# Throughput Maximization for Online Request Admissions in Mobile Cloudlets

Qiufen Xia†, Weifa Liang† and Wenzheng Xu†¶
†Research School of Computer Science, Australian National University, Canberra, ACT 0200, Australia
¶School of Information Science and Technology, Sun Yat-Sen University, Guangzhou, 510006, China
{qiufen.xia, wenzheng.xu}@anu.edu.au, wliang@cs.anu.edu.au

***Abstract*—In mobile cloud computing (MCC) paradigm, cloud service providers not only offer powerful cloud data centers but also provide small-scale cloudlets in some strategic locations for mobile users to access their rich resources. Due to the flexibility and locality of cloudlets, most requests of mobile users can be processed locally. However, the cloudlets usually have limited resources and processing abilities, which implies that they may not be capable to process every incoming request. Instead, some resource-intensive requests need to be sent to remote data centers for processing and such a processing is transparent to users. In this paper, we address the online request admission issue in a cloudlet with an objective to maximize the system throughput, for which we first propose a novel admission cost model to model critical resource consumptions. We then devise efficient control algorithms for online request admissions. We finally conduct experiments by simulations to evaluate the performance of the proposed algorithms. Experimental results indicate that the proposed algorithms are promising and outperform other heuristics.**

## I. Introduction

Many mobile devices such as smart phones and tablets are becoming increasingly popular, people now depend heavily on them to run various applications such as image processing, facebook, twitter, games, and emails for social and business purposes. However, due to small sizes and being powered by batteries, these portable and lightweight mobile devices have only limited energy to support their operations. To mitigate the severe energy constraint on mobile devices is to make use of the rich resources provided by mobile cloud computing (MCC) platforms. That is to offload data and computationally expensive tasks from mobile devices to cloud platforms through wireless networks [4], [6], [10].

In MCC environments, wireless mobile devices access the cloud through wireless communication such as Wi-Fi, 3G/4G, etc. However, it is well known that wireless communication is unreliable and constrained by its bandwidth. The long delay of data transfer between a mobile device and the cloud is unavoidable. Thus, offloading tasks from mobile devices to the cloud is not always a smart choice since the cloud is typically far from mobile users. To overcome the long delay by offloading tasks to the remote clouds, the cloud has to be moved closer to the mobile users in the form of the cloudlet [18], which consists of trusted, resource rich servers in vicinities of mobile users (e.g., near or co-located with a wireless access point) [14], [17]. Although a cloudlet may mitigate the delay latency, it does suffer serious drawbacks too. As the resources in a cloudlet are not as abundant as that in a powerful cloud, a cloudlet may run out of its resources quickly if its resources are carelessly allocated. Particularly, when many requests are executed at the same time, this will lead to the cloudlet overloaded.

In this paper, we focus on developing efficient control algorithms for online request admissions in the MCC environments with an objective to maximize the system throughput of the cloudlets, where each request can be represented by a demand resource vector in which each component is the amount of a specific resource demanded by the request. Our main contributions are as follows. We first propose a novel admission cost model to model different resource consumptions in a cloudlet. We then devise efficient control algorithms for online request admissions based on the proposed resource cost model. We finally conduct experiments by simulations to evaluate the performance of the proposed algorithms. Experimental results indicate that the proposed algorithms are very promising, in comparison with other heuristics.

The remainder of this paper is organized as follows. Section 2 introduces related work, followed by introducing the system model and problem definitions in Section 3. The online request and batch admission algorithms are proposed in Section 4 and Section 5, respectively. The performance evaluation of the proposed algorithms will be conducted in Section 6, and the conclusion is given in Section 7.

## II. Related work

Admission control is a key issue in the provision of guaranteed quality of service (QoS) in mobile cloud computing (MCC) environments. The design of admission control algorithms for MCC is especially challenging, given the limited and highly demand resources, and the mobility of users. Considerable research efforts have been taken in the past few years. Many existing works in literature focused on developing various admission control policies and resource allocation strategies. For example, several researchers investigated the admission control problem based on the Markov Decision Process. They aimed to either maximize the revenue of the system or minimize the cost of service providers based on the prediction information [2], [9], [11]. Specifically, Hoang *et al.* [9] proposed a linear programming solution for the problem by considering the QoS requirements of mobile users with an aim to maximize the revenue of the service providers. Liang

978-1-4799-0537-9/13/$31.00 ©2013 IEEE

*et al.* [11] formulated the adaptive resource allocation problem as a semi-Markov decision process to capture the arrivals and departures of users dynamically with an objective to maximize the rewards of the overall system through balancing between the resource utilization and the resource cost. Abundo *et al.* [2] employed the Markov forward-looking admission control policy for a service broker to maximize profits of resource providers while guaranteeing the QoS requirements of admitted users. Almeidaa *et al.* [1] presented a joint admission control and resource allocation scheme by formulating a convex optimization problem with an objective to minimize the cost of the service provider, i.e., maximize the revenue of the provider.

The above mentioned studies focused mainly on the CPU resource by ignoring other important resources such as memory, secondary storage, network bandwidth, while these resources also impact the system performance significantly. Srikantaiah *et al.* [15] considered the request consolidation problem in virtualized heterogeneous systems to minimize the energy consumption while meeting the performance requirement, for which they proposed an approach to optimize multiple resources, i.e., CPU and disk usage jointly, using the bin-packing algorithm. The proposed approach requires the optimal operating point from profiling data and calculates the Euclidean distance between the optimal point and the current workload allocation. However, finding such an optimal operating point is difficult due to its dependence on experimental data. In contrast to these studies, in this paper we approach the cloudlet resource allocation by responding user requests without the knowledge of future request arrival rate. We propose admission algorithms that can effectively and efficiently admit requests according to the current workload of each resource in the system with an objective to maximize the system throughput.

## III. Preliminaries

In this section, we first introduce the system model. We then define the problems precisely.

### A. System model

We consider a mobile cloudlet computing environment as shown in Fig. 1, where a cloudlet connecting to a remote powerful cloud computing platform through the Internet provides cloud services to a set of local wireless mobile users. Denote by $\{u_i \mid 1 \leq i \leq N\}$ the set of local mobile users, where $N$ is the number of mobile users. Due to stringent constraints on mobile devices such as limited battery lifetime, limited storage, and relatively weak computation capability, mobile users usually offload their computing intensive or large volume of data storage tasks to the cloudlet to save the limited resources of mobile devices. To this end, mobile users first send their requests in terms of amounts of resources needed to the cloudlet. The cloudlet then decides whether to admit the requests according to its resource availability and the admission costs of these requests. For the sake of convenience, we assume that time is slotted into equal *time slots*. We further assume that the system has no knowledge on the future request

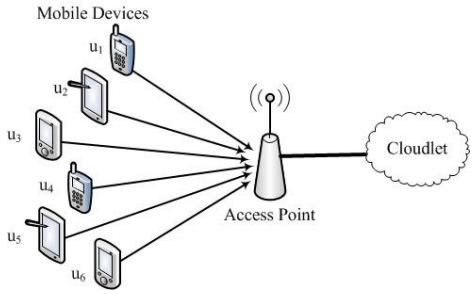



Fig. 1. The system model of mobile cloud computing

generation and arrival rates. The acceptance or rejection of a request by the system is made at the beginning of each time slot $t$.

More specifically, we assume that the cloudlet provides $K$ different resources. Let $C_k$ be the capacity of resource $k$ for all $k$ with $1 \leq k \leq K$. Let $H(t) = \langle H_1(t), \ldots, H_K(t) \rangle$ be the amounts of resources occupied by the admitted requests at time slot $t$. In all our discussions we assume that $H(t)$ is given. Let $A(t) = \langle A_1(t), \ldots, A_K(t) \rangle$ be the vector of available resources in the cloudlet at time slot $t$, then $A_k(t) = C_k - H_k(t)$ for all $k$ with $1 \leq k \leq K$. Denote by $r_i(t) = \langle r_{i,1}(t), \ldots, r_{i,K}(t); \tau_i \rangle$ the amounts of resource demanded by a request $r_i(t)$ at time slot $t$, where $r_{i,k}(t) \in \mathbb{Z}$ is the amount of resource $k$ requested and $\tau_i$ is the occupation period of the requested resources. We further assume that requests arrive one by one and use $\langle r_1, \ldots, r_N \rangle$ to denote the sequence of requests.

Within each time slot, we consider two different request admission scenarios: one is that only one request is evaluated, that is, the admission control algorithm will determine whether the request is admitted, depending on not only whether there are enough available resources for the request but also whether it is too high to admit the request in terms of the admission cost. The other is that multiple requests are evaluated, i.e., a set of requests will be admitted at each time slot.

### B. Problem definitions

Given a mobile cloudlet, each mobile user $u_i$ sends its request $r_i(t)$ at time slot $t$ to the cloudlet. Each request $r_i(t)$ consists of a set of specified amounts of resource demands on the cloudlet and the occupation period $\tau_i$, i.e., $r_i(t) = \langle r_{i,1}(t), \ldots, r_{i,K}(t); \tau_i \rangle$, where $r_{i,k}(t)$ is the amount of resource $k$ needed. Assuming that requests arrive one by one, and there is no knowledge of future request generation and arrival rates.

The *online request throughput maximization problem* is to determine whether an arrival request to be admitted or rejected by the system such that the system throughput is maximized for a specified time period $T$. The *system throughput* is the ratio of the number of *admitted requests* to the number of requests for period $T$, where a request is admitted if the request will be implemented by the cloudlet. Otherwise, it is rejected immediately at the current time slot, and can be resubmitted at future time slots. The admission decision of a request depends

on its requested resource availability and its admission cost. Once a request is admitted, its processing in the cloudlet may last more than one time slot. If it cannot be finished within the current time slot, it will continue occupying the system resources at the next time slot until it finally finishes. In other words, we assume that the request scheduling is non-preemptive scheduling.

The *online batch request throughput maximization problem* is to maximize the system throughput if multiple requests, instead of one request, will be admitted per time slot.

As each of $K$ different resources in a cloudlet can be treated as one dimension of a $K$-dimension bin with capacity $C_k$ of dimension $k$, the online request throughput maximization problem is equivalent to packing as many online requests as possible without knowing of future requests. If each request can be implemented within a single time slot, then its occupied resources can be released at the next time slot. Clearly, this special case of the problem is an online bin packing problem which is NP-hard [5]. Thus, the online request throughput maximization problem is NP-hard, too. Meanwhile, the online request throughput maximization problem is a special case of the online batch request throughput maximization problem when there is only one request considered at each time slot, thus, the latter is NP-hard, too.

## IV. Algorithm for the Online Request Throughput Maximization Problem

In this section, an algorithm for the online request throughput maximization problem is devised. Let $r_i(t) = \langle r_{i,1}(t), \ldots, r_{i,K}(t); \tau_i \rangle$ be the request being considered at this moment. The system proceeds as follows. It first checks whether the requested amount of each resource $r_{i,k}(t)$ can be met by the system. If not, the request is rejected immediately; otherwise the system calculates the admission cost of processing the request based on the load of each resource at this moment. If its admission cost is beyond a specified threshold of each resource in the system, the request will be rejected; otherwise, it is admitted by the cloudlet. In the following we propose the admission cost modeling of processing a request.

### A. Admission cost modeling

We here model the admission cost of each demand resource $k$ as a convex function of the quantity of the resource with an increasing marginal cost. That is, the cost of allocating a unit specific resource to a request increases with the demand quantity of that resource. The *unit admission cost* of using resource $k$ with demand $r_{i,k}(t)$ ($\neq 0$) by request $r_i(t)$ is defined as follows.

$$\zeta(r_{i,k}(t), H_k(t)) = a_k^{\frac{H_k(t)}{C_k}} (a_k^{\frac{r_{i,k}(t)}{C_k}} - 1), \tag{1}$$

where $a_k > 1$ is a constant and $H_k(t)$ is the amount of resource $k$ occupied by admitted requests at time slot $t$ prior to request $r_i(t)$.

From Eq. (1), it can be seen that $\zeta(r_{i,k}(t), H_k(t))$ is in the range of $(0, a_k - 1]$. Notice that the unit admission cost of demanding a resource is closely related to the demanding quantities of that resource. That is, a higher $r_{i,k}(t)$ means a higher admission cost of $r_i(t)$.

Denote by $\gamma(r_i(t), H(t))$ *the admission cost* of a request $r_i(t)$ with occupation period $\tau_i$ for all resources at time slot $t$, then

$$\begin{aligned} \gamma(r_i(t), H(t)) &= \tau_i \cdot \sum_{k=1}^{K} \zeta(r_{i,k}(t), H_k(t) \mid r_{i,k}(t) \neq 0) \\ &= \tau_i \cdot \sum_{k=1}^{K} a_k^{\frac{H_k(t)}{C_k}} (a_k^{\frac{r_{i,k}(t)}{C_k}} - 1). \end{aligned} \tag{2}$$

### B. Admission policy

Given a request $r_i(t)$, there is an admission control strategy that determines whether it will be admitted. That is, for each request $r_i(t)$, a threshold $B_k$ of the unit admission cost of using resource $k$ is given. Recall that the value range of a unit admission cost of using resource $k$ is in $(0, a_k - 1]$, $B_k$ thus is in the range of $(0, a_k - 1]$. Let $\theta_k$ be a constant with $\theta_k \in (0, 1]$, $B_k$ can be rewritten as $(a_k - 1) \cdot \theta_k$. For each request, *a threshold of the average admission cost* $B$ of using all demanded resources is also given, which is $B = \theta \cdot (a - 1) = \theta \cdot (\max_{k=1,\ldots,K}\{a_k\} - 1)$, where $\theta \in (0, 1]$. Let $||r_i(t)||$ be the number of resources requested by request $r_i(t)$, then $||r_i(t)|| \leq K$. Thus, a request $r_i(t)$ is admitted if it meets the following two inequalities:

(i) $\zeta(r_{i,k}(t), H_k(t)) \leq B_k$ for each its demanded amount of resource $k$, $r_{i,k}(t)$ with $r_{i,k}(t) \neq 0$;

(ii) $\frac{\gamma(r_i(t), H(t))}{||r_i(t)|| \cdot \tau_i} \leq B$.

### C. Algorithm

As mentioned in the previous subsection, the online request throughput maximization problem is equivalent to the online $K$-dimensional bin packing problem. Meanwhile, different dimensions (e.g., wireless communication bandwidth requirement, and the number of CPU instructions) have different attributes, we reduce this $K$-dimensional bin packing problem to one dimension bin packing problem with admission control by introducing the admission cost (see Eq. (2)). The detailed algorithm is described in **Algorithm** 1, which is also referred to as `Algorithm Online-OBO`.

*Theorem 1:* Given a mobile cloudlet environment and an online admission request sequence, there is an online algorithm for the request throughput maximization problem, which takes $O(K)$ time per request at each time slot.

*Proof:* Following **Algorithm** 1, there is only a single request $r_i(t)$ per time slot to be determined. To respond to this incoming request, it checks whether the request is admitted. If yes, it takes $O(K)$ time to update the amounts of occupied and available system resources. Otherwise, the request is rejected. Thus, the **Algorithm** 1 takes $O(K)$ time to decide whether to admit a request $r_i(t)$ at time slot $t$. ■

**Algorithm 1** Algorithm for the online request throughput maximization problem

**Input:** $B_k$, $B$, an arrival request $r_i(t)$, the occupied information $H(t)$ of resources at time slot $t$, where $r_i(t) = \langle r_{i,1}(t), \ldots, r_{i,K}(t); \tau_i \rangle$ and $1 \leq k \leq K$.

**Output:** Admit or reject request $r_i(t)$.

/* The admission cost of request $r_i(t)$ */;
$\gamma(r_i(t), H(t)) \leftarrow 0$;
**for** each $r_{i,k}(t)$ in $r_i(t)$ **do**
  Calculate $A_k(t) \leftarrow C_k - H_k(t)$, which is the available amount of resource $k$;
  **if** $r_{i,k}(t) > A_k(t)$ **then**
    Reject request $r_i(t)$;
    EXIT;
  **else**
    **if** $r_{i,k}(t) \neq 0$ **then**
      Calculate the cost $\zeta(r_{i,k}(t), H_k(t))$ by Eq. (1);
      **if** $\zeta(r_{i,k}(t), H_k(t)) \leq B_k$ **then**
        $\gamma(r_i(t), H(t)) \leftarrow \gamma(r_i(t), H(t)) + \tau_i \cdot \zeta(r_{i,k}(t), H_k(t))$
      **else**
        Reject the request;
        EXIT;
      **end if**
    **end if**;
  **end if**;
**end for**;
**if** $\frac{\gamma(r_i(t), H(t))}{||r_i(t)|| \cdot \tau_i} \leq B$ **then**
  /* Update the amounts of occupied resources */;
  $H(t) \leftarrow \langle H_1(t) + r_{i,1}(t), \ldots, H_K(t) + r_{i,K}(t) \rangle$;
  **return** Admit request $r_i(t)$
**else**
  Reject request $r_i(t)$;
  EXIT;
**end if**.

## V. Algorithm for the Online Batch Request Throughput Maximization Problem

In this section, we deal with multiple request admissions at each time slot by proposing an algorithm. Specifically, let $\Delta S(t)$ be the set of requests arrived at time slot $t$. We determine a subset $\Delta S'(t) \subseteq \Delta S(t)$ of requests to be admitted if not all the requests in $\Delta S(t)$ are admitted.

The basic idea for the online batch request maximization problem is to admit a set of requests one by one using a greedy strategy, according to the admission criteria made by the system. Specifically, given the available resources $A(t)$ and occupied resources $H(t)$ at time slot $t$, let $\Delta S(t)$ and $\Delta S'(t)$ be the set of requests arriving at time slot $t$ and the subset of these requests to be admitted by the system, respectively, where $\Delta S'(t) \subseteq \Delta S(t)$.

The proposed algorithm proceeds as follows. Initially, $\Delta S'(t) = \emptyset$. For each request $r \in \Delta S(t)$, compute its unit admission cost based on $H(t)$, using the similar admission criteria as we set for a single request in the previous section, Eq. (1) and Eq. (2). If a request does not meet the criteria, it is rejected and removed from $\Delta S(t)$. Otherwise, it becomes a candidate to be admitted. A candidate request with the minimum admission cost is then identified. If two candidate requests have the same minimum admission cost, the one with a smaller occupation period will be chosen to add to the admission set $\Delta S'(t)$, and $\Delta S(t) = \Delta S(t) - \Delta S'(t)$. Let $r_{t_1}$ be the request that has been admitted with $1 \leq t_1 \leq |\Delta S(t)|$. The system then updates its available resources $A'(t) = \langle A_1(t) - r_{t_1,1}(t), \ldots, A_K(t) - r_{t_1,K}(t) \rangle$. The algorithm continues to identify the next request from $\Delta S(t) - \{r_{t_1}(t)\}$ based on the updated available resources $A'(t)$ to see whether it can be admitted. This procedure continues until $\Delta S(t) = \emptyset$.

The detailed algorithm is described in **Algorithm** 2, which is also referred to as `Algorithm Online-Batch`.

*Lemma 1:* If a request is rejected in an iteration of the `while` loop of **Algorithm** 2, it will not be admitted in the rest of iterations at time slot $t$, i.e, it is removed from further consideration at time slot $t$.

*Proof:* We show this by contradiction. Assume that a request $r_i(t)$ is rejected in an iteration of the `while` loop, which means that (i) either its requested amount of a specific resource $k$ is larger than the available amount of resource $k$, if this is the case, it will not be admitted in the future as well because the available amount of resource $k$ becomes smaller and smaller with more and more requests being added to $\Delta S'(t)$; or (ii) the total admission cost of $r_i(t)$ is beyond the threshold. For each resource it requested, the resource becomes less than that of it in the iteration that $r_i(t)$ is rejected, the admission cost becomes larger in comparison with the one in which it was rejected for the first time, i.e, its admission cost is still larger than the given threshold, so it will be rejected. ■

*Theorem 2:* Given a cloudlet environment, there is an online algorithm for the batch request throughput maximization problem that takes $O(K|\Delta S(t)|^2)$ time at each time slot, where $\Delta S(t)$ is the set of requests at time slot $t$.

*Proof:* Following **Algorithm** 2, within the `while` loop, updating the information of occupied resources and calculating the amounts of available resources take $O(K)$ time, while checking whether the demanded amounts of resource $k$ of the request can be met takes $O(K)$ time. It finally takes $O(K)$ time to decide whether the admission cost $\gamma(r_i(t), r_{i,k}(t))$ of each required resource satisfies the threshold requirement. In total, there are $|\Delta S(t)|$ iterations. Thus, **Algorithm** 2 takes $\sum_{i=1}^{|\Delta S(t)|} O(i \cdot K) = O(K \cdot |\Delta S(t)|^2)$ time. ■

## VI. Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms and investigate the impact of different parameters on the algorithm performance.

**Algorithm 2** Algorithm for the online batch request throughput maximization problem

**Input:** A set of requests $\Delta S(t)$, $B_k$, $B$, $H(t) = \{H_1(t), \ldots, H_K(t)\}$.

**Output:** a subset of admitted requests $\Delta S'(t) \subseteq \Delta S(t)$ at time slot $t$.

$U \leftarrow \Delta S$; $\Delta S'(t) \leftarrow \emptyset$; $H'(t) \leftarrow H(t)$;
**while** $U \neq \emptyset$ **do**
  /* A variable indicating the minimum admission cost */; $min\_cost \leftarrow \infty$;
  /* A variable indicating the index of the request with the minimum admission cost */; $i_0 \leftarrow \infty$;
  **for** each request $r_i(t)$ in $U$ **do**
    /* The admission cost by processing request $r_i(t)$ */; $\gamma(r_i(t), H'(t)) \leftarrow 0$;
    **for** each $r_{i,k}(t)$ in $r_i(t)$ **do**
      Calculate the available amount of resource $k$: $A'_k(t) \leftarrow C_k - H'_k(t)$;
      **if** $r_{i,k}(t) > A'_k(t)$ **then**
        $U \leftarrow U - \{r_i(t)\}$; Reject request $r_i(t)$;
      **else**
        Calculate the unit admission cost $\zeta(r_{i,k}(t), H_k(t))$ of $r_i(t)$ by Eq. (1);
        **if** $\zeta(r_{i,k}(t), H_k(t)) > B_k$ **then**
          $U \leftarrow U - \{r_i(t)\}$; Reject request $r_i(t)$;
        **else**
          $\gamma(r_i(t), H'(t)) \leftarrow \gamma(r_i(t), H'(t)) + \tau_i \cdot \zeta(r_{i,k}(t), H_k(t))$;
        **end if**;
      **end if**;
    **end for**;
    **if** $\frac{\gamma(r_i(t), H'(t))}{||r_i(t)|| \cdot \tau_i} > B$ **then**
      $U \leftarrow U - \{r_i(t)\}$; Reject request $r_i(t)$;
    **end if**;
    **if** $\frac{\gamma(r_i(t), H'(t))}{||r_i(t)|| \cdot \tau_i} < min\_cost$ **then**
      $min\_cost \leftarrow \frac{\gamma(r_i(t), H'(t))}{||r_i(t)|| \cdot \tau_i}$;
      $i_0 \leftarrow i$;
    **else**
      **if** $\frac{\gamma(r_i(t), H'(t))}{||r_i(t)|| \cdot \tau_i} = min\_cost$ **then**
        Select the request $r_{i'}(t)$ with a smaller occupation period between the two requests, i.e., $min\_cost \leftarrow \frac{\gamma(r_{i'}(t), H'(t))}{||r_{i'}(t)|| \cdot \tau_{i'}}$;
        $i_0 \leftarrow i'$;
      **end if**;
    **end if**;
  **end for**
  $\Delta S'(t) \leftarrow \Delta S'(t) \cup \{r_{i_0}(t)\}$ where $r_{i_0}(t)$ has the minimum admission cost;
  $U \leftarrow U - \{r_i(t)\}$;
  Update the amounts of occupied resources $H'(t)$ by taking the occupied resources by $r_i(t)$ into $H'(t)$;
**end while**
**return** $\Delta S'(t) \subseteq \Delta S(t)$.

### A. Simulation environment

We consider a mobile cloudlet environment that consists of $n$ servers [8] and a set of mobile devices sending their requests to the cloudlet for processing as depicted in Fig. 1. The cloudlet contains four resources: CPU, memory, and disk storage with capacities 2.99 $GHz$, 8 $GB$ and 1024 $GB$ for each server, and the bandwidth capacity of the wireless access point with 75 $Mbps$ [13].

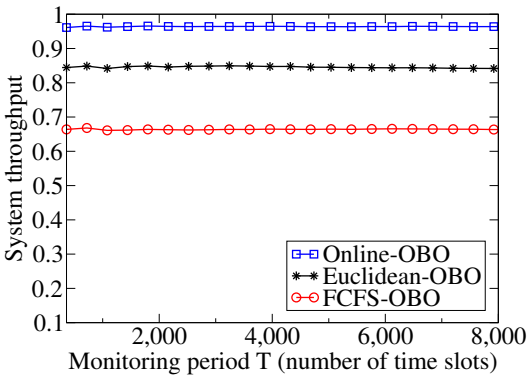
Recall that $B_k = \theta_k \cdot (a_k - 1)$ and $B = \theta \cdot (a - 1)$ in the section IV-B, in our default settings, we assume that all resources have the same threshold $B_k = B$. We set $\theta_k = \theta = 0.3$ and $a_k = a = 4$ for algorithms `Online-OBO` and `Online-Batch`. The number of server $n$ is 16 in the default setting. Unless otherwise specified, we will adopt these default settings in our simulations. Each value in all figures is the average of the results by applying the nominated algorithm for 20 different request sequences.

We assume that each time slot lasts 10 seconds [14] and the system monitoring time period is $T = 8,000$ time slots. We further assume that only a single request is evaluated at each time slot for the online request throughput maximization problem, while the number of requests at each time slot is randomly generated within the range of $[2, 10]$ for the online batch request throughput maximization problem. The requirements of resources by each request are set according to Amazon Small Instance [3] settings. If a request does require resource $k$, then the required amount of resource $k$ is generated randomly within the range of resource $k$ listed in Table. I, of which the *maximum occupation period* $\tau_{max}$ means the demanded occupation period for the system resources of a request is at most $\tau_{max}$.

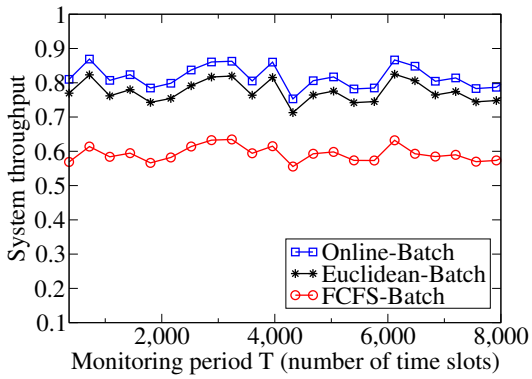
TABLE I
PARAMETERS OF REQUESTS

| Type | One-by-one | Batch |
|---|---|---|
| CPU power (GHz) | [1, 6] | [1, 6] |
| Memory (MB) | [1, 400] | [1, 400] |
| Storage (GB) | [0.06, 1.2] | [0.06, 1.2] |
| Bandwidth (Mbps) | [0.05, 1.5] | [0.05, 1.5] |
| Maximum occupation period (time slots) | 20 | 20 |

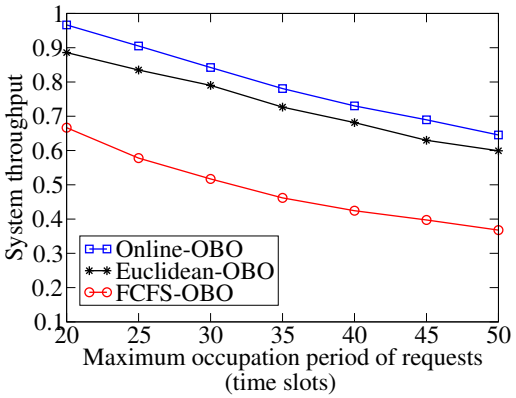
We consider two heuristics as our evaluation benchmarks. The first heuristic admits requests according to the First-Come-First-Service strategy, and a request will be admitted as long as the cloudlet can fulfil its resource demands, otherwise the request will be rejected immediately. We refer to this algorithm as `FCFS-OBO` and `FCFS-Batch` for the online request throughput maximization problem and the online batch request throughput maximization problem, respectively. The second heuristic [15] considers the request consolidation problem in a virtualized heterogeneous server system with an aim to minimize the operation energy consumption of servers. The proposed solution finds an optimal operating point which occurs at 70% CPU utilization and 50% disk utilization from profiling data through the calculating of the Euclidean distance between the optimal point and the current



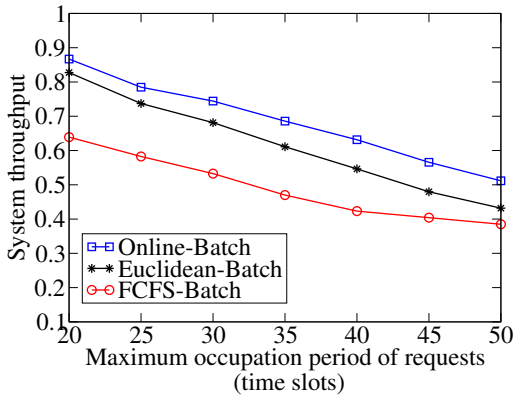

(a) The system throughput of algorithms `Online-OBO`, `Euclidean-OBO` and `FCFS-OBO` over different monitoring periods




(b) The system throughput of algorithms `Online-Batch`, `Euclidean-Batch` and `FCFS-Batch` over different monitoring periods




(c) The system throughput of algorithms `Online-OBO`, `Euclidean-OBO` and `FCFS-OBO` with various maximum occupation periods at $T = 8,000$ time slots




(d) The system throughput of algorithms `Online-Batch`, `Euclidean-Batch` and `FCFS-Batch` with various maximum occupation periods at $T = 8,000$ time slots

Fig. 2. Performance evaluations of different algorithms.

workload allocation [15]. As their objective is to minimize the energy consumption of the system, we aim to maximize the system throughput in a cloudlet environment, we treat the optimal point as the one that fully utilizes all resources in the system. That is, the optimal point for each resource is the capacity of the resource. We refer to the modified algorithms as algorithms `Euclidean-OBO` and `Euclidean-Batch` for the two problems.

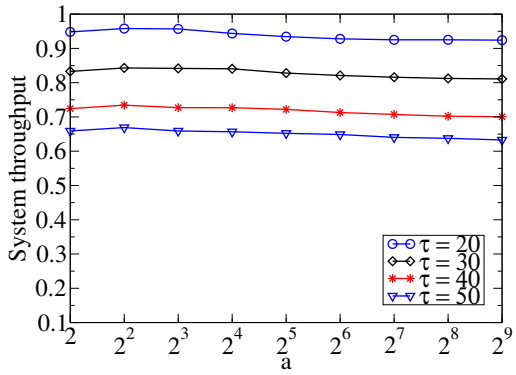
### *B. Algorithm performance evaluation*

We first evaluate the proposed algorithms against the benchmark algorithms in the default parameter settings under different monitoring periods and with various maximum occupation periods of requests. The performance curves of these algorithms are plotted in Fig. 2. From Fig. 2(a) and Fig. 2(b) we can see that the proposed algorithms `Online-OBO` and `Online-Batch` outperform their counterparts `Euclidean-OBO`, `FCFS-OBO`, `Euclidean-Batch` and `FCFS-Batch` in terms of the system throughput over different monitoring periods $T$. For example, the system throughput of `Online-OBO` is around $10\%$ higher than that of `Euclidean-OBO`, and $30\%$ higher than that of `FCFS-OBO`. Also, the system throughput of `Online-Batch` is around $4\%$ and $23\%$ higher than its counterparts of algorithms `Euclidean-Batch` and `FCFS-Batch`. The rationale behind is that the proposed algorithms will reject those requests that have large quantity of resource demands and occupation of the demanded resources for a long period, while these requests will reduce the future system throughput since they occupy large amounts of resources for long periods if they were admitted. In contrast, the other two heuristics failed to reject such requests. In addition, from Fig. 2(c) and Fig. 2(d), it can be seen that the proposed algorithms `Online-OBO` and `Online-Batch` significantly outperform all the other mentioned algorithms in terms of the system throughput with various maximum occupation periods $\tau_{max}$ of requests for a given monitoring period $T = 8,000$ time slots. It can also be seen from Fig. 2(c) and Fig. 2(d) that the system throughput decreases with the increase of the $\tau_{max}$, because a longer occupation period of a request leads to a higher load to the system, which generates a smaller system throughput.

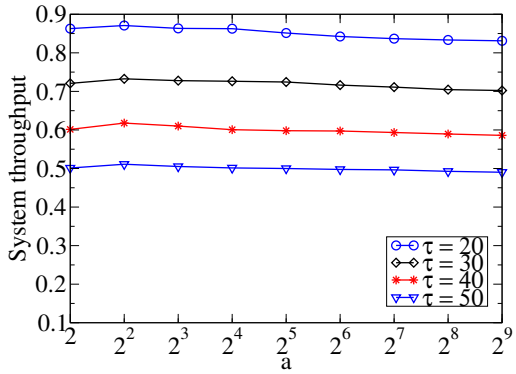
### C. Impact of parameters on algorithm performance

We then study the impact of different parameters on the algorithm performance under different maximum occupation periods $\tau_{max}$ at $T = 8,000$ time slots as follows.

*1) Impact of parameter $a$:* We first investigate the impact of the value of $a$ on the system throughput by varying $a$ from 2 to $2^9$. Fig. 3 plots the performance curves of algorithms `Online-OBO` and `Online-Batch` with various maximum occupation periods $\tau_{max}$. For the sake of convenience, we use $\tau$ to represent $\tau_{max}$ in Fig. 3, from which it can be seen that the system throughput increases with the growth of $a$. Specifically, the system throughput reaches the peak at $a = 4$, and follows decreasing. We thus set $a = 4$ in our default setting. Notice that the system throughput decreases with the growth of $\tau_{max}$ of requests, the arguments are similar as we did in Fig. 2(c) and Fig. 2(d).



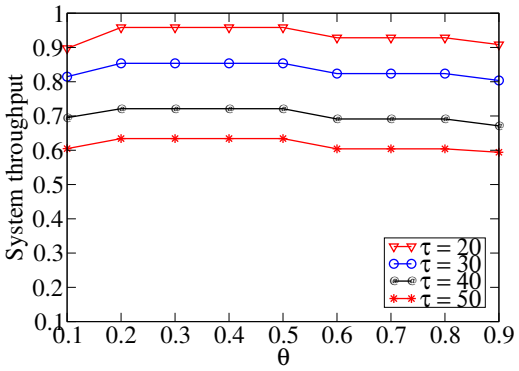

(a) The impact of $a$ on the system throughput of algorithm `Online-OBO` with various maximum occupation periods



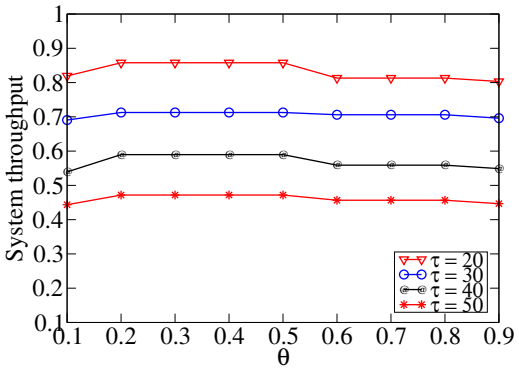

(b) The impact of $a$ on the system throughput of algorithm `Online-Batch` with various maximum occupation periods

Fig. 3. The impact of values of $a$ on the system throughput of algorithms `Online-OBO` and `Online-Batch` with various maximum occupation periods at $T = 8,000$ time slots.

*2) Impact of parameter $\theta$:* We then evaluate the impact of the threshold coefficient $\theta$ in $B_k = B = \theta \cdot (a-1)$ on the system throughput of algorithms `Online-OBO` and `Online-Batch` by varying $\theta$ from 0.1 to 0.9. Fig. 4 (a) and Fig. 4 (b) imply that the system throughput increases with the growth of $\theta$, reaches the peak at $\theta = 0.2$, then keeps stable until $\theta = 0.5$ and decreases subsequently. We thus choose $\theta = 0.3$. Recall that $a = 4$, the threshold $B_k = B = \theta \cdot (a-1) = 0.9$ in our default setting. The system throughput varies with the change of the threshold, this is because the system tends to reject requests when the threshold is small, thereby reducing the system throughput. However, when the threshold is large, requests with higher resource demands can be admitted, which lead to a reduction of the system throughput due to the large resource demands of such requests. That is, the optimal system throughput arises only when the threshold neither large nor small. Also, the system throughput decreases with the increase of $\tau_{max}$, following the similar arguments as we did in Fig. 2(c) and Fig. 2(d).




(a) The impact of $\theta$ on the system throughput of algorithm `Online-OBO` with various maximum occupation periods




(b) The impact of $\theta$ on the system throughput of algorithm `Online-Batch` with various maximum occupation periods

Fig. 4. The impact of $\theta$ on the system throughput of algorithms `Online-OBO` and `Online-Batch` with various maximum occupation periods at $T = 8,000$ time slots.

## VII. Conclusions

In this paper, we considered the online request throughput maximization problem in a cloudlet. We developed novel admission control algorithms through proposing a novel admission cost model to model different resource consumptions. We also conducted extensive experiments by simulations to evaluate the performance of the proposed algorithms against existing heuristics in terms of system throughput. Experimental results demonstrate that the proposed algorithms are promising and outperform the mentioned heuristics.

## REFERENCES


[1] J. Almeidaa, V. Almeidaa, D. Ardagnab, Í. Cunhaa, C. Francalancib, and M. Trubianc. Joint admission control and resource allocation in virtualized servers. *Journal of Parallel and Distributed Computing*, Vol. 70, No. 4, pp.344–362, 2010.

[2] M. Abundo, V. Cardellini, and F. Presti. Admission control policies for a multi-class QoS-aware service oriented architecture. *SIGMETRICS Performance Evaluation Review'12*, ACM, Vol. 39, No. 4, pp.89–98, 2012.

[3] Amazon EC2 instance types–measuring compute resources. *http://aws.amazon.com/ec2/instance-types/*.

[4] E. Cuervo, A. Balasubramanian, D. Cho, A Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: making smartphones last longer with code offload. *Proc. MobiSys'10*, ACM, 2010.

[5] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. *Approximation algorithms for NP-hard problems*, pp.46–93, 1997.

[6] S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan. How close is close enough? understanding the role of cloudlets in supporting display appropriation by mobile users. *Proc. of PerCom'12* , IEEE, 2012.

[7] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive Internet services. *Proc. of NSDI'08*, USENIX, 2008.

[8] Dell Data-center-in-a-Box. *http://www.dell.com*.

[9] D. T. Hoang, D. Niyato and P. Wang. Optimal admission control policy for mobile cloud computing hotspot with cloudlet. *Proc. of WCNC'12*, IEEE, 2012.

[10] K. Kumar and Y. Lu. Cloud computing for mobile users: can offloading computation save energy. *Journal of Computer*,Vol. 43, No. 4, pp.51-56, 2010.

[11] H. Liang, L. Cai, H. Shan, X. Shen, and D. Peng. Adaptive resource allocation for media services based on semi-Markov decision process. *Proc. of ICTC'10*, 2010.

[12] S. Lim, J. Huh, Y. Kim, and C. Das. Migration, assignment, and scheduling of jobs in virtualized environment. *Proc. of HOTCLOUD'11*, USENIX, 2011.

[13] S. Sesia, I. Toufik, and M. Baker. *LTE - the UMTS long term evolution: from theory to practice*. John Wiley, 2011.

[14] M. Satyanarayanan, R. Bahl, R. Cáceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing*, IEEE, Vol. 8, No. 4, pp.14–23, 2009.

[15] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. *Proc. of HotPower'08*, USENIX, 2008.

[16] T. Benson, A. Anand, A Akella, and M. Zhang. Understanding data center traffic characteristics. *SIGCOMM Computer Communication Review*, ACM, Vol. 40, No. 1, pp.92–99, 2010.

[17] T. Verbelen, P. Simoens, F. D. Turck, and B. Dhoedt. Cloudlets: bringing the cloud to the mobile user. *Proc. of MCS'12*, ACM, 2012.

[18] L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. *Proc. of CLOUD '12*, IEEE, 2012.