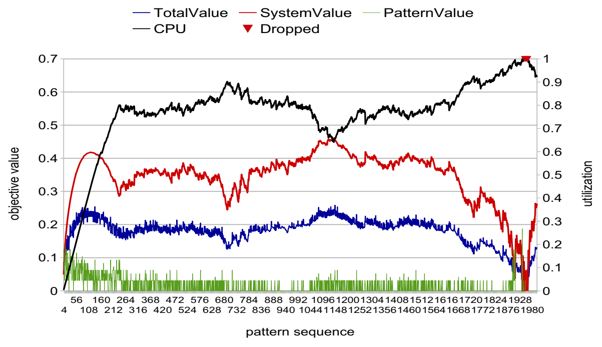

Fig. 3. Experiment 2: Time series (β_{loc} = 6; β_{usg} = 2).

We keep $\beta_{usg} = 2$. In Figure 3 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 3. 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.

We vary β_{loc} in the range $[2, 6]$. The corresponding values are provided in Table II, 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, the algorithm execution time is fairly short at slightly above 200 msec. As β_{loc} increases, the solution favors achieving a lower pattern value and, at the same time, a higher system value. The total value exhibit a convex behavior with a minimum at about $\beta_{loc} = 4$.

β_{loc}	Time	Total	System	Pattern	Satisfied
2	222.8	0.2578	0.1596	0.3563	0.0000
3	213.3	0.1858	0.1950	0.1767	0.0006
4	203.8	0.1634	0.2451	0.0816	0.0404
5	208.4	0.1657	0.2940	0.0374	0.2297
6	201.1	0.1765	0.3430	0.0100	0.7089
hard	188.0	0.1874	0.3748	0.0000	1.0000

TABLE II
EXPERIMENT 2: EFFECT OF β_{loc} (β_{usg} = 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 Table III, where we vary β_{usg} in columns and β_{loc} in rows. 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.

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.

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.

We fix $\beta_{loc} = 2$ and vary β_{usg} in the range $[0, 8]$. The corresponding values are provided in Table IV, 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, 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.

	β_{loc}/β_{usg}	0	2	4	6	8
System	2	0.2028	0.1596	0.1334	0.1187	0.1115
	4	0.2739	0.2451	0.2059	0.1727	0.1470
	6	0.3703	0.3430	0.2756	0.2202	0.1820
Pattern	2	0.3571	0.3563	0.3529	0.3657	0.3890
	4	0.0973	0.0816	0.1018	0.1349	0.1720
	6	0.0008	0.0100	0.0403	0.0749	0.1103
Total	2	0.2800	0.2578	0.2431	0.2422	0.2502
	4	0.1856	0.1634	0.1539	0.1538	0.1595
	6	0.1856	0.1765	0.1579	0.1476	0.1462

TABLE III
EXPERIMENT 2.

β_{usg}	Time	Total	System	Pattern	Satisfied
0	24.6	0.1768	0.3407	0.0129	0.8488
2	24.8	0.1695	0.3087	0.0304	0.6748
4	26.6	0.1518	0.2291	0.0745	0.3559
6	26.8	0.1492	0.1643	0.1340	0.1285
8	27.1	0.1533	0.1235	0.1831	0.0648
hard	21.8	0.1874	0.3748	0.0000	1.0000

TABLE IV
EXPERIMENT 3: EFFECT OF β_{usb} (β_{loc} = 2).

REFERENCES

[1] 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*, vol. 58, no. 2, pp. 1–12, March 2014.

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

[3] 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*, ser. CAC '14, IEEE. London, UK: IEEE Computer Society, September 8-12 2014.

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

[5] M. Unuvar, M. Steinder, and A. N. Tantawi, "Hybrid cloud placement algorithm," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*, Sept 2014, pp. 197–206.

[6] A. N. 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*, Aug 2012, pp. 3–10.

[7] ——, "On biasing towards optimized application placement in the cloud," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, October 2015.

[8] ——, "Quantitative placement of services in hierarchical clouds," in *Quantitative Evaluation of Systems*. Springer International Publishing, 2015, pp. 195–210.

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

[10] S. Andradóttir, "Accelerating the convergence of random search methods for discrete stochastic optimization," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 9, no. 4, pp. 349–380, 1999.

[11] G. Casella and E. I. George, "Explaining the gibbs sampler," *The American Statistician*, vol. 46, no. 3, pp. 167–174, 1992.