

# Online Algorithms for Joint Application-VM-Physical-Machine Auto-Scaling in a Cloud\*

Yang Guo  
Bell Labs, Alcatel-Lucent  
Holmdel, NJ 07733  
yang.guo@alcatel-lucent.com

Alexander L. Stolyar  
Bell Labs, Alcatel-Lucent  
Murray Hill, NJ 07974  
stolyar@research.bell-labs.com

Anwar Walid  
Bell Labs, Alcatel-Lucent  
Murray Hill, NJ 07974  
anwar@research.bell-labs.com

## ABSTRACT

We consider the joint application-VM-Physical-Machine assignment problem in a cloud environment, where application requests are serviced by Virtual Machines (VMs) residing on Physical Machines (PMs). The VMs are dynamically allocated to accommodate varying application needs, while being intelligently packed into PMs to minimize the number of hosting PMs. We consider a shadow routing based approach for this joint assignment problem. The proposed shadow algorithm employs a specially constructed virtual queueing system to dynamically produce an optimal solution that guides the actual application-to-VM and VM-to-PM assignments. The proposed algorithm runs continuously without the need to re-solve the underlying optimization problem “from scratch”, and adapts automatically to the changes in the application demands. We prove the asymptotic optimality of the shadow algorithm. The simulation experiments further demonstrate the algorithm’s good performance and high adaptivity.

## Categories and Subject Descriptors

G.1.6 [NUMERICAL ANALYSIS]: Optimization—*Convex programming, Linear programming*; G.3 [PROBABILITY AND STATISTICS]: Probabilistic algorithms, Queueing theory

## Keywords

Cloud, VM choice and placement, Shadow routing

## 1. INTRODUCTION

A Cloud data-center typically consists of a large number of physical machines (PMs). Virtualization technology enables cloud service providers to offer customers virtual machines (VMs) to run their applications, while facilitating the sharing of CPU, memory and storage resources. VMs can be

configured and allocated as needed to meet hourly, daily, or weekly demand variation. When an application’s demand for resources increases, additional VMs are allocated to meet the quality of service requirement; conversely, when the demand drops, resources can be freed to save cost. The cloud service providers may further wish to pack the allocated VMs into a small number of PMs so as to minimize operational costs, save energies, and/or generate higher revenues by accommodating more applications. The design of algorithms for the joint application-to-VM and VM-to-PM assignments that minimize the number of hosting PMs is a challenging research problem.

Considerable prior works exist on cloud resource management with the focus on either the application-to-VM assignments or VM-to-PM assignments. In application-to-VM assignment, the application requests are assigned to VMs to receive the service. The number of allocated VMs needs to be large enough so that the application requests can be serviced with satisfactory performance, i.e., satisfying the required *SLA (service-level agreement)*; yet not overly large to avoid unnecessary cost to users and resource wastage for cloud service providers. *Auto-Saling* is a popular feature that realizes the dynamic application-to-VM assignment, and is offered by many cloud service providers, e.g., Amazon [2]. [13] gives an extensive literature review of proposed auto-scaling techniques, which can be classified into two categories: *reactive techniques* that are based on predefined rules, and *proactive techniques* that involve load prediction. [16] incorporates user’s budget and QoS constraints to assign schedules for VM startups and shut-downs. [11] proposes a system and online algorithms for VM autoscaling based on load prediction. [8] consider an autoscaling solution which takes into account VM configuration and boot time. Nevertheless, the existing work on auto-scaling does not take the VM-to-PM assignment into consideration.

The problem of efficient VM-to-PM assignment, taken in isolation (i.e. disjointly from the application-to-VM assignment problem), fits into the general framework of *stochastic bin packing*. Classical stochastic bin packing problems are such that the “items” (VMs) arrive exogeneously, are packed into “bins” (PMs), and never leave the system. (See e.g. [6, 7] for good reviews, and also a recent paper [10].) In the VM-to-PM assignment problem, “items” (VMs) arrive in the system, are served for a random time, and then leave the system. This can be termed as a *stochastic bin packing with item departures*. The work in this direction

\* (Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

is more recent, motivated primarily by the VM-to-PM assignment in a network cloud, and includes, in particular, papers [9, 12, 14, 15, 22–24] (and to some extent [10]), which focus on different aspects of the problem at different generality levels. There are other recent VM-placement work, e.g., [1, 17, 19], which addresses the placement problem from different angles, ranging from minimizing network traffic, to shortening the inter-VM distance/latency and statistically sharing resources.

In this paper, we consider the joint assignment of applications to VMs and VMs to PMs, where applications are dynamically assigned to VMs to meet their performance requirements, while the VMs are configured (placed) in physical machines such that the cloud resources are efficiently utilized. In our framework, we explicitly consider (i) the varying demand load for applications, expressed in terms of arrival rates for individual applications and the required time to serve an application request; (ii) the VM type(s) an application may require, characterized by the computing, memory and storage resource required to handle certain number of application requests; and (iii) the physical machines and their resource configurations.

Our paper continues the direction of [9]; in particular, it follows the *shadow routing* approach, employing a specially constructed virtual queueing system, which in essence dynamically produces and maintains a solution to the underlying optimization problem that guides the actual assignment algorithm. The main contribution of [9] was to demonstrate how VM-to-PM packing constraints can be incorporated in the shadow routing framework. The distinctive feature of this paper is that we have a joint application-to-VM and VM-to-PM assignment problem. We show that the shadow routing can still be applied, but the corresponding virtual queueing system is necessarily more complicated (it has two tiers). The advantage of shadow routing approach is that it is simple and adaptive: no need to know a priori, or explicitly measure, the application request arrival rates; if the arrival rates change, the algorithm adapt automatically. Yet, as we show, the algorithm is asymptotically optimal. All these features are confirmed by our simulation experiments.

The rest of the paper is organized as follows. The formal model and the optimization problem are given in Section II. Section III defines the Shadow scheme (including the construction of the two-tier virtual queueing system). Section IV proves the asymptotic optimality of the Shadow scheme. In Section V we provide practical parameter settings, and also give the procedures for actual application-to-VM and VM-to-PM assignments, guided by the Shadow algorithm. In Section VI we present simulation results for the Shadow scheme where we test its accuracy, adaptability and robustness in the face of varying application request loads. In Section VII we consider the special and challenging case of sudden and appreciable changes in the application request loads and demonstrate how expedited algorithm convergence can be achieved. Section VIII concludes the paper with discussions.

## 2. MODEL AND PROBLEM STATEMENT

### 2.1 Model

There are several applications, indexed by  $i \in \mathcal{I} = \{1, \dots, I\}$ , and several classes of VMs, indexed by  $j \in \mathcal{J} = \{1, \dots, J\}$ . At any given time, each application  $i$  employs a group of VMs, denoted by a vector  $m_i = (m_{ij}, j = 1, \dots, J)$  to provide the service, where  $m_{ij}$  is the number of VMs of class  $j$  assigned to application  $i$ . User service requests for application  $i$  arrive at the rate  $\lambda_i$ . A service request for application  $i$  is assigned to one of the VMs of application  $i$  and is serviced by that VM. The service time of an application  $i$  request is  $\mu_i$ . A VM of class  $j$  can service an interger number  $w_{ij} \geq 0$  of concurrent application  $i$  requests.

Each class  $j$  VM needs several computing resources of different types when it is served; namely, the amount  $a_{jk} > 0$  of resources  $k = 1, \dots, K$ . The DC contains  $\beta > 0$  physical machines (PM), each of which has the amount  $A_k > 0$  of resource  $k \in \mathcal{K}_\ell$ . If a class  $j$  VM is instantiated in the DC, it is placed into one of the PMs, where the amounts  $a_{jk}$  of resources are allocated (if they are still available at that specific PM). This means, in particular, that a PM in the DC can simultaneously serve the numbers of different VM class types given by a vector  $s = (s_j, j = 1, \dots, J)$  if

$$\sum_i s_j a_{jk} \leq A_k, \quad \forall k \in \mathcal{K}_\ell. \quad (1)$$

Vectors  $s$  (with non-negative integer components) satisfying this condition we call (*feasible*) *configurations* of a PM in the DC. The set of configurations which are not dominated by convex combinations of other configurations is denoted by  $S$ .

We make a natural non-degeneracy assumptions: for each  $i$  there exist  $j$  and  $s$  such that  $w_{ij} \geq 1$  and  $s_j \geq 1$  (otherwise application  $i$  cannot be served at all); for each  $j$  there exists  $s$  such that  $s_j \geq 1$  (otherwise, we can simply exclude some  $j$  from the model).

### 2.2 Problem statement

Denote by  $\phi_s \geq 0$  the fraction of PMs in the DC that are used in the configuration  $s \in S$ . We want an algorithm that allocates fractions  $\phi_s$ , and numbers of VMs,  $m_{ij}$ , such that ideally they are close to the optimal solution of the following linear program:

$$\min_{\{m_{ij}\}, \{\phi_s\}, \rho} \rho, \quad (2)$$

subject to

$$m_{ij} \geq 0, \quad \forall (i, j), \quad \phi_s \geq 0, \quad \forall s, \quad (3)$$

$$\lambda_i / \mu_i \leq \sum_j w_{ij} \cdot m_{ij}, \quad \forall i, \quad (4)$$

$$\sum_i m_{ij} / \beta \leq \sum_{s \in S} s_j \phi_s, \quad \forall j, \quad (5)$$

$$\sum_{s \in S} \phi_s = \rho. \quad (6)$$

Note that in this LP the variables  $\phi_s$  and  $m_{ij}$  are non-negative *real* numbers; so that they have the meaning of

average values (of the corresponding "true" variables), such that the average system workload can be handled. The LHS of (4) is the aggregate (average) workload for application  $i$ , while the RHS of (4) is the aggregate (average) service capacity allocated this application. The constraint (5) might be easier to understand in the form  $\sum_i m_{ij} \leq \sum_{s \in S} s_j \phi_s \beta$ , with  $\phi_s \beta$  being the (average) number of PMs used in configuration  $s$  and, therefore,  $s_j \phi_s \beta$  being the (average) number of  $j$ -VMs served by PMs in configuration  $s$ . Finally, note that, although the meaning of  $\rho$  is the average fraction of the utilized PMs, the LP makes sense even if the optimal  $\rho$  is greater than 1 – if this is the case, the DC is overloaded.

### 3. SHADOW ROUTING BASED SCHEME

#### 3.1 Construction of the virtual queueing system

We will now construct an algorithm, which makes variable updates upon each application request arrival, and asymptotically solves the LP (2)-(6) (in the sense that will be specified below). The algorithm maintains and updates a virtual ("shadow") queueing system, and makes "routing" and "service" decisions based on the current state of that system, depicted schematically on Figure 1.

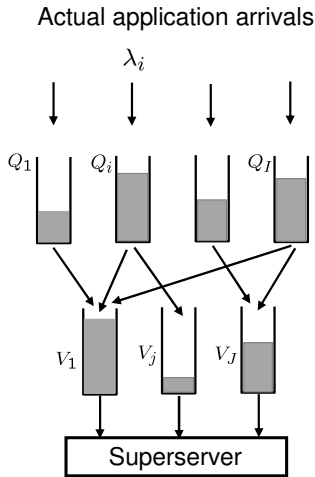


Figure 1: Virtual (shadow) queueing system

Assume for simplicity that the arrival flow of service requests for each application  $i$  is Poisson. (This is not crucial.) Then, the sequence of request arrivals is such that the type of each arrival is determined randomly, according to probabilities  $\lambda_i/\lambda$ , where  $\lambda = \sum_{i'} \lambda_{i'}$ , independently of other arrivals. For each application type  $i$  there is the associated virtual queue, whose length is denoted by  $Q_i$ . For each VM class  $j$  there is the associated virtual queue, whose length is denoted by  $V_j$ . (The virtual queues are just variables maintained by the algorithm – they are not physical queues where application requests, or VMs, or anything else, wait for service.) When a request, say of type  $i$ , arrives, the following sequence of events and decisions takes place, in sequence. First, the amount  $1/\mu_i$  of "work" is placed in  $Q_i$ , namely  $Q_i := Q_i + 1/\mu_i$ . Then, the algorithm must decide whether

or not to activate a virtual server ( $ij$ ), for one of the  $j$ . If ( $ij$ ) server is activated, the amount  $c_1 w_{ij}$  is removed from  $Q_j$  and the amount  $c_1/\beta$  is added to  $V_j$ , namely

$$Q_i := \max\{Q_i - c_1 w_{ij}, 0\}, \quad V_j := V_j + c_1/\beta.$$

Here  $c_1 > 0$  is a parameter. Finally, the algorithm must decide whether or not to activate a "superserver" that "serves" queues  $V_j$  in one of the modes  $s \in S$ ; if it is activated in mode  $s$ , then the amount  $c_2 s_j$  is removed from each of the  $V_j$ , namely

$$V_j := \max\{V_j - c_2 s_j, 0\}, \quad \forall j.$$

Here  $c_2 > 0$  is another parameter. It is easy to see that the choice of parameters  $c_1$  and  $c_2$  satisfying

$$c_1 > \max_i 1/\mu_i, \quad c_2 > c_1/\beta \quad (7)$$

is sufficient for the virtual queueing system to be able to "keep up" with the incoming load, in the sense that there exists an algorithm for activating servers ( $ij$ ) and the superserver, so that all virtual queues remain stable (do not run away to infinity).

Now, suppose we want an algorithm for activating servers ( $ij$ ) and the superserver, such that the average frequency of superserver activation is minimized, subject to the constraint that all virtual queues remain stable. The virtual queueing system and the problem we just described are within the framework of general model in [20], 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 activates servers ( $ij$ ) and activates different modes  $s$  of the superserver, essentially solve the LP (2)-(6).

**Remark.** We want to emphasize that the virtual queues are *not* buffers where actual application requests or 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 requests.

#### 3.2 Shadow algorithm

**Algorithm:** The algorithm maintains the virtual queues  $Q_i$  and  $V_j$ , and uses parameters  $c_1, c_2 > 0$ , as described above. In addition, there is a (small) parameter  $\eta > 0$ , (small) parameter  $\theta > 0$ , and parameter  $\alpha > 0$ .

Upon each new service request arrival, say of type  $i$  to be specific, the algorithm does the following (in sequence):

1.  $Q_i(t) := Q_i(t) + 1/\mu_i$ .

2. Set  $b_{\ell j} := 0$  for all  $(\ell j)$ . Compute

$$j^* \in \arg \max_j [\alpha w_{ij} Q_i - (1/\beta) V_j], \quad (8)$$

and if

$$\alpha w_{ij^*} Q_i - (1/\beta) V_{j^*} \geq 0 \quad (9)$$

do

$$Q_i := \max\{Q_i - c_1 w_{ij^*}, 0\}, \quad V_{j^*} := V_{j^*} + c_1/\beta, \quad b_{ij^*} := 1.$$

3. Set  $b_s = 0$  for all  $s \in S$ . Find configuration vector  $\sigma$  such that

$$\sigma \in \arg \max_{s \in S} \sum_{j \in \mathcal{J}} s_j V_j. \quad (10)$$

If condition

$$\eta \sum_j \sigma_j V_j \geq 1 \quad (11)$$

holds, set  $b_\sigma := 1$ . Update virtual queues  $V_j(t)$  as follows:

$$V_j := \max\{V_j - \sum_{s \in S} s_j b_s, 0\}, \quad \forall j \in \mathcal{J}.$$

4. Update variables  $\bar{p}_{\ell j}$  and  $\bar{\phi}_s$ .

$$\bar{p}_{\ell j} := (1 - \theta)\bar{p}_{\ell j} + \theta b_{\ell j}, \quad \forall (\ell, j),$$

$$\bar{\phi}_s := (1 - \theta)\bar{\phi}_s + \theta b_s, \quad \forall s.$$

**End of Algorithm**

## 4. ASYMPTOTIC OPTIMALITY OF SHADOW ALGORITHM

**PROPOSITION 1.** *Suppose all system parameters and all Shadow algorithm parameters, except maybe  $\eta$ , are fixed rational numbers. Assume that condition (7) holds. Suppose the input flows are Poisson, with fixed rates  $\lambda_i$ . Consider a sequence of systems with parameter  $\eta \rightarrow 0$ . Then, for any  $\eta$ , 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  $\bar{\phi}_s^{(\eta)}$  the steady-state probability that configuration  $s$  is chosen in (10) and condition (11) holds for it. Similarly, let  $\bar{p}_{ij}^{(\eta)}$  be the steady-state probability that the arriving request is of type  $i$ , index  $j$  is chosen in (8) and condition (9) holds. Then, as  $\eta \rightarrow 0$ , the sequence of vectors  $(\{\bar{\phi}_s^{(\eta)}\}, \{\bar{p}_{ij}^{(\eta)}\})$  is such that its any limiting point  $(\{\bar{\phi}_s\}, \{\bar{p}_{ij}\})$  satisfies (using notation  $\lambda = \sum_i \lambda_i$ )*

$$\lambda c_1 \bar{p}_{ij} = m_{ij}, \quad \forall (i, j),$$

$$\lambda c_2 \bar{\phi}_s = \phi_s, \quad \forall s \in S,$$

where vector  $(\{\bar{\phi}_s\}, \{m_{ij}\})$  is an optimal solution of LP (2)-(6).

*Proof of Proposition 1:* The virtual queueing process, viewed as a discrete time process at the times just after request 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 [20]), 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  $\eta$ . (Here we use condition (7), which guarantees that the system has sufficient capacity so that condition (55) in [20] holds.) Then, the property (AO-2) in section 4.9 of [20] can be established. In our case, it means that, as  $\eta \rightarrow 0$ , the algorithm solves the problem of minimizing the

frequency of superserver activations, subject to stability of virtual queues. Formally, if for each  $\eta$  we pick a stationary distribution, then, as  $\eta \rightarrow 0$ ,  $(\{\bar{\phi}_s^{(\eta)}\}, \{\bar{p}_{ij}^{(\eta)}\})$  converges to a set of optimal solutions  $(\{\bar{\phi}_s\}, \{\bar{p}_{ij}\})$  of the following LP:

$$\min_{\{\bar{p}_{ij}\}, \{\bar{\phi}_s\}, \bar{\rho}} \bar{\rho}, \quad (12)$$

subject to

$$\bar{p}_{ij} \geq 0, \quad \forall (i, j), \quad \bar{\phi}_s \geq 0, \quad \forall s, \quad (13)$$

$$(\lambda_i/\lambda)/\mu_i \leq \sum_j c_1 w_{ij} \bar{p}_{ij}, \quad \forall i, \quad (14)$$

$$\sum_i \bar{p}_{ij} c_1/\beta \leq \sum_{s \in S} s_j \bar{\phi}_s, \quad \forall j, \quad (15)$$

$$\sum_s \bar{\phi}_s = \bar{\rho}, \quad (16)$$

$$\sum_j \bar{p}_{ij} \leq 1, \quad \forall i, \quad \sum_s \bar{\phi}_s \leq 1. \quad (17)$$

Using again condition (7), it is easy to see that there exists an optimal solution to the LP (12)-(16), for which condition (17) holds. This means that any optimal solution of LP (12)-(17) is also optimal for LP (12)-(16). Finally, we observe that if we rewrite LP (12)-(16) in terms of variables  $m_{ij} = \lambda c_1 \bar{p}_{ij}$ ,  $\phi_s = \lambda c_2 \bar{\phi}_s$  and  $\rho = \lambda c_2 \bar{\rho}$ , we obtain problem (2)-(6). The result follows.  $\square$

## 5. IMPLEMENTATION ISSUES

### 5.1 Algorithm parameter setting

In view of condition (7), parameters  $c_1$  and  $c_2$  can be set as follows:  $c_1 = 1.01 \max_i 1/\mu_i$  and  $c_2 = 1.01 c_1/\beta$ .

The averaging parameter  $\theta$  can be set to for example to  $\theta = 0.001$ .

”Typical value” of one  $V_j$  is

$$D_1 = (1/\eta)(1/\max_s \sum_{j \in \mathcal{J}} s_j).$$

Maximum change of a  $V_j$  is:  $D_2 = \max[c_1/\beta, c_2 \max_{s,j} s_j]$ . We choose  $\eta$  such that

$$D_2/D_1 = 1/\gamma, \quad \gamma = 2, 5, 10.$$

Finally, a typical scale of a  $Q_i$  is  $[1/(\alpha\beta w_{ij})]$  times the typical value of  $V_j$ . (See step 2 of the algorithm.) To make sure that the typical values of  $Q_i$  are not too small compared to those of  $V_j$ , we set  $\alpha$  so that for all  $(ij)$ ,  $1/(\alpha\beta w_{ij}) \geq 1/3$ , namely  $\alpha = 3/[\beta \max_{ij} w_{ij}]$ .

### 5.2 Actual application request assignment algorithm

The algorithm at any time has access to the quantities  $\bar{p}_{ij}$ , maintained by the Shadow algorithm.

For each  $(ij)$  let  $X_{ij}$  be the number of applications  $i$  currently assigned to  $j$ -VMs. When a new  $i$ -application arrives

we do the following, in sequence:

1. Assign this  $i$ -application to VM type  $j$  which minimizes  $X_{ij}/[w_{ij}\bar{p}_{ij}]$ .
2. For this  $j$  we pick a PM configuration  $s$  according to VM-to-PM mapping algorithm below.
3. If there are VMs of the chosen above class  $j$ , serving applications  $i$  in PMs with the chosen above configuration  $s$ , we assign the application to a such available non-empty VM with the maximum number of applications in it; else (i.e. if such non-empty VM  $j$  does not exist), we request a new VM  $j$ , which is placed into an  $s$ -PM according to VM-to-PM mapping algorithm below.

When an application departs and leaves its VM empty, we treat this as a VM departure.

### 5.3 VM-to-PM mapping algorithm

Each non-empty PM at any given time has a designated configuration  $s \in S$ ; the designation  $s = (s_1, \dots, s_J)$  means that we will never place more than  $s_j$  class  $j$  VMs into this PM. A PM with designation  $s$  is referred to as an  $s$ -PM. Empty PMs do not have any designation. The following quantity is maintained for each  $s \in S$ :  $z_j(s)$  – the total number of class  $j$  VMs in  $s$ -PMs. In addition, the algorithm at any time has access to the quantities  $\bar{\phi}_s$ , maintained by the Shadow algorithm.

When a new class  $j$  VM is requested, the algorithm does the following (in sequence):

1. Compute configuration index

$$s' \in \arg \min_{s \in S : s_j > 0} z_j(s)/[s_j \bar{\phi}_s].$$

2. Among  $s'$ -PMs choose a PM with the maximal number of existing VM, but such that the existing number of class  $j$  VMs is less than  $s_j$  (so that the new class  $j$  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.

The above algorithm is identical to the VM-to-PM mapping algorithm (Algorithm B) in [9].

## 6. EVALUATION

In this section we evaluate the Shadow scheme using simulations. (Here, when we refer to the Shadow scheme, or algorithm, we mean the combination of the Shadow itself as defined in Section 3.2 and the actual assignment algorithms in Sections 5.2 and 5.3.) We consider a data center with a pool of 1,000 *Physical Machines* (PMs), with each PM being configured with 42 ECUs (*Elastic Compute Units*) and 96 GBytes of memory, where one ECU is equivalent to one 1GHz single core CPU as defined by *Amazon* [3]. The PM configuration is roughly equivalent to a HP ProLiant SL390s G7 server. We assume there are eight *Virtual Machine* (VM) types, as listed in Table 1. The *VM-size* tuples (#ECU : MemSize) of these VM types are identical to the 64-bit virtual machine types supported by *Amazon*. The maximal feasible configuration set, as defined in Section 2.1, contains 46 vectors.

We assume there are four types of applications supported by the data center. Denote by  $w_{ij}$  the number of concurrent application requests of type  $i$  that can be serviced by a VM of type  $j$ . Table 1 lists the values of  $\{w_{ij}\}$ . We select the values of  $\{w_{ij}\}$  such that App 1 to App 4 resemble the *CPU intensive application*, *interactive application*, *high-memory high-bandwidth application*, and *file access application*, respectively. Note that the designed algorithms are general and can handle arbitrary parameter settings.

We start with the experiments to examine the accuracy of the Shadow algorithm. We then investigate the adaptivity of the algorithms, and extend the Shadow algorithm to support *services*. The means to expedite the convergence in the face of sudden arrival rate change is studied in Section 7.

**Table 1: VM resource requirements and Number of concurrent application requests that can be serviced by a VM ( $\{w_{ij}\}$ )**

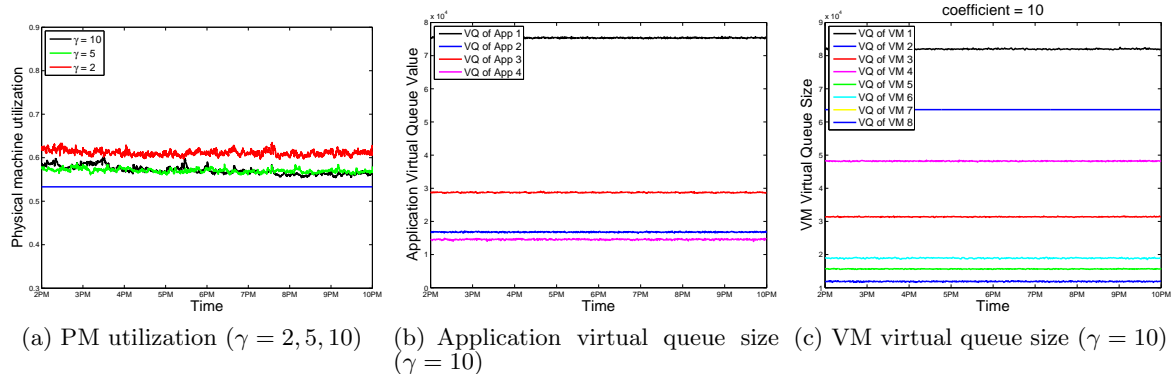
	CPU(ECU)	Mem(GB)	App1	App2	App3	App4
VM1	33.5	23	34	56	23	40
VM2	26	68.4	26	94	68	40
VM3	13	34.2	13	47	34	20
VM4	20	7	20	27	0	40
VM5	6.5	17.1	0	24	17	10
VM6	8	15	0	23	15	40
VM7	4	7.5	0	12	0	20
VM8	5	1.7	0	22	0	10

### 6.1 Datacenter PM utilizations and virtual queue sizes

In this experiment, the application requests arrive according to the Poisson process with the average arrival rate of 124 requests/sec. It resembles the scenario where all applications are equally active during the simulated time period. The service time of a request is a constant of 100 seconds. The algorithm parameters are set as described in Section 5.1. While not presented, the proposed algorithms work well with different arrival rates and service times. We remove the first two hours of simulated time as the warm up period. We will examine the adaptivity/responsiveness of the algorithms in Section 6.2, and present additional procedure for expedited convergence in Section 7.

Fig. 2(a) depicts the PM utilization with different coefficient ( $\gamma$ ) values. The larger the value of coefficient  $\gamma$ , the smaller the  $\eta$  value, and thus the Shadow algorithm is more accurate. (On the other hand, the convergence time is larger.) We also plot the optimal PM utilization value obtained using CPLEX optimizer [4] for the purpose of comparison. The optimal PM utilization is 53.4%. The PM utilizations with  $\gamma = 5$  and 10 are roughly the same, around 56.5%, while the PM utilization with  $\gamma = 2$  reaches 62%. The results indicate that the value of  $\gamma$  needs to be about 5 or greater to achieve good accuracy. We hence use  $\gamma = 10$  in the following experiments.

To look into the algorithm performance, we plot the virtual queue sizes associated with different application types and different virtual machine types in Fig. 2(b) and Fig. 2(c), respectively. In Shadow algorithms, the virtual queue sizes, scaled by factor  $\eta$ , converge to optimal dual variables. All virtual queue sizes remain approximately constant in Fig. 2(b)



**Figure 2: Physical machine (PM) utilizations with different coefficient values, and the virtual queue sizes for application and Virtual Machine (VM)**

and Fig. 2(c), a strong indication that the algorithm has converged to the optimal point. In addition, the values of application virtual queue sizes and the values of virtual machine virtual queue sizes are in the range of  $10^4$ , suggesting the rescaling of virtual queue sizes using the parameter  $c_1, c_2, \eta$  in Section 3.2 has been effective.

## 6.2 Adaptivity of the algorithms

We next investigate the adaptivity/responsiveness of the proposed algorithms. Applications' request rates typically vary over time and different applications exhibit different daily patterns. For instance, companies may run more CPU intensive applications, e.g., Map-Reduce type of applications, over-night than in the day-time. Meanwhile, applications such as virtual desktop become less requested in the evening than during the working hours. In this experiment, we assume the application request rate gradually change from 5pm to 7pm. Specifically, the request rate for application 1 increases linearly from 124 requests/second at 5pm to 208 requests/sec at 7pm, while the request rates for application 2, 3 and 4 decrease linearly from 124 requests/sec at 5pm to 35 requests/sec, 52 requests/sec, and 52 requests/sec at 7pm, respectively. The change of application request rates not only changes the optimal PM utilization, but also dynamically changes the optimal number of VMs allocated for different applications and optimal number of PMs assigned to different configurations.

Fig. 3(a) depicts the PM utilization over time. Without explicitly detecting the input rate change, the Shadow algorithm is able to automatically keep track of the new optimal operating point as it continues to run in the same manner as usual. Fig. 3(b) illustrates the number of VMs for some application types, i.e.,  $m_{ij}$ . We select the ones that are *active* during the transition period from 5pm to 7pm. The number of VMs adjusts nicely based on the current application request rates. Fig. 3(c) further depicts the number of PMs assigned to some configurations. There is a total of 46 configurations in  $S$ . We selected the configurations that are assigned to positive number of PMs. The results in Fig. 3 demonstrate that the Shadow algorithm is able to promptly respond to application request rate change and successfully keep track of the new optimal operating point.

## 6.3 Providing services

In the model considered so far (see Section 2), an application request is serviced by one VM. In practice, an application request can be multi-tiered and may need the services from multiple VMs. For instance, the Web application architecture consists of three tiers: *presentation tier*, *application tier*, and *data access tier*. Different tiers provide different functionalities and may require different types of VMs. The presentation tier communicates with other tiers, and displays the results to the client browsers. The application tier makes logic decisions and performs calculations. It also moves the processing data between presentation tier and data access tier. Finally, data access tier stores and retrieves the information from a data base or file system. The information is then passed back and forth to the application tier for processing, and then eventually back to the presentation tier to display to users. Different tiers require different types of computational resources.

For our purposes, let's define such request as a *service request*, where a service request employs one or multiple *applications*, as defined in Section 2, to complete the service. Note that different services may employ the same underlying applications, e.g., multiple services need to access the same data base thus the data-access application can be shared among them. Without loss of generality, we assume that application requests of the same type, belonging to different services, can share a VM. (If not, these applications can be treated as different application types within our model.) Let  $k$  indexes the service types,  $k \in \mathcal{K} = \{1, 2, \dots, K\}$ , and  $\lambda_k$  be the arrival rate of service  $k$ . Denote by  $\mathcal{I}_k \subseteq \mathcal{I}$  the subset of applications used by service  $k$ .

The optimization problem generalizes as follows:

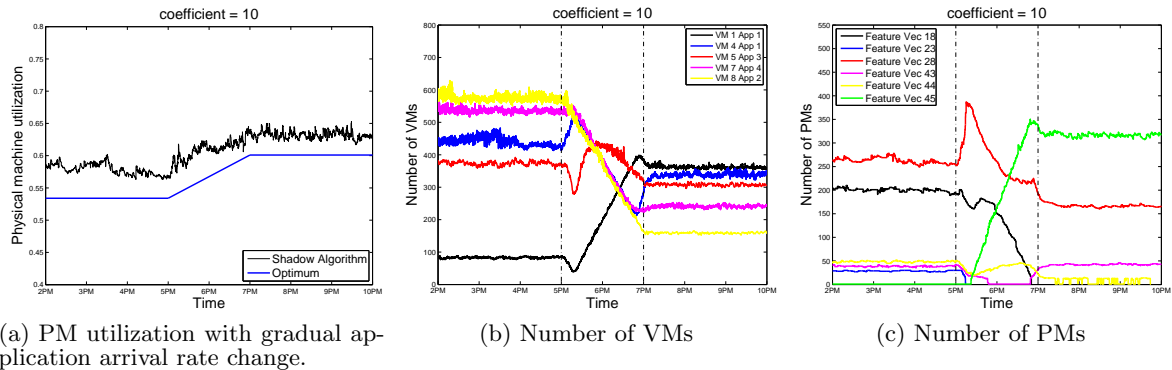
$$\min_{\{m_{ij}\}, \{\phi_s\}, \rho} \rho, \quad (18)$$

subject to

$$m_{ij} \geq 0, \quad \forall (i, j), \quad \phi_s \geq 0, \quad \forall s \quad (19)$$

$$\sum_{k: i \in \mathcal{I}_k} \lambda_k / \mu_i \leq \sum_j w_{ij} \cdot m_{ij}, \quad \forall i, \quad (20)$$

$$\sum_i m_{ij} / \beta \leq \sum_{s \in S} s_j \phi_s, \quad \forall j, \quad (21)$$



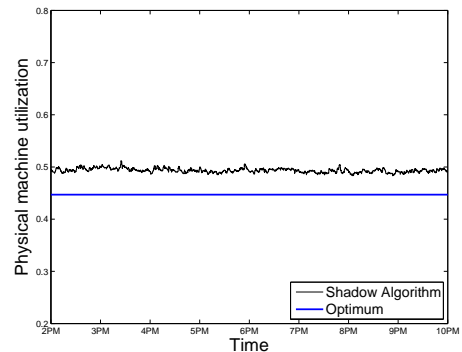
**Figure 3: Datacenter physical machine utilization and optimal number of PMs and VMs with gradual application arrival rate change**

$$\sum_{s \in S} \phi_s = \rho. \quad (22)$$

The problem (18) is same as the problem (2) except the constraints in (20) are more general. We note, however, that the Shadow algorithm, as described in Section 3.2, still applies as is – we simply treat each application request  $i$  as such, regardless of which service it belongs to.

We use an example to show how Shadow algorithms can solve the optimization problem (18). Suppose the cloud supports three types of services, where service 1 employs application 1, 2, and 4, service 2 employs application 2 and 4, and service 3 employs application 3 and 4. Application 4 (e.g., file access application) provides the common functionality that can be used by all three services; assume that application requests of type 4 can share a VM, regardless of which service they belong to. In contrast, assume that application requests of type 2, which belong to different services 1 and 2 *cannot* be “mixed” within same VM; therefore, the Shadow algorithm will “split” the application type 2 into two types 2.1 and 2.2, respectively.

We set the arrival rate of all three services to be 77 reqs/sec. Other parameters, such as  $\{w_{ij}\}$ ,  $\{\mu_i\}$ , remain the same as in the previous experiments. Fig. 4 depicts the PM-utility in comparison to the optimal utility. The optimal PM utility is 44.7%, while the PM utility using Shadow algorithm is 49.7% on average.

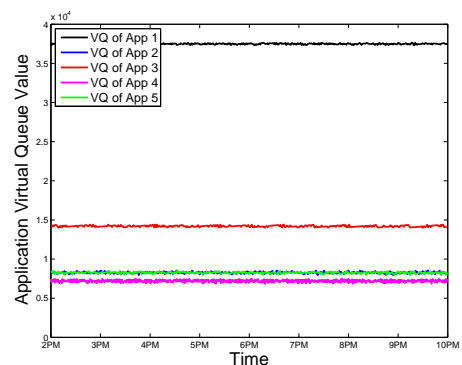


**Figure 4: PM utilization with three services**

Fig. 5 further depicts five virtual queues for Constraint (20) (as compared to four virtual queues in the experiments in Section 6.1). Note that the virtual queue sizes of queue 2 and queue 5 are identical and overlap with each other. The experiment shows how to apply the Shadow algorithm to the service model, and that it performs well.

## 7. EXPEDITED CONVERGENCE

One of the salient features of Shadow algorithm is its capability to *automatically* keep track of the optimal operating point without actively measuring the request arrival rate. Different request rates correspond to different optimal points and different optimal values of  $\{m_{ij}\}$  and  $\{\phi_s\}$ . The amount of PMs assigned to each configuration  $s$  hence needs to be adjusted accordingly so as to minimize the PM utility.



**Figure 5: Application virtual queue sizes with three services**

While Shadow algorithm is very capable in tracking the optimal point, the VM-to-PM mapping algorithm, as described in Section 5.3, also plays a pivotal role in the overall algorithm convergence speed. Below we examine how Shadow performs in face of a sudden application request rate change. We then propose a PM redesignation algorithm that can expedite the convergence.

The experiment parameters are the same as in the experiment in Section 6.2 except that the gradual transition period is replaced by a sudden transition period, i.e., the request rates for application 1, 2, 3, and 4 are suddenly changed at 6PM. The optimal PM utilization is 53.4% from time 2PM to 6PM and 60.3% from time 6PM to 10PM. Fig. 6 depicts the value of  $\phi_s$ , the number of PMs assigned to different configurations, and the PM utilization. The value of  $\phi_s$  is from Shadow algorithm, while the number of PMs are the result of VM-to-PM mapping algorithm. As shown in Fig. 6(a) and Fig. 6(b), both the Shadow and VM-to-PM mapping algorithm react to the sudden rate change quickly. The value of  $\bar{\phi}_s$  and the number of PMs reach new optimal point within tens of minutes.

The PM utilization, however, does suffer from a bump right after 6PM. The PM utilization shoots up to 81%, and then decreases gradually to the new optimal PM utilization point of 60.2%. This is due to the fact that optimal values of  $\{\phi_s\}$  change suddenly at 6PM. Thus PMs originally assigned to a given configuration may not be needed in the new optimal solution. This is obvious as shown in Fig. 3(c). Configuration 18 is assigned about 200 PMs in the time period before 6PM. After the sudden rate change, the number of optimal assigned PMs to configuration 18 is near zero. It takes time for the PMs assigned to the configuration 18 to leave the system, which causes the PM utilization rise during the sharp transition period. Specifically, before 6PM, the request rate for 4 applications are 124 requests/sec. After the 6PM, the request rate for app 1 is 208 req/sec, and that of app 2, 3, and 4 are 35req/sec, 52req/sec, and 52req/sec, respectively. The minimum number of required PMs before the change is 534 PMs. Since this is a sudden change, the PMs that are used to serve the 'old' request profile won't be released immediately at 6PM. Meanwhile, the extra 208 - 124 = 84 req/sec for application 1 are extra new requests that need to be accommodated. The minimal number of PMs needed to accommodate 84 req/sec (for application 1) is 210 PMs. Hence the required number of PMs right after 6PM can be estimated as 534+210=744 PMs, or 74.4%, as shown by the red dotted line in Fig. 6(c). We next describe the *PM-redesignation algorithm* that can mitigate the above PM utilization surge problem.

## 7.1 PM-redesignation algorithm

PM-redesignation algorithm is an additional optional procedure that can be run after a VM departs the system. Its purpose is to speed up the system transition to a new optimal regime, when sudden changes in the input flows and/or system parameters occur.

This algorithm has access to the quantities  $\bar{\phi}_s$ , maintained by the Shadow algorithm. After a new VM (of any class) departs from an s-PM, we denote by  $\hat{s} = (\hat{s}_1, \dots, \hat{s}_j)$  the *actual* configuration of this PM; namely,  $\hat{s}_j$  is the actual

number of  $j$ -VMs this PM contains. Clearly,  $\hat{s} \leq s$ . If  $\hat{s} = 0$ , i.e. it is the empty configuration, we leave this empty PM without any designation, and stop. If  $\hat{s} \neq 0$ , denote by  $S^*(\hat{s}) = \{s' \in S \mid \hat{s} \leq s'\}$ ; denote by  $Y_{s'}$  the current number of  $s'$ -PMs. Compute  $s^* = \arg \min_{s' \in S^*} Y_{s'} / \bar{\phi}_{s'}$ . If  $s^* \neq s$  and  $[Y_{s^*} + 1] / \bar{\phi}_{s^*} \leq [Y_s - 1] / \bar{\phi}_s$ ,  $s^*$  is identified as the PM-redesignation target. The PM is redesignated as an  $s^*$ -PM. In addition, every PM belonging to  $s$  is examined to see if it can be redesignated to  $s^*$ . If Yes, this PM will be redesignated as an  $s^*$ -PM.

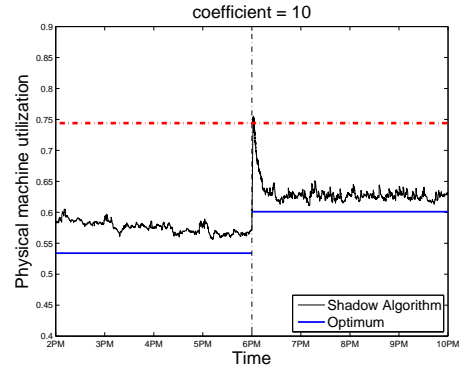


Figure 7: PM utilization with sudden application request rates change

Fig. 7 depicts the PM utilization after running PM-redesignation algorithm. The peak PM-utilization is cut down to 75.2%, close to the required 74.4%. Fig. 8 further depicts the number of redesignated PMs over time. As expected, the majority of PM redesignations (about 200 PM redesignations) happens just after the 6PM. During the stable phase, the number of PM redesignations is minimum.

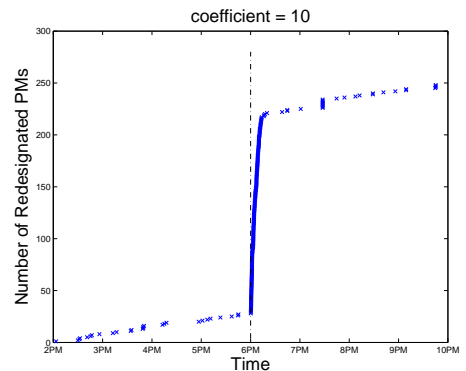


Figure 8: Number of Redesignated PMs with sudden application request rates change

## 8. DISCUSSION AND CONCLUSION

In this paper we presented an optimization framework that adaptively solves the application-to-VM and VM-to-PM assignment problem. Our approach integrates the interest of users in acquiring elastic resources that automatically scale



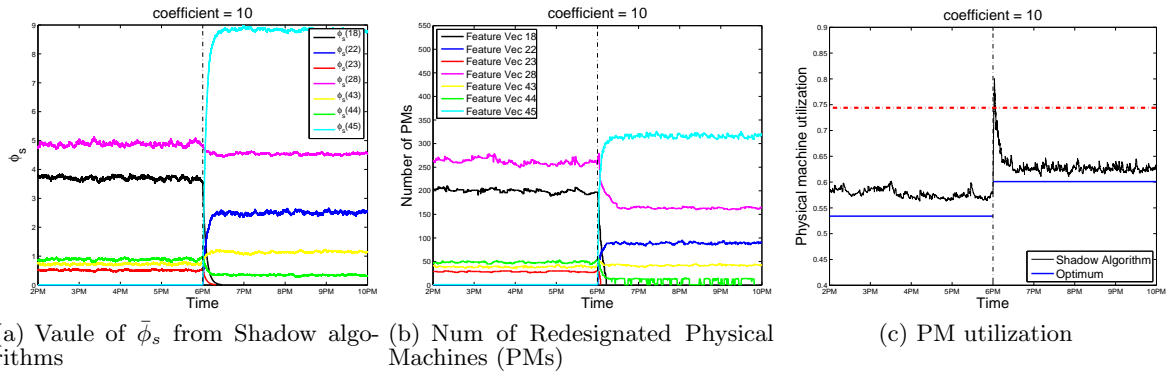


Figure 6: Value of  $\bar{\phi}_s$ , number of PMs, and PM utilization for a suddern request rate change at time 4pm.

with the application demands, with the interest of cloud service providers in packing VMs to efficiently utilize cloud resources. Extensive simulation results verify the proposed algorithms' efficiency, convergence, and adaptivity.

In the framework considered, the cloud customers do not decide on the type and number of VMs to be launched to meet the application demands. This decision is embedded in the optimization framework, and is made by the cloud provider/operator. Therefore, our framework is especially suitable for private clouds, where private cloud operators have the full control over the application-to-VM assignment and VM-to-PM assignment, and can develop models for application resource usage. Private clouds have experienced rapid growth in recent years [5] due in part to security and privacy concerns. For public cloud, additional mechanisms are needed to allow the cloud providers to steer the users' VM selection using, for example, pricing incentives. This is, however, outside the scope of this paper and is a topic of future research.

We assume in our model that the VMs can be brought up and torn down immediately. However, there is typically a boot and configuration delay which may be significant [18]. Our framework can readily incorporate the VM boot delay by inflating the job size of an application request so as to provide a sufficient resource margin. The issue of how to scale resources to reduce the number of VM boots and tear-downs may be worthy of further investigation.

## 9. REFERENCES

- [1] Mansoor Alicherry, T. V. Lakshman. Network aware resource allocation in distributed clouds. *INFOCOM-2012*.
- [2] Auto Scaling - Amazon Web Services, <http://aws.amazon.com/documentation/autoscaling/>
- [3] Amazon EC2 Instance Types, <http://aws.amazon.com/ec2/instance-types/>
- [4] Robert Fourer, David M. Gay, and Brian W. Kernighan, AMPL: A Modeling Language for Mathematical Programming, *Cengage Learning*, 2nd edition.
- [5] Travis Balinas, 110 Cloud Stats & Figures of 2012. Zenoss, Inc.
- [6] 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.
- [7] 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.
- [8] B. Dougherty, J. White, D. Schmidt. Model-driven Auto-scaling of Green Cloud Computing Infrastructure. *J. Future Generation Computer Systems*, Volume 28 Issue 2, February, 2012. pp. 371-378.
- [9] Y. Guo, A. L. Stolyar, A. Walid. Shadow-routing based dynamic algorithms for Virtual Machine placement in a network cloud. *INFOCOM-2013*. <http://ect.bell-labs.com/who/stolyar/publications/gpd-vm-paper-inf.pdf>
- [10] V. Gupta, A. Radovanovic. Online Stochastic Bin Packing. Preprint, 2012. <http://arxiv.org/abs/1211.2687>
- [11] C-L. Hung, Y-C. Hu and K-C. Li. Auto-Scaling Model for Cloud Computing System. *International Journal of Hybrid Information Technology*, Vol. 5, No. 2, April, 2012.
- [12] J.W. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang. Joint VM Placement and Routing for Data Center Traffic Engineering. *INFOCOM-2012*.
- [13] T. Lorido-Botran, J. Miguel-Alonso and J. Lozano. Auto-scaling Techniques for Elastic Applications in Cloud Environments. Technical Report EHU-KAT-IK-09-12, University of the Basque Country, 2012.
- [14] S.T. Maguluri, R. Srikant, L.Ying. Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters. *INFOCOM-2012*.
- [15] S.T. Maguluri, R. Srikant. Scheduling Jobs with Unknown Duration in Clouds. *INFOCOM-2013*.
- [16] M. Mao, J. Li and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. *Grid Computing (GRID)*, 2010.
- [17] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. *INFOCOM-2010*.
- [18] M. Mao and M. Humphrey. A Performance Study on the VM Startup Time in the Cloud. *IEEE Fifth International Conference on Cloud Computing, 2012*.
- [19] X. Meng, C. Isci, J. Kephart, L. Zhang, and E.

- Boulillet. Efficient resource provisioning in compute clouds via VM multiplexing. *Proc. ICAC, 2010*.
- [20] A. L. Stolyar. Maximizing Queueing Network Utility subject to Stability: Greedy Primal-Dual Algorithm. *Queueing Systems*, Vol. 50, pp. 401-457 (2005).
  - [21] 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.
  - [22] A. L. Stolyar. An infinite server system with general packing constraints. *Operations Research*, to appear. <http://arxiv.org/abs/1205.4271>
  - [23] A. L. Stolyar, Y. Zhong. A large-scale service system with packing constraints: Minimizing the number of occupied servers. *SIGMETRICS-2013*. <http://arxiv.org/abs/1212.0875>
  - [24] A. L. Stolyar, Y. Zhong. Asymptotic optimality of a greedy randomized algorithm in a large-scale service system with general packing constraints. 2013, submitted. <http://arxiv.org/abs/1306.4991>