Availability-Aware Energy-Efficient Virtual Machine Placement Algorithm

Zhouhan Yang[1], Liu Liu[2], Sanjukta Das[3], Ram Ramesh[3], Anna Ye Du[3], and Chunming Qiao[1]
[1] Department of Computer Science and Engineering, University at Buffalo (SUNY), Buffalo, 14228, USA
[2] Key Lab of Optical Fiber Sensing and Communications, Ministry of Education, UESTC, Chengdu, 610000 China
[3] Management Science and Systems Department, University at Buffalo (SUNY), Buffalo, 14228, USA
Email: {zhouhany, sdsmith4, rramesh, yedu, qiao}@buffalo.edu; liuliu27@uestc.edu.cn

Abstract —Availability, as a part of Service Level Agreement (SLA), is a critically important issue in cloud services, which are affected by server or network failures in datacenters. Cloud service providers seek to not only fulfill the SLA, but also simultaneously minimize their operating costs, which are dominated by the energy consumption. In order to minimize the impact of a server/switch failure on a single application, one spread out the VMs for the application across different racks. Although a higher availability can be achieved, the power consumption may increase significantly. In this paper, we develop a variance-based metric to measure the risk of either over provisioning or under provisioning availability by means of VM placement. We then propose algorithms to place VMs in online and offline manners, respectively. These algorithms aim to strike a balance between maximizing the availability and minimizing the energy, in order to reduce the operating cost for the service providers.

Index Terms—Cloud, availability, energy consumption, risk, dual

I. Introduction

As cloud computing technologies continue to advance, clients increasingly rent Virtual Machines (VM) on demand from Service Providers (SPs). Such a "pay as you go" model serves as a more economical and flexible alternative to retaining in-house IT infrastructure by saving on capital and operating cost. However, availability of these rented VMs become critically important, due to frequent failures within a datacenter, and several recent protracted service outages has already caused billions of dollars in lost productivities and revenues for these clients, who are expected to demand a more stringent Service Level Agreement (SLA) to meet their availability requirements. For SPs, failure to fulfill the availability requirement SLA may lead to loss of reputation in addition to SLA-specified penalties.

Availability typically refers to the uptime percentage of a service in a given (often finite) duration. We focus on cases where an application requires a minimum number of VMs (e.g. S) to be up and running for a high percentage of time (e.g. 99.9%). In order not to violate the availability requirement in the SLA, an SP needs to allocate a sufficient number of redundant VMs and place them strategically across different servers/racks to deal with correlated failures.

More specifically, in order to minimize the impact of the failures of servers and Top-of-Rack (ToR) switches on an application, one would like to spread out the VMs for the application to as many different servers/racks as possible. Of course doing so may result in significant energy consumption and is against the principle of most of the previous work on server/rack consolidation which ignores the impact of failures on the availability requirement.

In this paper, we approach the problem from the perspectives of a SP, who would like to reduce its total cost, dominated by the server cost and the energy cost, while providing a good availability guarantee by e.g., allocating a minimal number of redundant VMs in order to minimize the number of servers it needs to use to run all the applications, which helps to significantly reduce the capital and operating expenses of the SP. In particular, we will focus on availability-aware and energy-efficient VM placement, as a good VM placement can not only achieve a low risk of SLA violation without requiring a large number of redundant servers by spreading the VMs for one application across different servers/racks, but also reduce the energy consumption by appropriately consolidating the VMs belonging to different applications onto as a few server/racks as possible without increasing the risk of SLA violation.

Previous work has not studied VM placement in order to reduce energy consumption while considering the risk of violating the availability requirement of the applications due to possible server/switch failures. For example, a large body of papers have studied the VM placement problem with multiple objectives including energy saving [1], [2], bandwidth and computing resource utilization [3], [4], QoS or a combination of multiple objectives [2], [4], [5], but none of them specifically considered the impact of server/switch failures. A few studies have sought to improve fault tolerance by placing the duplicate data in close proximity to the computing

Manuscript received March 31, 2015; revised September 3, 2015.
This work was supported by the NSF under Grant No. CSR-1409809 and Google's Research Award HDTRA1-09-1-0032.
Corresponding author email: zhouhany@buffalo.edu.
doi:10.12720/jcm.10.9.647-658

resources. Both [6] and [7] proposed VM migration strategies assuming inter-rack server failures. The most closely related work is [8], which considered the impact of ToR switch failures, but focused on how to spread out the VMs to minimize an overall cost which includes the bandwidth cost, and a cost associated with VM failures. Our work in this paper differs from all the previous work by considering not only concurrent server and ToR switch failures, but also the minimization of an overall cost that is based on the energy consumption and the risk of violating the availability requirements (which, as mentioned earlier, is related to the server cost). Such an overall cost is more inline with the SP's perspective than any other existing cost functions in previous VM placement studies, as each of the energy cost and server cost accounts for about 50% of the entire cost of a datacenter [9].

Our contribution of this paper can be summarized as below:

- We propose a novel concept to characterize the risk of violating the availability requirement of applications in the situation where we have both ToR switch failures and individual server failures. We mathematically define the risk by using expectation and standard deviation of the number of VMs available to a given application.
- Based on a popular energy consumption model, we also establish a mathematical model combining the objectives to minimize the total energy cost and the risk of violating the availability requirement.
- Based on the mathematical model, we establish the dual model of the primal problem. We come up with an offline algorithm and an online algorithm to place the VMs efficiently. We also develop an online algorithm based on the primary-dual model, and prove that the online solution and optimal offline solution are within a bounded ratio to each other.

The paper is organized as follow. The availability-aware energy efficient VM placement problem is first described in Section II. A formal problem definition is presented in Section III, where we first define the expectation and standard deviation of the number of VMs available to an application in the presence of concurrent failures of a server and a ToR switch, and then characterize the risk of violating the availability requirement of an application accordingly. We also propose an overall cost inline with the SP's perspective that considers the minimization of both risk and energy consumption, we develop the objective function of the optimization problem under consideration. In Section IV and Section V, two heuristic algorithms, one offline and one online, to place VMs are presented. In Section VI, we establish the dual model and the online dual algorithm based on the model. In Section VII, we present the simulation results to compare the performance of the algorithms with that in [3]. Section VIII presents the conclusion and future work.

II. Availability-Aware Problem

The topology of datacenter in this paper is as Fig. 1 shows. There are three layers of switches: core switch, aggregate switch and ToR switch. All switches, racks and servers are identical. We consider two types of failures in this paper: the physical server failure and ToR switch failure. The former will cause all VMs mapped on the failed server also fail, while the latter results the involved VMs being rendered inaccessible. In any single slot, we assume that there will be a ToR switch failure, along with at most one physical server failure.

We use $m \in \mathcal{M} = \{1,2,\ldots M\}$ to represent racks, $n \in \mathcal{N} = \{1,2,\ldots N\}$ to represent the *nth* server in each rack, and $i \in \mathcal{I} = \{1,2,\ldots I\}$ stands for application set in the system. We consider a VM is available if it functions correctly and can be accessed by other VMs as needed. We use $\mu(p,i)$ to denote the expected number of available VMs belonging to application i under placement p when both ToR and physical server may fail. Let $Var(p,i)$ be the standard deviation of the number of available VMs belonging to i under placement p. The actual available VM number will vary from $\mu(p,i)-Var(p,i)$ to $\mu(p,i)+Var(p,i)$. Note that a significant fluctuation in the number of available VMs indicate high instability of the VM provisioning schemes, in that if the application requires $\mu(p,i)$ VM to be available, then a larger value of $\mu(p,i)+Var(p,i)$ means over-provisioning and a higher cost, while a smaller value of $\mu(p,i)-Var(p,i)$ means under-provisioning and increased SLA violation risk. Therefore, it is desirable to minimize the fluctuation range, which in turn lower risk of over provisioning and SLA violation in terms of availability.

Different number of VMs requested by applications will lead to the difference in the value of $\mu(p,i)$ and $Var(p,i)$. Therefore, in this paper, we use the metrics $risk(p,i) = \dfrac{Var(p,i)}{\mu(p,i)}$, which denotes the normalized standard deviation of the number of available VMs, to identify the risk faced by application i under placement p.

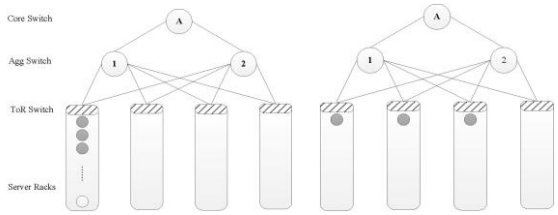
Suppose the failure probabilities of each rack and server are P_r and P_s, respectively. Then $\mu(p,i)$ can be obtained by computing the expected number of VMs available to application i under the following two possible events: (1) when the system have zero server failure and one ToR switch failure (defined as event B1); and (2) when the system has one server failure and one ToR switch failure (defined as event B2). We call the event B when the system has one server failure or one server failure along with a ToR switch failure. The

probabilities of event B, B1 and B2 denoted by $P(B)$, $P(B1)$ and $P(B2)$ respectively are:

$$\begin{aligned} P(B) &= P(B1)+P(B2) \\ P(B1) &= C_{MN}^{0} P_s^{0}(1-P_s)^{MN} \\ P(B1) &= C_{MN}^{1} P_s(1-P_s)^{MN-1} \end{aligned} \quad (1)$$

In addition, we use the metrics E to represent the total energy cost of the whole system. Based on the commonly used linear model [10], 11], the energy consumption of ToRs and physical servers can be separated into two parts: the static part produced once they are switching on, and the dynamic part varying depending on their workload.

To better understand the concepts, let us look at an example in Fig. 1. In this example, we have an application requests for three VMs. In Fig. 1, we placed the VMs to three servers in one rack. In Fig. 1(b), we place the three VMs to three different racks. Assuming P_s is 0.02, P_r is 0.02. For Placement one, by the definition of μ and Var, we can get μ is 2.16003 while Var is 1.1459. For placement two, the result of μ is 2.1, and Var is 0.4426. We can see from the example that, a wide distribution of VMs among different racks can much lower the risk computed by $\frac{Var}{\mu}$.



(a) (b)

Fig. 1. Example of VM placement.

TABLE I: DEFINITION OF NOTIONS

Notation	Definitation
A_i	The *ith* application in I, $1 \le i \le I$
$A(i,p,m,n)$	The number of VMs belonging to A_i deployed at server m in rack n
S_i	Total number of VMs required by application i
$V_{i,j}$	The *jth* VM belongs to application i, $1 \le j \le S_i$
V_i	The set of VMs belongs to application i, $\|V_i\| = S_i$
R_i	The size of the VM belonging to application i
$C(m,n)$	Total capacity of server n in rack m, $1 \le n \le N$, $1 \le m \le M$

III. PROBLEM FORMULATION

We start the problem formulation by defining several notations in Table I. Note that the formulation below is applicable to the general case where an application may require several VMs of different sizes $R_{i,j}$, (e.g., small, medium or large instances), and different servers may also have different capacities $C(m,n)$. In later sections, we will however describe algorithms assuming that all the VMs required by each application i have the same size (i.e, $R_i = R_{i,j} = R_{i,k}$) for any i, j and k. And all the servers have the same capacity C (i.e., $C = C(m,n) = C(p,q)$) for any m, n, p and q.

We use notation X_p^i to represent the placement of VM for each application.

$$X_p^i = \begin{cases} 1 \text{ if VMs belonging to application } i \\ \quad \text{are placed by configuration } p \\ 0 \quad \text{Otherwise} \end{cases} \quad (2)$$

For application i requiring S_i VMs, we use $\mu_{i,0}$ to denote the expected number of VMs that are available when distributing S_i VMs to as many different servers/racks (up to S_i racks) as possible, also let $Var_{i,0}$ denote the standard deviation of available VMs under this placement. As described before, we only have at most one rack failure together with a server failure in a single slot, such a VM placement should yield the smallest $Var_{i,0}$ and largest $\mu_{i,0}$.Hence, we define $risk_{i,0} = \frac{Var_{i,0}}{\mu_{i,0}}$, which should be close to the smallest risk (in terms of either SLA violation or over achieving the availability beyond the requirement) application i can possibly suffering. Note that with the lowest degree of server/rack consolidation, the system's overall risk computed by $risk_0 = \sum_i risk_{i,0}$ is close to the minimum. However, the energy consumption under this VM placement is maximized.

On the other hand, by consolidating all the VMs to as few racks as possible, we can achieve an overall energy cost E_0 close to the minimum energy usage. However, the risk as defined above, including that of SLA violation, under this placement is close to the maximum. This means that more backup servers are needed to satisfy the SLA requirement, which leads to the increase in the overall server cost and energy consumption. Therefore, there is a tradeoff to be carefully considered between

decreasing the average risks of applications and reducing the energy consumption when placing VMs in datacenters.

In this paper, we develop efficient algorithms to address this issue by carefully choosing the VM placement solutions.

Based on the tradeoff described above, we formulate the objective function as a multiple-objective problem given by:

$$\min\left(\frac{\sum_i\sum_{p\in P} risk(p,i)X_p^i - risk_0}{risk_0} + \theta\frac{\sum_i\sum_{p\in P} E(p,i)X_p^i - E_0}{E_0}\right) \quad (3)$$

where $E(p,i)$ denotes the energy consumption of application i under placement p (which will be Full-computed in Eq. (11) later), and $\sum_{p\in P}\sum_i E(p,i)$ is the overall energy cost of the system.

In addition, θ is a parameter that can be used to allow a flexible trade off between ***E*** and ***risk***, by setting θ to different value, assuming a value less than one if risk reduction is more important and larger than one if energy reduction is more important.

The objective function can be converted to:

$$\min\left(E_0\sum_i\sum_{p\in P} risk(p,i)X_p^i + \theta risk_0\sum_i\sum_{p\in P} E(p,i)X_p^i\right) \quad (4)$$

For each server, we should constrain that the total size of the VMs mapped on it can't exceed its capacity:

$$\sum_i A(i,p^i,m,n)R_i X_p^i \le c(m,n) \quad (5)$$

And for each VM, we can only place it on one server, which can be given as:

$$\sum_{p\in P} X_p^i = 1 \quad \forall i \quad (6)$$

We use $\sigma_{i,m}^B$, $\sigma_{i,m}^{B1}$ and $\sigma_{i,m}^{B2}$ to represent the expected number of unavailable VMs in V_i due to events B, B1 and B2, respectively, when the failed rack is m. Based on the definition of these events in Section II, we have:

$$\sigma_{i,m}^B = \sigma_{i,m}^{B1} + \sigma_{i,m}^{B2} \quad (7)$$

where $P(B1\mid B) = \frac{P(B\cdot B1)}{P(B)} = \frac{P(B1)}{P(B)}$, and $P(B)$ and $P(B1)$ are defined in Eq. (1).

For Event B2, which is defined as when the system has one ToR failure and one server failure, we consider two possible situations: The failed server is inside the failed rack m, or the failed server is outside rack m. The expected number of unavailable VMs due to event B2 (represented by $\sigma_{i,m}^{B2}$), should be the number of VMs mapped on failed rack m, plus the expected number of VMs that are unavailable due to a server failure outside rack m. It can be obtained by:

$$\begin{aligned}\sigma_{i,m}^{B2} &= \sum_n\sum_j X_{i,j}^{m,n} P(B2\mid B) \\ &\quad + \frac{M-1}{M}\sum_m\sum_n\sum_j X_{i,j}^{m,n} P_s P(B2\mid B)\end{aligned}$$

where $P(B2\mid B) = \frac{P(B\cdot B2)}{P(B)} = \frac{P(B2)}{P(B)}$. And $P(B2)$ is also defined in Eq. (1).

The expected number of available VMs in V_i can then be computed by subtracting S_i from the expected number of unavailable VMs due to event B as follows.

$$\begin{aligned}\mu_i &= S_i - \sum_m P_r\sigma_{i,m}^B \\ &= S_i - \sum_m P_r\sigma_{i,m}^{B1} - \sum_m P_r\sigma_{i,m}^{B2}\end{aligned}$$

Before deriving Var_i, we first define $N_{i,m}$, which stands for the expected number of available VMs in V_i when a specific rack m fails, as below:

$$\begin{aligned}N_{i,m} &= S_i - \sigma_{i,m}^B \\ &= S_i - \sigma_{i,m}^{B1} - \sigma_{i,m}^{B2}\end{aligned}$$

The Var_i can be computed based on μ_i and $N_{i,m}$ using the definition of standard deviation as follows.

$$Var_i = \sqrt{\frac{1}{|M|}\sum_m (N_{i,m} - \mu_i)^2} \quad (8)$$

To establish the energy model, we introduce two notations $Z^{m,n}$ and Z^m to indicate whether a server and a rack respectively, is active or not as follows:

$$Z^{m,n} = \begin{cases} 1 & \text{if server } n \text{ belonging to rack } m \\ & \text{is active} \\ 0 & \text{otherwise}\end{cases} \quad (9)$$

$$Z^m = \begin{cases} 1 & \text{if rack } m \text{ is active} \\ 0 & \text{otherwise}\end{cases} \quad (10)$$

Note that a server and a rack are considered active when it has at least one VM loaded on it. We now compute the total energy consumption $\sum_i\sum_{p\in P} E(p,i)$ in Eq. (3) as follows: the overall energy is from two parts: the racks and the servers. We build the model based on the most common linear model [10], [11]. As mentioned earlier, rack and server's energy usage consists of static part and dynamic part. We use e_0^R and e_0^S to denote the static energy consumption of active racks and servers respectively. The dynamic power consumption of a rack varies depending on the number of active ports, which is equal to the number of active servers in it. Let e_1^R be the maximum dynamic power consumption of a rack. And e_{max} be the maximum dynamic power consumption of a

server, which can be reached when the server's load is 100%. The energy usage of the system can be obtained by the following equations:

$$\sum_i \sum_{p \in P} E(p,i) = E_{rack} + E_{server} \quad (11)$$

where

$$E_{rack} = \sum_m (e_0^R Z^m + e_1^R \frac{\sum_n Z^{m,n}}{N}) \quad (12)$$

$$E_{server} = \sum_m \sum_n (Z^{m,n} e_0^S + e_{max} \frac{\sum_i \sum_j X_{i,j}^{m,n} R_{i,j}}{C^{m,n}}) \quad (13)$$

IV. Offline VM Placement Algorithm

As the optimization problem defined above is NP-hard, we first describe a heuristic offline algorithm called "Packing Then Distribute" (PTD) to solve the problem.

As in any offline algorithm, we assume that a set of application requests are given. As mentioned earlier, we will assume that all the VMs required by one application i has the same size R_i, although $R_i \neq R_j$ for $i \neq j$. Without loss of generality, let $R_1 \leq R_2, \cdots \leq R_I$, and denote the corresponding set of VMs by $V_1, V_2 \cdots V_I$, where $|V_i| = S_i$ is the number of VMs requested by application i.

The offline PTD algorithm has two phases: VM packing and VM distribution, respectively. In the first phase, it will try to place the VMs in set V_1 first, followed V_2 and so on. For each application, it assigns its VMs into as few servers/racks as possible to minimize the energy consumption. This can be accomplished by e.g. the first-fit server selection strategy, which tries to place the next requested VM in the first server in the first rack as long as the server has a sufficient capacity left, and otherwise, it will try the next serve in the same rack, and eventually if no server in the same rack can be used, the algorithm will try the (server in the) next rack. Other server selection strategies that results in a maximum server/rack consolidation can also be used. After all the VMs in V_I for the last application have been placed, the packing phase ends, and the algorithm calculates the corresponding energy consumption as E_0, which should be close to the minimum needed due to the nature of the VM packing strategy used so far.

In the second phase of PTD, the algorithm will try to distribute the VMs in V_i among different servers/racks in order to reduce the risk for application i due to either server or ToR switch failures, and in turn, reduce the overall risk as well. This is accomplished by changing the initial placement of some VMs in V_i from a more consolidated server/rack to less consolidated ones. This takes multiple iterations, and in each iteration, the algorithm changes the placement of one VM. This will result in a possibly lower $risk_i$, but a higher overall energy cost E. The algorithm recalculates the value of the objective function in Eq. (3), and will terminate if after the current iteration, this value doesn't differ from the value of the previous iteration by more than a pre-determined threshold.

Below, we describe how to determine the "source" and "destination" in each iteration during which the algorithm makes change the initial placement of a VM.

Finding the source: for each "active" rack m, and each application i whose VMs have been placed in this rack, we count the number of VMs in V_i and denote by $S_i' \leq S_i$, and choose the maximum S_i' over all i, and use this maximum number, denoted by D_m to represent the degree of consolidation of the rack m. The algorithm then randomly chooses a VM currently placed in the rack having the maximum D_m and moves the VM to a different rack.

Finding the destination: Suppose the VM to be moved belongs to application i, that is $\in V_i$. The algorithm first chooses a rack from all active racks whose S_i' is zero or the smallest. Within this rack, it chooses from all the active servers with sufficient capacity left the one that currently host zero or the least number of VMs $\in V_i$. If no such active server can be found, the algorithm considers two strategies: one is to power up an inactive server within this rack to host this VM, and the other is to go to another active rack having the second smallest S_i', and choose an active server in the same way as what has been described above. If it can succeed in placing the VM either way, the algorithm will compute the objective function in Eq. (3) corresponding to these two strategies and go with the better approach. On the other hand, if none of the above strategies works, since none of the servers in any active rack has sufficient capacity to accommodate the VM, the algorithm will eventually power up an inactive rack and a server within that rack to host the VM.

The pseudo-code of the algorithm is as follows:

Algorithm 1 Offine Packing Then Distribution: PTD

1: Input: V : the set of VMs; threshold

2: Output: T : The placement of VMs;

3: Let $V = \{V_1, V_2, \ldots, V_I\}$ be the set of VMs sorted in ascending order of R_i.

4: Place VMs of V_i from V onto a physical server sequentially;

5: while difference in the value Eq. (3) between this iteration and previous iteration $\geq threshold$

6: choose the VM j to move out as described in **Finding the source**;
7: choose the server n VM j move to as described in **Finding the destination**, move VM j to server n;
8: end while
9: Return: The placement T;

Remarks: Intuitively, VM packing reduces energy consumption but increases risk, while VM distribution does just the opposite. The overall objective in Eq. (3) is minimized at some degree of server/rack consolidation with some degree of risks. The above PTD algorithm starts at one extreme of having the highest degree of server/rack consolidation and risk, and then tries to move towards the middle (optimal) point by reducing the degree of server/rack consolidation and risk. One can envision another heuristic algorithm, called "distribute then pack" or DTP, which starts at the other extreme of having the lowest degree of server/rack consolidation and risk, and then tries to move towards the middle (optimal) point by increasing the degree of server/rack consolidation and risk. Similarly, one can also envision another heuristic which starts with a random VM placement (somewhere in between the two extremes) and then uses Tabu-search to find the optimal point. Due to space limitation, these and other offline heuristics will be omitted from this paper. Instead, we will describe an "online" algorithm that could also be used to process a given set of application requests as in the offline case.

V. Online VM Placement Algorithm

In this section, we describe an online algorithm which is useful when application requests come one at a time, and each time, we need to decide VM placement for the current request based on the current status of the servers/racks, without any knowledge about future application requests. We will focus on an online algorithm that does not involve any reconfiguration of the existing VMs. That is, once the VMs for an application are placed, no changes to their placement will be made (unless the application completes).

Our online algorithm works in two phases as follows. In the first phase, for application i requiring S_i VMs, we first estimate the number of servers needed for this application, denoted by M_i, to host these S_i VMs. More specifically, we estimate the frequency distribution (or in general, probabilistic distribution) of the ratio $Ratio_x = \frac{M_x}{S_x} \le 1$ for any application x. Once such a distribution is known, then when a request for S_i VMs arrives, the online algorithm will first generate a random number and then use it to determine the appropriate ratio to apply based on the known distribution and finally, determine the number of servers needed M_i.

The above estimation is based on statistically analyzing (as well as learning) the (ideal) relationship between M_j and S_j for a large number application j's. These application requests can either be based on the actual ones we have processed, or are synthesized (or generated via simulation). The basic idea is to use these requests as an input to an optimal or near optimal offline algorithm such as the PTD described above in order to find out the $ratio_i$ used by such algorithm.

In our experiments, we have generated six types of application requests, each including 5,000 requests for a total of 30,000 application requests. A request in Type $0 \le t \le 5$ needs $10+t$ VMs (the number of VMs requested thus varies from 10 to 15). The goal of this exercise is to obtain, for each type of requests, what is the frequency (or in general probabilistic) distribution of the ratio of the number of servers. We run our PTD algorithm using all the 30,000 requests as input. These requests are processed in the order dictated by the algorithm, and not by their types. At the end of the run, we found that the PTD algorithm assigns 5 servers for 14 Type 2 requests, 6 servers for 55 Type 1 requests, and 10 servers for 1083 Type 1 requests, etc.

Fig. 2 shows the histogram for three types of application requests obtained by running our PTD algorithm. As can be seen, the trend is the same for each type.

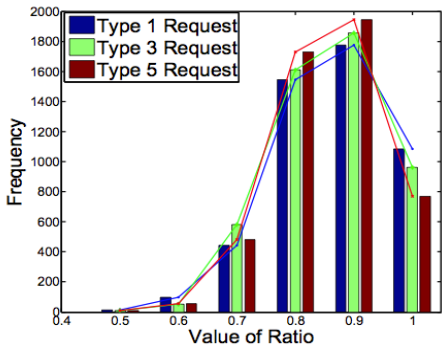


Fig. 2. Note how the caption is centered in the column.

The second phase of our online algorithm starts once M_i is determined. The basic idea is to try to find M_i active servers in as many (up to M_i) different racks as possible, and then evenly distribute the S_i VMs among these M_i servers as evenly as possible (by using e.g., a round-robin assignment strategy). This is to minimize the risk without affecting the overall energy consumption (given the energy consumption model we used). If there are only $h < M_i$ active servers, we will choose from two possible placement strategies as following: the first is distributing them in these h servers as evenly as possible and the other is to power up one or more currently inactive servers in different racks. The second strategy may result in a higher energy cost and a lower risk. The algorithm will compare values of the objective function in Eq. (3) corresponding to these two strategies, and choose the better one.

The pseudo-code of the online algorithm is described as Algorithm 2

Algorithm 2 Online Algorithm

1: Input: A: the set of applications to be mapped;
2: Output: T: The placement of VMs;
3: while $i \le I$ do
4: For $A_i \in A$, using frequency distribution of ratio to determine the number of servers needed to place VMs by $M_i = Ratio_i S_i$;
5 if a server j is active and has enough capacity left to accommodate the VMs, put it into set C, find all such servers;
6: If $|C| \ge M_i$ then
7: Evenly place the VMs to the servers from set C that locates in as many as possible racks, $i = i+1$;
8: else
9: Compute the value of Eq.(3) $\rightarrow e$ for two strategies: Distribute VMs in these |C| servers as evenly as possible; or power up one or more currently inactive servers in different racks. Choose the one can minimize the value $i = i+1$;
10: end if
11: end while
12: **Return**: The placement T;

VI. Dual Online Algorithm

To develop an online approximation algorithm, we first take a linear relation on the condition $X_p^i \in (0,1)$, and convert it to $X_p^i \ge 0$. Then the dual problem can be described as:

$$\max -\sum_i \lambda_i - \sum_{m,n} c(m,n)\delta(m,n) \tag{14}$$

which can be converted eq. (14) to:

$$\min \sum_i \lambda_i + \sum_{m,n} c(m,n)\delta(m,n) \tag{15}$$

with the constraints:

$$E_0 risk(p,i) + \theta risk_0 E(p,i) + \lambda_i + \sum_{m,n} A(i,p^i,m,n) R_i \delta(m,n) \ge 0 \tag{16}$$

$$\lambda_i \in R \quad \forall i \tag{17}$$

$$\delta(m,n) \ge 0 \tag{18}$$

It's obvious that when $\lambda_i \ge 0$, the dual constrain can always be satisfied. Since the inequality for λ_i holds for all mapping P, we can set

$$\begin{aligned} \lambda_i &= \max(-E_0 risk(p,i) - \theta risk_0 E(p,i) \\ &\quad - \sum_{m,n} A(i,p,m,n) R_i \delta(m,n)) \\ &= -\min(E_0 risk(p,i) + \theta risk_0 E(p,i) \\ &\quad + \sum_{m,n} A(i,p,m,n) R_i \delta(m,n)) \end{aligned} \tag{19}$$

According to the weak dual theory, any feasible solution associated with the dual problem is a lower bound to the primal optimal solution. Based on this conclusion, we develop an efficient online algorithm to be used when we don't have any information about the application requests arrived in the future. Furthermore we can prove that the performance obtained through our proposed online algorithm will be at worst no more than r times of the optimal solution.

For each arriving application request, the dual online algorithm computes a primal solution and a corresponding dual solution. As any feasible dual solution produces a lower bound of the optimal offline solution, we can show that the primal solution is also within a as small as possible factor of the optimal offline solution if we carefully design the algorithm. In the proposed dual online algorithm, the dual variable $\delta(m,n)$ is initialized to 0, we design the update of $\delta(m,n)$ in each iteration to ensure the dual to primal ratio is as small as possible. In the following subsections, we first describe the update of the dual variable, then we compute the primal to dual ratio. Finally, the dual online approximation algorithm is given, and the feasibilities of the dual and primal solutions are proved.

A. Primal to Dual Ratio

Let Z^* denote the optimal offline solution of the primal problem, Z_{on} denotes the online solution of the primal problem. Note that $Z^* \le Z_{on}$ since the online solution can at best match the optimal offline solution. There exists a ratio $r \ge 1$ such that:

$$Z_{on} \le rZ^* \tag{20}$$

for all input. So $Z^* \le Z_{on} \le rZ^*$. In general, we would like to get the ratio r as closely as possible to one.

Suppose p^* is the placement that minimizes $E_0 risk(p,i) + \theta risk_0 E(p,i) + \sum_{m,n} A(i,p,m,n) R_i \delta(m,n)$. In the online algorithm, we set $B = \frac{C(m,n)}{A(i,p^*,m,n)R_i}$, $a = (1+\frac{1}{B})^B - 1$. The initial value of $\delta(m,n)$ is 0, when there is an arriving application i, we will update the value of $\delta(m,n)$ to $\delta(m,n)(1+\frac{1}{B}) + \frac{E_0 risk(p^*,i) + \theta risk_0 E(p^*,i)}{A(i,p^*,m,n)R_i} \frac{1}{aB}$.

Let $\triangle\delta(m,n)$ denote the increment in $\delta(m,n)$. Let ε stand for the increment in the dual objective when we currently have an arriving application i. We have:

$$\varepsilon = \sum_{m,n} \triangle\delta(m,n)c(m,n) + \lambda_i$$
$$= \frac{a-1}{a}(E_0 risk(p^*,i) + \theta risk_0 E(p^*,i))$$
$$\leq OPT$$

where OPT is the optimal solution of the primal problem.

Note that $E_0 risk(p^*,i) + \theta risk_0 E(p^*,i)$ is the increment in the primal objective. As the way $\delta(m,n)$ updates, the dual to primal ratio is $\frac{a-1}{a}$. Furthermore, we have $E_0 risk(p^*,i) + \theta risk_0 E(p^*,i) \leq \frac{a}{a-1} OPT$. So the primal to optimal ratio $r = \frac{a}{a-1}$.

We use $|R(i,m,n)|$ to represent the total size of resources requested by application i on server m in rack n, then

$$a = (1+\frac{1}{B})^B - 1$$
$$= (1+\frac{|R(i,m,n)|}{C(m,n))})^{\frac{C(m,n)}{|R(i,m,n)|}} - 1$$

the value of r depends on the value of $\frac{|R_{i,m,n}|}{C_{m,n})}$. In our simulation, $\frac{|R(i,m,n)|}{C(m,n)}$ is ranging from 10%-30%. Fig. 3 shows the value of the primal online algorithm to optimal ratio r when sizes of VMs range from 10%-30%.

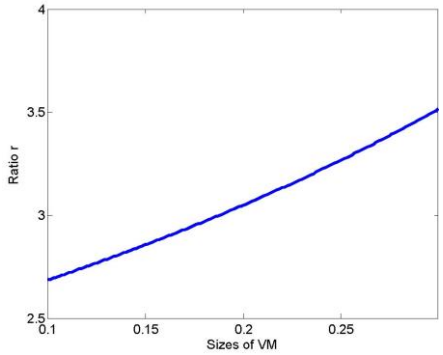


Fig. 3. Primal to optimal ratio.

B. Online Approximation Algorithm

As defined before, for an arriving application request i, we need to determine a mapping $p^* \in P$ such that $p^* = argmin(E_0 risk(p,i) + \theta risk_0 E(p,i) + \sum_{m,n} A(i,p,m,n) R_i \delta(m,n))$ at the first step. We are trying to find the mapping in a heuristic way. We first consolidate the VMs to as less servers/racks as possible, then we start to move the VMs belonging to application i to other servers as the way described in the distribution phrase of PTD offline algorithm. Then we will update the $\delta(m,n)$ in the manner we designed before. The complexity of this algorithm depends on the size of the placement set P.

The online VM allocation algorithm is shown in Algorithm 3.

Algorithm 3 Dual Online VM Allocation Algorithm

1: **Initialize**: $\delta(m,n) = 0$;

2: For current arriving request i, Find a mapping $p^* = argmin(E_0 risk(p,i) + \theta risk_0 E(p,i) + \sum_{m,n} A(i,p,m,n) R_i \delta(m,n))$

3: If the size of VMs mapped on a single server exceed its capacity limitation, find another mapping p^*

4: Place request i at p^*

5: Set

$$\delta(m,n) \leftarrow \delta(m,n)(1+\frac{1}{B}) + \frac{E_0 risk(p^*,i) + \theta risk_0 E(p^*,i)}{A(i,p^*,m,n)R_i} \frac{1}{aB}$$

6 Set

$$\lambda_i = -(E_0 risk(p^*,i) + \theta risk_0 E(p^*,i) + \sum_{m,n} A(i,p^*,m,n) R_i \delta(m,n))$$

C. Primal and Dual Feasibility

A solution is feasible as long as the constrains of the problem can be satisfied. In this subsection, we prove that the primal solution and dual solution obtained by Algorithm 3 are both feasible. Given the way the online algorithm is constructed, when finding a mapping p^*, the size of VMs mapped on a single server will not exceed its capacity limitation, so the primal solution is always feasible. Moreover, in step 2, we always choose the placement p^* that can maximize the value of λ_i, so the dual solution is feasible too. Since $\delta(m,n)$ only increases, the solution will remain feasible subsequently.

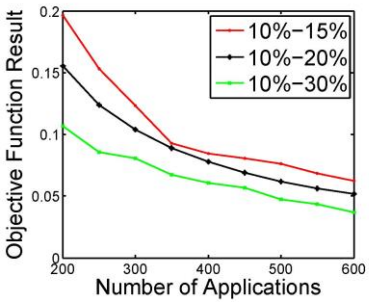
VII. Performance Evaluation

We first implement the offline algorithm, online algorithm and dual online algorithm in simulation, then generate a large number of application requests as input, and finally calculate the risk and energy consumption as well as the overall cost. We will evaluate the percentage increase from the minimum risk $risk_0$ obtained by distributing the VMs among servers/racks to the overall *risk*, and the percentage increase from the minimum system energy consumption E_0 obtained by consolidating the VMs to as less racks/servers as possible, to the overall energy consumption. We compute the overall percentage increase as defined in Eq. (3). We also implement the most relevant FT algorithm from [3], by minimizing one of its cost functions, termed FTC, we can expect the improvement in fault tolerance.

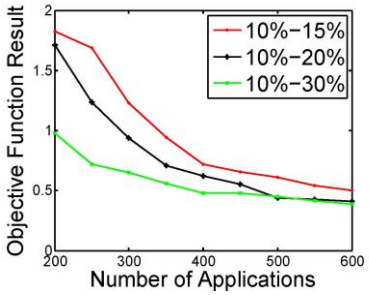
A. Simulation Setting

We did the simulation using the CloudSim [12] simulator. We simulate a datacenter with the same

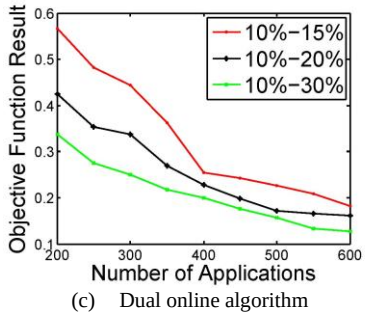
topology as that described in Section II, consisting of 500 racks, each having 100 physical servers. Each server has 2GHz 1-core cpu. The size of each VM required by various applications ranges from 10% to 30% of the resource capacity of each server. When computing the energy consumption by a rack (excluding the power consumed by its servers) using Eq. (16), we set its parameters as follow: e^R_{static} =0.75 and $e^R_{dynamic}$ =0.25, based on the model in [13]. Similarly, When computing the energy consumption by a server using Eq. (17), we set e^S_{static} =0.6 and $e^S_{dynamic}$ =0.4. Note that since a fully loaded server may consume 200W while a rack can consume up to 60W in addition according to [9], 1 unit of rack's power consumption from Eq. (16) needs to be converted to 0.3 units of server's power consumption, when computing the overall energy consumption using Eq. (15). As we think from the SP's perspective, the failure rate of switch and server is known in advance. We set P_r as 0.05 [14] and $P_s = 0.02$.



(a) PDT algorithm



(b) Online algorithm



(c) Dual online algorithm

Fig. 4. Impact of the number of applications and VM sizes on the performance of PTD, online, dual online algorithms.

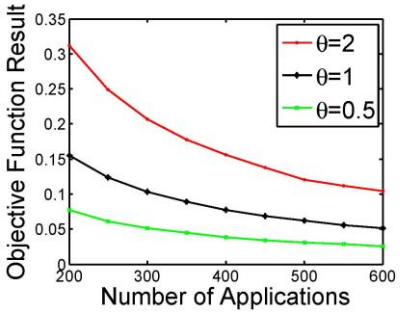
B. Simulation Result

Fig. 4 plots the simulation results from the PTD algorithm, online algorithm and the dual online algorithm. Three subcases are also shown where the requested VM size varies from 10%-15%, 10%-20% and 10%-30%, respectively. The figure shows that the percentage increase in the overall cost (assuming $\theta = 1$) reduces with the number of application requests.

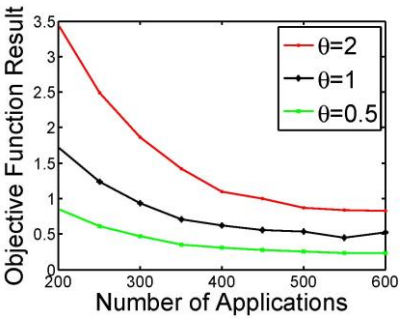
Such a reduction is mainly due to the fact that with more applications, the minimum energy consumption (E_0) needed is larger because of more number of VMs, hence the percentage increase calculated using Eq. (3) is smaller, while the *risk* is not much affected by the number of applications.

The reason that percentage increase in the overall cost is lower with a larger variation in the VM size requested is that, larger VM size will also result in a larger E_0, which will lead to a smaller percentage increment. Also, as the VM size is larger, VMs are more likely to distribute among the datacenter as a server can't host that many numbers of VMs due to the capacity limitation. This will result in lower increment of *risk* as well.

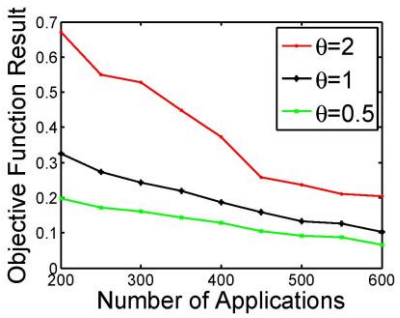
The results also suggest that the offline algorithm results in a lower overall cost increment per application than the online algorithm and dual online algorithm, which is expected.



(a) PDT algorithm



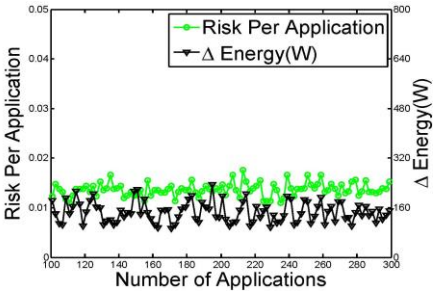
(b) Online algorithm



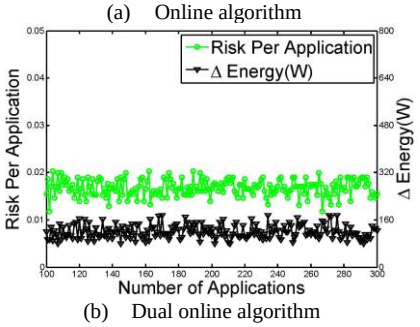
(c) Dual online algorithm

Fig. 5. Impact of θ value on PTD, online and dual online algorithms.

The impact of θ on the percentage increase in the overall cost using the algorithms is shown in Fig. 5 where the VM size varies from 10%-20%. As can be seen, when θ varies from 0.5 to 2, the differences in the percentage increase in the overall cost using the online algorithm is more pronounced than using the offline algorithm and dual online algorithm. In particular, when $\theta = 2$, the percentage increase in overall cost using the online algorithm reduces by 75% when the number of application requests increases from 200 to 600. This indicates that the offline algorithm and the dual online algorithm are particularly effective when reducing the energy consumption (as well as the risk) no matter how many applications are there, while pointing to the potential in reducing the energy consumption using the online algorithm when the number of applications is small. This is somewhat expected as the online algorithm spreads the VMs for an applications out among many servers/racks by modeling after what an offline algorithm would do when the number of applications is large. When the number of applications is small, such a strategy may end up consuming too much energy thus leading to a higher percentage increase in the overall cost.



(a) Online algorithm



(b) Dual online algorithm

Fig. 6. The risk value per application and increment in system's energy cost using the online algorithm and dual online algorithm.

To corroborate the above explanation, we first process 100 application requests using the online and dual online algorithm. We then process 200 more applications, and plot $risk_i = \frac{Var_i}{\mu_i}$ and the (absolute) increment in the system's total energy usage $\Delta Energy$ in watts, which is equal to the overall energy after the application request is accommodated minus the overall energy before the application request is accommodated. As shown in Fig. 6(a) and Fig. 6(b), these two curves fluctuate in a very small range, and are not influenced by the number of applications currently in the system.

We further compare our algorithms with the FT algorithm in [8] in Fig. 7 where the VMs size varies from 10%-20%, and $\theta = 1$.

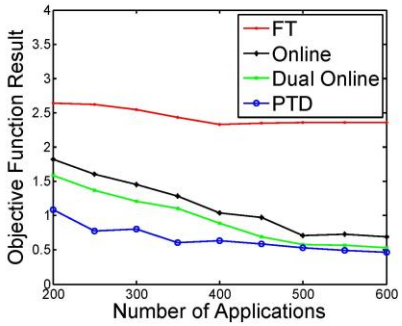


Fig. 7. Comparison result of PTD algorithm, online algorithm and FT algorithm under different number of applications

From Fig. 7, we can see that since the FT algorithm does not pay attention to the energy saving, it results in the highest percentage increase in the overall cost. The dual online algorithm has an improvement about 30% compared to the heuristic online algorithm. The offline PTD algorithm has the best performance, which is as expected.

VIII. Conclusions

In this paper, we have, for the first time, proposed available-aware and energy efficient VM placement algorithms to lower the risk of violating availability requirements of the applications while achieving as low energy consumption as possible. We have proposed and mathematically defined a measure to characterize such a risk taking into consideration of concurrent server and ToR switch failures. We have also proposed and mathematically established a model for the minimization of the overall cost including both the risk and the energy consumption. As the optimization problem is NP-hard, we have proposed two heuristic algorithms (online and offline) shown through simulations that they are quite effective, and also we design an approximation algorithm by using the dual model.

So far, we have assumed that the increase in the dynamic part of the energy consumption follows a linear function. In the future, we will extend our algorithms when such an energy consumption follows a nonlinear function. Load balancing is another factor we can take into consideration when placing the VMs. We will also explore other online algorithms which use a better (e.g., more adaptive) method to estimate the number of servers needed to host the VMs for a given application, and perform re-optimization through e.g., VM migration. Furthermore, to ensure availability guarantee, a Service Provider (SP) needs to jointly decide the number of redundant VMs to be allocated and the placement of all the primary and redundant VMs. It also needs to be able to calculate the availability in a more direct way based on more comprehensive model for the correlated failures inside a datacenter than the way used in this paper to calculate the risk. Finally, the SP needs to do more

accurate market-based cost-benefit analysis in order to design an appropriate SLA which stipulates the levels of availability guarantees, and the corresponding prices to be paid by the clients, and the penalties for SLA violation to be paid by the SP, by considering its CAPEX and OPEX for providing the services.

ACKNOWLEDGEMENTS

This research is supported in part by HDTRA1-09-1-0032, Google's Research Award and NSF CSR-1409809.

REFERENCES

[1] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proc. 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, May 2010, pp. 826–831.

[2] A. Dastjerdi, S. Garg, and R. Buyya, "Qos-aware deployment of network of virtual appliances across multiple clouds," in *Proc. IEEE Third International Conference on Cloud Computing Technology and Science*, Nov. 2011, pp. 415–423.

[3] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, New York, NY, USA: ACM, 2012, pp. 199–210.

[4] J. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffi c engineering," in *INFOCOM, Proceedings IEEE*, March 2012, pp. 2876–2880.

[5] M. Guazzone, C. Anglano, and M. Canonico, "Energy-efficient resource management for cloud computing infrastructures," in *Proc. IEEE Third International Conference on Cloud Computing Technology and Science*, Nov. 2011, pp. 424–431.

[6] M. Melo, P. Maciel, J. Araujo, R. Matos, and C. Araujo, "Availability study on cloud computing environments: Live migration as a rejuvenation mechanism," in *Proc. 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2013, pp. 1–6.

[7] M. Mihailescu, A. Rodriguez, and C. Amza, "Enhancing application robustness in infrastructure-as-a-service clouds," in *Proc. IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops*, June 2011, pp. 146–151.

[8] P. Bod´ık, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in *Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, New York, NY, USA: ACM, 2012, pp. 431–442.

[9] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[10] D. Kliazovich, P. Bouvry, and S. Khan, "Dens: Data center energy-effi cient network-aware scheduling," in *Proc. IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing Green Computing and Communications (GreenCom)*, Dec. 2010, pp. 69–75.

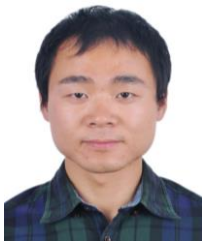
[11] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic rightsizing for power-proportional data centers," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1378–1391, Oct. 2013.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[13] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, "A power benchmarking framework for network devices," in *Proc. 8th International IFIP-TC 6 Networking Conference*, Berlin, Heidelberg: Springer-Verlag, 2009, pp. 795–808.

[14] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proc. ACM SIGCOMM 2011 Conference*, New York, NY, USA: ACM, 2011, pp. 350–361.

Zhouhan Yang is a PhD student in Computer Science and Engineering department, State University of New York at Buffalo. She received her B.S in Computer Science department at University of Science and Technology of China (USTC), Hefei, in 2010. Her research interest is in availability modeling of virtual networks in cloud computing and fault-tolerant virtual server architectures.



Liu Liu is pursuing his Ph.D. degree Communication and Information System at University of Electronic Science and Technology of China. His Research interests include resource management in data centers and cloud computing.

Sanjukta Das received her PhD in operations and information management in 2007 from the University of Connecticut. She is an Associate Professor at SUNY Buffalo in the Department of Management Science and Systems. Her research interests include resource allocation and contract design in cloud computing. Her research has been funded by Google and NSF. She has published extensively in journals such as INFORMS Journal on Computing and Information Systems Research. She serves as an Associate Editor for INFORMS Journal on Computing and as a Coordinating Editor for Information Systems Frontiers. She has served as a Guest Associate Editor for MIS Quarterly. She was a General Co-chair of the INFORMS Conference on Information Systems and Technology 2014 in San Francisco.

Ram Ramesh is Professor and Chair of Management Science & Systems department, School of Management, SUNY at Buffalo. His current research focuses on availability analytics and statistical modeling of cloud infrastructures, cloud market mechanisms and contract structures. He serves as an Editor-in-Chief of Information Systems Frontiers and an Area Editor of INFORMS Journal on Computing for the area "Knowledge Management and Machine Learning". He has published extensively in journals such as Information Systems Research, INFORMS Journal on Computing and IEEE TKDE. His research has been funded by NSF, Air Force Office of Scientific Research, Air Force Research Laboratory, Army Research Institute, Google, Raytheon, Samsung and Westinghouse.

Anna Ye Du received her PhD from SUNY Buffalo in 2010 and is a post-doctoral fellow in the Department of Management Science and Systems. Her research areas include statistical modeling of availability in computing systems, operations and economics of cloud computing markets, and analytics of health IT. Her research works have been widely published in journals such as Information Systems Research, INFORMS Journal on Computing, ACM Transactions on MIS and many more.

Chunming Qiao is Professor and hair of Department of Computer Science and Engineering at SUNY Buffalo. He directs the Lab for Advanced Network Design, Analysis, and Research (LANDER) with current focus on cyber transportation systems, cloud computing, and smartphone systems. He has published extensively with an h-index of about 60 (according to Google Scholar). Two of his papers have received best paper award from IEEE and Joint ACM/IEEE venues. He also has 7 US patents and has consulted for several major IT and telecom companies, including Cisco and Google. He has served on the editorial board of IEEE Transactions on Networks, and Transactions on Parallel and Distributed Systems. His research has been funded by Cisco, Google and NSF. He was elected to IEEE Fellow for his contributions to optical and wireless network architectures and protocols.