Shadow-Routing Based Dynamic Algorithms for Virtual Machine Placement in a Network Cloud

Yang Guo
Bell Labs, Alcatel-Lucent
Email: yang.guo@alcatel-lucent.com

Alexander L. Stolyar
Bell Labs, Alcatel-Lucent
Email: stolyar@research.bell-labs.com

Anwar Walid
Bell Labs, Alcatel-Lucent
Email: anwar@research.bell-labs.com

***Abstract*—We consider a *shadow routing* based approach to the problem of real-time adaptive placement of virtual machines (VM) in large data centers (DC) within a network cloud. Such placement in particular has to respect *vector packing* constraints on the allocation of VMs to host physical machines (PM) within a DC, because each PM can potentially serve multiple VMs simultaneously. Shadow routing is attractive in that it allows a large variety of system objectives and/or constraints to be treated within a common framework (as long as the underlying optimization problem is convex). Perhaps even more attractive feature is that the corresponding algorithm is very simple to implement, it runs continuously, and adapts automatically to changes in the VM demand rates, changes in system parameters, etc., without the need to re-solve the underlying optimization problem "from scratch". In this paper we focus on the *min-max-DC-load* problem. Namely, we propose a combined VM-to-DC routing and VM-to-PM assignment algorithm, referred to as *Shadow scheme*, which minimizes the maximum of appropriately defined DC utilizations. We prove that the Shadow scheme is asymptotically optimal (as one of its parameters goes to 0). Simulation confirms good performance and high adaptivity of the algorithm. Favorable performance is also demonstrated in comparison with a baseline algorithm based on VMware implementation [7], [8]. We also propose a simplified – "more distributed" – version of the Shadow scheme, which performs almost as well in simulations.**

I. Introduction

Effective resource management is one of the main challenges for cloud service providers [7], [8]. In cloud computing, the virtualization technology is employed to enable the resource sharing and dynamic system/network reconfigurations. Virtual machines (VMs) share the CPU/memory resources by residing on the same physical machine; and can be dynamically resized and migrated based on the application needs [4], [16]. The great flexibility allows the cloud service providers to offer users computing and storage services in a pay-as-you-go manner while achieving the economy of scale. The various types of resources contained in the cloud, however, come with different constraints. For instance, CPU and memory are typically confined to a single physical machine (PM) and can only be shared locally; while disk storage space is often offered as a utility service where all machines attached to the storage system can use the service. How to efficiently utilize different resources in the inherently dynamic, complex, and heterogeneous cloud environment is challenging [8].

In this paper we consider the VM placement problem to multiple data centers (DCs) or server clusters. We consider multiple types of Virtual machines (VMs) and physical machines (PMs), where a VM type is characterized by the vector of the required resource amounts; and a PM type is characterized by the vector of the resource amounts it possesses. Different DCs may consist of different PM types. Not all resources are associated with individual PMs – some of them (e.g. disk storage) exist as one resource pool associated with a DC. The placement problem consists of two parts (layers): (i) *routing layer* decides which DC a given VM should be routed to; and (ii) *DC layer* assigns an "arriving" VM to a specific PM within the DC – this assignment, of course, has to respect VM-to-PM "packing constraints", i.e. the total resource requirements of all VMs assigned to a PM cannot exceed the resource amounts at the PM (see (1) below). The latter constraints are sometimes called *vector packing* (see e.g. [5]). Packing constraints substantially complicate the VM placement problem, and require a sufficiently intelligent strategy to efficiently utilize the physical resources.

We propose a *shadow routing* based placement algorithm in this paper. This means that, roughly speaking, each arriving VM first "arrives" into a specially designed virtual queueing system, where it is routed to one of the (virtual) queues. The "service" in the virtual system is performed by a "superserver", whose "service rate vectors" are feasible "packing configurations" of VMs into PMs. The advantage of shadow routing approach is that it is simple and adaptive: no need to know a priori, or explicitly measure, the VM arrival rates; if the arrival rates change, the algorithm adapt automatically. Yet, as we show, the algorithm is asymptotically optimal. (The virtual system in our case is an instance of a general model of [13], and therefore our algorithm definition and its optimality are derived from the results of [13].) All these features are confirmed by our simulations, where we compare our algorithm, referred to as *Shadow scheme*, to a baseline scheme, which is along the lines of some of the currently used algorithms [8].

While our model is related to classical *stochastic bin packing* problems (see e.g. [5], [6] for good reviews), it is different in two important respects: first, the model is substantially more general in that we have multiple pools of different "bins" (PMs) and, second, the "items" (VMs) leave the system after a random service time. The work on models that are close to ours is very recent (see e.g. [9], [10]). Paper [9] addresses a real-time VM allocation problem, which in particular includes

packing constraints; the approach of [9] is close in spirit to Markov Chain algorithms used in combinatorial optimization. Paper [10] is concerned mostly with maximizing throughput of a system, where VM queueing delays are allowed and in fact are typical. (We are mostly interested in large-scale systems, where VM queueing delays are negligible.) Other VM placement related works, e.g. [1], [11], [12], focus on different aspects of VM placement, ranging from minimizing the network traffics, to shortening the inter-VM distance/latency and statistically sharing resources. Our study addresses the load balancing and VM packing problem.

Shadow routing approach has been applied previously to large-scale service systems, but without packing constraints [14]. A distinct feature of our work, and one of its main contributions, is that we demonstrate that packing constraints can be incorporated into the shadow routing framework, and moreover, it can be done in a computationally efficient way, amenable to practical implementations.

Layout of the rest of the paper. The formal model and the specific problem (min-max-DC-utilization) are given in Section II. Section III defines the Shadow scheme (including the construction of the virtual queueing system in Section III-A) and proves its asymptotic optimality. In Section IV we present simulation results for the Shadow scheme, including its comparison to a reasonable baseline algorithm and tests of its adaptability and robustness in different scenarios. Then, in Section V, we define a simplified ("more distributed") version of Shadow algorithm, which according to simulation results, still has superior performance compared to the baseline. In Section VI we show how Shadow scheme can be made robust not only to the changes in VM input rates, but also to the changes in average VM service times.

II. Model and problem statement

A. Model

There are several classes of VMs (jobs), indexed by $i \in \mathcal{I} = \{1, \dots, I\}$. Class i jobs arrive at the rate λ_i. Each class i job needs several computing resources of different types when it is served; namely, the amount $a_{ik} > 0$ of resources $k = 1, \dots, K$. When job i is placed for service (and is allocated the required amounts of resources), its average service time is $1/\mu_i$. After the service is complete, a job releases all resources allocated to it and leaves the system.

There are several Data Centers (DC), or server clusters, indexed by j. Computing resources $k = 1, \dots, K$, defined earlier, are of two different *kinds*: "pooled" resources $k \in \mathcal{K}_p = \{1, \dots, K'\}$ and "localized" ones $k \in \mathcal{K}_\ell = \{K'+1, \dots, K\}$. A DC j contains the total amount $\beta_{jk} > 0$ of the pooled resource $k \in \mathcal{K}_p$. A DC j also contains β_j^* physical machines (PM), each of which has the amount $A_{jk} > 0$ of localized resource $k \in \mathcal{K}_\ell$. If a class i VM is placed for service at (routed to) DC j, it is placed into one of the PMs, where the amounts a_{ik} of localized resources are allocated (if they are still available at that specific PM), and the amounts a_{ik} of pooled resource is allocated (if they are still available at the DC). This means, in particular, that a PM in DC j can simultaneously serve the numbers of different job types given by a vector $s = (s_i,\ i = 1, \dots, I)$ if

$$\sum_i s_i a_{ik} \le A_{jk}, \quad \forall k \in \mathcal{K}_\ell. \tag{1}$$

Vectors s (with non-negative integer components) satisfying this condition we call *feasible configurations* of a PM in DC j. The set of such vectors that are maximal (not dominated by others) is denoted by S_j.

For example, suppose (as in our simulation model, described later) that there are three computing resource types that each VM needs: Disk Storage, CPU and Memory, indexed by $k = 1, 2, 3$ respectively; suppose Disk Storage is a pooled resource, while CPU and Memory are localized ones. Then $K = 3$, $K' = 1$, $\mathcal{K}_p = \{1\}$, $\mathcal{K}_\ell = \{2, 3\}$.

B. Problem statement

For each DC j, we define its *PM-utilization* as the fraction of PMs in it that are non-idle, and define its *k-utilization* for each pooled resource k as the fraction of the resource that is in use. The problem is to route new VM requests to DCs, and assign VMs to PMs within each DC, in a way such that the maximum of all average PM-utilizations and all average k-utilizations, across all DCs, is minimized. This objective can be naturally thought of as load balancing (across resources of all kinds at all DCs); however, we emphasize that it is also the system capacity maximization objective, because – if achieved – it makes sure that the system can process the offered load if this is feasible at all. A very desirable feature of the routing/assignment scheme is that it is simple, parsimonious (not requiring a priori knowledge of input rates) and adaptive.

More precisely, denote by λ_{ij} the average rate at which an algorithm routes class i jobs to server j; by $\phi_{sj} \ge 0$ the average fraction of PMs in DC j that are used in the configuration $s \in S_j$. Then, we want an algorithm that "produces" rates λ_{ij} and fractions ϕ_{sj} such that ideally they solve the following linear program (with ρ being an additional variable, having the meaning of the maximum average utilization, and therefore being the objective function as well):

$$\min_{\{\lambda_{ij}\},\{\phi_{sj}\},\rho} \rho, \tag{2}$$

subject to

$$\lambda_{ij} \ge 0,\ \forall (i,j), \quad \phi_{sj} \ge 0,\ \forall (s,j), \tag{3}$$

$$\sum_i \lambda_{ij} a_{ik}/(\beta_{jk}\mu_i) \le \rho, \quad \forall j,\ \forall k \in \mathcal{K}_p, \tag{4}$$

$$\sum_j \lambda_{ij} = \lambda_i, \quad \forall i, \tag{5}$$

$$\lambda_{ij}/(\beta_j^*\mu_i) \le \sum_{s \in S_j} s_i \phi_{sj}, \quad \forall (j,i), \tag{6}$$

$$\sum_{s \in S_j} \phi_{sj} = \rho, \quad \forall j. \tag{7}$$

The LHS of (4) is the k-utilization of DC j, with λ_{ij}/μ_i being the average number of class i VMs in service at DC j. The constraint (6) might be easier to understand in the form $\lambda_{ij}/\mu_i \leq \sum_{s\in S_j} s_i \phi_{sj} \beta_j^*$, with $\phi_{sj}\beta_j^*$ being the average number of PMs in DC j used in configuration s and, therefore, $s_i\phi_{sj}\beta_j^*$ being the average number of i-VMs served by PMs in configuration s.

Clearly, unless the optimal value ρ of the LP (2)-(7) does not exceed 1, the system cannot possibly process the entire offered load. If optimal $\rho < 1$, then all offered load can be handled (i.e., the underlying stochastic process can be made stable), as long as queueing of the VMs waiting for service is allowed.

An important case is when the system is large-scale, in the sense of all λ_i, β_{jk} and β_j^* being large simultaneously. In this case, when optimal $\rho < 1$, not only all offered load can be handled, but the system can be controlled in a way such that there is essentially no queues (almost all VMs will be assigned for service immediately) and the fractions of used resources become (almost) non-random; namely, each fraction of PMs in configuration s in DC j stays close to an optimal ϕ_{sj}, and all PM- and k-utilizations are (almost) non-random, not exceeding ρ. Our simulation model in Section IV is not "extremely" large-scale, with the number of PM in one DC ranging from 100 to 350; however, as we will see, the system behavior under our proposed scheme does exhibit these desirable properties of large-scale systems.

Given the structure of the LP (2)-(7), the following fact is obvious. Let us reduce each set S_j of feasible configurations by removing configurations s that can be dominated (in the sense of vector inequality $\leq$) by a convex combination of other vectors in S_j; denote the reduced set by $\hat{S}_j$. Then, if in (2)-(7) we replace S_j with $\hat{S}_j$, this will not change the optimal value of the problem. Set $\hat{S}_j$ can be much smaller than S_j. (In our simulation model it is about three times smaller.) This substantially improves computational complexity of the scheme that we propose below.

III. Shadow routing based scheme

A. Construction of the virtual queueing system

We will now construct an optimal routing algorithm, which makes decisions upon each VM arrival. The algorithm "maintains" and updates a virtual ("shadow") queueing system, and the routing decisions are based on the state of that system. Assume for simplicity that the arrival flows of VMs of different classes are Poisson. (This is not crucial.) Then, the sequence of VM arrivals is such that the class of each arriving VM is i with probabilities λ_i/λ, where $\lambda = \sum_{i'} \lambda_{i'}$, independently of other arrivals. For each DC j there are associated virtual queues, labeled by (j,k), $k \in \mathcal{K}_p$, and (j,i), $i \in \mathcal{I}$; the lengths of those queues are denoted by Q_{jk} and Q_{ji}. (The virtual queues are just variables maintained by the algorithm – they are not physical queues where VMs, or anything else, wait for service.) When a VM arrives, and its class is i, the algorithm must immediately route it to one of the DCs; if the chosen DC is m, then the algorithm places the amounts of "work" $a_{ik}/(\beta_{mk}\mu_i)$ into queues (m,k), namely $Q_{mk} := Q_{mk} + a_{ik}/(\beta_{mk}\mu_i)$, and amount $1/(\beta_m^*\mu_i)$ of "work" into (only one) queue (m,i). After the routing decision for this VM is chosen, and corresponding updates are done, the algorithm must decide whether or not to activate a "superserver". If superserver is activated, then for each DC j a "service mode" $\sigma^j \in \hat{S}_j$ is chosen; then, the amount of "work" c is removed from each virtual queue (j,k), namely $Q_{jk} := \max\{Q_{jk} - c, 0\}$, and the amount of "work" $c\sigma_i^j$ is removed from each virtual queue (j,k), namely $Q_{ji} := \max\{Q_{jk} - c\sigma_i^j, 0\}$. Here $c > 0$ is a fixed parameter chosen to be large enough, so that if activated every time the superserver can definitely keep serving work from *all* virtual queues at the rates greater than the work arrives.

Now, suppose we want a joint routing and superserver activation algorithm, such that the average frequency of superserver activations is minimized, subject to the constraint that all virtual queues remain stable (do not run away to infinity). The virtual queueing system and the problem we just described are within the framework of general model in [13], which gives a general *asymptotically* optimal (in the sense specified below) algorithm, called *Greedy Primal-Dual* (GPD). When we apply GPD to our virtual system we obtain the algorithm given in the next subsection. We then show in Proposition 1 that the average rates at which this algorithm routes different type customers to the DCs are close to those given by an optimal solution to LP (2)-(7).

Remark. We want to emphasize that the virtual queues are *not* buffers where actual VM requests are placed for waiting; instead, they are no more than variables maintained by the routing algorithm. Therefore the length of virtual queues has no connection to the waiting times of actual VM requests. Moreover, if the routing algorithm keep the maximum average utilization close to the optimal value $\rho < 1$, then, as explained earlier, there will be essentially not waiting before VM placements. This will also be confirmed by our simulations. This remark applies to all virtual queues used by the algorithms in this paper.

B. Routing layer algorithm

The routing of VMs to Dcs is done by the following algorithm. Denote by R_i the subset of DCs j, such that at least one class i VM can fit into a PM, i.e. $a_{ik} \leq A_{jk}$ for all $k \in \mathcal{K}_l$.

Algorithm-A

There is a (small) parameter $\eta > 0$, parameter $c > 0$, and (small) parameter $\theta > 0$.

Upon each new actual VM arrival, say of class i to be specific, the algorithm does the following (in sequence):

1. Compute DC index

$$m \in \arg\min_{j\in R_i} [Q_{ji}/(\beta_j^*\mu_i) + \sum_{k\in\mathcal{K}_p} Q_{jk}a_{ik}/(\beta_{jk}\mu_i)],$$

route this VM to DC m and for this m do the following updates:

$$Q_{mk} := Q_{mk} + a_{ik}/(\beta_{mk}\mu_i), \quad \forall k \in \mathcal{K}_p,$$

$$Q_{mi} := Q_{mi} + 1/(\beta^*_m \mu_i).$$

2. For each DC j compute

$$\sigma^j \in \arg\max_{s \in \hat{S}_j} \sum_{i' \in \mathcal{I}} s_{i'} Q_{ji'}.$$

If condition

$$\eta \sum_j [\sum_{k \in \mathcal{K}_p} Q_{jk} + \sum_{i' \in \mathcal{I}} \sigma^j_{i'} Q_{ji'}] \geq 1 \quad (8)$$

holds, do the following updates:

$$Q_{jk} := \max\{Q_{jk} - c, 0\}, \quad \text{for all } j \text{ and } k \in \mathcal{K}_p,$$

$$Q_{ji'} := \max\{Q_{ji'} - c\sigma^j_{i'}, 0\}, \quad \text{for all } j \text{ and } i \in \mathcal{I}.$$

3. For each DC j update the following "configuration usage fractions":

$$\hat{\phi}_{sj} := \theta I(s, \sigma^j) + (1-\theta)\hat{\phi}_{sj}, \quad \text{for all } j \text{ and } s \in \hat{S}_j, \quad (9)$$

where $I(s, \sigma^j) = 1$ if s was the configuration σ^j computed in step 2 and condition (8) in step 2 held, and $I(s, \sigma^j) = 0$ otherwise.

End Algorithm-A

Parameter c is such that

$$c > \max_{i,j} \max_{k \in \mathcal{K}_p} a_{ik}/(\beta_{jk}\mu_i) \quad \text{and} \quad c > \max_{i,j} 1/(\beta^*_j \mu_i). \quad (10)$$

This is sufficient for the superserver to be able to "keep up" with the incoming load, which in turn guarantees that the virtual queues will be stable under Algorithm-A.

Step 3, which computes configuration usage fractions, is *not* needed for the routing itself, but the usage fractions $\hat{\phi}_{ij}$ are "fed into" the DC layer algorithm (Algorithm-B below) for the assignment of VM to PMs within each DC.

Proposition 1. *Suppose system parameters a_{ik}, β_{jk}, β^*_j, μ_i, c, are fixed rational numbers, such that condition (10) holds. Suppose the input flows are Poisson, with fixed rates λ_i. Consider a sequence of systems with parameter $\eta \to 0$. Then, for any η, the virtual queueing process is a positive recurrent countable discrete-time Markov chain. Moreover, stationary distributions of the processes are such that the following holds. Denote by $\bar{\phi}^{(\eta)}_{sj} = \mathbb{E}\hat{\phi}_{sj}$ the steady-state probability that configuration s is chosen as σ_j* and *the condition (8) holds (and therefore the superserver is activated), for a fixed parameter η; similarly, let $\bar{p}^{(\eta)}_{ij}$ be the steady-state probability that an arriving type i VM is routed to DC j. Then, as $\eta \to 0$, the sequence of vectors $(\{\bar{\phi}^{(\eta)}_{sj}\}, \{\bar{p}^{(\eta)}_{ij}\})$ is such that its any limiting point $(\{\bar{\phi}_{sj}\}, \{\bar{p}_{ij}\})$ satisfies (using notation $\lambda = \sum_i \lambda_i$)*

$$\lambda_i \bar{p}_{ij} = \lambda_{ij}, \quad \forall (i,j),$$

$$\lambda c \bar{\phi}_{sj} = \phi_{sj}, \quad \forall (s,j), \ s \in \hat{S}_j,$$

where vector $(\{\phi_{sj}\}, \{\lambda_{ij}\})$ is an optimal solution of LP (2)-(7).

Proposition 1 says that, when parameter η is small, Algorithm-A produces VM-to-DC routing rates, as well as configuration usage fractions, that are close to optimal in the sense of LP (2)-(7).

Proof of Proposition 1: The virtual queueing process, viewed as a discrete time process at the times just after VM arrivals, is obviously a discrete time countable Markov chain. (Rationality of parameters implies that there is only a countable number of states.) This Markov chain is stochastically stable (see section 4.9 of [13]), which in our case means that there is a finite number of positive recurrent classes of communicating states, reachable with probability 1 from any state, and therefore a stationary distribution exists for any η. (Here we use condition (10), which guarantees that the superserver has sufficient capacity to "keep up" with the amount of "work" arriving into virtual queues, and therefore implies condition (55) in [13].) Then, the property (AO-2) in section 4.9 of [13] can be established. In our case, it means that, as $\eta \to 0$, Algorithm-A solves the problem of minimizing the frequency of superserver activations, subject to stability of virtual queues. Formally, if for each η we pick a stationary distribution, then, as $\eta \to 0$, $(\{\bar{\phi}^{(\eta)}_{sj}\}, \{\bar{p}^{(\eta)}_{ij}\})$ converges to the set of optimal solutions $(\{\bar{\phi}_{sj}\}, \{\bar{p}_{ij}\})$ of the following LP:

$$\min_{\{\bar{p}_{ij}\}, \{\bar{\phi}_{sj}\}, \bar{\rho}} \bar{\rho}, \quad (11)$$

subject to

$$\bar{p}_{ij} \geq 0, \ \forall (i,j), \quad \bar{\phi}_{sj} \geq 0, \ \forall (s,j), \quad (12)$$

$$\sum_i (\lambda_i/\lambda) \bar{p}_{ij} a_{ik}/(\beta_{jk}\mu_i) \leq c\bar{\rho}, \quad \forall j, \ \forall k \in \mathcal{K}_p, \quad (13)$$

$$\sum_j \bar{p}_{ij} = 1, \quad \forall i, \quad (14)$$

$$(\lambda_i/\lambda)\bar{p}_{ij}/(\beta^*_j \mu_i) \leq \sum_{s \in S_j} s_i c \bar{\phi}_{sj}, \quad \forall (j,i), \quad (15)$$

$$\sum_{s \in S_j} \bar{\phi}_{sj} = \bar{\rho}, \quad \forall j. \quad (16)$$

If we rewrite this LP in terms of variables $\lambda_{ij} = \lambda_i \bar{p}_{ij}$, $\phi_{sj} = \lambda c \bar{\phi}_{sj}$ and $\rho = \lambda c \bar{\rho}$, we obtain problem (2)-(7). The result follows. □

C. Parameter setting for Algorithm-A in implementations

To explain a reasonable choice of parameter setting, we appeal to the general results of [13], according to which, as $\eta \to 0$, the scaled virtual queue lengths ηQ_{jk} and ηQ_{ji} in steady-state converge to an optimal set of non-negative finite dual variables, q^*_{jk} and q^*_{ji}, corresponding to constraints (13) and (15), respectively. Therefore, for the system with finite η to behave near optimally, it is necessary that the variables ηQ_{jk} and ηQ_{ji} are "almost constant", i.e. do not fluctuate much about corresponding levels q^*_{jk} and q^*_{ji}. Since any set of optimal duals satisfies the condition $\sum_j [\sum_{k \in \mathcal{K}_p} q^*_{jk} + \sum_{i' \in \mathcal{I}} \sigma^j_{i'} q^*_{ji'}] = 1$ (compare this to (8)), we also conclude that for near-optimality, the condition (8) needs to hold approximately as equality at all times in steady-state.

These observations motivate the parameter choices suggested next, and our simulations confirm that they work as desired.

It is sufficient that parameter c satisfies (10). But it should not be much larger, so as not to cause very large increments of virtual queues' values in one step of the algorithm. In our simulations we use $c = 1.01 \max\{H_1, H_2\}$, where H_1 and H_2 are the right-hand sides of the inequalities in (10).

The key parameter of the algorithm is η – as Proposition 1 shows, the smaller the η the more precise the algorithm is when the system is in steady-state. However, the time for the virtual system to make transition to a "new" steady-state when the input rates change, is of the order $O(1/\eta)$ (see general results in [13]) – for this reason it is undesirable to make η too small. As discussed above, parameter η should be small enough so that the increments of Q_{jk} in one step of the algorithm (whose absolute values are upper bounded by c) are sufficiently small w.r.t. "typical values" of Q_{ji} and Q_{jk}; such "typical value" we assume to be $1/(J(K'+I)\eta)$, obtained from the condition $\eta[\sum_{jk} Q_{jk} + \sum_{ji} Q_{ji}] \approx 1$, being in turn a "crude version" of (8) holding as equality. (Here $J(K'+I)$ is the total number of virtual queues.) Then, in our simulations, we use three different values of η satisfying

$$\frac{c}{1/(J(K'+I)\eta)} = \frac{1}{\gamma}, \quad \text{with } \gamma = 10, 5, 2.$$

The initial state of the virtual queues is irrelevant, as far as actual operation of the algorithm is concerned, because the algorithm runs continuously.

The averaging parameter θ, used in Step 3, is not crucial. We use $\theta = 0.01$ in our simulations.

D. Data Center layer algorithm – Actual assignment of VMs to PMs within DC

We now give the algorithm for assigning VMs routed to a DC j, to PMs in that DC.

Algorithm-B

Each non-empty PM within DC j at any given time has a designated configuration $s \in \hat{S}_j$; the designation $s = (s_1, \ldots, s_I)$ means that we will never place more than s_i class i VMs into this PM. A PM with designation s is referred to as an s-PM. Empty PMs do not have any designation; a PM designation, once assigned, never changes until the PM gets empty. The following quantity is maintained for each $s \in \hat{S}_j$: $z_i^j(s)$ – the total number of class i VMs in s-PMs (in DC j). In addition, the algorithm at any time has access to the quantities $\hat{\phi}_{sj}$ (only for the DC j), maintained by the Algorithm-A.

When a new class i VM "arrives" in DC j, the algorithm does the following (in sequence):

1. Compute configuration index

$$s' \in \underset{s \in \hat{S}_j \,:\, s_i > 0}{\arg\min} \; z_i^j(s)/[s_i \hat{\phi}_{sj}].$$

2. Among s'-PMs choose a PM with the maximal number of existing VM, but such that the existing number of class i VMs is less than s_i (so that the new class i VM can still fit), and assign the VM to this PM. If no such s'-PM is available, we place the VM into an empty PM and designate the PM as s'-PM.

End Algorithm-B

Remark 1. In the model and algorithm, we assume there is one PM type in each DC. The model and algorithm can easily be generalized for the case when a DC contains several large blocks of PMs, with the same PM type in each block.

Remark 2. In our simulations of the Shadow algorithm, we use a slightly improved version of Step 2 of Algorithm-B. Namely, we assume that a VM migration between PMs is allowed (as it is in real systems [7]), and therefore for each configuration s' it is easy to migrate, if necessary, some VMs between s'-PMs so that the number of occupied s'-PMs is exactly $\lceil \max_i z_i^j(s')/s'_i \rceil$. ($\lceil \xi \rceil$ is the smallest integer not exceeding ξ.) If migration is not employed, this number is obviously the lower bound, and our results are somewhat optimistic.

From now on, the entire shadow routing based scheme, consisting of Algorithm-A (running at routing layer) and Algorithm-B (running at each DC), will be referred to as *Shadow scheme* or *Shadow algorithm*.

IV. Simulation results

In this section we evaluate the Shadow scheme using simulations. The simulation model is such that there is one pooled resource, disk storage, and two localized resources, CPU and Memory. A set of six datacenters is considered. The disk storage space amounts for six datacenters are the same and set to be 360TB. The localized resource amounts (at a physical machine) are defined by the *PM-size* touple $(\#\mathrm{ECU} : \mathrm{MemSize})$, where one ECU (*Elastic Compute Unit*) roughly equals to one 1GHz single core CPU as defined by *Amazon*, and MemSize (*Memory Size*) is in the unit of GBytes. Within each data center all PMs are same, that is have the same PM-size. The PM-size touples for the physical machines at these six data centers are $\{(42:96), (40:96), (26:72), (32:96), (20:8), (12:16)\}$. The PM-sizes $(42:96), (40:96)$ and $(32:96)$ are roughly equivalent to HP ProLiant SL390s G7/BL460c G7/BL460c G6 servers with 96GB memory. The PM-sizes $(26:72), (20:8)$ and $(12:16)$ are possible physical hardware configurations used by Amazon in supporting high memory instance, high CPU instance, and standard instance [15]. The number of maximal feasible configurations (or size of S_j) for six DCs are $\{161, 146, 42, 70, 3, 4\}$, and the reduced number of maximal feasible configurations (or size of $\hat{S}_j$) are $\{46, 31, 10, 23, 3, 3\}$. We use $\{\hat{S}_j\}$ in our simulation. The number of physical machines for the six data centers are set to be $\{100, 100, 150, 150, 200, 350\}$. The numbers are chosen such that the different datacenters have roughly similar total amount of CPU resources. We use eight types of VMs, as listed in Table I. The *VM-size* touples $(\#\mathrm{ECU} : \mathrm{MemSize} : \mathrm{DiskSize})$ of these VM types are identical to the 64-bit virtual machine types supported by *Amazon* [2]. We assume that the flow of incoming VMs is a Poisson process with average inter-arrival time of 2 seconds. The type of each arriving VM is independent of other VMs,

and is determined according to the distribution Dist-I or Dist-II given in the last two rows of Table I. Dist-I used in the experiments unless indicated otherwise. The life-time (or, service time) of VMs follows the truncated normal distribution with the average life-time of 20 minutes and the standard deviation of 5 min. The Shadow algorithm parameters are set as described in Section III.

TABLE I
VM RESOURCE REQUIREMENTS AND TYPE DISTRIBUTION

	VM1	VM2	VM3	VM4	VM5	VM6	VM7	VM8
CPU(ECU)	33.5	26	13	20	6.5	8	4	5
Mem(GB)	23	68.4	34.2	7	17.1	15	7.5	1.7
Disk(TB)	1.69	1.69	0.85	1.69	0.42	1.69	0.85	0.35
Dist-I	0.15	0.15	0.1	0.15	0.1	0.2	0.1	0.05
Dist-II	0.05	0.05	0.1	0.05	0.15	0.1	0.3	0.2

A. Datacenter PM- and disk utilizations using Shadow scheme

Fig. 1 and 2 depict the PM and disk storage utilizations at the six datacenters with three different values of algorithm parameter η, corresponding to coefficient $\gamma = 2$, 5, and 10. The simulation is warmed up for a period of two hours and then runs for 18 hours. We also calculate the optimal utilization level ρ by solving the linear program offline using the Matlab Optimization Toolbox, and plot it on the figures for the purpose of comparison. The offline optimal solution indicates that the PM-utilization constraints are binding only at a subset of DCs; and none of the disk storage utilization constraints are binding. The simulation results shown in Fig. 1 and Fig. 2 are consistent with all these facts: some of the PM-utilization curves in Fig. 1 oscillate closely around the optimal utilization level, while all disk storage utilization curves are below optimal utilization level. In addition, the plots show that the algorithm works well with different values of η, and thus is not very sensitive to this parameter, *as long as it is within a reasonable range as discussed in Section III.*

Table II lists the observed probabilities that a requesting VM belongs to VM type i and is placed at DC j (rounded to 0.001). Again, the results are close to the optimum (computed offline). VM type 1, requiring 33.5ECU, 23GB memory, and 1.69TB disk storage (see Table I) can only be hosted at DC1 (42:96) and DC2 (40:96). Interestingly, DC1 also hosts a significant number of VM6s which, together with VM1, efficiently utilizes DC1 physical machine's CPU resource. DC2 hosts a significant number of VM5s that, together with VM1, efficiently utilizes DC2 physical machine local resources. VM4 is a good fit for physical machine in DC5, while VM6 is a good fit for physical machine in DC6. So majority of VM4 and VM6 are placed in DC5 and DC6. VM2 fit DC3 and DC4 well, and two VM3 is equivalent to VM2, thus can also be hosted by DC3 and DC4. In addition, VM2+VM7, VM2+VM8, 2*VM3+VM7, and 2*VM3+VM8 are good matches for DC4. Overall the Shadow algorithm does a nice job closely tracking the optimal "packing" of VMs into PMs at different DCs, thus very efficiently utilizing the system resources.

TABLE II
PROBABILITY THAT A REQUESTING VM BELONGS TO VM TYPE i AND IS PLACED AT DC j USING SHADOW ALGORITHM ($\times 100$)

	VM1	VM2	VM3	VM4	VM5	VM6	VM7	VM8
DC1	7.5	0	0	0	1.6	4.2	2.2	0
DC2	7.5	0	0	0	6.2	0	0.8	0.2
DC3	0	7.7	5.0	0.1	0.9	0.3	0.2	0
DC4	0	7.3	4.9	0.6	1.1	0.8	5.9	2.8
DC5	0	0	0	14.3	0	0	0	0.1
DC6	0	0	0	0	0	14.5	0.9	1.9

For the purpose of comparison, we implement a *Baseline algorithm*, which is along the lines of the algorithms used, for example, in the current *VMware* implementations [8]. At the routing layer, Baseline algorithm "pretends" that all resources – not just disk storage – are pooled resources. Namely, it treats each localized resource k at DC j as a pooled resource, whose total amount β_{jk} is equal to the total amount of this resource in all of the PMs, namely $\beta_{jk} = \beta_j^* A_{jk}$. It routes an incoming VM request to the DC m that can accommodate the requesting VM and has the minimum max-utilization across the three resources (Disk storage, CPU, Memory). Denote by u_{jk} the used amount of resource type k at DC j. We route an incoming VM to a DC m satisfying

$$m \in \arg\min_{j \in R_i} \Big\{ \max_{k \in \mathcal{K}_p} \{u_{jk}/\beta_{jk}\} \Big\},$$

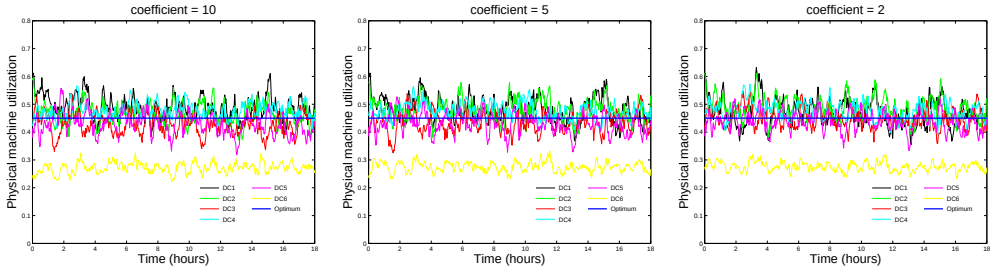
where $\mathcal{K}_p = \{1, \ldots, K\}$, because we assumed that *all* resource types are pooled.

At the DC layer, once the VM, say of type i, is admitted into DC m, the algorithm chooses the "best-fit" physical machine to host the VM. By best-fit we mean that the addition of this VM incurs the smallest increase of utilization of an individual PM. Denote by $\mathcal{N}_m = \{1, \ldots, \beta_m^*\}$ the set of PMs at DC m; and by v_{nk} the used amount of resource type $k \in \mathcal{K}_\ell$ at PM n. The VM is placed at PM n', where:

$$n' \in \arg\min_{n \in \mathcal{N}_m} \Big\{ \max_{k \in \mathcal{K}_\ell} \{(v_{nk} + a_{ik})/A_{mk}\} - \max_{k \in \mathcal{K}_l} \{v_{nk}/A_{mk}\} \Big\}.$$

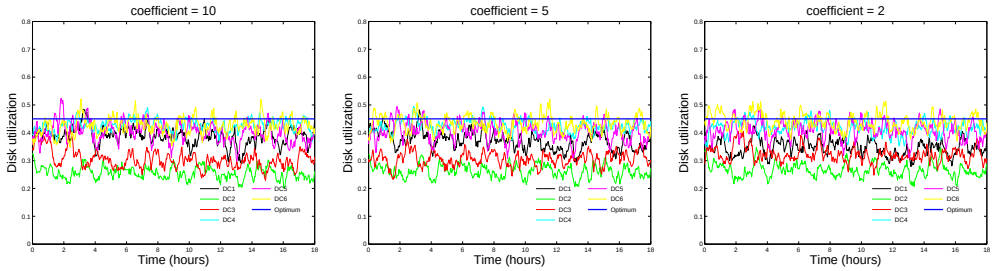
Remark 3. We note that, with the best-fit algorithm at the DC layer of Baseline, there is *no* straightforward way to improve packing efficiency via VM migrations, as was the case for the Shadow scheme (see Remark 2 in Section III.)

Fig. 3 depicts the PM-utilization and disk utilization using Baseline algorithm. The optimal utilization level is also shown on the figures. Compared to the PM- and disk utilization under the Shadow scheme, Baseline algorithm incurs $> 20\%$ increase in maximum utilization level. Table III illustrates the observed probability that a requesting VM belongs to VM type i and is placed at DC j using the Baseline algorithm (rounded to 0.001). Comparing to Table II, DCs tend to host more types of VMs, and less effective in pairing up matching VMs. For instance, VM1 and VM6 is a good pair for DC1, and VM1 and VM5 is a good pair for DC2. These two pairings are not as effectively employed by Baseline algorithm as by Shadow algorithm. Similarly, VM2+VM7, VM2+VM8, 2*VM3+VM7, and 2*VM3+VM8 are not well utilized in DC4, and VM4 entirely misses the best-match DC5.



(a) Shadow scheme ($\gamma = 10$) (b) Shadow scheme ($\gamma = 5$) (c) Shadow scheme ($\gamma = 2$)

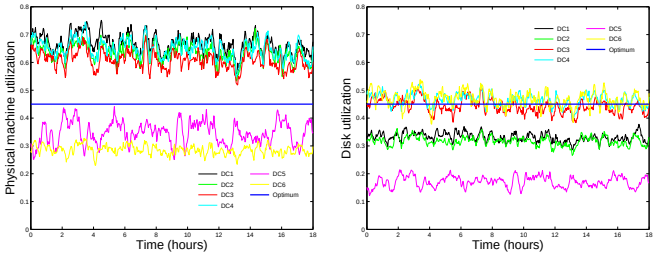
Fig. 1. Datacenter physical machine utilization using shadow routing



(a) Shadow scheme ($\gamma = 10$) (b) Shadow scheme ($\gamma = 5$) (c) Shadow scheme ($\gamma = 2$)

Fig. 2. Datacenter disk utilization using shadow routing

In other words, as expected, Baseline does a much less precise "packing" of VMs into PMs, which results in the efficiency loss.



(a) PM utilization (b) Disk utilization

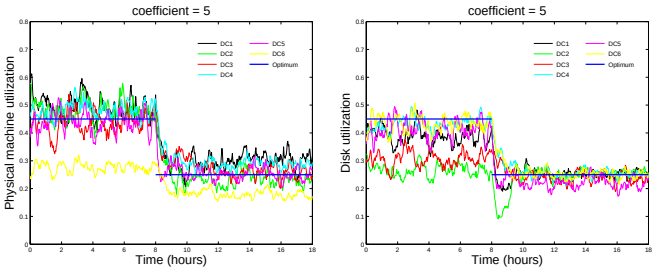
Fig. 3. Datacenter physical machine utilization and disk utilization using Baseline algorithm

TABLE III
PROBABILITY THAT A REQUESTING VM BELONGS TO VM TYPE i AND IS PLACED AT DC j USING BASELINE ALGORITHM ($\times 100$)

	VM1	VM2	VM3	VM4	VM5	VM6	VM7	VM8
DC1	7.7	2.4	0.7	0.9	0.7	0.3	0.0	0.0
DC2	7.3	2.3	0.7	1.0	0.6	0.2	0	0
DC3	0.0	0.0	6.2	9.3	6.2	1.9	0.0	0.0
DC4	0.0	10.4	2.4	3.8	2.3	0.7	0.0	0.0
DC5	0.0	0.0	0.0	0.0	0.0	0.0	10.1	5.1
DC6	0.0	0.0	0.0	0.0	0.0	16.7	0.0	0.0

B. Responsiveness of Shadow algorithm

We next examine the responsiveness of Shadow algorithm. We start with the investigation of how Shadow scheme reacts to a sudden change of VM arrival process. Specifically, the VM type distribution, as defined in Table I, is changed from Dist-I to Dist-II at the beginning of the 8-th hour (see Fig. 4). The change of the distribution leads to the change of individual VM request rates, thus different optimal routing rates and different optimal utilization level (shown in the figure). The results in Fig. 4 demonstrate that the Shadow algorithm is able to promptly respond to such sudden change and manage to keep track of the new optimal operating point. We emphasize that the algorithm responds *automatically* – it does *not* do any explicit "input rate change detection", but rather continues to run in the same manner as usual.

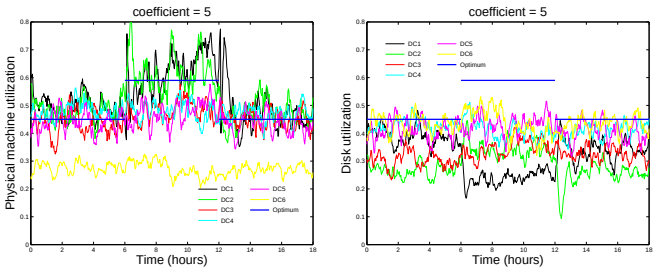


(a) PM utilization (b) Disk utilization

Fig. 4. Datacenter physical machine and disk utilization with sudden VM arrival rate change

Datacenters go through maintenance periods when a fraction of the physical machines or disk storages are temporarily taken offline for maintenance service. In the following, we examine how Shadow algorithm performs when some physical machines are taken offline for a period of time and then reinstalled. We choose to take half of the physical machines (50 physical machines) in Datacenter 1 offline at the 6-th hour and bring them back online at the 13-th hour (see Fig. 5). When the number of physical machines in DC j changes, the Shadow algorithm changes the corresponding value of β_j^*,

but otherwise continues to run as is. Even though the change is at DC 1 only, after this change the constraints on PM-utilization remain binding at some other DCs as well (DC 2 in addition to DC 1, to be precise). This fact is reflected in Fig. 5(a). Again, the algorithm is able to effectively respond to the sudden configuration change.



(a) PM utilization (b) Disk utilization

Fig. 5. Datacenter PM and disk utilization with datacenter maintenance

V. Simplified Shadow scheme

The Shadow scheme in Section III is (asymptotically) optimal in a large-scale system. A potential disadvantage of that scheme is that its "centralized" part – Algorithm-A, performed at the routing layer – requires a quite detailed information about each DC, namely the set of configurations $\hat{S}_j$. Furthermore, and more importantly, the routing layer needs to communicate current values of the fractions $\hat{\phi}_{sj}$ to each of the DCs, so that Algorithm-B can be run at the DC layer.

We now propose a simplified, *approximate* scheme, which does not require any information to be passed from the routing layer down to the DC layer, and requires routing layer to "know" much less about each DC. We refer to it as Simplified Shadow scheme. At the routing layer we "pretend" (as we did in Baseline algorithm) that each localized resource k at DC j is a pooled resource whose total amount β_{jk} is equal to the total amount of this resource in all of the PMs, namely $\beta_{jk} = \beta^*_j A_{jk}$. The values of β_{jk} for all resources (pooled and localized) is essentially all the routing layer will need to know about each DC. We then apply a *special case* of Algorithm-A, with $K' = K$, i.e. $\mathcal{K}_p = \{1, \dots, K\}$ containing all resource types, and $\mathcal{K}_l$ being empty. At DC layer, each DC j will run a special case of Algorithm-A, which is "confined" to this DC only and takes as input the flow of VMs routed to this DC by the routing layer. The exact definition of this scheme is given next.

A. Routing layer algorithm

The routing of VMs to DCs is done by the following algorithm. Recall we assume that all resource types are pooled, $\mathcal{K}_p = \{1, \dots, K\}$, with $\beta_{jk} = \beta^*_j A_{jk}$ for those resource types that (in reality) are localized.

Algorithm-C

There is a (small) parameter $\eta > 0$, parameter $c > 0$.

Upon each new actual VM arrival, say of class i to be specific, the algorithm does the following:

1. Compute DC index

$$m \in \arg\min_{j \in R_i} \sum_{k \in \mathcal{K}_p} Q_{jk} a_{ik} / (\beta_{jk} \mu_i),$$

route this VM to DC m and for this m do the following updates:

$$Q_{mk} := Q_{mk} + a_{ik}/(\beta_{mk}\mu_i), \quad \forall k \in \mathcal{K}_p.$$

2. If condition

$$\eta \sum_j \sum_{k \in \mathcal{K}_p} Q_{jk} \geq 1$$

holds, do the following updates:

$$Q_{jk} := \max\{Q_{jk} - c, 0\}, \quad \text{for all } j \text{ and } k \in \mathcal{K}_p.$$

End Algorithm-C

B. Data Center layer algorithm

This algorithm runs on each DC j. The set $\mathcal{K}_l$ is the set of localized resources in our original model.

Algorithm-D

There is a (small) parameter $\eta > 0$, parameter $c > 0$, and (small) parameter $\theta > 0$.

Upon each new actual VM arrival (into DC j), say of class i to be specific, the algorithm does the following (in sequence):

1. Do the following update:

$$Q_{ji} := Q_{ji} + 1/(\beta^*_j \mu_i).$$

2. Compute

$$\sigma^j \in \arg\max_{s \in \hat{S}_j} \sum_i s_i Q_{ji}.$$

If condition

$$\eta \sum_i \sigma^j_i Q_{ji} \geq 1 \tag{17}$$

holds, do the following updates:

$$Q_{ji} := \max\{Q_{ji} - c\sigma^j_i, 0\}, \quad \text{for each } i.$$

3. Update

$$\hat{\phi}_{sj} := \theta I(s, \sigma^j) + (1-\theta)\hat{\phi}_{sj}, \quad \text{for all } s \in \hat{S}_j, \tag{18}$$

where $I(s, \sigma^j) = 1$ if s was the configuration σ^j computed in step 2 and condition (17) in step 2 held, and $I(s, \sigma^j) = 0$ otherwise.

4. Compute configuration index

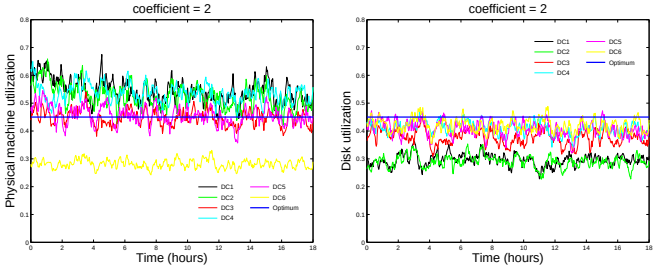
$$s' \in \arg\min_{s \in \hat{S}_j \,:\, s_i > 0} z^j_i(s)/[s_i \hat{\phi}_{sj}].$$

5. Among s'-PMs choose a PM with the maximal number of existing VM, but such that the existing number of class i VMs is less than s_i (so that the new class i VM can still fit), and assign the VM to this PM. If no such s'-PM is available, we place the VM into an empty PM and designate the PM as s'-PM.

End Algorithm-D

C. Simulation

Fig. 6 depicts the PM- and disk utilizations using the Simplified Shadow scheme. (Similarly to the Shadow scheme, we implement a slightly improved version of Step 5 of Algorithm-D. See Remark 2 in Section III.) As expected, the Simplified shadow scheme is not optimal and the maximum of PM-utilizations is about 10% higher than that achieved by Shadow scheme. The max PM-utilization of the Simplified Shadow, however, is about 10% better than that of the Baseline algorithm.



(a) PM utilization (b) Disk utilization

Fig. 6. Datacenter physical machine utilization and disk utilization with simplified routing strategy

VI. Real-time adjustment of the mean service times

Our algorithms do not use any information on the VM input rates λ_i, which is a key feature making them adaptive. The algorithms do however use the mean VM service times $1/\mu_i$ as parameters. Although the mean service times are much less subject to dramatic unexpected changes over time (compared to input rates), they can in fact change. Therefore it is reasonable to have a feedback loop which adjusts the mean service time parameters based on actual observed service time samples. To this end, we implement the following procedure. Every time a type i VM departs the systems, with its actual service time being T, we update the parameter $1/\mu_i$ used by the shadow routing as

$$1/\mu_i := \theta_1 T + (1-\theta_1)(1/\mu_i);$$

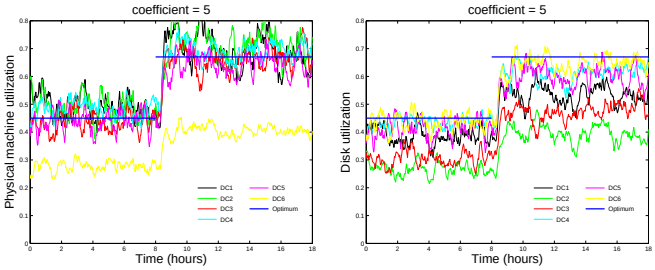
θ_1 is a small parameter – we use $\theta_1 = 0.01$

A. Simulation

In this experiment, we change the average VM life-time from 20 minutes to 30 minutes at the 8-th hour. Fig. 7 depicts the PM- and disk utilizations using the VM life-time sample mean as the input to the shadow routing algorithm. We also plot the optimum as the reference. The impact of using estimated VM life-time is marginal.

VII. Conclusions

We proposed a shadow routing based scheme for the problem of VM allocation in large heterogeneous data centers or server clusters, and demonstrated – both analytically and via simulations – its good performance, robustness and adaptability. In particular, we addressed one of the key challenges in this



(a) PM utilization (b) Disk utilization

Fig. 7. Datacenter physical machine utilization and disk utilization with sample based mean service time estimation

kind of allocation problems: the need to observe VM-to-PM packing constraints. We showed how the packing constraints can be incorporated to produce an (asymptotically) optimal solution and – equally importantly – demonstrated that this can be done in a way that is computationally feasible in practical implementations.

References

[1] Mansoor Alicherry, T. V. Lakshman. Network aware resource allocation in distributed clouds. *INFOCOM-2012*.

[2] Amazon EC2 Instance Types, *http://aws.amazon.com/ec2/instance-types/*

[3] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica and M. Zaharia, Above the clouds: A Berkeley view of cloud computing, *Communications of the ACM*, April 2010.

[4] M. de Assuncao, A. di Costanzo and R. Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters, *Proceedings of ACM international symposium on High performance distributed computing (HPDC)*, 2009, pp. 141-150.

[5] N. Bansal, A. Caprara, M. Sviridenko. A New Approximation Method for Set Covering Problems, with Applications to Multidimensional Bin Packing. *SIAM J. Comput.*, 2009, Vol.39, No.4, pp.1256-1278.

[6] J. Csirik, D. S. Johnson, C. Kenyon, J. B. Orlin, P. W. Shor, and R. R. Weber. On the Sum-of-Squares Algorithm for Bin Packing. *J.ACM*, 2006, Vol.53, pp.1-65.

[7] A. Gulati, G. Shanmuganathan, A. Holler, I. Ahmad. Cloud Scale Resource Management: Challenges and Techniques. *HotCloud-2011*.

[8] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, X. Zhu. VMware Distributed Resource Management: Design, Implementation and Lessons Learned. *VMware Technical Journal*, 2012, Vol.1, No.1, pp. 45-64. http://labs.vmware.com/publications/vmware-technical-journal

[9] J.W. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang. Joint VM Placement and Routing for Data Center Traffic Engineering. *INFOCOM-2012*.

[10] S.T. Maguluri, R. Srikant, L.Ying. Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters. *INFOCOM-2012*.

[11] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. *INFOCOM-2010*.

[12] X. Meng, C. Isci, J. Kephart, L. Zhang, and E. Boulillet. Efficient resource provisioning in compute clouds via VM multiplexing. *Proc. ICAC, 2010*.

[13] A. L. Stolyar. Maximizing Queueing Network Utility subject to Stability: Greedy Primal-Dual Algorithm. *Queueing Systems*, Vol. 50, pp. 401-457 (2005).

[14] A. L. Stolyar, T. Tezcan. Control of systems with flexible multi-server pools: A shadow routing approach. *Queueing Systems*, 2010, Vol.66, pp.1-51.

[15] Huan Liu's Blog. *http://huanliu.wordpress.com/2010/06/14/amazons-physical-hardware-and-ec2-compute-unit/*.

[16] X. Wang, H. Xie, R. Wang, Z. Du, L. Jin, Design and implementation of adaptive resource co-allocation approaches for cloud service environments. *3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, Aug. 2010.