Prioritization of Overflow Tasks to Improve Performance of Mobile Cloud

Haleh Khojasteh, Jelena Mišić, and Vojislav B. Mišić
Ryerson University, Toronto, Canada M5B 2K3

***Abstract*—Mobile devices may offload their applications to a virtual machine running on a cloud host. This application may fork new tasks which require virtual machines of their own on the same physical machine. Achieving satisfactory performance level in such a scenario requires flexible resource allocation mechanisms in the cloud data center. In this paper we present two such mechanisms which use prioritization: one in which forked tasks are given full priority over newly arrived tasks, and another in which a threshold is established to control the priority so that full priority is given to the forked tasks if their number exceeds a predefined threshold. We analyze the performance of both mechanisms using a Markovian multiserver queueing system with two priority levels to model the resource allocation process, and a multi-dimensional Markov system based on a Birth-Death queueing system with finite population, to model virtual machine provisioning. Our performance results indicate that the threshold-based priority scheme not only performs better, but can also be tuned to achieve the desired performance level.**

***Index Terms*—performance; emerging technologies; mobile cloud computing.**

I. Introduction

The tension between resource-hungry applications such as face recognition, natural language processing, interactive gaming, and augmented reality, and resource- and energy-constrained mobile devices poses a significant challenge for current and future mobile platform development. Mobile cloud computing, where mobile devices can offload some computational jobs to the cloud is envisioned as a promising approach to address such a challenge [1]. While stationary clouds incur relatively long setup times, are leased for long time periods, and enjoy uninterrupted network connectivity, requests offloaded from mobile devices usually require quick response, may be infrequent, and are subject to variable network connectivity [2]. Also, the volume of workload to be offloaded may not be known in advance since many of the offload requests are the consequence of decisions made by the (generally unpredictable) human user of the device.

In this work, we address the elasticity in mobile cloud computing with a solution that allocates resources for on-demand job requests in the mobile clouds. We assume that the decision to offload has been made and focus on the allocation of cloud resources to the offloaded applications sent to a cloud data center. Namely, jobs offloaded by mobile devices are executed by virtual machines (VMs) hosted on physical machines (PMs) in a mobile cloud. During their lifetime, these jobs (also referred to as primary tasks) can fork new, secondary tasks; a job is completed when all the forked tasks complete their service. As secondary tasks need to communicate with the primary task as well as with each other, their allocated VMs should preferably be hosted on the same PM as the parent task's VM. However, the host PM may not have the resources required to execute the secondary task, which is then queued as 'overflow' tasks in order to find a new 'home'.

The proposed solution manages these two types of tasks as two service classes using a queueing model based on integration of multi-dimensional Markov system and Birth-Death queueing systems with multiple servers and finite population ($M/M/L//L$), inspired by the Birth-Death queueing systems developed in [3]. We consider soft bounds on completion times and limit the number of secondary tasks in order to prevent resource hogging. We also consider priority differentiation between the tasks, which is implemented using two mechanisms. In the first mechanism, overflow tasks are always serviced before any regular tasks, be they primary or secondary. In the second, we impose a threshold for the number of overflow tasks in the input queue. As long as the number of overflow tasks is below the threshold, a probabilistic selection similar to Weighted Fair Queueing [4] is used; otherwise, only overflow tasks are serviced until their number drops below the threshold.

The paper is organized as follows: in Section II, we survey existing work on resource allocation in mobile cloud computing, but also some relevant research results pertaining to stationary clouds. Section III describes the proposed resource allocation module while Section IV describes the virtual machine provisioning module and integration of the two modules. Section V discusses the performance of our system and the related outcomes. Section VI includes the implementation discussions. Section VII concludes the paper and discusses some directions for future work.

II. Related Work

Several research studies have proposed solutions to address the issues of computational power and battery lifetime of mobile devices by offloading computing tasks to cloud. CloneCloud [5] extends the concept of VM-based clone cloud offloading from LAN surrogates to cloud servers, and introduces an OS supporting VM migration. MAUI [6] has provided method-level code offloading based on the .NET framework. MAUI aimed to optimize energy consumption of a mobile device by estimation and evaluating the tradeoff between the energy consumed by local processing versus the transmission of code and data for cloud offloading. A framework for moving smartphone application processing to the cloud center using the concept of smartphone virtualization was introduced in ThinkAir [7]; it addresses lack of scalability by creating VM of a complete smartphone system on the cloud. CMcloud [8] is a mobile-to-cloud offloading platform which attempts to minimize both the server costs and the user service fee by offloading as many mobile applications to a single server as possible, while trying to satisfy the target performance of all applications. To achieve such goals, CMcloud exploited architecture performance modeling and server migration techniques. In Properly Offloading Mobile Applications to Clouds (POMAC) framework [9], other than offloading decision making technique, an offloading mechanism was designed through method interception at Dalvik virtual machine level to allow mobile applications to offload their computation intensive methods.

In most of the works related to resource allocation in mobile cloud computing, there are some trade-offs among power consumption, QoS parameters and costs. These objectives are usually dependent on cloud resources, applications profiles and network parameters. COSMOS (Computation Offloading as a Service for Mobile Devices) system [2] received mobile user computation offload demands and allocated them to a shared set of compute resources that was dynamically acquired (through leases) from a commercial cloud service provider.

The partitioning of elastic mobile datastream applications was formulated in [10] as on optimization problem by minimizing the cost function which is combination of communication energy and computation energy.

In [11], a model has been built to incorporate pertinent characteristics of the workflow software and network hardware devices. Then, the objective functions have been constructed which guide the offloading decisions. A heuristic algorithm was presented that produced offloading plans according to these objective functions and their variations.

In [12], offloading requests were sent in bundles so that, the period of time that the network interface stays in the high-power state can be reduced. Two online algorithms were presented, collectively referred to as Ready, Set, Go (RSG), that make near-optimal decisions on how offloading requests from multiple applications are to be best coalesced.

In the work presented in [13], in order to realize resource allocation, authors have estimated a cost model for each VM running on a server in the cloud and they have calculated the sum of the costs required to run the physical resources required on the server.

In another approach to connecting mobile devices to cloud servers in [14], authors have proposed Hermes, a polynomial time approximation scheme (FPTAS) algorithm to solve the latency problem.

The model proposed in [15] is based on the wireless network cloud (WNC) concept and a multi-objective linear optimization approach using an event-based finite state model and dynamic constraint programming method has been used to determine the appropriate transmission power, process power, cloud offloading and optimum QoS profiles.

In [16], a study on virtual machine deployment was presented together with an evaluation of the impact of VM deployment and management for application processing by analyzing the parameters such as VM deployment and execution time of applications. The work presented in [17], has analyzed the impact of performance metrics on the execution of applications (cloudlets).

The work in [18] has presented a task scheduling and resource allocation scheme which used the continually updated data from the loosely federated General Packet Radio Service (GPRS) to automatically select appropriate mobile nodes to participate in forming clouds.

Resource provisioning in stationary cloud computing has been extensively studied. By taking advantage of Lyapunov optimization techniques, an online decision algorithm was designed for request distribution which achieves the average response time arbitrarily close to the theoretically optimum and controls the outsourcing cost based on a given budget [19].

The work in [20] has proposed two different mechanisms, which reflect two different classical economic approaches for fairly allocating resources: the Nash Bargaining (NB) mechanism and the Lexicographically Max-Min Fair (LMMF) mechanism.

The work presented in [21] has proposed a randomized auction mechanism based on an application of smoothed analysis and randomized reduction, for dynamic VM provisioning (pricing tailor-made VMs on the spot) and pricing in geo-distributed cloud data centers. An online procurement auction mechanism to address the resource pooling issue in cloud storage systems was presented in [22].

We note that Markov models with multiple priority classes have been used in different fields. For example, a Markov chain flow decomposition for a two-class priority queue in presented in [26]. Also, threshold-based priorities have been utilized in the development of Markov models. For instance, in [27], a multi-server queueing system with two priority classes was used with a threshold defined according to number of servers in the system. In another similar approach, a threshold based Markov chain system has been deployed to model elastic and inelastic traffic flows in TCP-friendly admission control; the threshold was defined according to some inelastic flow parameters [28].

Similar to the work presented in this paper, some of the cloud resource allocation solutions have used queueing theory: e.g., authors in [23] proposed a performance model for systems with dynamic service demand where job size in number of tasks varies during service. It is assumed that the size of a job in number of tasks varies randomly during the time that job is in the system. The arrival of the jobs to the system is according to a Poisson process with parameter λ jobs/sec. Also, it is assumed that a new arriving job to the system initially demands service for a single task. A job generates random number of tasks according to a Poisson process with parameter α task/job/sec during its service time in the system. It is assumed that each task requires a VM for its execution and task execution times are exponentially distributed. Service time of a job begins with its arrival to the system and it is completed when there are no more tasks belonging to that job left in the system. In this model, a job has a general service time distribution.

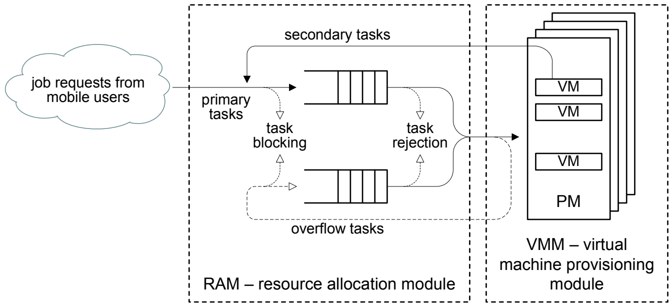

secondary tasks
job requests from mobile users
primary tasks
task blocking
task rejection
VM
VM
VM
PM
overflow tasks
RAM – resource allocation module
VMM – virtual machine provisioning module

Fig. 1. Overview of the system model.

In [24] a resource allocation model for IaaS cloud datacenters has been presented which is based on cloud federation mechanism. The arrival of the jobs is either according to a homogeneous Poisson process or a Markov Modulated Poisson Process (MMPP) which allows time variations in the arrival rate. Also, each job requires a single VM to complete its service; service times are exponentially distributed and mean service time is a function of the number of busy VMs on a server. The system has a finite queue, which is managed according to the FCFS discipline. The system queue has a finite size and once it is full, further requests are rejected. A federation threshold is defined on the number of jobs waiting in this queue; when the limit of threshold is reached, jobs will be redirected to another queue called upload queue. The jobs waiting in upload queue will be transferred to the other cloud datacenters participating in the cloud federation. Jobs in this solution include a single task and the model is not suitable for on-demand job requests as their size varies during the service time.

The cloud resource allocation model suggested in [25] includes fault recovery. In this model, arrival of jobs is according to a general stochastic process and each job has random number of tasks. Each task requires a single VM and task service times are exponentially distributed, which results in independent task completion times. The system has a finite queue and each task of a job takes a position in the queue. A job is rejected if all of its tasks are not accepted in the system. It is assumed that all the tasks in a job will start to get service simultaneously. The system has been modeled as a $GI^{[X]}/M/S/N$ queue where N corresponds to the maximum number of allowed tasks in the system. Also, the VM failure rate in this model is a Poisson process and VM recovery times are exponentially distributed. This model is not appropriate for on-demand job requests in mobile cloud system as all the tasks in a job are supposed to start getting service at the same time.

III. Resource allocation

Fig. 1 provides a schematic overview of the proposed solution. We assume that jobs offloaded from a mobile device arrive according to a Poisson process with arrival rate λ. When the job request reaches the data center, it is queued in the new task queue. Tasks which can't be admitted due to a full queue are blocked.

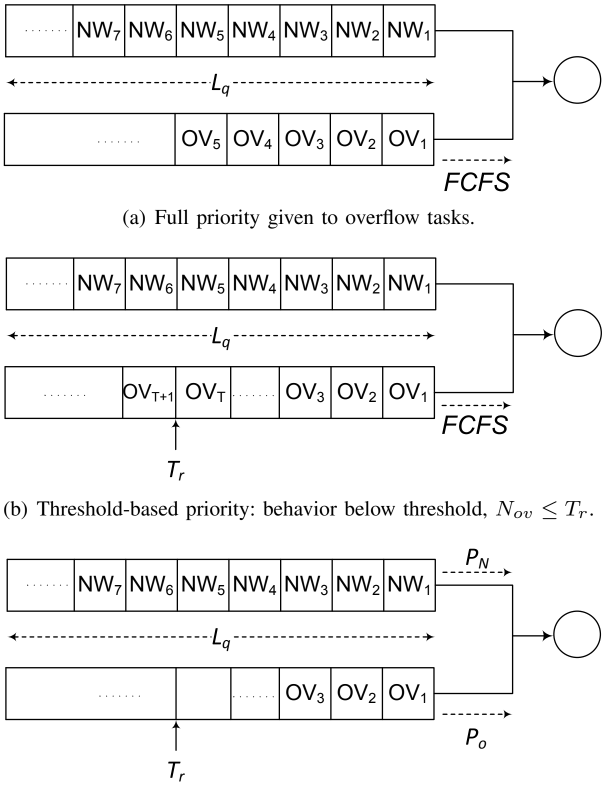
Once the task reaches the head of the queue, the resource allocation module (RAM) attempts to find a PM that can accommodate the task – i.e., a single VM equipped with appropriate OS and applications, running on a PM that has sufficient spare capacity. If such a PM is found, the appropriate VM will be instantiated with the queued task; otherwise, the task is rejected. Let $1/\beta$ denote the mean look up time to find appropriate PM in the server pool. RAM is modeled as a multi-dimensional Continuous Time Markov chain (CTMC) presented in Figs. 3 and 4; more detailed explanation of this chain is given below.

If the computational resources of the VM allocated to the primary task are insufficient, additional VMs are forked to fulfill the requirements. Forking creates a secondary VM, which is an independently executing clone of the primary VM [30]. Secondary or forked tasks are queued and processed similar to the primary tasks, but with an important constraint: namely, that all communications with the mobile device must be routed through the primary VM. This constraint has two consequences in practice: first, all VMs running secondary tasks must end before the VM running the parent primary task. Second, secondary VMs should be instantiated on the same PM that host the VM running the parent task which facilitates communication between them.

However, if the PM running the parent (primary) VM has no spare capacity for a secondary one, the secondary task will not be immediately blocked. Instead, it will be returned to the RAM as an overflow task; these tasks are routed through a dedicated queue, separately from the newly arrived primary tasks and first-time secondary tasks. We note that an overflow task can still be blocked if the required computational capacity can't be found when that task reached the head of the overflow queue.

Using separate queues allows us to prioritize secondary and, in particular, overflow tasks. The objective is twofold: first, to reduce the wait time for mobile applications; second, to minimize the probability that an offloaded job will have to be aborted because it is unable to fork the required secondary task. Both of these goals, in fact, strive to increase user satisfaction. A similar scenario is observed in cellular networks, where handover calls, which are continuations of existing calls, are always given priority over new calls [31].

Prioritization is implemented in two ways. In the simpler approach, hereafter referred to as full priority scheme, priority is always given to overflow tasks. In this scheme, which is schematically shown in Fig. 2(a), primary and first-time secondary task requests in the new arrivals queue will not get service as long as there is a task waiting in the overflow queue. However, this may lead to unnecessary penalization of tasks in the new arrivals queue, which is why we have also considered another approach, hereafter referred to as threshold-based priority scheme. In this scheme a threshold is set in the overflow queue. As long as the number of overflow tasks in the queue, N_{ov}, is not above the threshold, T_r, i.e., $N_{ov} \leq T_r$, as shown in Fig. 2(b), overflow tasks and new incoming tasks get service according to the probabilities of P_o and P_N, respectively, similar to Weighted Fair Queueing (WFQ) method [4]. However, once the number of tasks in the



(a) Full priority given to overflow tasks.

(b) Threshold-based priority: behavior below threshold, $N_{ov} \leq T_r$.

(c) Threshold-based priority: behavior above threshold, $N_{ov} > T_r$.

Fig. 2. The service order in the dual input queue system.

TABLE I
PARAMETER DEFINITION.

Parameter	Description
N	Number of servers
N_{ov}	Number of waiting overflow tasks in RAM
m	Number of available VMs on the PM
λ	Incoming task rate
$1/\beta$	Mean look up time to find a PM
$1/\gamma$	Mean clean up time in RAM
λ_i	Primary task arrival rate into the PMs
λ_{ci}	Secondary task generation rate in a job
μ	Mean service time of primary tasks
d	Mean service time of secondary tasks
ρ	Offered load
L_q	Size of queues in RAM
L	Maximum number of secondary tasks in a job
c	Minimum number of jobs accommodable on a PM
O_i	Incoming overflow rate from the RAM to VMM
O_o	Outgoing overflow rate from the VMM to RAM
T_r	Threshold of number of waiting overflow tasks in RAM
P_o	Probability of giving service to the overflow tasks
P_N	Probability of giving service to the new incoming tasks
P_s	Successful provisioning probability
P_{bq}	Blocking probability due to full RAM
P_{br}	Rejection probability due insufficient resources
P_{rj}	Total rejection probability
φ	Basic instantiation/ Full transition rate
φ_x	Partial transition rate

overflow queue exceeds the threshold, i.e., when $N_{ov} > T_r$, the system exclusively services overflow tasks, as in the first approach, until the number of tasks in the overflow queue drops below the threshold. In both cases, tasks from either queue will be serviced in FCFS order. The computational details of P_o and P_N are presented in Section III-B.

We will now describe the queueing model for both schemes in more detail, using the parameters listed in Table I.

A. Resource allocation with full priority of overflow tasks

Resource allocation in case overflow tasks have full priority, i.e., new arrival tasks will not be served as long as there is an overflow task in the appropriate queue, is modeled as a Markovian multiserver queueing system with two priority levels. In this model, illustrated in Fig. 3, states are labeled as (i, j, k) where i and j indicate the number of tasks in the overflow and new arrivals queue, respectively, and k denotes the admission mode: 'A' means that a task is accepted while 'R_o' ('R_N') means that an overflow (new) task is rejected. The length of the queue buffer is L_q.

New tasks arrive with rate of λ, while overflow tasks arrive at a rate of O_o. If this task is accepted, the system moves to the state $(0, j-1, A))$ at a rate of $P_s\beta$, where P_s is the probability of finding appropriate VM in the Virtual Machine provisioning Module (VMM), which we will derive in Section IV below, and $1/\beta$ is the look-up time needed to find a suitable PM. Otherwise, the system moves to $(0, j+1, A)$ which means the new task is added to the waiting new tasks. As overflow tasks have absolute priority, new arrivals – primary as well as first-time secondary ones – can get service only if there are no overflow tasks in the queue, which corresponds to $j = 0$, i.e., the first row of the model.

If the queue is full, a new task cannot be admitted and the system moves from state $(0, L_q, A)$ to $(0, L_q, R_N)$ at a rate of $\beta(1-P_s)$. The target state $(0, L_q, R_N)$ denotes blocking of a new arrival task; it is shown shaded in the top right corner of Fig. 3. When the task is rejected, the system moves back to $(0, L_q, A)$ at a clean-up rate of $\gamma = 10\beta$.

Overflow tasks arrive at a rate of O_o: if such a task gets service (and they always have priority), the system moves to the first neighboring state up with a rate of $P_s\beta$; otherwise, the task is queued and the system moves to the next state below the current state. Yet overflow tasks can also be rejected if the queue is full, which is represented by the additional shaded states below the bottom line with states (L_q, j, A). The rate of rejection is $\beta(1-P_s)$, as is the case with new arrivals; the system goes back to (L_q, j, A) at a clean-up rate of γ.

A task may be blocked due to a full queue; it occurs with the probability

$$P_{bq} = \sum_{i=0}^{L_q-1} \pi(i, L_q, A) + \sum_{j=0}^{L_q} \sum_{k \in S_2} \pi(L_q, j, k) + \pi(0, L_q, R_N) \tag{1}$$

where $S_2 = \{A, R_o\}$.

A task can also be rejected due to insufficient resources with

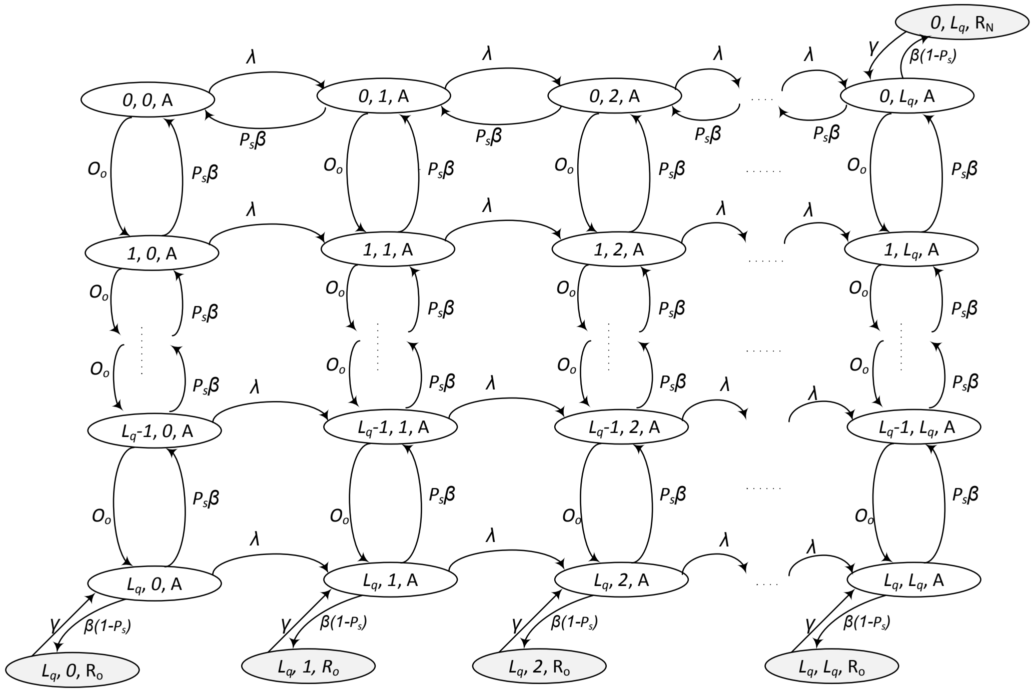


Fig. 3. Markovian model of resource allocation in case overflow tasks have full priority.

the probability

$$P_{br} = \frac{\beta(1-P_s)}{\gamma}\pi(0, L_q, R_N) + \sum_{j=1}^{L_q} \frac{\beta(1-P_s)}{\gamma}\pi(L_q, j, R_o) \tag{2}$$

The total rejection probability is the sum of the two:

$$P_{rj} = P_{bq} + P_{br} \tag{3}$$

B. Resource allocation with threshold-based priority of overflow tasks

Resource allocation in case of threshold-based priority of overflow tasks over new arrivals is modeled with a two-dimensional Markovian model illustrated in Fig. 4. The model behaves in a manner similar to the previous one, but with an important distinction: namely, acceptance depends on the length of the overflow task queue. (As before, both new arrival and overflow queue can accommodate up to L_q tasks.) If the overflow queue contains more than T_r tasks, only overflow tasks are serviced as long as the queue length is above the threshold. If the overflow queue contains T_r or fewer tasks, tasks to be accepted are taken from one or the other queue: new tasks are accepted with a probability of $P_N = \frac{\lambda}{\lambda+O_o}$ while overflow tasks are accepted with a probability $P_o = 1 - P_N = \frac{O_o}{\lambda+O_o}$, where λ and O_o denote arrival rates for new and overflow tasks, respectively.

As before, shaded states outside of the two-dimensional chain denote rejection states pertaining to new arrivals (in the top right) and overflow tasks (at the bottom).

In this case, task blocking probability is

$$P_{bq} = \sum_{i=0}^{T_r} \sum_{k \in S_1} \pi(i, L_q, k) + \sum_{j=0}^{L_q} \sum_{k \in S_2} \pi(L_q, j, k) \tag{4}$$

where $S_1 = \{A, R_N\}$ and $S_2 = \{A, R_o\}$, and task rejection probability is

$$\begin{aligned} P_{br} = &\sum_{i=1}^{T_r} \frac{\beta(1-P_N P_s)}{\gamma}\pi(i, L_q, R_N) \\ &+ \sum_{j=1}^{L_q} \frac{\beta(1-P_s)}{\gamma}\pi(L_q, j, R_o) \end{aligned} \tag{5}$$

Total rejection probability is, then, equal to their sum: $P_{rj} = P_{bq} + P_{br}$.

IV. Virtual machine provisioning

Virtual Machine provisioning Module (VMM) manages instantiation, provisioning and deployment of VMs. Fig. 5 shows a multi-dimensional Continuous Time Markov chain (CTMC) that models the VMM within a PM. States are labeled as (i, j, k) where i indicates the tasks waiting to be serviced, while j and k denote the number of jobs (i.e., primary tasks) and secondary tasks currently in service, respectively. O_i denotes the incoming overflow rate coming in from the RAM, while O_o represents the rate of tasks that can't be accommodated in the PM that will be returned to the RAM as overflow. The task arrival rate to each PM, λ_i, can be obtained as

$$\lambda_i = \frac{\lambda(1-P_{bq})}{N} \tag{6}$$

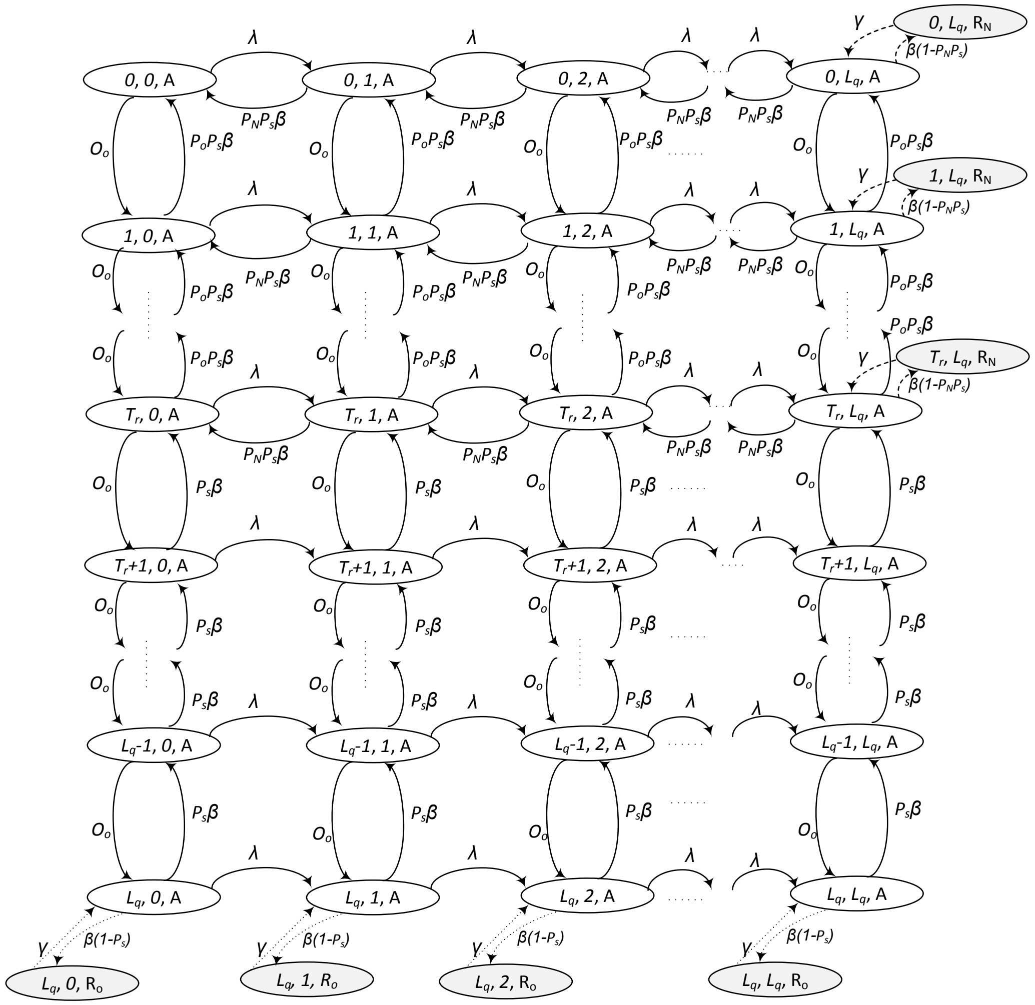


Fig. 4. Markovian model of resource allocation in case overflow tasks have threshold-based priority.

where P_{bq} is the blocking probability obtained from the RAM (which will be explained below) and N is the number of PMs in the system.

In Fig. 5, the main plane of the multi-dimensional Markov model illustrates the waiting queues and serving status of primary VMs. Service times for both primary and secondary tasks are exponentially distributed with mean values of μ and d, respectively, which includes the time needed for forking. The shaded states represent secondary task queues which can have up to L tasks. Secondary tasks are generated with a rate of λ_{ci}; they are modeled as Birth-Death queueing systems with finite population of L customers and L servers ($M/M/L//L$). This approach is inspired by the Birth-Death queueing systems with finite population presented in [3]. In this case, the stationary probability of kth state of the secondary task queue can be obtained as

$$p_k = p_0 \left(\frac{\lambda_{ci}}{d}\right)^k \binom{L}{k} \tag{7}$$

where p_0 is

$$p_0 = \left[\sum_{k=0}^{L} \left(\frac{\lambda_{ci}}{d}\right)^k \binom{L}{k}\right]^{-1} = \frac{1}{(1+\lambda_{ci}/d)^L} \tag{8}$$

by replacing the value of p_0 in equation 7, p_k is calculated as

$$p_k = \begin{cases} \frac{\left(\frac{\lambda_{ci}}{d}\right)^k \binom{L}{k}}{(1+\lambda_{ci}/d)^L} & 0 \le k \le L \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Mean number of tasks in the secondary task queue is

$$\overline{N_t} = \sum_{k=0}^{L} k p_k = \frac{\sum_{k=0}^{L} k \left(\frac{\lambda_{ci}}{d}\right)^k \binom{L}{k}}{(1+\lambda_{ci}/d)^L} = \frac{L\lambda_{ci}/d}{1+\lambda_{ci}/d} \tag{10}$$

Assuming the number of tasks in an offloaded job is limited to one primary and L secondary ones, and that each PM can accommodate up to m VMs, we have to make sure that $m > L$ so that at least one job (i.e., its primary task and all of its secondary tasks) can be accommodated on a single PM. In fact, a PM can accommodate at least $c = \left\lfloor \frac{m}{L+1} \right\rfloor$ jobs. However, not all jobs will have the maximum number of $L+1$ tasks;

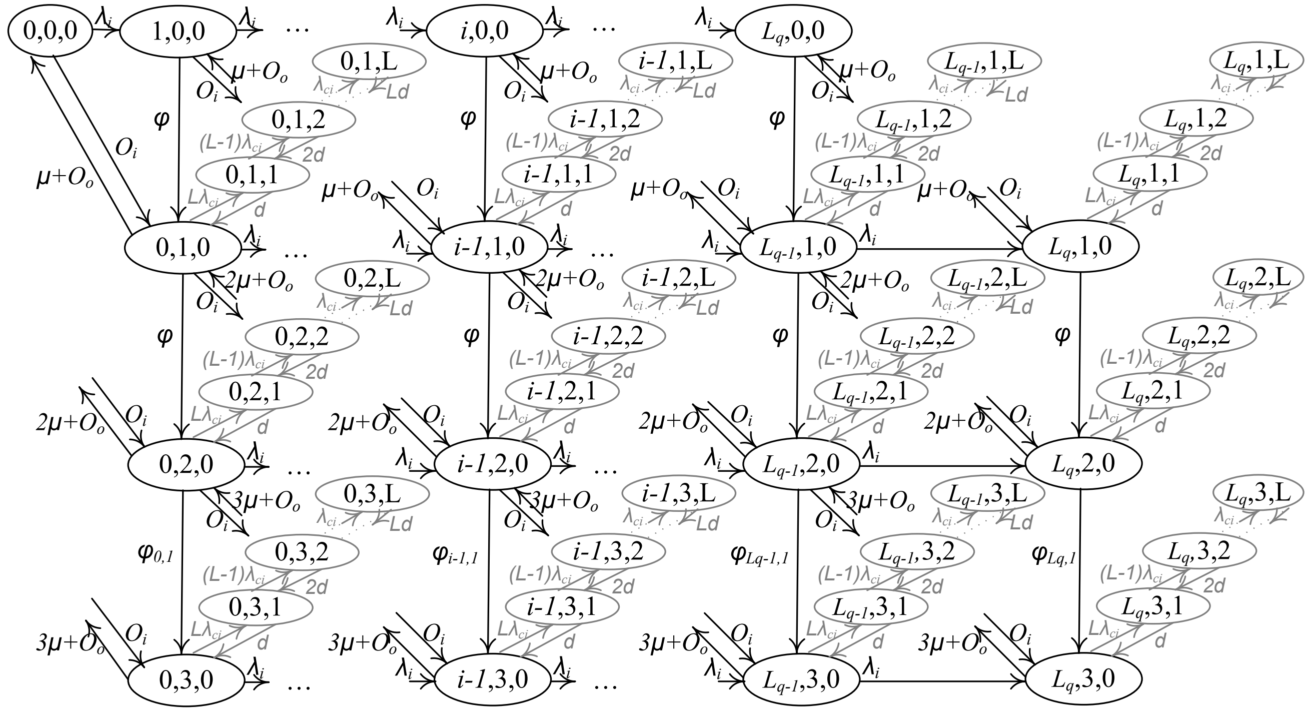


Fig. 5. Virtual machine provisioning model of a PM with the ability to accept maximum three jobs (see text for details).

therefore, if the number of jobs is limited to c, chances are that some of the VMs on the PM will be underused, so we can conservatively assume that a PM can accommodate $c+\frac{c}{2}$ job requests.

Assuming $m=10$ and $L=4$, we obtain $c=\left\lfloor\frac{10}{5}\right\rfloor=2$, and the number of jobs allowed on a PM is $c+\frac{c}{2}=3$; this last number is used in the provisioning module illustrated in Fig. 5. However, when the job number reaches c, the transition rate of moving to serve the next job changes. This transition rate is φ as long as the number of current jobs (i.e., primary tasks) on a PM is below c. When the limit of c jobs is reached, the transition rate changes to

$$\varphi_{i,x}=\varphi Pt_{i,x} \tag{11}$$

where i is the number of jobs waiting for service, while x indicates the number of serviced jobs above c; $Pt_{i,x}$ is the transition coefficient which can be obtained as the ratio of the sum of steady-state probabilities where the length of each secondary queue is larger than $\overline{N_t}$, and the sum of steady-state probabilities for full length of each secondary queue. As the average number of expected secondary tasks in a single secondary queue $\overline{N_t}$ was calculated in (10), the transition coefficient is

$$Pt_{i,x}=\frac{\sum\limits_{h=1}^{c+x}\sum\limits_{k=\overline{N_t}}^{L}p(i,j_h,k)}{\sum\limits_{r=1}^{c+x}\sum\limits_{k=1}^{L}p(i,j_r,k)} \tag{12}$$

where $c+x$ is the number of jobs in service and i is the number of jobs waiting for service.

Probability of overflow, i.e., that a task request cannot be deployed on the PM, is

$$P_{na}=p(L_q,0,0)+\sum_{k=1}^{L}\sum_{j=1}^{c+\frac{c}{2}}p(L_q,j,k)+\sum_{i=0}^{L_{q-1}}\sum_{y\in\Phi}P_y \tag{13}$$

where $p(i,j,k)$ indicates the steady-state probability of the corresponding state and P_y is a member of Φ, the set of products of probabilities of states corresponding to secondary queues for which the sum of the corresponding secondary VMs exceeds the capacity of secondary VMs on a PM. The probability that the total number of secondary VMs in a PM exceeds the available number of VMs is a combinatorial probability which can be computed as a sum of products in (13). Φ is represented as

$$\Phi=\left\{\prod_{l=1}^{c+\frac{c}{2}}p(i,j_l,k_l)\Big|\sum_{l=1}^{c+\frac{c}{2}}k_l>m-(c+\frac{c}{2})\right\} \tag{14}$$

where $m-(c+\frac{c}{2})$ denotes the number of allowed secondary VMs on a PM.

Then, probability of successful provisioning on a PM is

$$P_s=1-P_{na}^{N} \tag{15}$$

Finally, the overflow rate generated in a PM is derived from the probability that a task is blocked due to lack of resources, which may be obtained analogously to the Erlang B formula in a truncated system consisting of two independent queues in a multi-dimensional Markov system [32]:

$$O_o = \frac{\frac{(\lambda_i/\mu)^{c+\frac{c}{2}}}{(c+\frac{c}{2})!} \cdot \frac{(\lambda_{ci}/d)^{m-(c+\frac{c}{2})}}{[m-(c+\frac{c}{2})]!}}{\sum_{i=0}^{c+\frac{c}{2}} \sum_{j=0}^{m-(c+\frac{c}{2})} \frac{(\lambda_i/\mu)^i}{i!} \cdot \frac{(\lambda_{ci}/d)^j}{j!}} \tag{16}$$

V. Performance Evaluation

A. Practical considerations

The analytical model has been solved using Maple 16 from Maplesoft Inc. [29].

To evaluate the performance of the proposed mobile cloud system, we have solved the model described above in a number of different scenarios. As presented above, the overall model consists of two interactive stochastic modules which are solved as follows.

We assume that all the transition rates in the module are equal to φ and then solve the model to obtain the steady-state probability for all the states and overflow rate. Then, we calculate the $Pt_{i,x}$ for every level beyond c jobs in the system; according to these new transition rates, we solve the model again and compute the new values of steady-state probability of all the states. Using these values, we can calculate P_s. As the overflow rate is independent of steady-state probabilities, it is not needed to calculate it again. This procedure is shown as pseudo-code in Algorithm 1.

Algorithm 1 First Time Solving of VMM Module

```
Assume all transition rates equal to $\varphi$ and solve VMM;
Compute outgoing overflow rate, $O_o$;
Calculate $Pt_{i,x}$ coefficients for all states $i$ and levels $x$;
Solve VMM again with new transition rates $\varphi$ and $\varphi_{i,x}$);
Calculate $P_s$ with new values;
```

The successful provisioning probability P_s and the overflow rate O_o obtained in this manner are used as input parameters to solve the RAM module. It computes the task blocking probability, P_{bq}, which is the input parameter to VMM module. The overall model consists of two interactive stochastic modules The associated pseudocode is shown in Algorithm 2. Iteration ends when the difference between the values of probabilities in successive iterations drops below a predefined threshold (we have used $\Delta = 10^{-6}$). Note that transition rates are obtained only once, in the first pass of the VMM.

B. Task blocking probability

In the first scenario, we have varied the offered load: first, by keeping mean task service time μ fixed while varying mean task arrival rate λ; and second, by varying mean task service rate at fixed mean task arrival rate. The offered load was calculated as $\rho = \frac{\lambda}{mN\mu}$, where $N = 100$ is the total number of PMs in the system, each of which had up to $m = 10$ VMs. The queue capacity was set to $L_q = 50$ for both queues, while the threshold in the overflow queue was set to $T_r = 30$.

Task blocking probability obtained in this manner is shown in Figs. 6(a) and 6(b). As expected, probability of task blocking increases with the offered load. Overflow tasks are given priority – and, consequently, easier access to resources – under both full and threshold-based priority mechanisms, which is why the blocking probability is much lower for such tasks. However, when threshold-based priority is applied, blocking probability for newly arrived tasks is noticeably lower, while that for overflow tasks is slightly higher. This indicates that the performance for one or the other type of tasks may be adjusted within certain limits. In the worst case, less than 4% of overflow tasks are blocked.

Algorithm 2 The Integrated model Algorithm

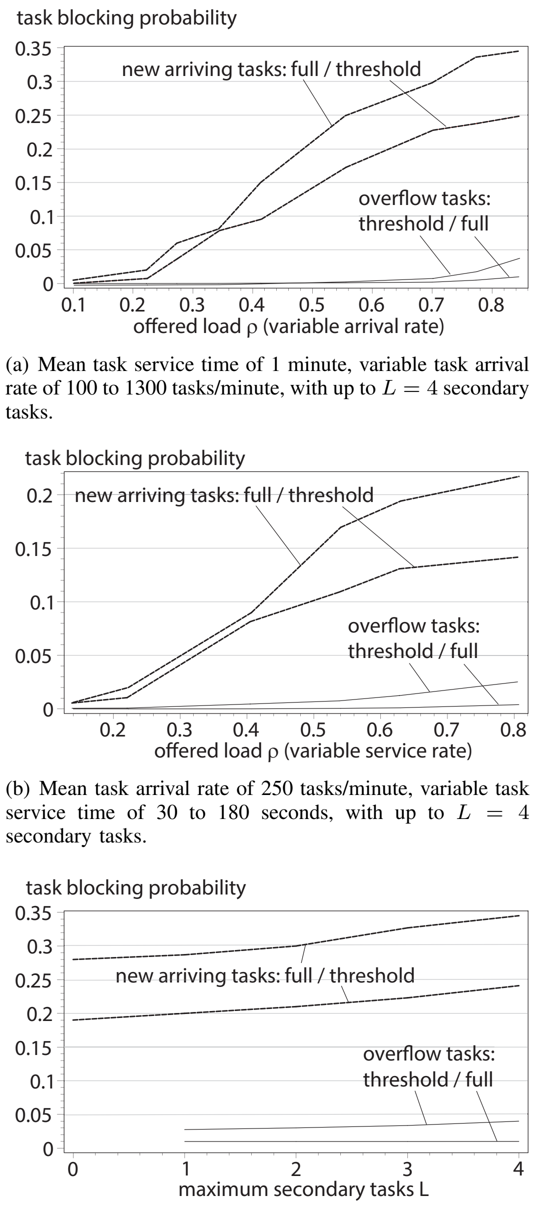
```
**Input:** Initial successful provisioning probability and overflow rate: $P_{s0}, O_{o0}$;
**Output:** Blocking probability in the RAM: $P_{bq}$;
count = 0; maximum = 30; $\Delta$ = 1;
$P_{bq0} \longleftarrow$ RAM $(P_{s0}, O_{o0})$;
**while** $\Delta \geq 10^{-6}$ **do**
    count $\longleftarrow$ count +1;
    $P_s \longleftarrow$ VMM $(P_{bq0})$;
    $O_o \longleftarrow$ VMM $(P_{bq0})$;
    $P_{bq1} \longleftarrow$ RAM $(P_s, O_o)$;
    $\Delta \longleftarrow |(P_{bq1} - P_{bq0})|$;
    $P_{bq0} \longleftarrow P_{bq1}$;
    **if** count == maximum **then**
        break;
    **end if**
**end while**
**if** count == maximum **then**
    **return** -1;
**else**
    **return** $P_{bq0}$;
**end if**
```

We have also investigated the blocking probability under fixed offered load but with a variable limit to the number of secondary tasks L; the results obtained under both service policies are shown in Fig. 6(c). Note that the case $L = 0$ corresponds to the absence of secondary tasks which, by extension, means that there are no overflow tasks; consequently, there is no corresponding data value for threshold-based priority curve. Again, new arriving tasks suffer a higher blocking rate which slowly increases with the task forking limit L; overflow tasks, on the other hand, are not affected much due to the dual-queue prioritization mechanism presented above. As before, threshold-based prioritization provides for much better performance for new tasks than its full priority counterpart.

We note that a rough upper bound for the probability that a job does not complete because a forked task is ultimately blocked may be obtained as the product of mean length of secondary task queue and probability of overflow, $P_{na}\overline{N_t}$.

C. Mean task delay

As for the mean task delays, threshold-based prioritization offers lower values (i.e., better performance), as can be seen in Fig. 7. As can be expected, mean delays increase rather sharply with the offered load. As the system operates well below saturation, rise in delay values is approximately linear.



(a) Mean task service time of 1 minute, variable task arrival rate of 100 to 1300 tasks/minute, with up to $L = 4$ secondary tasks.

(b) Mean task arrival rate of 250 tasks/minute, variable task service time of 30 to 180 seconds, with up to $L = 4$ secondary tasks.

(c) Mean task service time of 1 minute, mean task arrival rate of 1300 tasks/minute, maximum number of secondary tasks variable from $L = 0$ to 4.

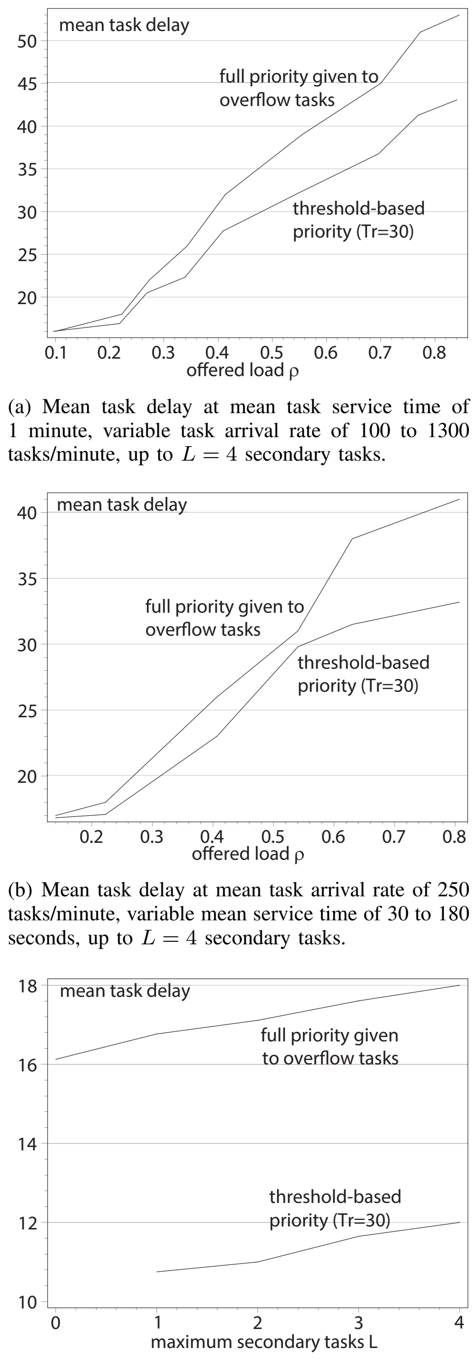
Fig. 6. Task blocking probability.

In case of variable task forking limit, the rise is somewhat milder, but this may be due to the comparatively low value of offered load utilized to generate data for Fig. 7(c).

We note that for the same offered load in the both priority cases, the cloud center generally appears to be more sensitive to the task arrival rate than to mean service time. This is due to the overhead imposed by the waiting times and provisioning processes which increases with the number of tasks but is independent of the task service time.

D. Utilization and the impact of queue threshold T_r

Server utilization is shown in Fig. 8. Under both prioritization policies, utilization increases with the offered load.



(a) Mean task delay at mean task service time of 1 minute, variable task arrival rate of 100 to 1300 tasks/minute, up to $L = 4$ secondary tasks.

(b) Mean task delay at mean task arrival rate of 250 tasks/minute, variable mean service time of 30 to 180 seconds, up to $L = 4$ secondary tasks.

(c) Mean task delay at mean task arrival rate of 250 tasks/minute and mean task service time of 1 minute, maximum number of secondary tasks variable from $L = 0$ to 4.

Fig. 7. Mean task delays.

As the system does not enter saturation, the rate of rise is approximately linear in both cases, although some flattening may be observed at offered load $\rho = 0.7$ and above. We note that utilization is slightly lower when full priority is given to the overflow tasks, compared to the threshold-based policy, since the number of overflow tasks is low.

Finally, Fig. 9 shows the effect of queue threshold in the threshold-based prioritization policy on task blocking

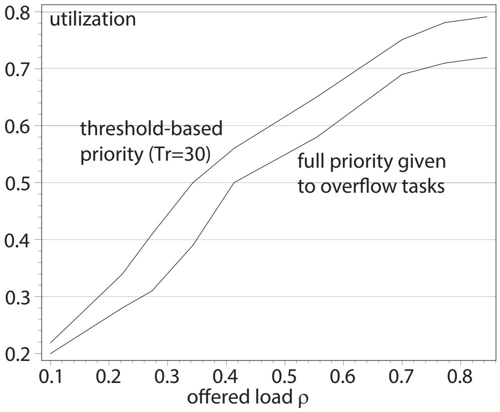


Fig. 8. Server utilization at constant service time of 1 minute, arrival rate of 100 to 1300 tasks/minute, with up to $L = 4$ secondary tasks.

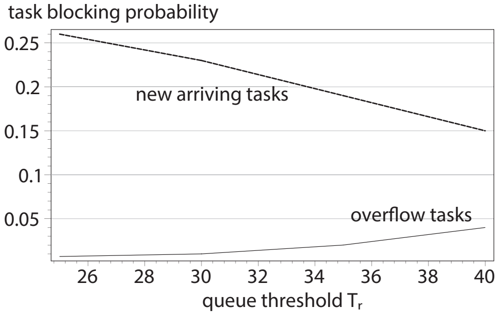


Fig. 9. Task blocking probability at constant service time of 1 minute and constant arrival rate 1000 tasks/minute, with up to $L = 4$ secondary tasks.

probability. As the threshold moves closer to the queue size of $L_q = 50$, value of the blocking probability for new and overflow tasks are getting closer to each other as the probability that the threshold will be exceeded diminishes. Conversely, lower values of the threshold push the system to behave in a manner closer to that under full priority – in fact, full priority policy is equivalent to a threshold-based one with threshold value of $T_r = 0$.

Overall, the threshold-based policy allows the cloud operator to fine-tune the performance of the cloud, as there are a number of parameters which can be adjusted to provide the desired values, or ranges thereof, for critical performance indicators such as mean delay and task blocking.

VI. Discussions

There are some points worth mentioning regarding the implementation of our resource allocation model.

We have assumed that both RAM and VMM modules are located in the same cloud datacenter.

It is possible to compute the time complexity of our algorithms through a hybrid solution of simulation and analytical modeling. The Maple engine can be integrated into simulation environments such as Simulink, NS2 simulator or OPNET Modeler. The Maple algorithms and data structures can be exported to the simulation blocks and the time spent in different steps can be obtained.

Choosing the optimal size of primary and overflow tasks queues, L_q, can be complicated. If the size of the queues is large, tasks will wait for long time in the queues to get served; whereas, if the size is small, significant number of new and overflow tasks will get blocked. Blocking the overflow tasks prevents the completion of the corresponding jobs.

Setting a limit for the maximum number of forked tasks, L, is also non-trivial. If the chosen limit is low, the running jobs cannot be completed as they are in need of more VMs for their secondary tasks. Another drawback is that the PM will accept more jobs than it can serve appropriately. Generally, selecting low values of L will incur under-provisioning of the resources. On the other hand, if the chosen size for L is high, the secondary task queues in the PM will not get full. Another pitfall is that defining large secondary task queue will prevent the PM from accepting more new jobs. Therefore, choosing high values of L will result in under-utilization in the cloud system.

The arrival rate can be considered as the components of Poisson processes with different intensities during the daytime or nighttime; therefore, the interarrival time of the tasks in components can be exponential. We have provided the spectrum of offered load in our scenarios and investigated the effect of the variability of arrival rate and service time on the system separately. We have assumed that the service times in different steps are exponential. Although we are aware that the service times are in nature sub-exponential, with assuming them as exponential, we have investigated the worst case scenario and evaluated the upper bound of the variables. In this paper, we have considered the general case of hypoexponential distribution as the service time of the system and we have decomposed the service time distribution into a linear combination of structured exponential distributions in different steps of the model [3].

VII. Conclusion and Future Work

We have developed two priority schemes for resource allocation of on-demand job requests in a cloud server pool based on giving different priorities to the overflow tasks with or without a predefined threshold. Unlike many of existing works, our model does not sacrifice the complexity of offloading problem just to make it solvable. Instead, complexity is addressed through the use of two interacting stochastic models which are solved through fixed point iteration to achieve any desired error level.

We have investigated the impact of task arriving rate, service time and the size of offloaded job on the performance metrics for both priority schemes. Also, we have evaluated the effect of threshold location on the threshold-based priority scheme. Our results confirm that threshold-based priority presents better system performance than full priority of overflow tasks.

Our next step will be the modification of threshold-based priority scheme in order to adjust the location of threshold in the overflow queue. Finding the best location of threshold is an optimization problem and the position can change according to the performance metrics, different policies adopted by cloud computing providers or cloud system's requirements. Also, we have observed that the performance metrics do not change linearly with regard to the offered load, which indicates that finding the settings of parameter values that would lead to optimal values of performance metrics is non-trivial.

References

[1] X. Chen. Decentralized Computation Offloading Game For Mobile Cloud Computing. *IEEE Trans. Parallel and Distributed Systems*, **26**(4):974–983, March 2015.

[2] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik and E. Zegura. COSMOS: computation offloading as a service for mobile devices. *Proc. 15th ACM Int. Symp. Mobile ad hoc networking and computing (MobiHoc)*, pp. 287–296, Philadelphia, PA, USA, August 2014.

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

[4] A. Demars, S. Keshav and S. Shenker, Analysis and Simulation of a Fair Queuing Algorithm. *ACM SIGCOMM '89*, pp. 3–26, Austin, TX, USA, September 1989.

[5] B. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. *The 6th ACM Conf. Computer systems (ACM EuroSys)*, pp. 301–314, Salzburg, Austria, April 2011

[6] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl. MAUI: making smartphones last longer with code offload. *the 8th Int. ACM Conf. Mobile Systems, Applications, and Services (MobiSys)*, pp. 49–62, San Francisco, CA, USA, June 2010.

[7] S. Kosta, A. Aucinas, P. Hui, R. Mortier and X. Zhang. ThinkAir: dynamic resource allocation and parallel execution in the cloud for mobile code offloading. *Proc. INFOCOM*, pp. 945–953, Orlando, FL, USA, March 2012.

[8] D. Chae, J. Kim, J. Kim, J. Kim, S. Yang, Y. Cho, Y. Kwon and Y. Paek. CMcloud: Cloud Platform for Cost-Effective Offloading of Mobile Applications. *14th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing*, pp. 434–444, Chicago, IL, May 2014.

[9] M. A. Hassan, K. Bhattarai, Q. Wei and S. Chen. POMAC: Properly Offloading Mobile Applications to Clouds. *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, pp. 1–6, Philadelphia, PA, June 2014.

[10] N. Kaushik and J. Kumar. A Computation Offloading Framework to Optimize Energy Utilization in Mobile Cloud Computing Environment. *Int. J. of Computer Applications & Information Technology*, **5**(2):61–69, April/May 2014.

[11] B. Gao, L. He, L. Liu, K. Li and S. A. Javis. From Mobiles to Clouds: Developing Energy-aware Offloading Strategies for Workflows. *ACM/IEEE 13th Int. Conf. Grid Computing (GRID)*, pp. 139–146, Beijing, China, September 2012.

[12] L. Xiang, S. Ye, Y. Feng, B. Li and B. Li. Ready, Set, Go: Coalesced Offloading from Mobile Devices to the Cloud. *Proc. INFOCOM*, pp. 2373–2381, Toronto, Canada, April 2014.

[13] M. J. O'Sullivan and D. Grigoras. Integrating mobile and cloud resources management using the cloud personal assistant. *Simulation Modelling Practice and Theory*, **50**:20–41, January 2015.

[14] Y. Kao, B. Krishnamachari, M. Ra and F. Bai. Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing *Proc. INFOCOM*, pp. 1894–1902, Hong Kong, China, April 2015.

[15] S. Vakilinia, D. Qiu and M. M. Ali. Optimal multi-dimensional dynamic resource allocation in mobile cloud computing. *EURASIP Journal on Wireless Communications and Networking*, **201**:1–14, November 2014.

[16] M. Shiraz, S. Abolfazli, Z. Sanaei and A. Gani. A study on virtual machine deployment for application outsourcing in mobile cloud computing. *The Journal of Supercomputing*, **63**(3):946–964, March 2013.

[17] M. Shiraz and A. Gani. Mobile Cloud Computing: Critical Analysis of Application Deployment in Virtual Machines. *Int. Conf. on Information and Computer Networks (ICICN)*, **27**:11–16, February 2012.

[18] A. Khalifa and M. Eltoweissy. Collaborative Autonomic Resource Management System for Mobile Cloud Computing. *The Fourth Int. Conf. on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING)*, pp. 115–121, Valencia, Spain, May 2013.

[19] Y. Niu, b. Luo, F. Liu, J. Liu and B. Li. When Hybrid Cloud Meets Flash Crowd: Towards Cost-Effective Service Provisioning. *Proc. INFOCOM*, pp. 1044–1052, Hong Kong, China, April 2015.

[20] D. Zarchy, D. Hay and M. Schapira. Capturing Resource Tradeoffs in Fair Multi-Resource Allocation *Proc. INFOCOM*, pp. 1062–1070, Hong Kong, China, April 2015.

[21] X. Zhang, C. Wu, Z. Li and F. C. M. Lau. A Truthful $(1-\epsilon)$-Optimal Mechanism for On-demand Cloud Resource Provisioning *Proc. INFOCOM*, pp. 1053–1061, Hong Kong, China, April 2015.

[22] J. Zhao, X. Chu, H. Liu, Y. Leung and Z. Li. Online Procurement Auctions for Resource Pooling in Client-Assisted Cloud Storage Systems. *Proc. INFOCOM*, pp. 576–584, Hong Kong, China, April 2015.

[23] S. Vakilinia, M. M. Ali and D. Qiu. Modeling of the resource allocation in cloud computing centers. *Computer Networks*, **91**:453–470, November 2015.

[24] D. Bruneo. A Stochastic Model to Investigate Data Center Performance and QoS in IaaS Cloud Computing Systems. *IEEE Transactions on Parallel and Distributed Systems*, **25**(3):560–569, March 2014.

[25] B. Yang, F. Tan and Y. Dai. Performance Evaluation of Cloud Service Considering Fault Recovery. *Journal of Supercomputing*, **65**:426–444, July 2013.

[26] W. Caraballo and T. G. Robertazzi. Markov Chain Flow Decomposition for a Two Class Priority Queue. *Conference on Information Sciences and Systems*, p. 1, Baltimore, MD, USA, March 2003.

[27] M. Harchol-Balter, T. Osogami, A. Scheller-Wolf and A. Wierman. Multi-server queueing systems with multiple priority classes. *Queueing Systems: Theory and Applications*, **51**(3-4):331–360, December 2005.

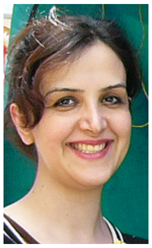
[28] A. S. Tam, D. Chiu, J. C. S. Lui and Y. C. Tay. A Case for TCP-Friendly Admission Control. *14th IEEE Int. Workshop on Quality of Service (IWQoS)*, pp. 229–238, New Haven, CT, USA, June 2006.

[29] Maplesoft Inc. Maple 16. http://www.maplesoft.com/products/maple/. *Last visited: July 15, 2015.*

[30] H. A. Lagar-Cavilla, J. A. Whitney, A. Scannell, P. Patchin, S. M. Rumble, E. Lara, M. Brudno and M. Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *ACM EuroSys*, pp. 1–12, Nuremberg, Germany, April 2009.

[31] W. Stallings. *Data and Computer Communications.* Pearson/Prentice Hall, 2007.

[32] D. Bertsekas and R. Gallager *Data Networks (2nd Edition).* Prentice Hall, 1991.



Haleh Khojasteh has received her PhD in Computer Science from Ryerson University, Toronto, Ontario, Canada in 2015. She received her MS (2011) degree in Computer Science from Ryerson University, and her BS degree in Electrical and Computer Engineering from Shahid Beheshti University, Tehran, Iran. Her research interests are cloud computing and wireless networks with emphasis on mathematical modeling and performance analysis.

Jelena Mišić (M91, SM08) is Professor of Computer Science at Ryerson University in Toronto, Ontario, Canada. She has published over 90 papers in archival journals and more than 120 papers at international conferences in the areas of wireless networks, in particular wireless personal area network and wireless sensor network protocols, performance evaluation, and security. She serves on editorial boards of *IEEE Transactions on Vehicular Technology*, *Computer Networks*, *Ad hoc Networks*, *Security and Communication Networks*, *Ad Hoc & Sensor Wireless Networks*, *Int. Journal of Sensor Networks*, and *Int. Journal of Telemedicine and Applications*. She is a Senior Member of IEEE and Member of ACM.

Vojislav B. Mišić is Professor of Computer Science at Ryerson University in Toronto, Ontario, Canada. He received his PhD in Computer Science from University of Belgrade, Serbia, in 1993. His research interests include performance evaluation of wireless networks and systems and software engineering. He has authored or co-authored six books, 20 book chapters, and over 200 papers in archival journals and at prestigious international conferences. He serves on the editorial boards of *IEEE transactions on Cloud Computing*, *Ad hoc Networks*, *Peer-to-Peer Networks and Applications*, and *International Journal of Parallel, Emergent and Distributed Systems*. He is a Senior Member of IEEE, and member of ACM and AIS.