Cloudlet Load Balancing in Wireless Metropolitan Area Networks

Mike Jia, Weifa Liang, Zichuan Xu, and Meitian Huang
Research School of Computer Science
The Australian National University
Canberra, ACT 2601, Australia
Email: u5515287@anu.edu.au, wliang@cs.anu.edu.au, edward.xu@anu.edu.au, u4700480@anu.edu.au

***Abstract*—With advances in wireless communication technology, more and more people depend heavily on portable mobile devices for businesses, entertainments and social interactions. Although such portable mobile devices can offer various promising applications, their computing resources remain limited due to their portable size. This however can be overcome by remotely executing computation-intensive tasks on clusters of near by computers known as cloudlets. As increasing numbers of people access the Internet via mobile devices, it is reasonable to envision in the near future that cloudlet services will be available for the public through easily accessible public wireless metropolitan area networks (WMANs). However, the outdated notion of treating cloudlets as isolated data-centers-in-a-box must be discarded as there are clear benefits to connecting multiple cloudlets together to form a network. In this paper we investigate how to balance the workload between multiple cloudlets in a network to optimize mobile application performance. We first introduce a system model to capture the response times of offloaded tasks, and formulate a novel optimization problem, that is to find an optimal redirection of tasks between cloudlets such that the maximum of the average response times of tasks at cloudlets is minimized. We then propose a fast, scalable algorithm for the problem. We finally evaluate the performance of the proposed algorithm through experimental simulations. The experimental results demonstrate the significant potential of the proposed algorithm in reducing the response times of tasks.**

I. Introduction

In recent years, advances in mobile computing have enabled mobile users to experience a plethora of engaging applications. However, while the resource demands of newly developed applications continue growing, the computing capacity of mobile devices remains limited due to their portable sizes. A traditional approach to overcoming the resource poverty of mobile devices is to leverage the rich computing resource of clouds. A mobile device can reduce its workload and prolong its battery life by offloading its computation-intensive tasks to a remote cloud for execution [7], [20]. However, one significant limitation of offloading tasks to a remote cloud is that the distance between the mobile users and the remote cloud is usually long. Such long distances cause long delays and even lags in applications with heavy user interactions, downgrading the user experiences. To mitigate this limitation, many researchers have suggested the use of clusters of computers called *cloudlets*, placed within user networks to support mobile devices by executing offloaded tasks [16], [17], [18], [19], [20], [22], [23].

A cloudlet is a trusted, resource-rich cluster of computers wirelessly connected to its nearby mobile users [17]. By providing low-latency access to rich computing resource, cloudlets can dramatically improve the performance of mobile applications, through allowing mobile users offload their applications to near by cloudlets [6], [17]. Although cloudlets are often defined as isolated "data centers in a box" [17], there are clear benefits to connecting multiple cloudlets together to form a network. Recent studies [13], [22], [23] discussed how cloudlets could be deployed in public wireless metropolitan area networks (WMANs), as a complimentary service to Wi-Fi Internet access. Metropolitan areas have a high population density, thus cloudlets can be accessible to a large number of users. This improves the cost-effectiveness of cloudlets as they are less likely to be idle. Furthermore, due to the size of such a network, WMAN service providers can take advantage of economies of scale when offering cloudlet services through WMANs, making cloudlet services more affordable to the general public.

A major challenge that WMAN service providers face is how to allocate different user task requests to different cloudlets so that the loads among the cloudlets in the network are well balanced, thereby shortening the response times of offloaded tasks. A typical solution to this problem is to allocate user requests to their closest cloudlets to minimize the network access delay. This approach however has been demonstrated to be inadequate in the WMAN setting [13], [22], [23]. Specifically, the vast number of users in the network means that the workload of each individual cloudlet will be highly volatile. If a cloudlet is suddenly overwhelmed with user requests, the response times of tasks at that cloudlet will increase dramatically, causing lags in the user applications and degrading user experiences. To prevent such issues from happening, it is critical to allocate user requests to different cloudlets such that the workload among the cloudlets is well balanced, thus reducing the average response time of the offloaded tasks. In this paper, we deal with such a load balancing problem among the cloudlets in response to dynamic demands of user requests, by devising an efficient scheduling algorithm to allocate user requests to different cloudlets.

In this paper we devise a load balancing algorithm for cloudlets within a WMAN, to reduce the average response time of offloaded tasks, for which we first introduce a system model to capture the response times of tasks, and formulate a novel optimization problem with the objective of finding a redirections of tasks between a given set of cloudlets in a network such that the maximum of the average response times of offloaded tasks among the cloudlets is minimized. We then propose a fast and scalable algorithm for the problem. We finally evaluate the performance of the proposed algorithm through experimental simulations. The experimental results demonstrate the significant potential of our algorithm in reducing response times of user tasks and maximizing user experiences.

The rest of the paper is organized as follows. Section II reviews related works. Section III introduces the system model and problem definition. Section IV gives a detailed description of the cloudlet placement algorithm, and Section V presents a case study and simulation results. A conclusion is drawn in Section VI.

II. Related Works

Offloading mobile applications to a remote cloud to overcome the limitations of mobile devices has been studied for over a decade [2], [16]. Generally, the model for application offloading systems in mobile cloud computing consists of a client component on the mobile device and a server component on a remote cloud [5], [7]. The client component is responsible for monitoring network performance, predicting computation requirements of mobile applications, and estimating execution time on local devices and on the cloud. Using this information, the client component can decide how much to offload. Recent works such as ThinkAir [15], Virtual Smartphone [4], and CloneCloud [5] have made use of virtual machines (VMs) as a platform for task offloading. A VM image of a mobile device is transferred to the cloud, and tasks are remotely executed on the device's VM in the cloud, using task offloading operations. In addition, there are studies propose efficient offloading solutions to improve QoS requirements of users [12], smartly admit user requests to cloudlets [20], [21], jointly consider offloading to remote clouds and cloudlets [3], [10]. For example, Hoang *et al.* [12] aimed to maximize the revenue of service providers by proposing a linear programming solution for task offloading to cloudlets. Xia *et al.* [20], [21] proposed efficient online algorithms to admit user requests to a cloudlet dynamically. Gelenbe *et al.* [10] consider the offloading of user applications between a local cloud and a remote cloud, by incorporating energy consumption and QoS requirements of users.

The main drawback of offloading tasks to a remote cloud is the latency between mobile users and the remote cloud, which can disrupt the user experiences in interactive applications such as mobile games. Cloudlets overcome this drawback by providing low-latency accesses to rich computing resource, which can dramatically improve the performance of mobile applications. Application offloading to cloudlets generally follows a VM-based approach [11], [17], where a mobile device rapidly instantiates a compressed VM image of the application and transfers it to a cloudlet. The device then can remotely execute tasks on the cloudlet. Once a task has been executed, its result will be returned to its mobile user. By doing so, cloudlet resources could become constrained if there is a large number of task requests from users. Two recent studies [3], [13] presented a system model where the average response time of tasks at a cloudlet can be estimated, using queuing theory [14]. They further proposed that a cloudlet can offload some of its tasks to a remote cloud for execution under extreme loads. Verbelen *et al.* [18] proposed a fine grained approach to cloudlet offloading, where a mobile application would be dynamically partitioned at run-time into independent remotely executable components. This approach has the advantage of distributing its components among multiple cloudlets, thereby maximizing parallelism. Furthermore, the number of offloaded components can be dynamically scaled down if the network connectivity is poor, providing a smoother user experience.

Several recent studies further broadened the definition of cloudlets to include ad-hoc computers in the network. Verbelen *et al.* [18], [19] proposed such a cloudlet architecture, creating a framework which enables ad-hoc discovery of nearby devices in the network to share resources. In this paper we restrict our definition of cloudlets to being a cluster of computers co-located with an Access Point (AP).

The topic of cloudlet placements within wireless networks has also been explored recently in [13], [22], and [23]. Jia *et al.* [13] showed that a strategic placement of a limited number of cloudlets in a WMAN can greatly improve the performance of mobile user devices, and presented an algorithm for solving the cloudlet placement problem. Xu *et al.* [22], [23] formulated a capacitated cloudlet placement problem that places K cloudlets to some potential locations in a wireless metropolitan aware network with the objective to minimize the average cloudlet access delay between the mobile users and the cloudlets serving their requests. They devised approximation algorithms for the problems with guaranteed approximation ratios when all cloudlets have identical computing capacities.

III. Preliminaries

In this section we describe the system model and define the cloudlet load balancing problem.

A. System Model

We assume that a WMAN provider has set up K cloudlets $\{1, 2, \ldots, K\}$ at fixed locations in the WMAN.The cloudlets are co-located at access points (APs) in the network, and all cloudlets are connected with each other via network connection. We assume that user applications are dynamically partitioned into discrete offloadable tasks that can be processed at any of the K cloudlets. Each user will offload his/her tasks to a near by AP with a cloudlet, and the cloudlet can either add incoming tasks to its own queue for processing or redirect some of its tasks to another cloudlet in the network (See Fig. 1).

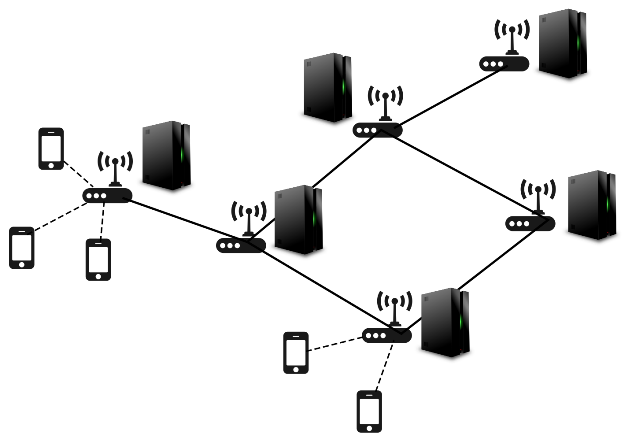


Fig. 1: A WMAN with APs

We model the cloudlets as *M/M/n* queues, where each cloudlet $i \in \{1, \ldots, K\}$ has n_i servers with the service rate μ_i at cloudlet i. Due to the rapidly changing nature of user demands, the rate of incoming requests can fluctuate wildly at each cloudlet over time. Therefore, we assume that the incoming user tasks at cloudlet i randomly arrive according to the Poisson process with arrival rate λ_i. The average response time of tasks at cloudlet i consists of the queuing time and the service time of tasks. Denote by T_i a function of a given task arrival rate λ to calculate the average response time of tasks at cloudlet i,

$$T_i(\lambda) = \frac{C\left(n_i, \frac{\lambda}{\mu_i}\right)}{n_i \mu_i - \lambda} + \frac{1}{\mu_i}, \tag{1}$$

where

$$C(n, \rho) = \frac{\left(\frac{(n\rho)^c}{n!}\right)\left(\frac{1}{1-\rho}\right)}{\sum_{k=0}^{n-1} \frac{(n\rho)^k}{k!} + \left(\frac{(n\rho)^c}{n!}\right)\left(\frac{1}{1-\rho}\right)}. \tag{2}$$

Eq. (2) is known as Erlang's C formula [14].

As task arrival rates at different cloudlets can be significantly different, some cloudlets may be overloaded while others may be under-utilized. We assume that all cloudlets are reachable from each other, and that a cloudlet can redirect a fraction of its task stream to another cloudlet. Denote by $f(i, j)$ the amount of task flow from cloudlet i to cloudlet j when $i \neq j$, see Fig. 2. There are the following constraints on $f(i, j)$.

$$f(i,j) = \begin{cases} -f(j,i), & \text{if } i \neq j \\ 0, & \text{otherwise} \end{cases}, \quad \forall i, j \in \{1, \ldots, K\} \tag{3}$$

$$\sum_{i=1}^{K} \sum_{j=1}^{K} f(i,j) = 0, \tag{4}$$

$$\sum_{j=1}^{K} \max\{f(i,j), 0\} \leq \lambda_i, \qquad \forall i \in \{1, \ldots, K\}. \tag{5}$$

Eq. (3) ensures that for any two cloudlets i and j, the flow in terms of task rate, from cloudlet i to cloudlet j is the negative of the flow from cloudlet j to cloudlet i. As the flow of tasks from any given cloudlet i to itself is zero, we have $f(i, j) = 0$. Eq. (4) ensures that all flow is conserved. Finally, Eq. (5) ensures that the sum of all outgoing task flows from cloudlet i is no greater than its incoming task arrival rate λ_i.

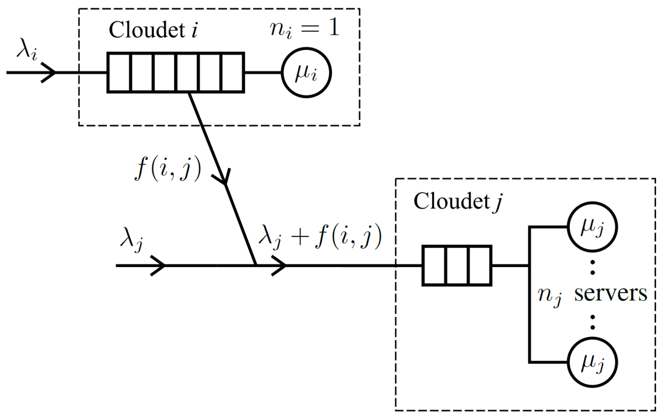


Fig. 2: Flow of tasks from cloudlet i to cloudlet j

We assume that all offloaded tasks have the same packet size, thus the delay incurred in transferring any task between a pair of APs through the network is identical. To model such a network delay in the WMAN, denote by $d \in \mathbb{R}^{K \times K}$ the network delay matrix, where entry $d_{i,j}$ represents the communication delay in relaying a task from cloudlet i to cloudlet j. We assume that the flow of incoming redirected tasks $f(i, j) < 0$ at cloudlet i has a delay of $-f(i, j) \cdot d_{i,j}$. We can then calculate the sum $T_{net}(i)$ of all network delays of incoming tasks from other cloudlets to cloudlet i as

$$T_{net}(i) = \sum_{j=1}^{K} \left| \max\{f(i,j), 0\} \cdot d_{j,i} \right|. \tag{6}$$

Using Eqs. (1) and (3), we can calculate $D(i)$, the average response time of tasks at cloudlet i, as:

$$D(i) = T_i(\overline{\lambda}_i) + T_{net}(i), \tag{7}$$

where $\overline{\lambda}_i$ is the final incoming task flow that will be processed at cloudlet i,

$$\overline{\lambda}_i = \lambda_i - \sum_{i=1}^{K} f(i,j). \tag{8}$$

Table I summarizes the symbols that are used in this paper.

B. Problem Definition

The Cloudlet Load Balancing Problem in G is defined as follows. Given a set of cloudlets $\{1, \ldots, K\}$, where each cloudlet i with task arrival rate λ_i and n_i servers with service rate μ_i for all $i \in \{1, \ldots, K\}$, the problem is to find a set of inter-cloudlet task flows $f = \{f(i,j) \mid i, j \in \{1, \ldots, K\}\}$ under the constraints given in Eqs. (3), (4), and (5), such that

Symbol	*Definition*
K	The number of cloudlets in the network
λ_i	Initial task arrival rate/workload for cloudlet i
n_i	The number of servers at cloudlets i
$f(i,j)$	The flow of tasks redirected from cloudlet i to cloudlet j
$\overline{\lambda}_i$	Resultant task arrival rate for cloudlet i given the set of inter cloudlet task flows $\{f(i,j) \mid i,j \in \{1,\ldots,K\}\}$
$T_i(\lambda)$	The average response time of tasks at cloudlet i when the arrival rate of tasks is λ
$T_{net}(i)$	Total network delay of incoming tasks arriving at cloudlet i given the set of inter cloudlet task flows $\{f(i,j) \mid i,j \in \{1,\ldots,K\}\}$
$d_{i,j}$	Network delay between AP i and j
$D(i)$	The average response time of tasks at cloudlet i
ϕ_i	The fraction of task stream to be redirected from or to cloudlet i
$\overline{D}$	An average response time of tasks in cloudlets
$T_{\min}$	$\min\{T_i(\lambda_i) \mid 1 \le i \le K\}$
$T_{\max}$	$\max\{T_i(\lambda_i) \mid 1 \le i \le K\}$
V_s	The set of overloaded cloudlets, i.e., $V_s = \{i \mid T_i(\lambda_i) > \overline{D}\}$
V_t	The set of underloaded cloudlets, i.e., $V_s = \{j \mid T_j(\lambda_j) \le \overline{D}\}$
ϵ, θ, δ	Accuracy thresholds that are used in the algorithm
Δ	The difference between the outgoing task flow from overloaded cloudlets and the incoming task flow to underloaded cloudlets
$\overline{D}'$	$\max\{D(j) \mid j \in V_t\}$
$G = (V,E)$	A flow network that is used to balance the workload of cloudlets in the algorithm
$u(i,j)$	The capacity of edge $\langle i,j \rangle$ in the flow network G
$c_{i,j}$	The delay cost of edge $\langle i,j \rangle$ in the flow network G

TABLE I: Symbols

the maximum of average response times of tasks among the K cloudlets, $\max_f\{D(i) \mid i \in \{1,\ldots,K\}\}$, is minimized.

IV. Algorithm

In this section, we propose a fast, scalable heuristic for the Cloudlet Loading Balancing Problem. The basic idea is to find an average response time of tasks $\overline{D}$ first, and then determine how much of its workload of each cloudlet should be redirected to other cloudlets such that the average response time of tasks at the cloudlet is approximately $\overline{D}$.

A. The average response time of tasks

Our objective is to minimize the maximum average response time of tasks in cloudlets. As the flow of tasks is conserved within the system (Eq. (4)), all tasks must eventually be executed. It is clear that we must try to redirect some tasks to and from cloudlets such that every cloudlet has the same average response time of tasks. The approach taken in this paper is to identify an average response time of tasks among the cloudlets and decide the out-going workload of each overloaded cloudlet and in-coming workload of each underloaded cloudlet. It then decides the specific flow of task streams from the overloaded cloudlets to the underloaded cloudlets.

Let $\overline{D}$ be the average response time of tasks. To find $\overline{D}$ and the amount of outgoing/incoming workload for each cloudlet, we estimate the value of $\overline{D}$ and iteratively refine the estimation until the average response time of tasks at every cloudlet is within a given bound ϵ of $\overline{D}$.

We begin by examining the value range of $\overline{D}$. Let $T_{\max} = \max\{T_i(\lambda_i) \mid 1 \le i \le K\}$ and $T_{\min} = \min\{T_i(\lambda_i) \mid 1 \le i \le K\}$, then it is obvious that $\overline{D}$ is in the value interval of $[T_{\min}, T_{\max}]$.

We choose $(T_{\min} + T_{\max})/2$ as the initial value of $\overline{D}$. We then partition all cloudlets into two disjoint sets, the set V_s of overloaded cloudlets:

$$V_s = \{i \mid T_i(\lambda_i) > \overline{D}\},$$

and the set V_t of underloaded cloudlets:

$$V_t = \{j \mid T_j(\lambda_j) \le \overline{D}\}.$$

For each overloaded cloudlet $i \in V_s$ we need to determine the amount of task flow that should be redirected from the cloudlet's arrival stream λ_i so that the task response time at cloudlet i is within an accuracy threshold ϵ of $\overline{D}$, i.e., we need to find a value of ϕ_i such that

$$\left|\overline{D} - T_i(\lambda_i - \phi_i)\right| \le \epsilon, \tag{9}$$

where ϵ is a given threshold.

For each underloaded cloudlet $j \in V_t$ we must determine ϕ_j the amount of task flow to arrive at cloudlet j so that the task response time at the cloudlet is within an acceptable bound of

$\overline{D}$, i.e.,

$$\left|\overline{D} - T_j(\lambda_j + \phi_j)\right| \leq \epsilon, \tag{10}$$

as illustrated by Fig 3.

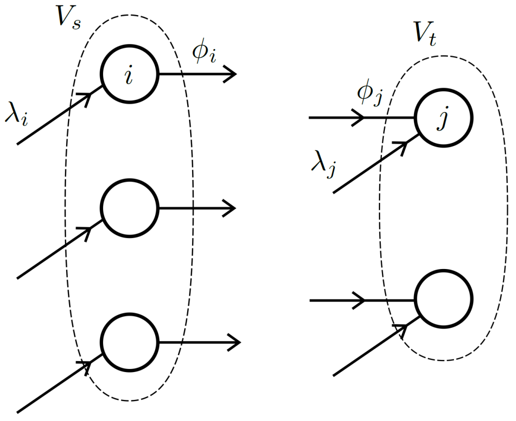


Fig. 3: An illustration of finding values of ϕ_i underloaded and ϕ_j for overloaded cloudlets

Finding values of ϕ_i and ϕ_j to satisfy Eqs (9) and (10) requires iterative trials that could take $O(\frac{1}{\epsilon})$. However, if we precalculate the required workload to achieve the average response time of tasks $\overline{D}$ within ϵ, we can find values of ϕ_i and ϕ_j in $O(1)$ time.

Once the desired outgoing flow ϕ_i for each overloaded cloudlet $i \in V_s$ and the desired incoming flow ϕ_j for underloaded cloudlets $j \in V_t$ have been calculated, we need to determine the value of $f(i,j)$, i.e., the fraction of task flow that should be redirected from cloudlet i to cloudlet j, for all $i \in V_s$ and $j \in V_t$ such that the accumulated network delay is minimized. Denote by Δ the difference between the outgoing task flow from overloaded cloudlets and the incoming task flow to underloaded cloudlets, i.e.,

$$\Delta = \sum_{i \in V_s} \phi_i - \sum_{j \in V_t} \phi_j. \tag{11}$$

If $\Delta = 0$, then the flow of tasks can be perfectly matched. If $\Delta > 0$, then there is a surplus of outgoing tasks. This implies that the chosen $\overline{D}$ is too small, as many cloudlets are trying to offload tasks from themselves to lower their average response time of tasks in order to meet $\overline{D}$. In this case we must therefore increase the lower bound $T_{\min} \leftarrow \overline{D}$. On the other hand, if $\Delta < 0$, then there is a deficit of outgoing tasks and $\overline{D}$ is too high; thus we must decreasing the upper bound $T_{\max} \leftarrow \overline{D}$. We choose $(T_{\max} + T_{\min})/2$ to be $\overline{D}$ in the next iteration, and find an appropriate $\overline{D}$, using binary search. The search will terminate when the accumulated flow Δ falls within a given threshold δ, i.e., $|\Delta| \leq \delta$.

Since redirecting tasks from overloaded cloudlets to underloaded cloudlets will incur a network delay at the destination cloudlet, we should further adjust $\overline{D}$ such that at every underloaded cloudlet $j \in V_t$ the sum of queueing time and network delay $T_{net}(j)$ of all tasks to cloudlet j is as close to $\overline{D}$ as possible: $T_j(\lambda_j) + T_{net}(j) = \overline{D}$. To this end, there must be a deficit of outgoing tasks from overloaded cloudlets, i.e., $\Delta < 0$.

We determine a set of inter-cloudlet flow f for a given $\overline{D}$, and we let $\overline{D}' = \max\{D(j) \mid j \in V_t\}$. If the difference between $\overline{D}$ and $\overline{D}'$ is within a given threshold θ, done; otherwise, we will need to further refine $\overline{D}$. Namely, if $\overline{D} < \overline{D}'$, then there are more outgoing tasks from overloaded cloudlets, thus we need to increase $\overline{D}$ to reduce ϕ_i for each overloaded cloudlet $i \in V_s$. If $\overline{D} > \overline{D}'$, then there are insufficient outgoing tasks from overloaded cloudlets, hence we should lower $\overline{D}$ to allow the overloaded cloudlets to redirect more tasks to the underloaded cloudlets. We choose $\overline{D} \leftarrow (\overline{D} + \overline{D}')/2$ to be the next iteration of $\overline{D}$ and continue this until the difference between $\overline{D}$ and $\overline{D}'$ is within θ. The details of this algorithm are given in Algorithm 1.

The rest is to find the values of $f(i,j)$ for all $i \in V_s$ and $j \in V_t$ that incur minimum delays, given an average response time $\overline{D}$ of tasks and the value of ϕ_i for each cloudlet $i \in \{1, \ldots, K\}$.

B. Minimum-latency flow

Having determined the amounts of outgoing and incoming flows of tasks ϕ_i for each cloudlet i, we need to determine the redirected task flow $f(i,j)$ for each overloaded cloudlet $i \in V_s$ to each underloaded cloudlet $j \in V_t$. We transform the problem of determining the amounts of outgoing flows of overloaded cloudlets to underloaded cloudlets into a minimum-cost maximum flow problem. Namely, we construct a flow network $G = (V, E)$ from the WMAN as follows.

We first construct a virtual source node s, and a virtual sink node t, and partition cloudlets into the set V_s of overloaded cloudlets and the set V_t of underloaded cloudlets based on a given $\overline{D}$. This gives us the set of vertices $V = V_s \cup V_t \cup \{s, t\}$. We then add a directed edge from s to each node in V_s, a directed edge from each node in V_t to t, and a directed edge from each node in V_s to each node in V_t. This gives us the set of edges $E = \{\langle s, i\rangle \mid i \in V_s\} \cup \{\langle j, t\rangle \mid i \in V_t\} \cup \{\langle i, j\rangle \mid i \in V_s, j \in V_t\}$.

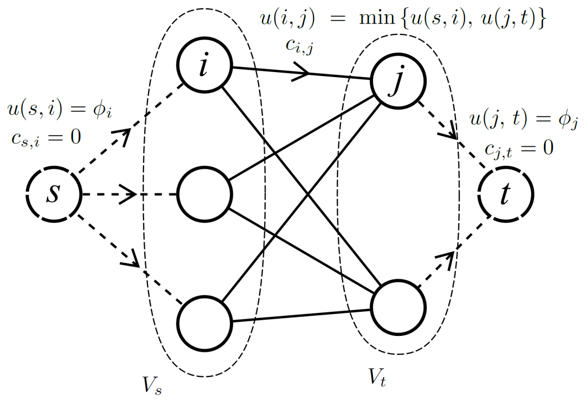


Fig. 4: A flow network

Algorithm 1 *Cloudlet Load Balance Algorithm*

Input: The task arrival rate λ_i, the number of server n_i, and the service rate μ_i of each cloudlet $i \in \{1, \ldots, K\}$; the network delay matrix $d_{i,j}$ $(i, j \in \{1, \ldots, K\})$; and tuning parameters θ, ϵ, δ.

Output: Inter-cloudlet task flow $f(i,j)$ for each pair of cloudlets $i, j \in \{1, \ldots, K\}$.

```
1: $T_{\max} \leftarrow \max\{T_i(\lambda_i) \mid 1 \le i \le K\}$;
2: $T_{\min} \leftarrow \min\{T_i(\lambda_i) \mid 1 \le i \le K\}$;
3: $\Delta \leftarrow \infty$;
4: /* find the initial value of $\overline{D}$ using binary search */
5: while ($|\Delta| > \delta$) do
6:     $\overline{D} \leftarrow (T_{\max} + T_{\min})/2$;
7:     $V_s \leftarrow \{i \mid T_i(\lambda_i) > \overline{D}\}$;
8:     $V_t \leftarrow \{i \mid T_i(\lambda_i) \le \overline{D}\}$;
9:     for each overloaded cloudlet $i \in V_s$ do
10:        Find a value of $\phi_i$ such that $\left|\frac{\overline{D} - T_i(\lambda_i - \phi_i)}{\overline{D}}\right| \le \epsilon$;
11:    end for
12:    for each underloaded cloudlet $j \in V_t$ do
13:        Find a value of $\phi_j$ such that $\left|\frac{\overline{D} - T_j(\lambda_j + \phi_j)}{\overline{D}}\right| \le \epsilon$;
14:    end for
15:    $\Delta \leftarrow \sum_{i \in V_s} \phi_i - \sum_{j \in V_j} \phi_j$;
16:    if $|\Delta| > 0$ then
17:        $T_{\min} \leftarrow \overline{D}$;
18:    else
19:        $T_{\max} \leftarrow \overline{D}$;
20:    end if
21: end while
22: /* adjust the value of $\overline{D}$ for the network delay between cloudlets */
23: $V_s \leftarrow \{i \mid T_i(\lambda_i) > \overline{D}\}$;
24: $V_t \leftarrow \{j \mid T_j(\lambda_j) \le \overline{D}\}$;
25: $\overline{D}' \leftarrow \infty$;
26: while ($|\overline{D} - \overline{D}'| > \theta$) do
27:    for each $i \in V_s$ do
28:        Find a value of $\phi_i$ such that $\left|\frac{\overline{D} - T_i(\lambda_i - \phi_i)}{\overline{D}}\right| \le \epsilon$;
29:    end for
30:    for each $j \in V_t$ do
31:        Find a value of $\phi_j$ such that $\left|\frac{\overline{D} - T_j(\lambda_j + \phi_j)}{\overline{D}}\right| \le \epsilon$;
32:    end for
33:    Calculate $f(i,j)$, by invoking Procedure minLatencyFlow($\overline{D}, V_s, V_t, \epsilon$);
34:    for each $j \in V_t$ do
35:        Calculate $D(j)$, using Eq.(7);
36:    end for
37:    $\overline{D}' \leftarrow \max\{D(j) \mid j \in V_t\}$;
38:    $\overline{D} \leftarrow (\overline{D} + \overline{D}')/2$;
39: end while
```

Denote by $u(i,j)$ the capacity of edge $\langle i, j\rangle$ in G. We begin by setting the edge capacities for the edges between the source node s and each overloaded cloudlet node $i \in V_s$ to ϕ_i, i.e., $u(s,i) = \phi_i$ for all edges $\langle s, i\rangle$ with $i \in V_s$, and the flow capacity of each overloaded cloudlet nodes V_t to ϕ_j, i.e., $u(j,t) = \phi_j$ for all edges $\langle j, t\rangle$ with $j \in V_t$. We then assign the cost of edges between the source s and cloudlet nodes in V_s to zeros, i.e., $c_{s,i} = 0$ for each $i \in V_s$. Similarly, we set the latency cost of the edges between the sink node t and cloudlet nodes in V_t to zeros, i.e., $c_{j,t} = 0$ for each $j \in V_t$. For each edge $\langle i, j\rangle$ from an overloaded cloudlet $i \in V_s$ to an underloaded cloudlet $j \in V_t$, we set its flow capacity to be the minimum of the incoming capacity at i and the outgoing capacity at j, i.e., $u(i,j) = \min\{u(s,i), u(j,t)\}$, and its cost to $d_{i,j}$, i.e., $c_{i,j} = d_{i,j}$. Fig. 4 illustrates the construction of the flow network $G(V,E)$.

Having constructed the flow network G, it can be seen that the problem of routing outgoing flows from overloaded cloudlets to underloaded cloudlets is transformed to the problem of finding a minimum cost maximum flow in G from s to t in terms of network latency. That is, our optimization objective is to

$$minimize \sum_{\langle i,j\rangle \in E} f(i,j) \cdot c_{i,j}, \tag{12}$$

subject to the following constraints.

$$f(i,j) \le u(i,j), \qquad \forall i, j \in V \tag{13}$$

$$f(i,j) = -f(j,i), \qquad i \ne s \text{ or } j \ne t \tag{14}$$

$$\sum_{j \in V} f(i,j) = 0, \qquad i \ne s \text{ or } j \ne t \tag{15}$$

where $f(i,j) \cdot c_{i,j}$ is the amount of network delay incurring by transferring tasks from cloudlet i to cloudlet j.

Procedure 1 *minLatencyFlow*

Input: An average response time $\overline{D}$ of tasks; a set of overloaded cloudlets V_s; a set of underloaded cloudlets V_t; and a tuning parameter ϵ

Output: Redirected task flow $f(i,j)$ for all $i, j \in \{1, \ldots, K\}$.

```
1: /* Construct a flow network with latency weighted edges. */
2: $V \leftarrow \{1, \ldots, K\} \cup \{s, t\}$;
3: $V_s \leftarrow \{i \mid i \in \{1, \ldots, K\}, T_i(\lambda_i) > \overline{D}\}$;
4: $V_t \leftarrow \{j \mid i \in \{1, \ldots, K\}, T_j(\lambda_j) \le \overline{D}\}$;
5: $E \leftarrow \emptyset$;
6: for each overloaded cloudlet $i \in V_s$ do
7:     $E \leftarrow E \cup \{\langle s, i\rangle\}$;
8:     Find a value of $\phi_i$ such that $\left|\frac{\overline{D} - T_i(\lambda_i - \phi_i)}{\overline{D}}\right| \le \epsilon$;
9:     $u(s,i) \leftarrow \phi_i$;
10:    $c_{s,i} \leftarrow 0$;
11: end for
12: for each underloaded cloudlet $j \in V_t$ do
13:    $E \leftarrow E \cup \{\langle j, t\rangle\}$;
14:    Find a value of $\phi_j$ such that $\left|\frac{\overline{D} - T_j(\lambda_j + \phi_j)}{\overline{D}}\right| \le \epsilon$;
15:    $u(j,t) \leftarrow \phi_j$;
16:    $c_{j,t} \leftarrow 0$;
17: end for
18: for each $i \in V_s$ do
19:    for each $j \in V_t$ do
20:        $E \leftarrow E \cup \{\langle i, j\rangle\}$;
21:        $u(i,j) \leftarrow \min\{u(s,i), u(j,t)\}$;
22:    end for
23: end for
24: Find a minimum-cost maximum-flow in the flow network $G(V,E)$, subject to edge capacities $u(u,v)$ for $\langle u, v\rangle \in E$, by invoking the transportation algorithm in [9].
```

This is clearly the Hitchcock Transportation Problem [9], which aims to find a maximum flow in G from s to t such that the total cost of the flow is minimized, which can be solved, using the Transportation Algorithm [8], [9] within $O(K^4)$ time complexity, where $K = |V|$ is the number of cloudlets in G. `Procedure` 1 details the construction of the flow network G for a given average response time $\overline{D}$ of tasks.

In summary, the proposed algorithm is to identify an average response time $\overline{D}$ of tasks and to determine the out-going workload for each overloaded cloudlet and the in-coming workload for each underloaded cloudlet, based on the value of $\overline{D}$. We begin with binary search to find $\overline{D}$ such that the outgoing task demand from overloaded cloudlets is within a given threshold δ of the incoming task demand from underloaded cloudlets. We then compute the specific flow of task streams from overloaded cloudlets to underloaded cloudlets, using a transportation algorithm [9]. We finally calculate the total network latency caused by the flow of tasks through refining $\overline{D}$ iteratively until the value of $\overline{D}$ is within the bound θ of $\overline{D}'$, where $\overline{D}' = \max\{D(j) \mid j \in V_t\}$.

V. Simulation

In this section we evaluate the performance of the proposed algorithm in simulation environments. We begin by explaining the simulation settings, and then evaluating the performance of the proposed algorithm, using different networks.

A. Simulation Environment

We first study a hypothetical structure of a cloudlet network. We adopted a similar method used in [13] to generate WMANs. We assume that the positions of cloudlets in a WMAN follow a scale-free distribution, and we generate random scale-free cloudlet network topologies, using the Barabasi-Albert Model [1]. Similar to [13], we assume that network delay between two APs in the network is proportional to their physical distance. As distances between real APs are essentially random, we assign the network delay between each pair of directly linked cloudlets randomly, according to the Normal distribution: $0.1 \leq \mathcal{N}(0.15, 0.05) \leq 0.2$. This randomizes the delay in the network while preserving the triangle distance inequality, as any pair of nodes with 2 degrees of separation has an intermediary distance of at least 0.2.

For each cloudlet i, we assign its service rate μ_i by sampling the Normal distribution $\mathcal{N}(5, 2) > 0$, and the number of servers by sampling the Poisson distribution with a mean of 3. The task arrival rate λ_i at cloudlet i is determined by the Normal distribution $0 < \mathcal{N}(15, 6) < \mu_i \cdot n_i - 0.25$. Notice that the arrival rate λ_i must not exceed $\mu_i \cdot n_i$, otherwise, the queue time at cloudlet i will be infinite.

B. Simulation Results

Fig. 5(a) plots the probability distributions of response times of tasks with and without adopting the proposed algorithm, respectively. As these figures indicate that the proposed algorithm narrows the range of cloudlet task response times between 0.4 and 0.8 seconds. The range is larger than our accuracy bound $\theta = 0.1$, which is partially due to network delay causing a mismatch between outgoing task demands at overloaded cloudlets, and incoming task demands at underloaded cloudlets.

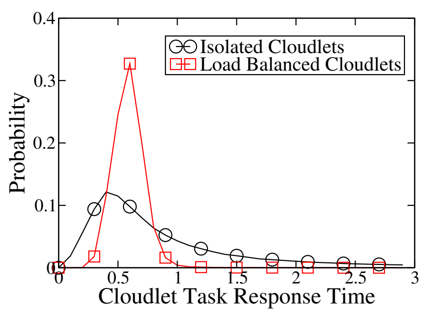
Symbol	*Definition*	*Default*
K	Number of cloudlets in the network	20
λ_i	Average task arrival rate at cloudlet i	15
μ_i	Service rate of servers at cloudlet i	5
n_i	Average number of servers at cloudlet i	3
$c_{i,j}$	Network delay between AP p_i and p_j	0.15
θ	Termination quality of $\overline{D}$ and $\overline{D}_t$.	0.1
ϵ	Accuracy bound for calculating task demand ϕ_i for cloudlet i	0.05

TABLE II: Default simulation settings

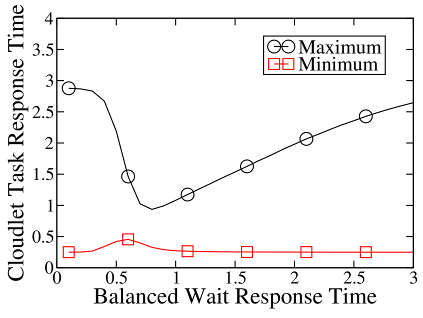
Fig. 5(b) shows the curves of the maximum and minimum response times of tasks after running the proposed algorithm with the average response time $\overline{D}$ of tasks being set at a fixed value. As previously discussed, when $\overline{D}$ is too low, the outgoing task demands from overloaded cloudlet are higher than the incoming task demands from the underloaded cloudlets. This limits the amount of flow between cloudlets, resulting in a wider range of cloudlet task response times. Similarly, when $\overline{D}$ is too high, the total outgoing task demand is lower than the total incoming task demand, and task flow in the network is limited. When $\overline{D}$ is close to zero, all the cloudlets are considered "overloaded" and consequently no flow can be redirected. When $\overline{D} = 0.5$, the minimum response time of tasks increases slightly as the algorithm identifies the first underloaded cloudlet j, and flow is redirected to meet the underloaded cloudlet's incoming task demand ϕ_j. The minimum response time of tasks begins to drop as $\overline{D}$ continues to increase, because more cloudlets are considered underloaded. It is worth noting that the minimum cloudlet task response time reaches a local maximum when $\overline{D} = 0.6$, whereas the maximum cloudlet task response time reaches the local minimum at $\overline{D} = 0.8$. The difference between these two stationary points correspond to the network delay between the overloaded and underloaded cloudlets.

Fig. 5(c) examines the maximum response time of tasks when there is no task flow and after task flow is calculated by the proposed algorithm, with the growth of the number of cloudlets in the network. As there are more and more cloudlets in the system, the likelihood of an overloaded cloudlet increases in the system increases, and so the blue line tends towards the maximum cloudlet task response time limit we have imposed (by restricting cloudlet task arrival rate $\lambda_i < \mu_i \cdot n_i - 25$). As can be seen, the performance of our algorithm is unaffected as we increase the number of cloudlets in the network.

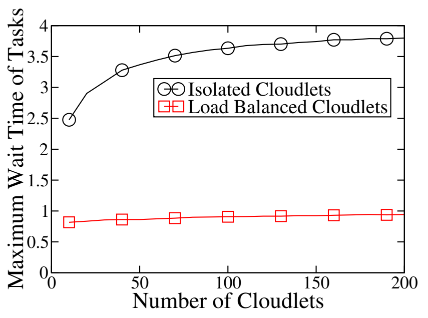
Fig. 5(d) shows the running time of the proposed algorithm with the increase of the number of cloudlets in the network. The running time was obtained based on a machine with a 3.40 GHz Intel i7 Quadcore CPU and 16GiB RAM.The running time increases dramatically as the number of cloudlets increases. However, the flow can still be calculated on a



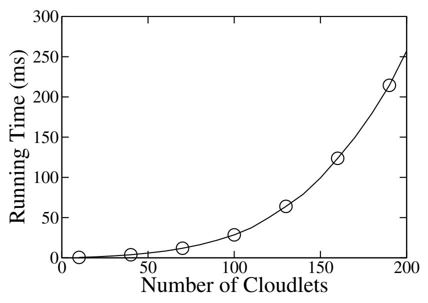
(a) Distributions of task response times



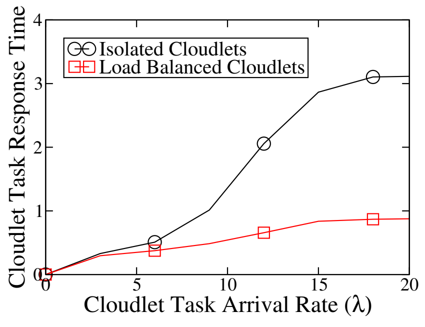
(b) Task response time for the selected average response times of tasks



(c) Maximum response times of tasks in WMANs with different number of cloudlets



(d) Running times of the proposed algorithm in WMANs with different numbers of cloudlets



(e) Task response times with different task arrival rates

Fig. 5: Performance evaluation of the proposed algorithm

desktop machine within less than 0.3 seconds for a network with 200 cloudlets. Large city areas typically span no more than several hundred square kilometers (km), so by placing a cloudlet every 2–3 km^2, even a large WMAN can be supported by a few hundred cloudlets. Refreshing the algorithm results every few seconds is sufficient to ensure that the redirection of cloudlet tasks in the network is up to date, while user demands shift around the network.

Fig. 5(e) compares the maximum cloudlet task response time in a static network, and after flow is calculated by the proposed algorithm, with increasing the average arrival rate of tasks under sampling the normal distribution, from which it can be seen that the maximum cloudlet task response time can increase dramatically as cloudlet task arrival rate increases, while the maximum cloudlet task response time increases at a low rate. A plateau is reached as cloudlet arrival rate λ_i for cloudlet i is capped for each cloudlet, even as the average cloudlet arrival rate according to the normal distribution.

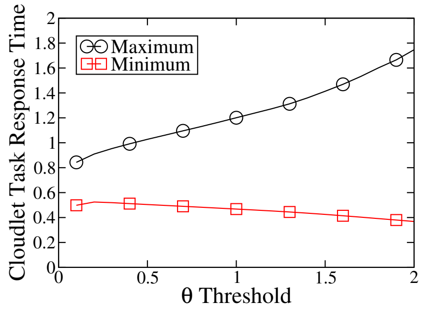
Fig. 6 studies the two accuracy measures in the proposed algorithm: θ and ϵ, where θ determines how close the maximum underloaded cloudlet task response time (after network flow) is to the chosen $\overline{D}$ cloudlet task response time before the solution is acceptable, while ϵ controls how accurate the cloudlet task demand ϕ_i is for each cloudlet i.

Fig. 6(a) plots the maximum and minimum cloudlet task response time in the network against the algorithm's main termination accuracy θ. With the increase of the value of θ, the termination condition of the proposed algorithm becomes more lax, and the number of incremental refinements of $\overline{D}$ is reduced. This results in the maximum cloudlet task response time increasing and the minimum cloudlet task response time decreasing.

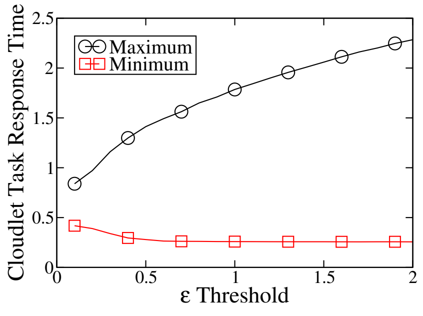
Fig. 6(b) studies the maximum and minimum cloudlet task response time in the network against the algorithm's accuracy bound ϵ of the task demand ϕ_i for given cloudlet i. As ϵ increases, the error margin of task demand at each cloudlet increases too, resulting in the algorithm redirecting the correct amount of flow for a given balanced cloudlet task response time $\overline{D}$. Despite the accuracy bound θ being unchanged, the proposed algorithm is less likely to converge towards the optimal $\overline{D}$, and as a result, the algorithm can terminates with a suboptimal flow.

Fig. 6(c) shows the running time the proposed algorithm as θ increases. Clearly, the curve is asymptotic as θ approaches zero, and as θ increases, the running time converges towards 0.6 milliseconds. While the results presented here are not conclusive, there is clearly a trade-off between the running time accuracy and the maximum cloudlet task response time delivered by the algorithm. The ϵ accuracy bound does not necessarily affect the running time of the algorithm, as we can precalculate the required load λ for any cloudlet task response D within ϵ accuracy for all cloudlets i. In this case, ϵ only affects the per-processing time whenever cloudlet hardware is installed or upgraded.

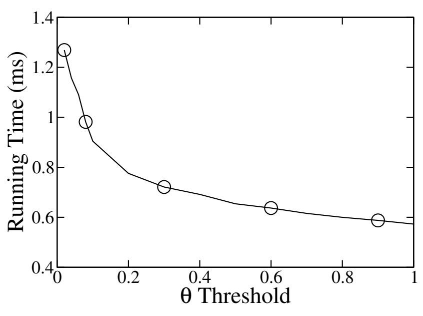
To conclude, the proposed algorithm can significantly reduce the maximum cloudlet task response time in a network



(a) θ threshold impact on task response time



(b) ϵ threshold impact on task response time



(c) θ threshold impact on running time

Fig. 6: The impacts of thresholds θ and ϵ on the performance of the proposed algorithm

of cloudlets, while the running time of the proposed algorithm for large numbers of cloudlets, is modest.

VI. Conclusion

Cloudlets are an important technology that provides performance improvements to mobile applications. As wireless Internet availability continues growing, public available and easy-to-access cloudlets will be vital to the future mobile computing. In this paper, we studied the problem of cloudlet load balancing in a WMAN, and proposed a fast and scalable algorithm for workload balancing among the cloudlets to shorten the average response time of tasks. Simulation results demonstrated that the proposed algorithm is promising.

References

[1] R. Albert, H. Jeong, and A. Barabási. Internet: Diameter of the world-wide web. *Nature*, Vol. 401, pp. 130–131, 1999.

[2] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I Yang. The case for cyber foraging. *Proceedings of 10th SIGOPS European Workshop*, pp. 87–92, ACM, 2002.

[3] V. Cardellini, V. D. N. Personé, V. D. Valerio, F. Facchinei, V. Grassi, F. Lo Presti, and V. Piccialli. A game-theoretic approach to computation offloading in mobile cloud computing. Technical Report, http://www.optimizationonline.org/DBHTML/2013/08/3981.html, 2013.

[4] E. Y. Chen and M. Itoh. Virtual smartphone over ip. *Proceedings of 14th International Symposium on World of Wireless Mobile and Multimedia Networks*, pp. 1–6, IEEE, 2010.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. *Proceedings of EuroSys*, pp. 301–314, ACM, 2011.

[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. *Proceedings of International Conference on Pervasive Computing and Communications*, pp. 122–127, IEEE, 2012.

[7] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. *Proceedings of 8th International Conference on Mobile Systems, Applications, and Services*, pp. 49–62, ACM, 2010.

[8] A. Dolan and J. Aldous. *Networks and Algorithms: An Introductory Approach.* J. Wiley & Sons Chichester, 1993.

[9] L. R. Ford and D. R. Fulkerson. *A simple algorithm for finding maximal network flows and an application to the Hitchcock problem.* Citeseer, 1955.

[10] E. Gelenbe, R. Lent, and M. Douratsos. Choosing a local or remote cloud. *Proceedings of 2nd International Symposium on Network Cloud Computing and Applications*, pp. 25–30, IEEE, 2012.

[11] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan. Just-in-time provisioning for cyber foraging. *Proceedings of 11th International Conference on Mobile Systems, Applications, and Services*, pp. 153–166. ACM, 2013.

[12] D. T. Hoang, D. Niyato, and P. Wang. Optimal admission control policy for mobile cloud computing hotspot with cloudlet. *Proceedings of Wireless Communications and Networking Conference*, IEEE, 2012.

[13] M. Jia, J. Cao, and W. Liang. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. To appear in *IEEE Transactions on Cloud Computing*, April, 2015.

[14] L. Kleinrock. *Queueing Systems, volume i: Theory.* pp. 101–103, 1975.

[15] 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. *Proceedings of INFOCOM*, pp. 945–953, IEEE, 2012.

[16] M. Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications*, Vol. 8, issue 4, pp. 10–17, IEEE, 2001.

[17] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing*, Vol. 8, issue 4, pp. 14–23, IEEE, 2009.

[18] T. Verbelen, P. Simoens, F. D. Turck, and B. Dhoedt. Cloudlets: Bringing the cloud to the mobile user. *Proceedings of 3rd workshop on Mobile Cloud Computing and Services*, pp. 29–36, ACM, 2012.

[19] T. Verbelen, P. Simoens, F. D. Turck, and B. Dhoedt. Leveraging cloudlets for immersive collaborative applications. *Pervasive Computing*, Vol. 12, issue 4, pp. 30–38, IEEE, 2013.

[20] Q. Xia, W. Liang, and W. Xu. Throughput maximization for online request admissions in mobile cloudlets. *Proceedings of 38th Conference on Local Computer Networks*, pp. 589–596, IEEE, 2013.

[21] Q. Xia, W. Liang, Z. Xu, and B. Zhou. Online algorithms for location-aware task offloading in two-tiered mobile cloud environments. *Proceedings of 7th International Conference on Utility and Cloud Computing*, pp. 109–116, IEEE/ACM, 2014.

[22] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo. Capacitated cloudlet placements in wireless metropolitan area networks. *Proceedings of 40th Conference on Local Computer Networks*, IEEE, 2015.

[23] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo. Efficient algorithms for capacitated cloudlet placements. To appear in *IEEE Transactions on Parallel and Distributed System*, Dec., 2015.