Analysis of a Pool Management Scheme for Cloud Computing Centers

Hamzeh Khazaei, *Student Member, IEEE,* Jelena Mišić, *Senior Member, IEEE,*
Vojislav B. Mišić, *Senior Member, IEEE,* and Saeed Rashwand, *Student Member, IEEE*

Abstract—In this paper, we propose an analytical performance model that addresses the complexity of cloud centers through distinct stochastic sub-models, the results of which are integrated to obtain the overall solution. Our model incorporates the important aspects of cloud centers such as pool management, compound requests (i.e., a set of requests submitted by one user simultaneously), resource virtualization and realistic servicing steps. In this manner we obtain not only a detailed assessment of cloud center performance, but also clear insights into equilibrium arrangement and capacity planning that allows servicing delays, task rejection probability, and power consumption to be kept under control.

Index Terms—cloud computing, performance analysis, response time, blocking probability, pool management schema, power consumption, interacting Markov models.

1 Introduction

Cloud Computing is a computing paradigm in which different computing resources such as infrastructure, platforms and software applications are made accessible over the Internet to remote user as services [1]. Infrastructure-as-a-Service (IaaS) cloud providers, such as Amazon EC2 [2] and IBM Cloud [3], deliver, on-demand, operating system (OS) instances provisioning computational resources in the form of virtual machines deployed in the cloud providers data center. A cloud service differs from traditional hosting in three principal aspects. First, it is provided on demand; second, it is elastic since users that use the service have as much or as little as they want at any given time (typically by the minute or the hour); and third, the service is fully managed by the provider [4], [5]. Due to dynamic nature of cloud environments, diversity of user requests, and time dependency of load, providing agreed quality of service (QoS) while avoiding over-provisioning is a difficult task [6].

Service availability and response time are two important quality measures in cloud's users perspective [7]. Quantifying and characterizing such performance measures requires appropriate modeling; the model ought to cover vast parameter space while being tractable. A monolithic model may suffer from intractability and poor scalability due to large number of parameters [8]. Instead, in this paper we develop and evaluate tractable functional sub-models and their interaction model and solve them iteratively. We construct separate sub-models for different servicing steps in a complex cloud center and then the overall solution is obtained by iteration over individual sub-model solutions [9]. We assume that the cloud center consists of a number of Physical Machines (PM) that are allocated to users in the order of task arrivals. More specifically, user may share a PM using virtualization technique. A cloud user may ask for more than one virtual machine (VM) by submitting a single compound request, hereafter referred to as a *super-task* [10].

Our proposed model embraces multiple aspects and provides insight into their interactions of today's cloud centers. The main features of our analytical model can be listed as follows:

- Assumes Poisson arrival of user requests;
- Incorporates compound user requests by introducing super-tasks;
- Supports high degree of virtualization (e.g., up to 10 VMs on each PM);
- Captures different delays imposed by cloud centers on user requests;
- Describes the dynamics of server pools and provides the steady-state pools arrangement;
- Characterizes the service availability at cloud center;
- Provides information for capacity planning;
- Offers tractability and scalability;

The rest of the paper is organized as follows. In Section 2, we survey related work in cloud center performance analysis. Section 3 presents the model and the details of the analysis. Section 4 introduces the stochastic sub-models and their interactions. Section 5 presents the numerical results obtained from the analytical model. Finally, Section 6 summarizes our findings and concludes the paper.

2 Related Work

Cloud computing has attracted considerable research attention, but only a small portion of the work done so far

- *H. Khazaei and S. Rashwand are with the Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada. E-mails: {hamzehk,rashwand}@cs.umanitoba.ca*
- *J. Mišić and V. B. Mišić are with the Department of Computer Science, Ryerson University, Toronto, Ontario, Canada. E-mails: {jmisic,vmisic}@scs.ryerson.ca*

has addressed performance issues by rigorous analytical models. Many research works carried out a measurement-based performance evaluation of the Amazon EC2 [2], [11], IBM Blue Cloud [3] or other cloud providers in the context of scientific computing [12], [13], [14], [15], [16], [17], [18], [19]. Here we survey those that proposed a general analytical model for performance evaluation of cloud centers.

In [20], the authors studied the response time in terms of various metrics, such as the overhead of acquiring and realizing the virtual computing resources, and other virtualization and network communication overhead. To address these issues, they have designed and implemented C-Meter, a portable, extensible, and easy-to-use framework for generating and submitting test workloads to computing clouds. In [21], a cloud center is modeled as the classic open network with single task arrival, from which the distribution of response time is obtained, assuming Poisson arrival process and exponentially distributed service time. Using the distribution of response time, the relationship among the maximal number of tasks, the minimal service resources and the highest level of services was found.

In [22], the cloud center was modeled as an $M/M/m/m + r$ queuing system from which the distribution of response time was determined. The authors computed waiting, service and execution periods assuming that they are independent (which is not realistic according to the authors' own argument). Also the waiting and servicing time do not capture virtualization or instance creation.

Our earlier work [23], [7], [10], [24] presents monolithic analytical models which are much more restrictive than our current work in terms of extendability, simplicity and computation complexity. Also, [23], [7], [10], [24] do not address the concept of virtualization as well as heterogeneous server pools. All physical machines were assumed to be up and running in one homogenous server pool. Moreover, proposed models do not consider common servicing steps in cloud computing centers. For instance, only waiting time in the global queue was characterized.

The performance of cloud computing services for scientific computing workloads was examined in [12]. The authors quantified the presence of Many-Task Computing (MTC) users in real scientific computing workloads. Then, they performed an empirical evaluation of the performance of four commercial cloud computing services including Amazon EC2. Also, the authors compared the performance characteristics and cost models of clouds and other scientific computing platforms, for general and MTC-based scientific computing workloads.

In [25], the authors quantified the resiliency of IaaS cloud with respect to changes in demand and available capacity. Using a stochastic reward net based model for provisioning and servicing requests in a IaaS cloud, they quantified the resiliency of IaaS cloud with respect to two key performance measures, namely, task rejection rate and total response delay. The scope of this work is limited to the effects of abrupt changes in arrival process and the number of physical machines.

A hierarchical modeling approach for evaluating quality of experience in a cloud computing environment was proposed in [26]. Due to simplified assumptions, authors applied classical Erlang loss formula and $M/M/m/K$ queuing system for outbound bandwidth and response time modeling respectively.

In [9], the authors proposed a general analytical model for an end-to-end performance analysis of a cloud service. They illustrated their approach using IaaS cloud with three pools of servers: hot, warm and cold, using service availability and provisioning response delays as the key QoS metrics. The proposed approach reduces the complexity of performance analysis of cloud centers by dividing the overall model into sub-models and then obtaining the overall solution by iteration over individual sub-model solutions. However, proposed approach has some constraints as follows:

- The model is limited to single task arrival per request;
- The proposed model does not consider interactions among pools so no insight into steady-state arrangement of the pools can be obtained.
- The effect of virtualization has not been accounted for explicitly.

We recently proposed a fine-grained performance model [27] in which we tried to address some of the shortcomings mentioned above, including super-tasks that permit users to submit multiple tasks at once, and a high degree of virtualization (i.e., up to 10 VMs on a single PM) in the model. However, the model in [27] assumes a fixed number of PMs in each pool and does not consider dynamic behavior of server pools.

As can be seen, no reported work covers all aspects of cloud center performance as well as pool management issues. This has motivated us to develop a comprehensive model to analyze cloud center operation.

3 Analytical Model

In IaaS cloud, when a request is processed, a pre-built or customized disk image is used to create one or more VM instances [25]. In this work we assume that pre-built images fulfill all user requests. We assume that Physical Machines (PMs) are categorized into three server pools: hot (i.e., with running VMs), warm (i.e., turned on but without running VM) and cold (i.e., turned off) [9]. All PMs and VMs are homogeneous and each PM has two Virtual Machine Monitors (VMM) that configure and deploy VMs on the PM. Instantiation of a VM from an image and provisioning it on hot PMs has the minimum delay compared to warm and cold PMs. Warm PMs require more time to be ready for provisioning and cold PMs require additional time to be started up before a VM instantiation. In this work we allow users to submit super-tasks containing several tasks, each of which requires a single VM. The number of unit tasks in a super-task (i.e., its size) is a generally distributed random variable, while unit tasks have independent and identically distributed service times.

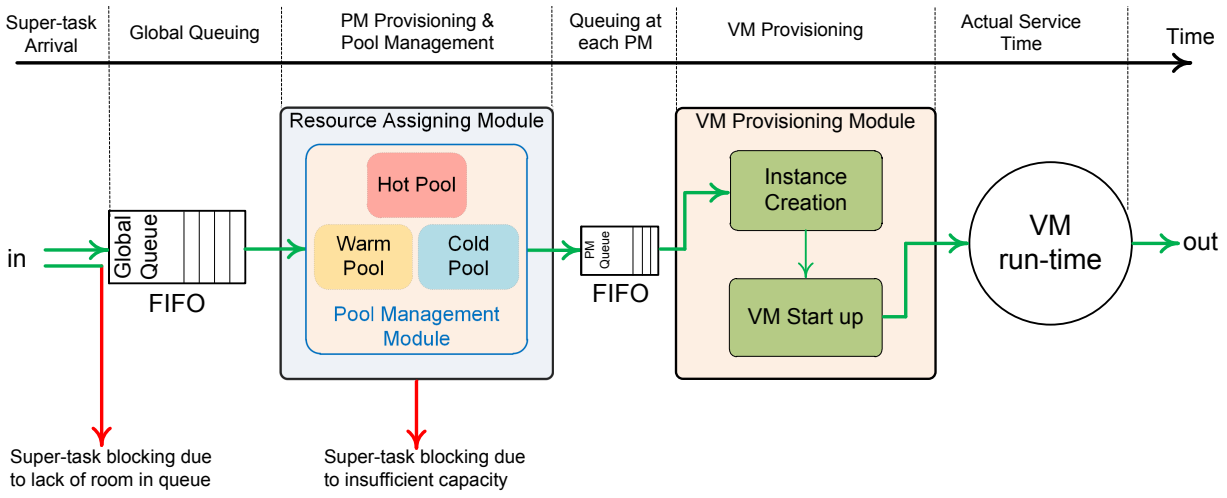


Fig. 1. The steps of servicing and corresponding delays.

If a user submits a super-task, it is very likely that individual tasks within the super-task require a high degree of interaction. As a result, we assume that all tasks within a super-task should be provisioned on the same PM. This assumption has two consequences. First, the super-task size must be limited to the number of VMs on a single PM. Second, all tasks within a super-task must begin service at the same time, otherwise the super-task will be rejected. The latter policy is often referred to as *total acceptance policy* [28]. The other approach, known as partial acceptance, may split a super-task so that individual tasks run on different PMs. While this policy may reduce the super-task rejection probability, it may also increase inter-task communication overhead and idle waiting, and, consequently, extend the overall service time. In this work, we assume total acceptance policy, while the performance of partial acceptance policy remains a promising topic for future work.

The steps incurred in servicing a super-task are shown in Fig. 1. User requests (super-tasks) are submitted to a global finite queue and then processed on a first-in, first-out basis (FIFO). A super-task that finds the queue full, will be rejected immediately. Once the super-task is admitted to the queue, it must wait until the Resource Assigning Module (RAM) processes it (first delay), as follows. First, RAM checks the hot pool for a PM with sufficient capacity. If such a PM can't be found, RAM moves on to the warm pool. If a PM can't be found there, RAM moves on to the cold pool. Eventually, a super-task is either assigned to a PM (second delay), or rejected due to insufficient system capacity. Once the super-task is assigned to one of the PMs, it will have to wait in that PM's input queue (third delay) until the VM Provisioning Module (VMPM) instantiates and deploys the necessary VMs (fourth delay), when the actual service starts. The overall response time is, then, the sum of the delays mentioned above and the actual service time. When a running task finishes, the capacity used by the corresponding VM is released and becomes available for servicing the next (super-)task.

The management of pools and movement of PMs among them is the primary responsibility of the Pool Management Module (PMM). While the first criterion that governs PMM operation is to minimize the cloud center response time and task rejection probability, the obvious solution that would keep all PMs in the hot pool (i.e., always on) would lead to excessive energy consumption. Hence the PMM needs to switch idle PMs from hot to warm pool, or from warm to cold pool, after a certain interval of inactivity, the duration of which may be altered to suit the traffic load and other operational parameters, as will be explained below.

To model the behavior of this system, we design three stochastic sub-models, each of which captures the details of the respective management module (RAM, VMPM, and PMM) and combine them into an overall model. then, we solve this model to compute the cloud performance metrics: task rejection probability and mean response delay as functions of variations in workload (super-task arrival rate, super-task size, task mean service time, number of VMs per PM), system capacity (i.e., number of PMs in each pool), and design parameters such as the rate by which idle hot PM powers down to either warm or cold pool. We describe our analysis in detail in the following sections, using the symbols and acronyms listed in Table 1.

4 Integrated Stochastic Models

As mentioned in section 2, due to high complexity of cloud centers, the proposed monolithic models were hard to analyze and extend. By introducing interacting analytical sub-models, the complexity of system is divided so that they can be analyzed accurately in order of processing steps. Moreover, the sub-models are scalable enough to capture the flexibility of the cloud computing paradigm.

In this paper, we implement the sub-models using interactive Continuous Time Markov Chain (CTMC). The sub-models are interactive such that the output of one sub-model is input to the other ones and vice versa. Table 2 shows the

TABLE 1
Symbols and corresponding descriptions.

Symbol	Description
RAM	Resource Assigning Module
VMPM_x	Virtual Machine Provisioning Module for a PM in the pool x; x could be hot, warm or cold
PMM	Pool Management Module
RASM	Resource Allocation Sub-Model
VMPSM_x	Virtual Machine Provisioning Sub-Model for a PM in the pool x; x could be hot, warm or cold
PMSM	Pool Management Sub-Model
MSS	Maximum Super-task Size
m	Maximum No. of VMs on a PM
p_i	Probability of having a super-task with size i
λ_{st}	Mean arrival rate of super-tasks
L_q	Size of global input queue
P_h, P_w, P_c	Probability of successful search in hot, warm and cold pool
$1/\alpha_h, 1/\alpha_w, 1/\alpha_c$	Mean look up time in hot, warm and cold pool
BP_q	Blocking probability due to lack of room in the global queue
BP_r	Blocking probability due to lack of capacity in the cloud center
P_{reject}	Total probability of blocking ($BP_q + BP_r$)
P_i	The probability by which a hot PM becomes idle
1/R	Mean searching time for finding an idle hot PM
$1/SU_w$	Mean start up time for a warm PM to become a hot PM
$1/SU_c$	Mean start up time for a cold PM to become a hot PM
$1/FR_w$	Mean time in which the lack of resource (PM) in the hot pool is detected and a PM from warm pool becomes hot
$1/FR_c$	Mean time in which the lack of resource (PM) in the hot pool is detected and a PM from cold pool becomes hot
σ	Desired No. of PMs that PMM tries to maintain in the warm pool for promoting the resiliency of the cloud center
$\overline{wt}$	Mean waiting time in global queue
$\overline{lut}$	Mean look up delay among pools
ϕ_h, ϕ_w, ϕ_c	Instantiation rate of a VM on a PM in the hot, warm or cold pool
$\lambda_h, \lambda_w, \lambda_c$	Arrival rate to a PM in the hot, warm and cold pool
N_h, N_w, N_c	Number of PMs in the hot, warm and cold pool
$\overline{PM_{wt}}$	Mean waiting time in a PM queue
$\overline{pt}$	Mean VM provisioning time
$\overline{td}$	Mean total delay for a task before getting into actual service

modules and their corresponding sub-models, which will be discussed in detail in the following sections.

4.1 Resource Allocation Sub-Model

The resource allocation process is described in the Resource Allocation Sub-Model (RASM) shown in Fig. 2. RASM is a

TABLE 2
Modules and their corresponding sub-models.

Module	Stochastic sub-model
RAM	RASM
VMPM_hot	VMPSM_hot
VMPM_warm	VMPSM_warm
VMPM_cold	VMPSM_cold
PMM	PMSM

two dimensional CTMC that records the number of super-tasks in the global queue as well as the current pool on which provisioning is taking place. Since the number of potential users is high and a single user typically submits super-tasks at a time with low probability, the super-task arrival can be adequately modeled as a Poisson process [29] with rate λ_{st}. Each state of Markov chain is labeled as (i, j), where i indicates the number of super-tasks in queue and j denotes the pool on which the leading super-task is under provisioning. State $(0, 0)$ indicates that system is empty which means there is no request under provisioning or in the queue. Index j can be h, w or c that indicate current super-task is undergoing provisioning on hot, warm or cold pool respectively. The global queue size is L_q and one more super-task can be at the deployment unit for provisioning thus, the capacity of system is $L_q + 1$.

Let P_h, P_w and P_c be the success probabilities of finding a PM that can accept the current super-task in the hot, warm and cold pool respectively. We assume that $1/\alpha_h$, $1/\alpha_w$ and $1/\alpha_c$ are the mean look up delays for finding an appropriate PM in hot, warm and cold pool respectively. Upon arrival of first super-task, system moves to state $(0, h)$ which means the super-task will be provisioned immediately in the hot pool. Afterwards, depending on the upcoming event, three possible transitions can occur:

(a) Another super-task has arrived and system transits to state $(1, h)$ with rate λ_{st}.
(b) A PM in hot pool accepts the super-task so that system moves back to state $(0, 0)$ with rate $P_h\alpha_h$.
(c) None of PMs in hot pool has enough capacity to accept the super-task, so the system examines the warm pool (i.e., transit to state $(0, w)$) with rate $(1 - P_h)\alpha_h$.

On state $(0, w)$, RASM attempts to provision the super-task on warm pool. If one of the PMs in the warm pool can accommodate the super-task, the system will get back to $(0, 0)$, otherwise RASM examines the cold pool (i.e., transition from state $(0, w)$ to $(0, c)$). If none of the PMs in cold pool can provision the super-task, the system moves back from $(0, c)$ to $(0, 0)$ with rate $(1 - P_c)\alpha_c$ which means the user request (super-task) will get rejected due to insufficient resources in the cloud center. During the provisioning in cold pool, another super-task may arrive and takes the system to state $(1, h)$ which means there is one super-task in deployment unit and one awaiting in global queue. Finally, the super-task under provisioning decision leaves the deployment unit, once it receives a decision from RASM and the next super-task at the head of global queue

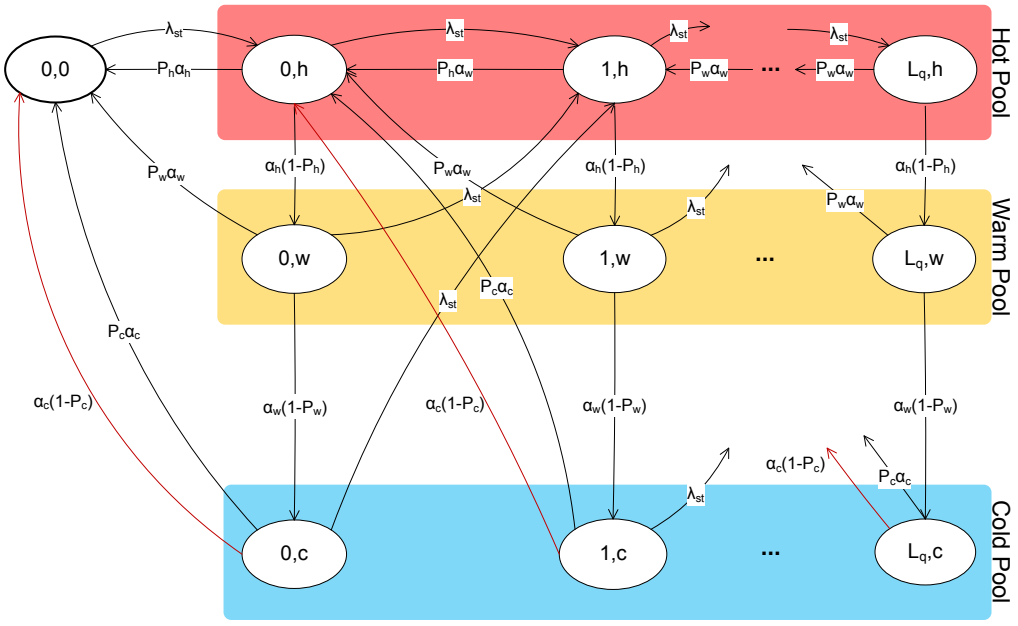


Fig. 2. Resource Allocation Sub-Model (RASM).

will go under provisioning. In this sub-model, arrival rate of super-task (λ_{st}) and look up delays ($1/\alpha_h, 1/\alpha_w$ and $1/\alpha_c$) are exogenous parameters and success probabilities (P_h, P_w and P_c) are calculated from the VM provisioning sub-model. The VM provisioning sub-model will be discussed in the next section.

Using steady-state probabilities $\pi_{(i,j)}$, blocking probability can be calculated. Super-tasks may experience two kinds of blocking events:

(a) Blocking due to a full global queue occurs with the probability of

$$BP_q = \pi_{(L_q,h)} + \pi_{(L_q,w)} + \pi_{(L_q,c)} \quad (1)$$

(b) Blocking due to insufficient resources (PMs) at pools occurs with the probability of [30]

$$BP_r = \sum_{i=0}^{L_q} \frac{\alpha_c(1-P_c)}{\alpha_c + \lambda_{st}} \pi_{(i,c)} \quad (2)$$

The probability of reject is, then, $P_{reject} = BP_q + BP_r$. In order to calculate the mean waiting time in queue, we first establish the probability generating function (PGF) for the number of super-tasks in the queue [31], as

$$Q(z) = \pi_{(0,0)} + \sum_{i=0}^{L_q} (\pi_{(i,h)} + \pi_{(i,w)} + \pi_{(i,c)}) z^i \quad (3)$$

The mean number of super-tasks in queue is [28]

$$\overline{q} = Q'(1) \quad (4)$$

Applying Little's law [32], the mean waiting time in the global queue is given by (first delay):

$$\overline{wt} = \frac{\overline{q}}{\lambda_{st}(1 - BP_q)} \quad (5)$$

Look up time among pools can be considered as a Coxian distribution with 3 steps (Fig. 3).

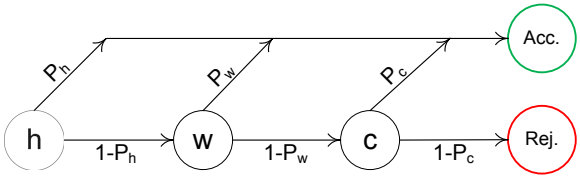


Fig. 3. Three steps of look-up delays among pools.

Therefore according to [33], the look-up time (second delay) can be calculated as follows.

$$\overline{lut} = \frac{1/\alpha_h + (1-P_h)((1/\alpha_w) + (1-P_w)(1/\alpha_c))}{1 - BP_q} \quad (6)$$

4.2 VM Provisioning Sub-Model

Virtual Machine Provisioning Sub-Model (VMPSM) captures the instantiation, deployment and provisioning of VMs on a PM. VMPSM also incorporates the actual servicing of each task (VM) on a PM. Note that each task within super-task is provisioned with an individual VM, but RASM makes sure that all tasks within a super-task are provisioned at the same PM. Fig. 4 shows the VMPSM (a CTMC) for a PM in the hot pool. As we assume homogeneous PMs, VMs, and tasks, all VMPSMs for the PMs in the hot pool are identical in terms of arrival and instantiation rates. The same holds for the PMS in the warm and cold pool – they can be modeled with the same VMPSM, though with a different arrival and instantiation rates. Consequently, each pool (hot, warm and cold) can be modeled as a set of VMPSMs.

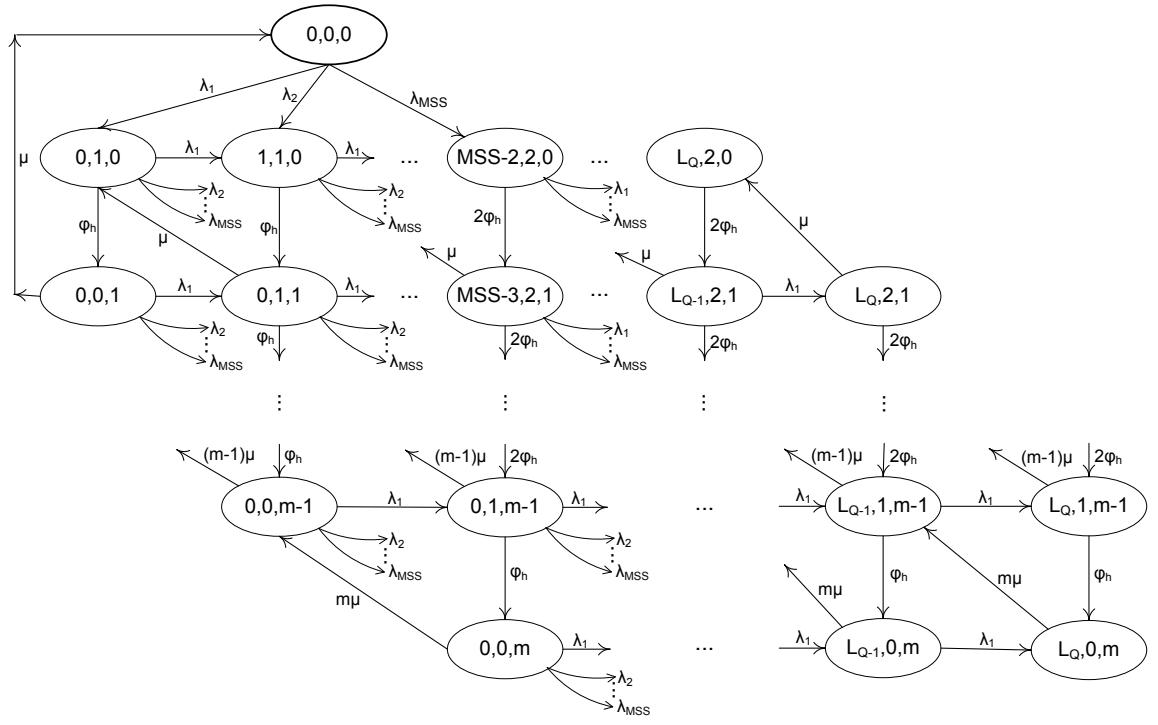


Fig. 4. Virtual Machine Provisioning Sub-Model for a PM in the hot pool (VMPSM_hot).

Each state in Fig. 4 is labeled by (i, j, k) in which i indicates the number of tasks in PM's queue, j denotes the number of tasks that is under provisioning by VMMs and k is the number of VM that are already deployed on the PM. Since we assume two VMMs installed on each PM, at most tasks can be provisioned simultaneously on a PM. Note that we set the queue size at each PM to m, the maximum number of VMs that can be deployed on a PM. Let ϕ_h be the rate at which a VM can be deployed on a PM at hot pool and μ be the service rate of each VM. So, the total service rate for each PM is the product of number of running VMs by μ. State $(0, 0, 0)$ indicates that the PM is empty and there is no task either in the queue or in the instantiation unit. Depending on the size of arriving super-task, model transits to one of the states in $\{(0, 1, 0),\ (0, 2, 0),\ \cdots\ ,(MSS - 2, 2, 0)\}$. Since all tasks within a super-task are to be provisioned at the same PM, maximum super-task size (MSS) is smaller or equal to m. The arrival rate to each PM in the hot pool is given by:

$$\lambda_h = \frac{\lambda_{st}(1 - BP_q)}{N_h} \tag{7}$$

in which N_h is the number of PMs in the hot pool. Note that BP_q, used in (1), is obtained from RASM. In Fig. 4, $\{\lambda_i, i = 1..MSS\}$ is given by $\lambda_i = \lambda_h p_i$; here p_i is the probability of having a super-task of size i. We examine truncated geometric and uniform distributions for numerical experiment.

The state transition in VMPSM can occur due to super-task arrival, task instantiation or service completion. From state $(0, 0, 0)$, system can move to state $(0, 2, 0)$ with rate λ_2 (i.e., super-task with size 2). From $(0, 2, 0)$, system can transit to $(0, 1, 1)$ with rate ϕ_h (i.e., instantiation rate) or upon arriving a super-task with size k, moves to state $(k, 2, 0)$. From $(0, 1, 1)$, system can move to $(0, 0, 2)$ with rate ϕ_h, transits to $(0, 1, 0)$ with rate μ (i.e., service completion rate), or again upon arriving a super-task with size k, system can move to state $(k - 1, 2, 1)$ with rate λ_k. A PM can not accept a super-task, if there is not enough room to accommodate all tasks within a super-task. Suppose that $\pi^h_{(i,j,k)}$ is the steady-state probability for the hot PM model (Fig. 4) to be in the state (i, j, k). Using steady-state probabilities, we can obtain the probability that at least one PM in hot pool can accept the super-task for provisioning. At first we need to compute the probability that a hot PM cannot admit a super-task for provisioning (P^h_{na}). Let F_h be the free capacity of the hot PM at each state:

$$F_h(i, j, k) = m - (i + j + k)$$

P^h_{na} is then given by:

$$P^h_{na} = \sum_{x \in \xi} \sum_{t=F_h+1}^{MSS} \pi^h_x p_t \tag{8}$$

where p_t is the probability of having a super-task of size t and ξ is a subset of VMPSM's states in which there is a chance of blocking for super-tasks:

$$\xi = \{(i, j, k) |\ F_h(i, j, k) < MSS\}$$

Therefore, probability of successful provisioning (P_h) in the hot pool can be obtained as

$$P_h = 1 - (P^h_{na})^{N_h} \tag{9}$$

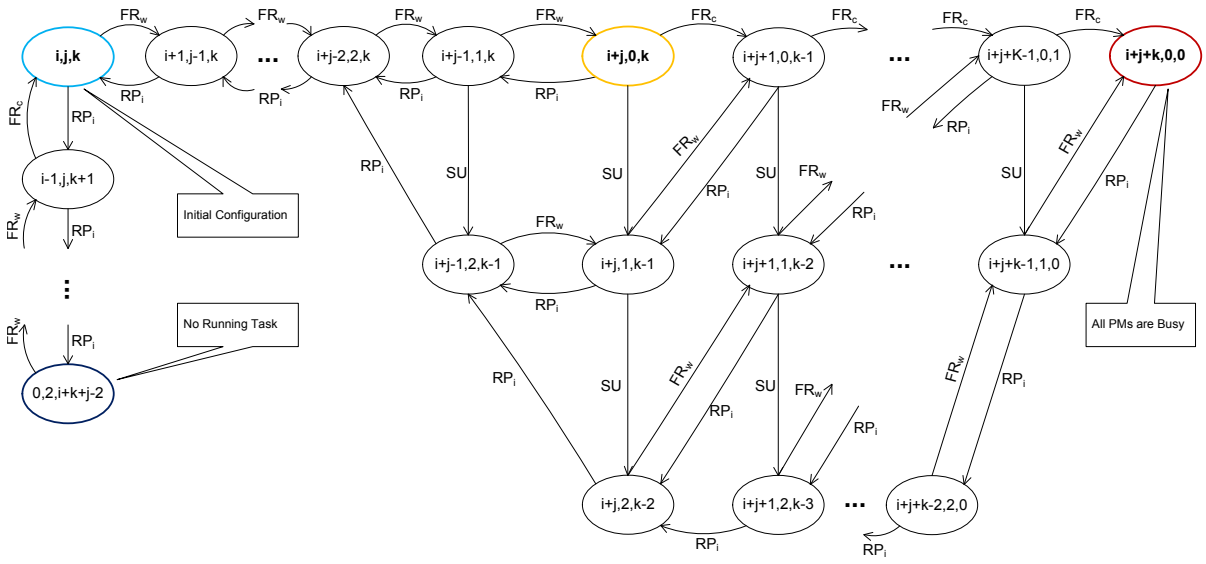


Fig. 5. Pool Management Sub-Model (PMSM).

Note that P_h is used as an input parameter in the resource allocation sub-model (Fig. 2). The provisioning model for warm PM is the same with the one for hot PM, though, there are some differences in parameters:

(a) The arrival rate to each PM in the warm pool is

$$\lambda_w = \frac{\lambda_{st}(1 - BP_q)(1 - P_h)}{N_w} \tag{10}$$

where N_w is the number of PMs in warm pool.

(b) Every PM in warm pool requires extra time to be ready (hot) for first instantiation. This time is assumed to be exponentially distributed with mean value of $1/\gamma_w$.

Instantiation rate in warm pool is the same as in hot pool (ϕ_h). Like VMPSM for a hot PM (Fig. 4), the model for a warm PM is solved and the steady-states probabilities ($\pi^w_{(i,j,k)}$), are obtained. The success probability for provisioning a super-task in warm pool is:

$$P_w = 1 - (P^w_{na})^{N_w} \tag{11}$$

A PM in the cold pool can be modeled as in the warm and hot pool but with different arrival rate and start up time. The effective arrival rate at each PM in the cold pool is given by:

$$\lambda_c = \frac{\lambda_{st}(1 - BP_q)(1 - P_h)(1 - P_w)}{N_c} \tag{12}$$

in which N_c is the number of PMs in the cold pool. A cold PM (off machine), requires longer time than a warm PM to be ready (hot) for the first instantiation. We assume that the start up time is also exponentially distributed with mean value of $1/\gamma_c$. The success probability for provisioning a super-task in the cold pool is given by:

$$P_c = 1 - (P^c_{na})^{N_c} \tag{13}$$

From VMPSM, we can also obtain the mean waiting time at PM's queue (third delay: $\overline{PM_{wt}}$) and mean provisioning time (fourth delay: $\overline{pt}$) by using the same approach as the one that led to Eq. (5). As a result, the total delay before starting of actual service time, is given by:

$$\overline{td} = \overline{wt} + \overline{lut} + \overline{PM_{wt}} + \overline{pt} \tag{14}$$

Note that all success probabilities (i.e., P_h, P_w and P_c) are input parameters to the resource allocation sub-model (Fig. 2).

4.3 Pool Management Sub-Model

Pool Management Sub-Model (PMSM) captures the details of pools management in the cloud center. We model PMSM as a three dimensional CTMC, Fig. 5. The system starts with state (i, j, k) which indicates that i, j and k PMs are in the hot, warm and cold pool respectively. If next search in hot pool was successful (probability of P_h), system will remain in the starting state, state (i, j, k), otherwise PMSM will bring one of the warm PMs into the hot pool (requires some time). Such transition occurs with the rate of FR_w which is defined as:

$$FR_w = \frac{1}{\lambda_{st} BP_q (1 - P_h) + (1/SU_w)} \tag{15}$$

in which $(1/SU_w)$ is the mean time that a warm PM becomes a hot PM.

In state (i, j, k), if one of the hot PM becomes idle (i.e., no running VM), the PMSM moves the idle hot PM to cold pool (i.e., turn off) by going to state $(i - 1, j, k + 1)$. If all PMs become idle, PMSM will move all the hot PMs to the cold pool and maintain a minimum number of PMs (here we assume 2 PMs) in the warm pool. As can be seen from Fig. 5, state $(i + j - 1, 1, k)$, PMSM turns on a cold PM machine whenever the number of warm PMs gets lower than 2. If the system is in state $(i + j, 0, k)$, in which there is no warm PM at the moment, upon arrival of a new super-task, PMSM will try to accommodate the new super-task

on the current hot PMs. In case of lack of capacity for new arrival, PMSM borrows a PM from cold pool with the rate FR_c, moving to state $(i+j+1, 0, k-1)$.

$$FR_c = \frac{1}{\lambda_{st} BP_q (1 - P_h) + (1/SU_c)} \tag{16}$$

where $(1/SU_c)$ is the mean time that a cold PM becomes a hot PM.

The state $(i+j+k, 0, 0)$ indicates that all PMs in the cloud center are hot and servicing users' tasks. At any state, if the number of PMs in the warm pool is less than a predefined minimum (e.g., 2 PMs in Fig. 5), an idle hot PM is moved to the warm pool at rate RP_i, otherwise the idle hot PM is moved to the cold pool (i.e., shut down).

4.4 Interaction among Sub-Models

The interactions among sub-models are depicted in Fig. 6. VM provisioning sub-models (VMPSM) compute the steady-state success probabilities (P_h, P_w and P_c) that at least a PM in one of the pools (hot, warm and cold, respectively) can accept the super-task. These success probabilities are used as input parameters to the resource allocation sub-model (RASM). The hot PM sub-model (VMPSM_hot) computes P_h which is the input parameter for both warm and cold sub-models. The warm PM sub-model (VMPSM_warm) computes P_w which is the input parameter for the cold sub-model (VMPSM_cold). In addition, the hot PM model provides P_h and the probability by which a hot PM becomes idle (P_i) for pool management sub-model (PMSM). In return, PMSM computes N_h, N_w and N_c as input for hot, warm and cold PM sub-models respectively. The resource allocation sub-model (RASM) computes the blocking probability, BP_q, which is the input parameter to VM provisioning sub-models as well as PMSM. Table 3 shows the sub-models and their corresponding outputs.

TABLE 3
Sub-models and their outputs.

Sub-model	Output(s)
RASM	BP_q, BP_r
VMPSM_hot	P_h, P_i
VMPSM_warm	P_w
VMPSM_cold	P_c
PMSM	N_h, N_w, N_c

As can be seen, there is an inter-dependency among sub-models. This cyclic dependency is resolved via fixed-point iterative method [34] using a modified version of successive substitution approach. For numerical experiments the successive substitution method (Algorithm 1) is continued until the difference between the previous and current value of blocking probability in the global queue (i.e., BP_q) is less than 10^{-6}. Usually the integrated model converges to the solution in less than 8 iterations.

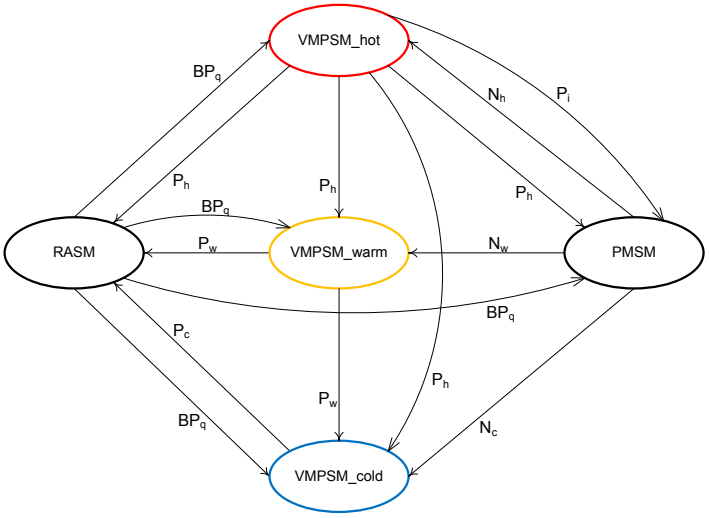


Fig. 6. Interaction diagram among sub-models.

Algorithm 1 Successive Substitution Method

Input: Initial success probabilities in pools: P_{h0}, P_{w0}, P_{c0}
Input: Initial idle probability of a hot PM: P_{i0}
Output: Blocking probability in Global Queue: BP_q

```
counter ← 0; max ← 10; diff ← 1
BP_q0 ← RASM (P_h0, P_w0, P_c0)
[N_h, N_w, N_c] ← PMM (P_h0, P_i0)
while diff ≥ 10^-6 do
    counter ← counter + 1
    [P_h, P_i] ← VMPSM_hot (BP_q0, N_h)
    P_w ← VMPSM_warm (BP_q0, P_h, N_w)
    P_c ← VMPSM_cold (BP_q0, P_h, P_w, N_c)
    [N_h, N_w, N_c] ← PMM (P_h, P_i)
    BP_q1 ← RASM (P_h, P_w, P_c)
    diff ← |(BP_q1 − BP_q0)|
    BP_q0 ← BP_q1
    if counter = max then
        break
    end if
end while
if counter = max then
    return -1
else
    return BP_q0
end if
```

4.5 Scalability and flexibility of integrated model

A cloud center may scale up or down in terms of global queue size, number of PMs in each pool and the degree of virtualization which are referred as design parameters. Table 4 shows the relationship between the number of states in each sub-model and their associated design parameters.

As can be seen from Table 4, the number of states in each sub-model has a linear dependency on design parameters which guarantees the scalability of the integrated model. Note that VMPSMs are identical for all PMs in the same pool. Therefore, we need only to solve three VMPSMs (i.e.,

TABLE 4
Relationship between the size of sub-models and design parameters.

Sub-Model	Design Parameters	No. of States: f(i)
RASM	L_q	f(L_q) = $3L_q$ + 1
VMPSM	m	f(m) = 3, if m=1 f(m) = 6, if m=2 f(m) = $2f(m-1) - f(m-2) + 1$, if m>2
PMSM	N_w, N_c, σ	f(N_w, N_c, σ) = $N_w + N_c + 1 + \sigma \sum_{s=1}^{\sigma}(N_w + N_c - \sigma - s)$

VMPSM_hot, VMPSM_warm and VMPSM_cold) regardless of how many PMs are in the cloud center.

5 NUMERICAL VALIDATION

The resulting sub-models have been implemented and solved using Maple 15 from Maplesoft, Inc. [35]. Under different configurations and parameter settings, we obtain two important performance metrics, namely, rejection probability of super-tasks (STs) and total response delay. We show the effects of changing arrival rate, task service time, number of PMs in each pool, number of VMs in each PM and super-task size on the said performance indicators. We also obtain the steady-state arrangement of pools (i.e., number of PMs in each pool) under above-mentioned parameter setting as well as various pool checking rates.

Table 5 presents the range of individual parameter values. Also, in all subsequent discussions and diagrams, the notation $[h, w, c]$ will refer to a cloud center with h, w, and c PMs (i.e., servers) in the hot, warm, and cold pool, respectively.

Our first set of experiments investigates the effects of task service time on rejection probability and total delay imposed by the cloud center on super-tasks before getting into service. In the first experiment, the results of which are shown in Fig. 7, super-tasks have one task each, while each PMs are permitted to run only two VMs; the super-task arrival rate is constant at 1000 STs/hr. As can be seen from Fig. 7(a), increasing the mean service time of individual tasks leads to an almost linear increase in rejection probability, while the increase of the system capacity (i.e., the number of PMs in each pool) reduces the rejection probability. The latter effect is more pronounced for tasks with longer service time, in particular for those with service time over 80 minutes. At the same time, extending system capacity has a negligible impact of rejection probability for tasks shorter than 60 min. These results allow us to identify a suitable arrangement for each service time, by applying a trade-off between the performance gain and the expenditure of required resources. Regarding total delay, Fig. 7(b) shows that, for all pool arrangements, longer service times will result in longer delays, since the actual service time has a direct effect on the amount of waiting time of queued super-tasks. This effect is more noticeable

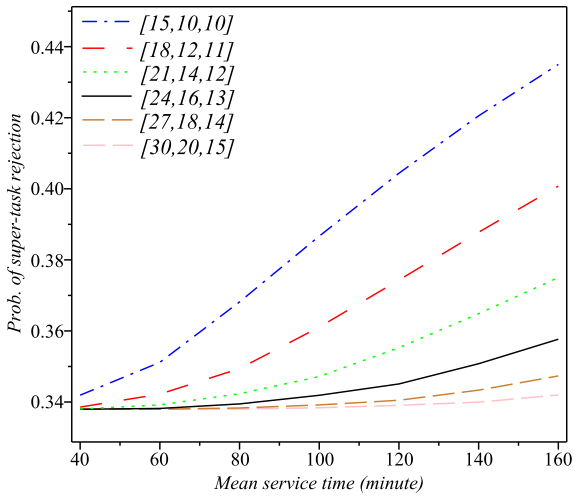
TABLE 5
Range of parameters for numerical experiment.

Parameter	Range	Unit
Arrival rate	400 . . 1000	super-tasks/hour
Mean service time	40 . . 160	minute
No. of PMs in the hot pool	15 . . 30	N/A
No. of PMs in the warm pool	10 . . 20	N/A
No. of PMs in the cold pool	10 . . 15	N/A
No. of VMMs on each PM	2	N/A
No. of VMs on each PM	2 . . 10	N/A
Mean Look-up rate in the hot pool	625	searches/hour
Mean Look-up rate in the warm and cold pool	825	searches/hour
Required time for a warm PM to become hot	1 . . 5	minute
Required time for a cold PM to become hot	5 . . 10	minute
VM deployment time on a hot PM	1 . . 5	minute
First VM deployment delay on a warm PM	3 . . 10	minute
First VM deployment delay on a cold PM	8 . . 15	minute
Size of global queue	50 . . 100	super-tasks
Queue size at PMs	2 . . 10	tasks

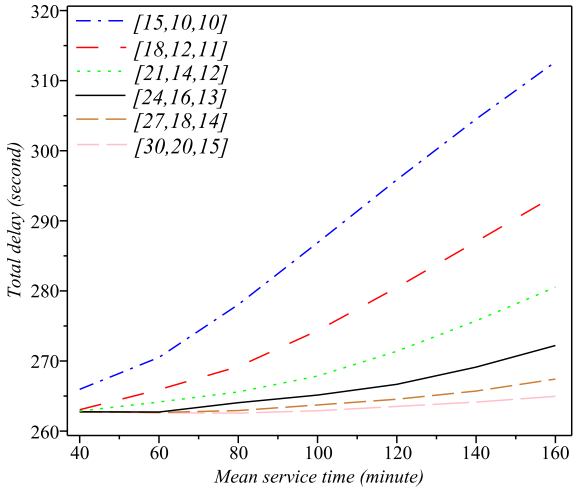
for 'weaker' configurations such as [15, 10, 10] (i.e., 15 PMs in the hot pool, and warm and cold pools with 10 PMs each), as opposed to the 'stronger' ones such as [30, 20, 15] where the delay increases is barely perceptible in the observed range of mean service times.

System performance when the mean service time is fixed at 100 minutes, but the super-task arrival rate (which, in this case, is equal to the task arrival rate) is variable, is shown in the diagrams in Fig. 8. Again, the rejection probability, shown in Fig. 8(a), exhibits a sharp rise above a certain value of (super-)task arrival rate, which is mainly due to insufficient size of the global (input) queue. However, the total delay, shown in Fig. 8(b), exhibits a sharp increase at an even lower value of the (super-)task arrival rate; when the rejection rate becomes non-negligible, the delay flattens because the actual number of tasks serviced is decreasing. Thus a simple increase in the size of the global queue is ill advised, as it would reduce the rejection probability at the expense of much higher delays. Consequently, if a cloud provider observes unacceptable delay and/or task rejection performance, a prudent course of action would be to increase the system capacity (i.e., the number of PMs) first, and extend the global queue only if the observed total response time is well below the prescribed limit.

Our second set of experiments focuses on the impact of super-task size on system performance. We have conducted the calculations using both geometric and uniform distribution for the number of tasks in a super-task. In the former case, we truncate the distribution at the value of maximum super-task size (MSS) which corresponds to the maximum number of VMs running on a single PM, so the probability

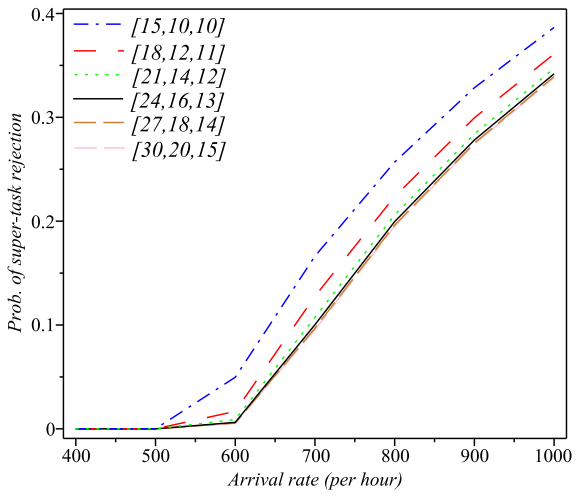


(a) Super-task rejection probability.

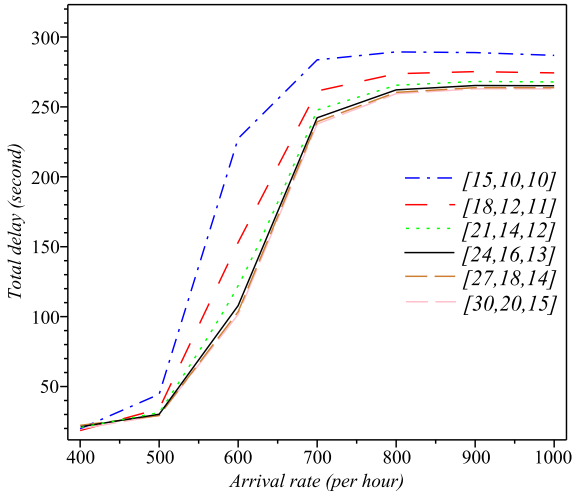


(b) Total delay on a super-task.

Fig. 7. Performance metrics w.r.t service time: 2 VMMs on each PM, 1 VM on each VMM, arrival rate = 1000 STs/hour, super-task-size = 1.



(a) Super-task rejection probability.



(b) Total delay on a super-task.

Fig. 8. Performance metrics w.r.t arrival rates: 2 VMMs on each PM, 1 VM on each VMM, service time = 100 min, super-task-size = 1.

of having a super-task of size i is

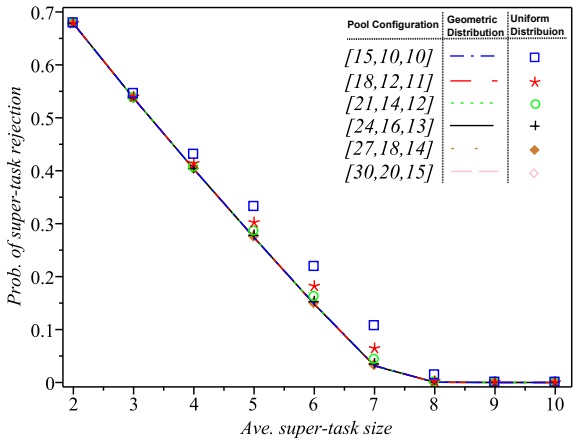
$$p_i = \begin{cases} (1-p)^{i-1}p, & \text{if } i < \text{MSS} \\ (1-p)^{MSS-1}, & \text{if } i = \text{MSS} \end{cases} \quad (17)$$

in which p is the parameter of geometric distribution. In the latter case (i.e., for uniform distribution) the probability of having a super-task of size i is

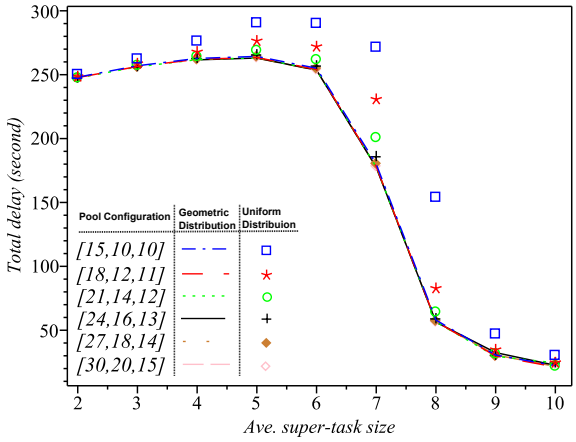
$$p_i = \frac{1}{MSS} \quad (18)$$

Regarding the value of MSS, it is worth noting that our model does not impose any restriction whatsoever on the degree of virtualization, but we still need a practical limit. We are aware that a mainframe class computer can indeed run tens, or perhaps even a hundred VMs that correspond to a standard PC machine. However, a recent survey has shown that 61% of the respondents run less than 10 VMs per PM, while a mere 5% run more than 25 VMs on a PM [36]. Moreover, VMs running on multi-core CPUs are vulnerable to cache interference, as a VM running on one core may degrade the performance of a VM running on another core [37], [38]. Hence, we have assumed that each PM runs two VMMs, each of which manages up to five VMs, for a total of up to 10 VMs per PM.

From the diagrams in Fig. 9(a), we can see that an increase in super-task size leads to a reduction of rejection probability for both geometric and uniform distributions. As for the total task delay, shown in Fig. 9(b), the curves show a relatively flat peak, and then fall off rapidly. This is due to the fact that 4500 tasks/hour is the aggregate task arrival rate: as the mean super-task size increases, the actual super-task arrival rate will decrease. In case of geometric distribution (shown with lines), larger mean size means lower variability of actual sizes, which makes the resulting rejection probability and total delay lower. In fact, in the boundary case where the average super-task size is

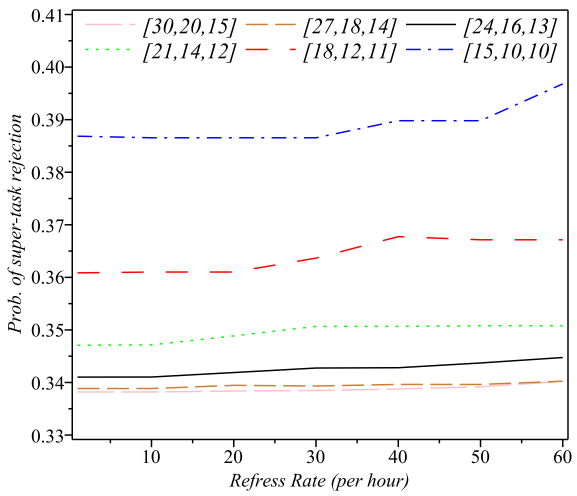


(a) Super-task rejection probability.

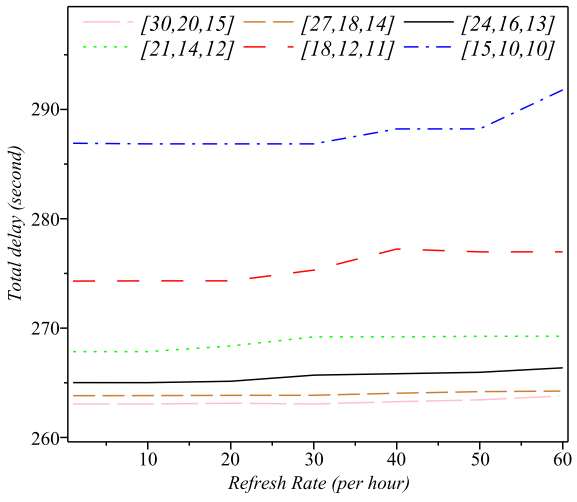


(b) Total delay of a super-task.

Fig. 9. Performance metrics w.r.t super-task size: 2 VMMs, 5 VMs on each VMM, service time = 100 min, arrival rate = 4500 tasks/hour.



(a) Super-tasks rejection probability.



(b) Total delay on a super-task.

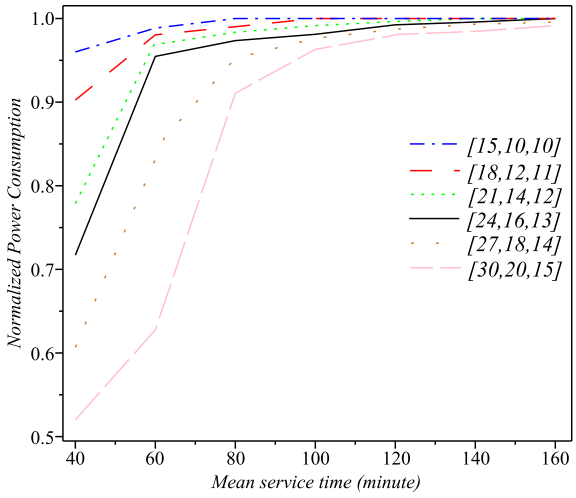
Fig. 10. Performance metrics w.r.t hot pool checking rate: 2 VMs on each PM, 1 VM on each VMM, super-task-size = 1, arrival rate = 1000 STs/hour, service time = 100 min.

10, all super-tasks have the same size, and the super-task arrival rate is one-tenth of the aggregate rate. As the result, each incoming super-task will obtain an entire PM which, together with the corresponding reduction in super-task arrival rate, guarantees a shorter waiting time in the global queue and virtual elimination of blocking at that queue. The processing delay becomes, then, the dominant part of the total delay. We note that lower variability of super-task size has been shown to increase the server utilization in multi-server systems as well [39], [40].

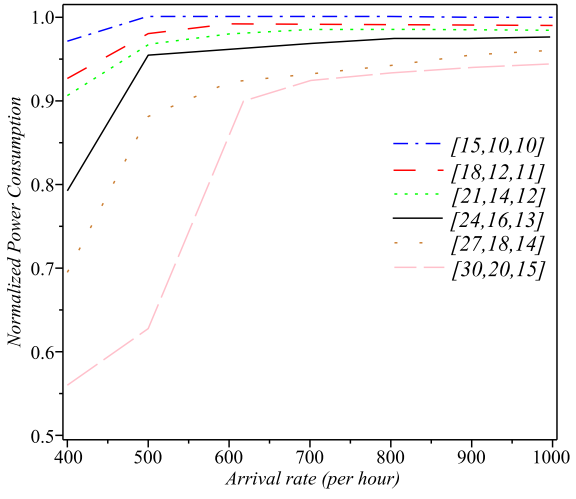
Similar behavior can be observed in the case of uniform distribution (shown with symbols), although the rejection probability and total delay are somewhat higher than in the case of geometric distribution. The rationale for this behavior is that super-tasks of all sizes are equi-probable under uniform distribution, whereas larger size super-tasks (which are more likely to get rejected under the total acceptance policy) have lower probability under geometric distribution. Also, larger super-tasks release more capacity when finished, which helps reduce the overall delay.

Our next experiment investigates the effects of pool checking rate on the performance metrics. The pool checking rate is the rate by which PMM searches for idle hot PMs and switches them to the warm or cold pool, respectively. Note that an idle hot PM would be first switched to the warm pool, and later to the cold pool; however, if there are more than σ PMs in the warm pool, it will be immediately moved to the cold pool (i.e., shut down). In previous experiments the mean pool checking rate was fixed at 15 checks per hour. For obvious reasons, higher pool checking rate will result in lower power consumption, however a rate that is too high may bring about the ping-pong effect. Namely, a recently shut-down PM may be needed soon so PMM has to turn it on again, and frequent back and forth movement of PMs may lead to longer delays and higher rejection probability, not to mention reduced hardware reliability.

In our experiments we have calculated the performance



(a) Normalized Power consumption: arrival rate = 500 (per hour).



(b) Normalized Power consumption: service time = 60 min.

Fig. 11. Normalized power consumption: 2 VMMs on each PM, 1 VM on each VMM, super-task-size = 1.

metrics for a number of different pool configurations. As can be seen from Fig. 10, 'stronger' configurations easily outperform the 'weaker' ones. When power consumption, shown in Fig. 11, is also taken into account, the results reveal that any given pool arrangement has a unique pool check rate that results in optimum balance between power consumption, on one side, and availability and throughput, on the other side. From the results shown in Figs. 10 and 11, it can be seen that for [15,10,10] arrangement the optimum pool check rate is 30/hour, while for [18,12,11] the best rate is 20/hr. For other configurations, a pool check rate of 10/hour can be considered as a good choice.

Using PMM, we also study the steady-state arrangement of pools in a cloud center. In other words, we try to identify the most probable arrangement of PMs in the pools in a long run. We examine the model for vast variety of parameter space and present the results for 3 distinct settings (Table 6) in the Table 7.

In Table 7, the first column indicates the initial configuration of pools by which we start the experiment. Then under 3 different settings and for 6 pool configurations the steady-state arrangements are identified (the arrangements that are in bold face). Under such arrangement the rejection probability is zero and the cloud center imposes the minimum delay on its users' requests. For instance, for setting (A) the best arrangement is **[41,5,7]**. Such steady-state configuration suggests a cloud provider to arrange its server pool accordingly in order for achieving the highest user satisfaction. However, if some amount of rejection or delay is tolerable, our experiment could suggest other configurations that can balance cloud providers' interests such as rejection probability, response delay, power consumption, and etc.

TABLE 6
Three configuration settings for the experiment.

Design Parameters	Setting (A)	Setting (B)	Setting (C)
Mean arrival rate (hour)	500	600	450
Mean service time (min)	80	60	100
Mean pool check rate (hour)	60	30	50
Desired No. of warm PMs	10	15	20
No. of VM on each PM	2	2	10
No. of VMM on each PM	2	2	2
Mean super-task size	1	2	4
Global queue size	50	75	100

TABLE 7
Steady-state configuration of pools.

Initial Configuration	Steady-State Configurations		
	Setting (A)	Setting (B)	Setting (C)
[15, 10, 10]	[33, 2, 0]	[34, 1, 0]	[34, 1, 0]
[18, 12, 11]	[38, 2, 1]	[40, 0, 1]	[39, 1, 1]
[21, 14, 12]	[40, 6, 1]	[45, 0, 2]	**[41,3,3]**
[24, 16, 13]	**[41,5,7]**	[48, 5, 0]	[39, 2, 12]
[27, 18, 14]	[40, 2, 17]	[51, 8, 0]	[39, 10, 10]
[30, 20, 15]	[38, 1, 26]	**[52,5,8]**	[39, 9, 17]

We also quantify the effects of PMM on power consumption for different arrival rates as well as service rates. The normalized power consumption can be calculated as

$$Pow([N_h, N_w, N_c]) = \frac{N_h}{N_t} + \frac{N_w}{N_t} * \psi \qquad (19)$$

in which $N_t = N_h + N_w + N_c$ and ψ is the ratio of power consumption of a warm PM to a hot PM (set to 60% according to [41], [42]). Therefore, when all PMs are busy the normalized power consumption of the cloud center is 1. As can be seen in the Fig. 11, in case of weaker configurations (i.e., [15,10,10] and [18,12,11]), PMM cannot achieve significant power saving since all of PMs are in hot mode most of the time. However, in case of larger configurations, PMM can achieve substantial power savings, even up to 50%, by turning down the idle PMs while still maintaining an acceptable level of QoS.

6 CONCLUSIONS

In this paper, we have developed an interacting analytical model that captures important aspects including resource assigning process, virtual machine deployment, pool management and power consumption of nowadays cloud centers. The performance model can assist cloud providers to predict the expected servicing delay, task rejection probability, steady-state arrangement of server pools and power consumption. We carried out extensive numerical experiments to study the effects of various parameters such as arrival rate of super-tasks, task service time, virtualization degree, super-task size and pool check rate on the task rejection probability, response time and normalized power consumption. The behavior of cloud center for given configurations has been characterized in order to facilitate the capacity planning, SLA analysis, cloud economic analysis and trade offs by cloud service providers. Using the proposed pool management model, the most appropriate arrangement of server pools and the amount of required electricity power can be identified in advance for anticipated arrival process and super-task characteristics.

In future work, we plan to design an availability model on top of our performance model in order to fully support different what-if analysis (e.g., SLA analysis, trade-offs, optimization) during design, implementation, testing and operational phases of a cloud service and, possibly, suggesting an architecture for a software tool to integrate proposed models and algorithms.

REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, 2009.

[2] An amazon.com company, "Amazon Elastic Compute Cloud, Amazon EC2," Website, Last accessed Oct. 2012, http://aws.amazon.com/ec2.

[3] IBM, "IBM Cloud Computing," Website, Last accessed July 2012, http://www.ibm.com/ibm/cloud/.

[4] M. Martinello, M. Kaniche, and K. Kanoun, "Web service availability–impact of error recovery and traffic model," *Reliability Engineering System Safety*, vol. 89, no. 1, p. 616, 2005.

[5] N. Sato and K. S. Trivedi, "Stochastic modeling of composite web services for closed-form analysis of their performance and reliability bottlenecks," *Service Oriented Computing ICSOC 2007*, pp. 107–118, 2007.

[6] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini, "QoS-aware clouds," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, Jul. 2010, pp. 321–328.

[7] H. Khazaei, J. Mišić, and V. B. Mišić, "Performance analysis of cloud computing centers using $M/G/m/m+r$ queueing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, p. 1, 2012.

[8] F. Longo, R. Ghosh, V. K. Naik, and K. S. Trivedi, "A scalable availability model for infrastructure-as-a-service cloud," *Dependable Systems and Networks, International Conference on*, pp. 335–346, 2011.

[9] R. Ghosh, K. S. Trivedi, V. K. Naik, and D. S. Kim, "End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach," in *Proceedings of the 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing*, 2010, pp. 125–132.

[10] H. Khazaei, J. Mišić, and V. B. Mišić, "Performance analysis of cloud centers under burst arrivals and total rejection policy," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, Dec. 2011, pp. 1–6.

[11] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May. 2011, pp. 104–113.

[12] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 931–945, Jun. 2011.

[13] L. Youseff, R. Wolski, B. Gorda, and R. Krintz, "Paravirtualization for hpc systems," in *In Proc. Workshop on Xen in High-Performance Cluster and Grid Computing*. Springer, 2006, pp. 474–486.

[14] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: The montage example," *2008 SC International Conference for High Performance Computing Networking Storage and Analysis*, vol. C, no. Nov., pp. 1–12, 2008.

[15] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for science grids: a viable solution?" *Proceedings of the 2008 international workshop on Dataaware distributed computing DADC 08*, pp. 55–64, 2008.

[16] E. Walker, "Benchmarking Amazon EC2 for high-performance scientific computing," *LOGIN*, vol. 33, no. 5, pp. 18–23, 2008.

[17] L. Wang, J. Zhan, W. Shi, Y. Liang, and L. Yuan, "In cloud, do mtc or htc service providers benefit from the economies of scale?" *Proceedings of the 2nd Workshop on ManyTask Computing on Grids and Supercomputers MTAGS 09*, vol. 2, pp. 1–10, 2010.

[18] S. Saini, D. Talcott, D. Jespersen, J. Djomehri, H. Jin, and R. Biswas, "Scientific application-based performance comparison of sgi altix 4700, ibm power5+, and sgi ice 8200 supercomputers," *2008 SC International Conference for High Performance Computing Networking Storage and Analysis*, pp. 1–12, Dec. 2008.

[19] S. Alam, R. Barrett, M. Bast, M. R. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, and J. S. Vetter, "Early evaluation of IBM BlueGene," *2008 SC International Conference for High Performance Computing Networking Storage and Analysis*, pp. 1–12, May 2008.

[20] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, "C-meter: A framework for performance analysis of computing clouds," in *CCGRID09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 472–477.

[21] K. Xiong and H. Perros, "Service performance and analysis in cloud computing," in *IEEE 2009 World Conference on Services*, 2009, pp. 693–700.

[22] B. Yang, F. Tan, Y. Dai, and S. Guo, "Performance evaluation of cloud service considering fault recovery," in *First Int'l Conference on Cloud Computing (CloudCom) 2009*, Dec. 2009, pp. 571–576.

[23] H. Khazaei, J. Mišić, and V. B. Mišić, "Modeling of cloud computing centers using $M/G/m$ queues," in *The First International Workshop on Data Center Performance*, Mar. 2011.

[24] ——, "Performance of cloud centers with high degree of virtualization under batch task arrivals," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. PrePrints, p. 1, 2012.

[25] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi, "Quantifying resiliency of IaaS cloud," *IEEE Symposium on Reliable Distributed Systems*, vol. 0, pp. 343–347, 2010.

[26] H. Qian, D. Medhi, and K. S. Trivedi, "A hierarchical model to evaluate quality of experience of online services hosted by cloud computing," in *Integrated Network Management (IM), IFIP/IEEE International Symposium on*, May. 2011, pp. 105–112.

[27] H. Khazaei, J. Mišić, and V. B. Mišić;, "A fine-grained performance model of cloud computing centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. PrePrints, p. 1, 2012.

[28] H. Takagi, *Queueing Analysis*. North-Holland, 1993, vol. 2: Finite Systems.

[29] G. Grimmett and D. Stirzaker, *Probability and Random Processes*, 3rd ed. Oxford University Press, Jul 2010.

[30] D. P. Heyman and M. J. Sobel, *Stochastic Models in Operations Research*. Dover, 2004, vol. 1.

[31] H. Takagi, *Queueing Analysis*. North-Holland, 1991, vol. 1: Vacation and Priority Systems.

[32] L. Kleinrock, *Queueing Systems, Volume 1, Theory*. Wiley-Interscience, 1975.

[33] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, 2nd ed. Wiley, 2001.

[34] V. Mainkar and K. S. Trivedi, "Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative models," *Software Engineering, IEEE Transactions on*, vol. 22, no. 9, pp. 640–653, Sep. 1996.

[35] Maplesoft, Inc., "Maple 15," Website, Last accessed Mar. 2011, http://www.maplesoft.com.
[36] SearchDataCenter.com, "The data center purchasing intentions survey report," Special Report, 2008, http://searchdatacenter.techtarget.com.
[37] G. Somani and S. Chaudhary, "Application performance isolation in virtualization," in *IEEE International Conference on Cloud Computing, CLOUD '09.*, Sep. 2009, pp. 41–48.
[38] K. Ye, X. Jiang, D. Ye, and D. Huang, "Two optimization mechanisms to improve the isolation property of server consolidation in virtualized multi-core server," in *12th IEEE International Conference on High Performance Computing and Communications (HPCC)*, Sep. 2010, pp. 281–288.
[39] S. C. Borst, "Optimal probabilistic allocation of customer types to servers," *SIGMETRICS Perform. Eval. Rev.*, vol. 23, pp. 116–125, May. 1995.
[40] J. Sethuraman and M. S. Squillante, "Optimal stochastic scheduling in multiclass parallel queues," *SIGMETRICS Perform. Eval. Rev.*, vol. 27, pp. 93–102, May. 1999.
[41] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," *SIGPLAN Not.*, vol. 44, no. 3, pp. 205–216, Mar. 2009.
[42] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Performance Evaluation*, vol. 67, no. 11, pp. 1155–1171, 2010.

Hamzeh Khazaei received his B.S. (2004) and M.S. (2008) degrees in computer science from Amirkabir University of Technology (Tehran Polytechnic), Iran. Currently he is a Ph.D candidate in Department of Computer Science, University of Manitoba, Canada. His research interests include cloud computing, modeling and performance evaluation of computing systems and queuing theory.

Jelena Mišić (M91, SM08) received her PhD degree in Computer Engineering from University of Belgrade, Serbia, in 1993. She is currently Professor of Computer Science at the Ryerson University, Toronto, Canada. Her current research interests include wireless networks and security in wireless networks. She is a member of IEEE and ACM.

Vojislav B. Mišić received his PhD in Computer Science from University of Belgrade, Serbia, in 1993. He is currently Professor of Computer Science at Ryerson University in Toronto, Ontario, Canada. His research interests include wireless networks and systems and software engineering.

Saeed Rashwand received his M.Sc. degree in Computer Science from Amirkabir University of Technology, Iran in 2008. He is currently a Ph.D. candidate in the department of Computer Science, University of Manitoba, Canada. His current research interests include performance evaluation, design, and implementation of wireless body area networks, wireless sensor networks, mobile ad-hoc networks, and wireless local area networks as well as cloud computing.