QoS-Aware Cloudlet Load Balancing in Wireless Metropolitan Area Networks

Mike Jia, Weifa Liang, *Senior Member, IEEE*, Zichuan Xu, *Member, IEEE*, Meitian Huang, and Yu Ma

Abstract—With advances in wireless communication technology, more and more people depend heavily on portable mobile devices for business, entertainments and social interactions. This poses a great challenge of building a seamless application experience across different computing platforms. A key issue is the resource limitations of mobile devices due to their portable size, however this can be overcome by offloading computation-intensive tasks from the mobile devices to clusters of nearby computers called cloudlets through wireless access points. 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 However, the outdated notion of treating cloudlets as isolated data-centers-in-boxes must be discarded as there are clear benefits to connecting multiple cloudlets together to form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 among cloudlets in an WMAN to optimize mobile application performance. We first introduce a novel system model to capture the response time delays of offloaded tasks and formulate an optimization problem with the aim to minimize the maximum response time of all offloaded tasks. We then propose two algorithms for the problem: one is a fast heuristic, and another is a distributed genetic algorithm that is capable of delivering a more accurate solution compared with the first algorithm, but at the expense of a much longer running time. We finally evaluate the performance of the proposed algorithms in realistic simulation environments. The experimental results demonstrate the significant potential of the proposed algorithms in reducing the user task response time, maximizing user experience.

Index Terms—Cloudlet placements, mobile user allocation, task response time minimization, mobile task offloading, mobile cloud computing

1 Introduction

In recent years, advances in mobile computing have enabled users to experience a plethora of engaging applications. However as resource demands of newly developed applications continue to grow, the computing capacity of mobile devices remain limited, due to their portable size. A traditional approach to overcoming the resource poverty of mobile devices is to leverage the rich computing resources in remote clouds. A mobile device can lighten its workload and prolong its battery life by offloading computation-intensive tasks to remote clouds for execution [12]. However, one significant limitation of offloading tasks to remote clouds is the distance between users and these remote clouds. Long delays between the clouds and users can cause lag in applications with heavy user interactions, upsetting user experiences. To minimize the response time delays of offloaded tasks to remote clouds, researchers have suggested the use of clusters of computers called cloudlets deployed within user networks to support mobile devices, by executing offloaded tasks on local cloudlets [23], [24], [25], [26], [29].

A cloudlet is a trusted, resource-rich cluster of computers wirelessly connected to its nearby mobile users [24]. By providing low-latency access to their rich computing resources, cloudlets can dramatically improve the performance of mobile applications. If no cloudlet is available nearby, it is often assumed that a user can offload its application to the remote cloud, or run the application on his mobile device [11], [24]. Although cloudlets are often defined as isolated "data centers in a box", there are clear benefits to connecting multiple cloudlets together to form a network. A recent study [16], [32], [33] discussed how cloudlets could be deployed in a public wireless metropolitan area networks (WMANs), as a complimentary service to Wi-Fi Internet access. Metropolitan areas usually have a high population density, meaning that cloudlets will 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 the network, WMAN service providers can take advantage of economies of scale when offering cloudlet services through the WMAN, making cloudlet services more affordable to the general public.

A major problem that WMAN service providers face is how to allocate user task requests to different cloudlets so that the workload among cloudlets in the WMAN are well balanced, thereby shortening the response time delay of tasks and enhancing user experience in the use of their service. A typical solution to this problem is to allocate user requests to their closest cloudlets to minimize the network delay, however this approach has been demonstrated to be inadequate in an WMAN setting [16]. Specifically, the vast

- *M. Jia, W. Liang, M. Huang, and Y. Ma are with the Research School of Computer Science, Australian National University, Canberra, ACT 0200, Australia. E-mail: {u551528, u4700480, u5108648}@anu.edu.au, wliang@cs.anu.edu.au.*
- *Z. Xu is with the School of Software, Dalian University of Technology, Dalian 116620, P. R. China. E-mail: z.xu@dlut.edu.cn.*

Manuscript received 23 Feb. 2017; revised 11 Oct. 2017; accepted 15 Dec. 2017. Date of publication 2 Jan. 2018; date of current version 9 June 2020. (Corresponding author: Mike Jia.)
Recommended for acceptance by A. Vasilakos.
Digital Object Identifier no. 10.1109/TCC.2017.2786738

number of users in the network means that the workload at each individual cloudlet will be highly volatile. If a cloudlet is suddenly overwhelmed with user requests, the task response time at the cloudlet will increase dramatically, causing lag in the user applications and degrading user experiences. To prevent some cloudlets from being overloaded, it is crucial to assign user requests to different cloudlets such that the workload amongst the cloudlets is well balanced, thereby reducing the maximum response time of offloaded tasks.

In this paper we deal with the QoS-aware load balancing problem among cloudlets in an WMAN in response to the dynamic resource demands of user requests, by devising efficient scheduling algorithms to allocate user requests to different cloudlets. Specifically, we devise two load balancing algorithms for cloudlets within an WMAN, to reduce the maximum response time of offloaded tasks from mobile users that consists of queuing and processing time delays at each cloudlet and routing time delays of packets between users and cloudlets.

The main contributions of this paper are as follows.

- We first introduce a novel system model to capture the response time delays of offloaded tasks and formulate a novel optimization problem—the cloudlet load balancing problem (CLBP).
- We then propose a fast heuristic algorithm that delivers a feasible solution to the problem. We also develop a distributed genetic algorithm that delivers a more accurate solution to the problem at the expense of a longer running time.
- We finally evaluate the performance of the proposed algorithms in realistic simulation environments. Experimental results demonstrate the significant potential of the proposed algorithms in reducing the user task response time and maximizing user experience.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 introduces the system model and problem definition. Section 4 gives a detailed description of the fast heuristic algorithm. Section 5 proposes the distributed genetic algorithm. Section 6 presents the simulation results, and the conclusion is drawn in Section 7.

2 Related Works

Offloading mobile applications to a remote cloud to overcome the limitations of mobile devices has been studied in the past decade [4], [7], [23], [30], [31]. In general, 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 [9], [12]. The client component is responsible for monitoring network performance, predicting computation requirements of mobile applications, and estimating execution times on both local devices and the cloud. Using this information, the client component can decide how many of its tasks are to be offloaded. Recent works such as ThinkAir [20], Virtual Smartphone [8], and CloneCloud [9] 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 then remotely executed on the device's VM in the cloud, using task offloading operations.

The main drawback of offloading tasks to a remote cloud is the latency between mobile users and the remote cloud, which can disrupt user experiences in interactive applications such as mobile games. Cloudlets overcome this drawback by providing low-latency accesses to rich computing resource locally, which can dramatically improve the performance of mobile applications. Application offloading to cloudlets generally follows a VM-based approach [15], [24], where a mobile device rapidly instantiates a compressed VM image of the application and transfers it to a cloudlet, and the task then can be executed 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 [6], [16] presented a system model where the average response time of tasks at a cloudlet can be estimated, using queuing theory [19]. They further proposed that a cloudlet can offload some of its tasks to a remote cloud for execution under extreme loads. Verbelen et al. [25] 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 and executing them in parallel. 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. [25], [26] proposed such a cloudlet architecture, creating a framework which enables ad-hoc discovery of nearby devices in the network to share resources, while Wan et al. [28] proposed an architecture of integrating mobile cloud computing and vehicular cyber-physical systems.

A fundamental problem for cloudlets in an WMAN is how to balance the workload among the cloudlets, considering that workload balance plays a vital role in enhancing QoS of services and user experiences. Although load balancing has been extensively studied in centralized data centers, there are essential differences in load balancing between cloudlets and centralized clouds. Specifically, in a centralized data center, there is a centralized queue for all incoming user requests, the workload balancing and task allocations among servers in the data center is performed by a centralized scheduler called the hypervisor [10], [34]. In such a scenario, the task transfer delay and processing delay between servers in the data center is several orders of magnitude less than the task transfer and processing delays between different cloudlets in an WMAN [3], [33], since the bandwidth and computing resources within a data center is usually abundant. In contrast, in a distributed cloudlet environment, user requests are admitted by the network through their access points (APs), and cloudlets are usually co-located at the APs. Associated with each cloudlet is a queue of waiting tasks, and the QoS requirements of requests is now an important concern. Balancing the workload among the waiting queues at different cloudlets while minimizing the maximum task response time (delay) among offloaded tasks is challenging. In addition, due to limited processing capacities of cloudlets, the wait time delay of a task at a cloudlet is a non-linear

function of the workload (processing capacity) of the cloudlet. How to consider such a distributed queuing delay feature in the workload balance among cloudlets is another challenge. Furthermore, the transfer delay by redistributing tasks from one cloudlet to another cloudlet cannot be ignored. Although there are several studies on load balancing of cloudlets in WMANs [16], [31], [33], none of these studies focused on minimizing the maximum task response time through workload balancing among cloudlets.

In this paper, we restrict our definition of a cloudlet to being a cluster of computers co-located with an Access Point (AP) in an WMAN. The topic of cloudlet placement within wireless networks has been explored in [16] and [32], [33]. For example, the authors in [16] showed that the strategic placement of a limited number of cloudlets in an WMAN can greatly improve the performance of mobile user devices, and presented an algorithm for the cloudlet placement problem. In their other studies [32], [33], the authors formulated a capacitated cloudlet placement problem that places K cloudlets to some potential locations in an WMAN with the objective to minimize the average cloudlet access delay between mobile user requests and the cloudlets serving the requests of the mobile users under the computing capacity constraints on different cloudlets. They devised approximation algorithms with guaranteed approximation ratios for the problem. Although they dealt with the average delay of offloaded tasks, they didn't incorporate the queuing time delay and the work load at each cloudlet into consideration. The authors [17] recently considered QoS-aware task offloading to WMANs by assuming that different requests may have different network function service requirements, for which they developed an efficient heuristic for dynamic request admissions by utilizing existing instances of virtualized network functions in cloudlets. In addition, Ceselli et al. [7] studied the problem of cloudlet placements and the assignment of APs to cloudlets, by proposing a heuristic algorithm. Their objective is to minimize the operational cost of the edge network. Instead, the objective in this paper is to minimize the maximum task response time through balancing the workload among the cloudlets in an WMAN.

It must be mentioned that this study is an extended version of our previous work in a conference paper [18]. The main differences between this extension and its conference version lie in (1) the development of another algorithm—the distributed genetic algorithm, which delivers a more accurate solution at the expense of a longer running time; and (2) another set of experiments to validate the concepts and evaluate the performance of the proposed algorithms.

3 Preliminaries

In this section we first introduce our system model. We then define the problem precisely.

3.1 System Model

We assume that an WMAN service provider has set up K cloudlets $\{1, 2, \ldots, K\}$ at fixed locations in the WMAN. The cloudlets are co-located at access points in the network,

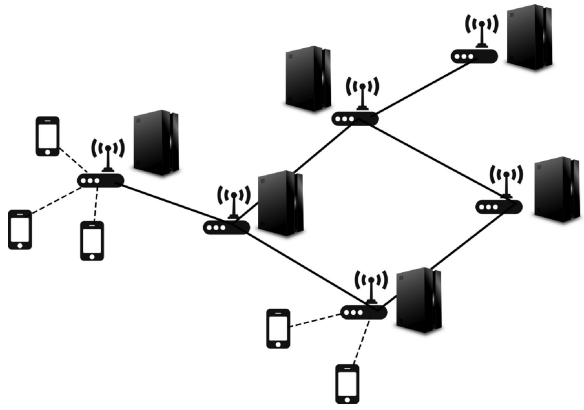


Fig. 1. An WMAN with APs.

and all cloudlets are connected to each other via network connection. We assume user applications are dynamically partitioned into discrete offloadable tasks that can be processed at any of the K cloudlets. Users will offload tasks to a nearby AP with a cloudlet, and the cloudlet can either choose to add the task to its own queue or redirect the task to another cloudlet in the network (See Fig. 1).

We model the cloudlets as $M/M/n$ queues, where cloudlet i has a number of servers n_i with service rate μ_i, for $i \in \{1, 2, \ldots, K\}$. Due to the rapidly changing nature of user demands, the rate of incoming requests can fluctuate wildly at each cloudlet over time. As such, we assume that the incoming user tasks at each cloudlet i arrive randomly according to the Poisson process with arrival rate λ_i. The average wait time of a task in cloudlet i consists of the queuing time and the service time of the task at cloudlet i. We use T_i, which is a function of a given task arrival rate λ, to calculate the average task wait time 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 [19].

As task arrival rates at different cloudlets can be significantly different, some cloudlets may be overloaded while others may be underloaded. We assume that all cloudlets are reachable from each other, and each cloudlet can redirect a fraction of its tasks to another cloudlet. We use $f(i, j)$ to denote the amount of task flow from cloudlet i to cloudlet j, for $i \neq j$ (see Fig. 2). We thus have the following constraint on $f(i, j)$

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

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

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

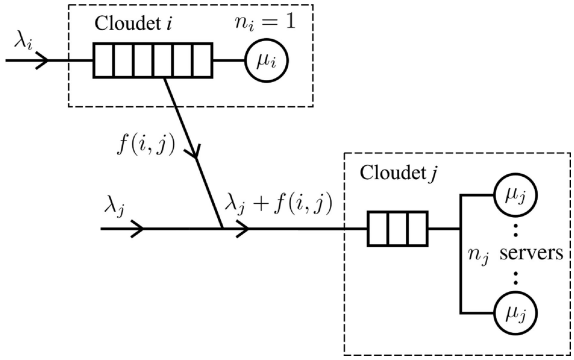


Fig. 2. The flow of tasks from cloudlet i to cloudlet j.

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, while Eq. (5) ensures that the sum of all outgoing task flows from cloudlet i (we ignore the incoming flow by summing the maximum of $f(i,j)$ and 0 for each cloudlet j) is less than its incoming task arrival rate λ_i.

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

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

Having Eqs. (1) and (3), the average task response time $D(i)$ of all tasks that are executed on cloudlet i can be calculated as follows:

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

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

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

A summary of system model and algorithm symbol definitions is given in Table 1.

3.2 Problem Definition

The Cloudlet Load Balancing Problem in an WMAN 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 $i \in \{1, 2, \ldots, K\}$ (for convenience we henceforth refer to the set of these given network parameters as NET), the problem is to find a set of inter-cloudlet task flows $f = \{\, f(i,j) \,|\, i,j \in \{1,2,\ldots,K\}\}$ under the constraints given in Eqs. (3), (4), and (5), such that the maximum task response time $D(i)$ is minimized, i.e.,

$$\min \max_{f}\ D(i). \tag{9}$$

TABLE 1
System and Algorithm Symbols

Symbol	*Definition*
K	Number of cloudlets in the network
$f(i,j)$	The flow of tasks from cloudlet i to cloudlet j.
λ_i	Initial task arrival rate/workload for cloudlet i
μ_i	Service rate of servers at cloudlet i
n_i	Number of servers at cloudlet i
$T_i(\lambda)$	Task response time at cloudlet i when the arrival rate of tasks is λ
$c_{i,j}$	Network delay between AP p_i and p_j
$T_{net}(i)$	Total network delay of incoming tasks arriving at cloudlet i given the set of inter cloudlet task flows $\{\delta_{i,j} \,|\, i,j \in \{1,2,\ldots,K\}\}$
$D(i)$	Task response time given the set of inter cloudlet task flows $\{\delta_{i,j} \,|\, i,j \in \{1,2,\ldots,K\}\}$
NET	The set of all given network parameters: $\{\lambda_i,\, n_i,\, \mu_i,\, c_{i,j} | i,j = 1,2,\ldots,K\}$
θ	Termination quality of $\overline{D}$ and $\overline{D}_t$.
ϵ	Accuracy bound for calculating task demand ϕ_i for cloudlet i
N	Maximum number of generations
R_{mut}	Rate of gene mutation
R_{mig}	Rate of gene migration
P	Gene population size on island
P_{mig}	Population percentage of migrant genes
S	Number of survivors in each generation

In this paper we propose two algorithms for the problem that strive for the tradeoff between the solution accuracy obtained by each algorithm and the running time of the algorithm.

4 Heuristic Algorithm

In this section we propose a heuristic for CLBP. The approach is to identify a balanced task response time $\overline{D}$ and then decide the out-going or in-coming workload of each cloudlet, based on $\overline{D}$. As we reduce the task response time of overloaded cloudlets by redirecting some of their tasks to underloaded cloudlets, the task response times of underloaded cloudlets are increased. By carefully directing workload (flow) between cloudlets, the tasks processed among the cloudlets will have roughly the same response time. To this end, we compute the task flow from overloaded cloudlets to underloaded cloudlets, using a transportation algorithm [14]. This procedure continues until the given accuracy bound is met.

4.1 Balancing Task Response Time

To find the balanced task response time $\overline{D}$ and decide the amount of outgoing/incoming workload for each cloudlet, we guess the value of $\overline{D}$ and iteratively improve it until it is within a given accuracy bound. We begin by examining the range of the average task wait time at cloudlets.

Let $T_{max} = \max_{1 \le i \le K} \{T_i(\lambda_i)\}$ and $T_{min} = \min_{1 \le i \le K} \{T_i(\lambda_i)\}$, then the value of $\overline{D}$ is in the range between T_{min} and T_{max}. We assign $\overline{D} = \frac{T_{max} + T_{min}}{2}$ as its initial value. We then partition all cloudlets into two disjoint sets, the set V_s of overloaded cloudlets

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

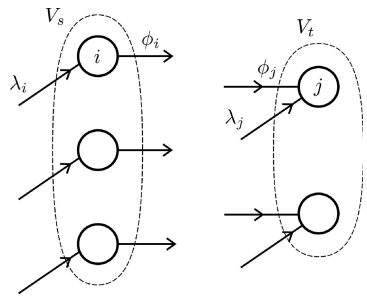


Fig. 3. Finding optimal outgoing and incoming workload for each cloudlet in an WMAN.

and the set V_t of underloaded cloudlets

$$V_t = \{\, j \,|\, T_j(\lambda_j) \leq \overline{D}\}.$$

For each overloaded cloudlet in $i \in V_s$, we decide the task demand ϕ_i of cloudlet i, which is the amount of task flow that should be redirected from its arrival task flow λ_i, such that its task response time is within a given accuracy ϵ of $\overline{D}$

$$\left|\overline{D} - T_i(\lambda_i - \phi_i)\right| \leq \epsilon, \tag{10}$$

where ϵ is a given threshold.

For each underloaded cloudlet $j \in V_t$, we decide the task demand ϕ_j of cloudlet j, which is the amount of task flow to arrive at cloudlet j such that its task response time is within the accuracy ϵ of $\overline{D}$

$$\left|\overline{D} - T_j(\lambda_j + \phi_j)\right| \leq \epsilon. \tag{11}$$

Once we have calculated ϕ_i for each overloaded cloudlet $i \in V_s$ and ϕ_j for each underloaded cloudlet $j \in V_t$, we can then determine the value of the redirected task flow $f(i,j)$ at the minimum cost, in terms of the network delay from overloaded cloudlets to underloaded cloudlets. Fig. 3 is an illustrative diagram of the calculation of ϕ_i and ϕ_j for all i and j with $1 \leq i \leq |V_s|$ and $1 \leq j \leq |V_t|$.

Because redirecting tasks from overloaded cloudlets to underloaded cloudlets will incur network delays at underloaded cloudlets, we should further adjust $\overline{D}$ such that the sum of the task response time at each underloaded cloudlet $j \in V_t$ and the incoming network delay $T_{net}(j)$ of all tasks to cloudlet j is nearly equal to $\overline{D}$, i.e., $T_j(\overline{\lambda}_j) + T_{net}(j) \approx \overline{D}$. Let $\overline{D'} = \max_{j \in V_t} \{T_j(\lambda_j)\}$. If the difference between $\overline{D}$ and $\overline{D'}$ is within a certain bound of accuracy θ, we are done; otherwise, we need to further refine $\overline{D}$. If $\overline{D} < \overline{D'}$, this means that we must reduce the amount of outgoing tasks from overloaded cloudlets, and we need to increase $\overline{D}$ by reducing ϕ_i for each overloaded cloudlet $i \in V_s$. Otherwise, we should increase the amount of outgoing tasks from overloaded cloudlets, and lower $\overline{D}$ to allow overloaded cloudlets to redirect more tasks to underloaded cloudlets. We choose $\overline{D} \leftarrow \frac{1}{2}(\overline{D} + \overline{D'})$, and recursively search for $\overline{D}$ and continue this procedure until the difference between $\overline{D}$ and $\overline{D'}$ is within the accuracy bound θ.

The rest is to find the minimum latency flow $f(i,j)$ when given a balanced task response time $\overline{D}$. The detailed algorithm is given in Algorithm 1.

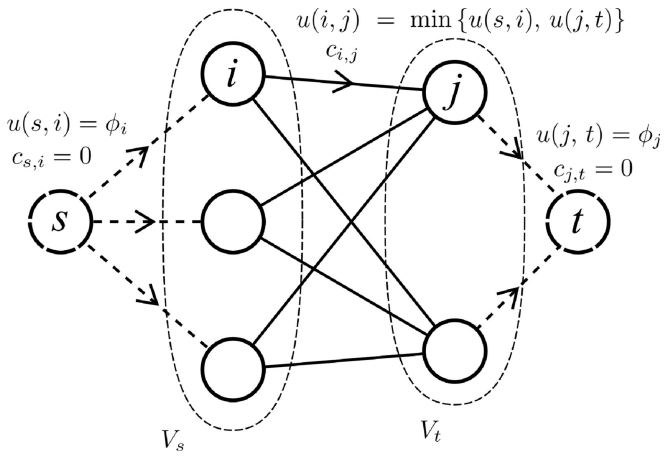


Fig. 4. Minimum-latency flow.

Algorithm 1. *CLBP-Heuristic Algorithm*

Input: NET, θ, ϵ
Output: $f(i,j)$, $i,j \in \{1,2,\ldots,K\}$.

1: $T_{max} \leftarrow \max_{1 \leq i \leq K} \{T_i(\lambda_i)\}$;
2: $T_{min} \leftarrow \min_{1 \leq i \leq K} \{T_i(\lambda_i)\}$;
3: $\overline{D} \leftarrow \frac{T_{max}+T_{min}}{2}$;
4: $V_s \leftarrow \{\, i \,|\, T_i(\lambda_i) > \overline{D}\}$;
5: $V_t \leftarrow \{\, j \,|\, T_j(\lambda_j) \leq \overline{D}\}$;
6: $\overline{D}' \leftarrow \infty$;
7: **while** $|\overline{D} - \overline{D}'| > \theta$ **do**
8: **for** each $i \in V_s$ **do**
9: calculate ϕ_i such that $\left|\frac{\overline{D} - T_i(\lambda_i - \phi_i)}{\overline{D}}\right| \leq \epsilon$;
10: **for** each $j \in V_t$ **do**
11: calculate ϕ_j such that $\left|\frac{\overline{D} - T_j(\lambda_j + \phi_j)}{\overline{D}}\right| \leq \epsilon$;
12: $\Phi \leftarrow \{\phi_k \,|\, k \in V_s \cup V_t\}$
13: calculate $f(i,j)$ for each $i,j \in \{1,2,\ldots,K\}$ by invoking Procedure `minLatencyFlow`(V_s, V_t, Φ);
14: **for** each $j \in V_t$ **do**
15: calculate $D(j)$ by Eq. (7);
16: $\overline{D}' \leftarrow \max_{j \in V_t}\{D(j)\}$;
17: $\overline{D} \leftarrow \frac{1}{2}(\overline{D} + \overline{D}')$;

4.2 Minimum-Latency Flow

Once we have determined the amount of outgoing or incoming task flow ϕ_k for each cloudlet k, we then determine for each outgoing task flow from an overloaded cloudlet i to each incoming task flow of underloaded cloudlet j, the value of the redirected task flow $f(i,j)$. To this end, we reduce the problem of routing the outgoing task flow from overloaded cloudlets to underloaded cloudlets to the minimum-cost maximum flow problem in an auxiliary flow graph $G = (V, E)$ derived from the original network as follows (see Fig. 4).

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

Denote by $u(i,j)$ the capacity of edge $\langle i,j\rangle \in E$. The edge capacity of each edge from the source node s to a cloudlet node $i \in V_s$ is set as $u(s,i) = \phi_i$ for each edge $\langle s,i\rangle$, and the edge capacity of each edge from an underloaded cloudlet node $j \in V_t$ to the sink node t is set as $u(j,t) = \phi_j$ for each edge $\langle j,t\rangle$. The latency cost of each edge from the source node s to an overloaded cloudlet node $i \in V_s$ is set as zero, i.e., $c_{s,i} = 0$. Similarly, the latency cost of each edge from each underloaded cloudlet node $j \in V_t$ to the sink node t is set as zero, i.e., $c_{j,t} = 0$. For each edge $\langle i,j\rangle$, $i,j \neq s,t$ from an overloaded cloudlet i to an underloaded cloudlet j, its edge capacity is set as $u(i,j) = \min\{u(s,i),\, u(j,t)\}$.

Having constructed the flow graph G, it can be seen that the problem of routing outgoing task flow from overloaded cloudlets to underloaded cloudlets is reduced to finding a minimum-cost and maximum-flow in G from s to t, i.e.,

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

subject to the following constraints:

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

$$\sum_{i\in V\setminus\{s\}} f(s,i) = \sum_{j\in V\setminus\{t\}} f(j,t), \qquad i \neq s \text{ or } j \neq t \tag{14}$$

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

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

This is clearly an instance of the Hitchcock Transportation Problem [14], and can be solved within $O(K^4)$ time, using a Transportation Algorithm [13], [14], where K is the number of cloudlets. The details are given in Procedure 1.

Procedure 1. *minLatencyFlow*

Input: V_s, V_t, $\{\phi_k \mid k \in V_s \cup V_t\}$
Output: $f_{i,j}$, $i,j \in \{1,2,\ldots,K\}$.

/* Construct the flow network with latency weighted edges. */
$V \leftarrow \{1,2,\ldots,K\} \cup \{s,\, t\}$;
$E \leftarrow \emptyset$;
for each $i \in V_s$ **do**
  $E \leftarrow E \cup \{\langle s,i\rangle\}$;
  $u(s,i) \leftarrow \phi_i$;
  $c_{s,i} \leftarrow 0$;
for each $j \in V_t$ **do**
  $E \leftarrow E \cup \{\langle j,t\rangle\}$;
  $u(j,t) \leftarrow \phi_j$;
  $c_{j,t} \leftarrow 0$;
for each $i \in V_s$ **do**
  for $j \in V_t$ **do**
    $E \leftarrow E \cup \{\langle i,j\rangle\}$;
    $u(i,j) \leftarrow \min\{u(s,i),\, u(j,t)\}$;
Solve the transportation problem in G using adapted Hungarian algorithm.

5 Distributed Genetic Algorithm

Although later experimental results indicate that the proposed heuristic, `Algorithm 1` in the previous section can deliver a feasible solution quickly, the solution may not be sufficiently accurate. In this section we devise a distributed genetic algorithm for CLBP that improves the accuracy of the solution at the expense of a longer running time. The key to this distributed algorithm is to perform fine-grain workload balancing among cloudlets iteratively through gene mutations until the solution converges on a given threshold.

5.1 Genetic Algorithm Operations

Genetic algorithms (GAs) have been widely used for combinatorial optimization problems. Traditional GAs maintain a population of solutions encoded as "genes" where only the fittest genes are bred to produce successively fitter generations of genes. However, genetic algorithms are often computation intensive and scale poorly. To overcome this, we design a distributed genetic algorithm for CLBP by leveraging the computing power of cloudlets. We first partition the cloudlets into overloaded and underloaded cloudlets, using each cloudlet $p \in \{1,\ldots,K\}$ as a partition reference. We then solve the CLBP problem using a distributed GA to find the fittest gene for each partition. We finally select the fittest gene among all partitions as the solution to the problem.

We begin by discussing the representation of genes. A gene is simply an encoded version of the task flow matrix f. While it is possible to use f directly as the gene representation, it is not the most efficient way. Many potential solutions using f as the gene will contain cloudlets that both receive and redirect flow, which guarantees that the solution is sub-optimal. By partitioning the cloudlets into overloaded and underloaded cloudlets, and representing only the task flow from overloaded to underloaded cloudlets, the potential solution space is substantially reduced. This in turn significantly reduces the number of iterations to an acceptable solution. If we sort the cloudlets according to their local task response times, we can partition the cloudlets using the task response time of a given cloudlet p as a reference to give us the sets V_s and V_t of overloaded and underloaded cloudlets as defined in the previous section. Let $g(i,j)$ denote the amount of task flow from cloudlet $i \in V_s$ to cloudlet $j \in V_t$, for $i \neq j$, where g is a $|V_s| \times |V_t|$ matrix. It can be seen that g has a one-to-one mapping to the task flow variable f, i.e., every gene has a unique corresponding flow matrix. We have the following constraint on $g(i,j)$:

$$\sum_{j\in V_t} g(i,j) \leq \lambda_i,\ \forall i \in V_s \tag{16}$$

$$\sum_{i\in V_s} g(i,j) \ <\ n_j\cdot\mu_j,\ \forall j \in V_t, \tag{17}$$

where Eq. (16) limits any given cloudlet $i \in V_s$ from redirecting more tasks than available according to its task arrival rate, and Eq. (17) limits any given cloudlet $j \in V_t$ from having a total incoming task flow of more than $n_j\cdot\mu_j$, as this would result in an infinite queue time at the cloudlet (see Eq. (1)).

Our initial gene population is constructed by randomly populating g_i with uniformly selected random numbers in the range $(0,\lambda_i)$. If a randomly generated gene violates one of the constraints, we randomly decrease values in the relevant row or column until the constraints are met.

Denote by P the given number of genes maintained in the gene population. The fitness of each gene is evaluated

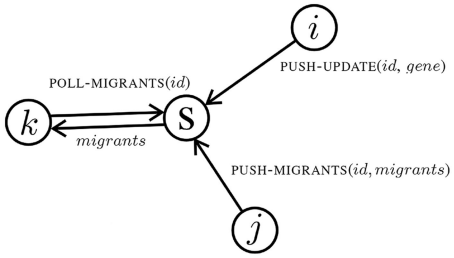


Fig. 5. Interactions between the supervisor and islands.

using our problem objective (defined in Formula (9)) as our fitness function. We then select only the fittest genes to survive to the next generation and repopulate the genepool. Two genes g_1 and g_2 breed to create an offspring gene g by taking the mean of each value

$$g(i,j) = \frac{1}{2}(g_1(i,j) + g_2(i,j)).$$

Denote by S the number of surviving genes that will persist into the next generation. We use the roulette selection to select S survivors and randomly crossbreed the survivors until the genepool population has been replenished. As we aim to minimize the fitness function, we take the reciprocal of each gene's fitness metric when performing roulette selection.

Denote by R_{mut} the mutation rate of gene offsprings, i.e., R_{mut} is the probability of an offspring gene being selected for mutation. We follow the mutation procedure given in [27], which shuffles values from randomly selected columns and rows. We continue to evolve the genepool until either an accuracy threshold is met or the maximum number of generations N is reached. In the final generation, the fittest gene is selected as the task flow between the overloaded cloudlets V_s and underloaded cloudlets V_t.

5.2 Distributed Algorithm

As the genetic operations described are performed on individual genes, independent of the rest of the population, these operations can be performed in parallel by the cloudlets in the network. However distributed genetic algorithms are more than just the parallel implementations of traditional sequential genetic algorithms. Numerous studies [1], [5], [21], [22] have shown that partitioning a large gene population into local groups called "islands" can encourage efficient selection of successful genetic traits, and generate better results than a traditional GA. In natural evolution, individuals rarely have the potential to mate with any partner in the entire population. Instead, breeding tends to be limited to sub-groups or neighborhoods, allowing the species at large to diversify. By limiting the reproductive access of genes to their local groups (islands), but occasionally allowing migrations of genes between islands, the genepool is less likely to stagnate, resulting in fitter genes.

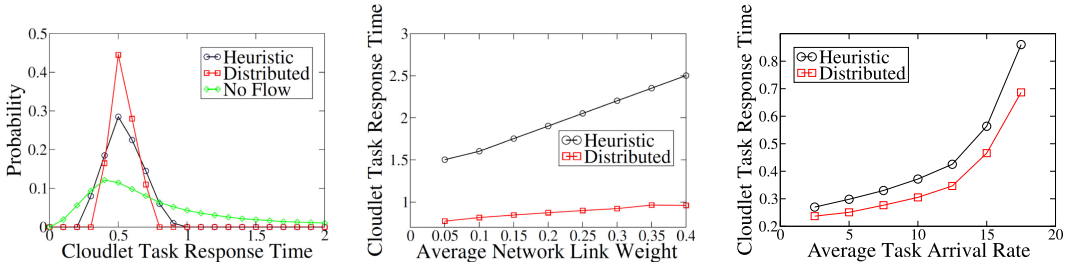
In the proposed distributed algorithm, we treat every cloudlet in the network as an island node that maintains and evolves its own genepool, using the genetic operations described. We further introduce R_{mig} to denote the migration rate of genes, where each generation has probability R_{mig} of migrating genes to another island, and we denote by P_{mig} the percentage of the population that will be migrated. Genes are selected for migration using the roulette method, similar to the selection of survivors. In each generation, islands will also poll the supervisor for in-bound migrant genes. If there exist inbound migrant genes at the island, the new genes are placed in the next generation of the genepool for breeding (for details see `Procedure` 2). We leave the task of coordinating the migration of genes across islands to a cloudlet acting as the supervisor.

Procedure 2. *CLBP-DGA-Island*(id, V_s, V_t)—Create an Island for Evolving Gene Population

Input: $NET, S, P, R_{mut}, R_{mig}, P_{mig}, N$
Output: $f(i,j)$, $i,j \in \{1,2,\ldots,K\}$.

```
genepool ← ∅;
for i ← 1 to P do
    Create a random gene according to Eqs. (18), (19);
    genepool ← genepool ∪ {gene};
for gen ← 1 to N do
    Sort genes by fitness;
    Push fittest gene to supervisor;
    if Random(0,1) < R_mig then
        Select P_mig · P genes to migrate;
        Push selected genes to supervisor for migration;
        Remove selected genes from genepool;
    genepool' ← ∅;
    Poll supervisor for inbound migrant_genes;
    if ∃ inbound migrant_genes then
        genepool' ← genepool' ∪ {migrant_genes};
    Select S survivor_genes from genepool;
    genepool' ← genepool' ∪ {survivor_genes};
    while |genepool'| < P do
        Select two parent genes p1 and p2 from genepool';
        child ← crossover(p1, p2);
        if Random(0,1) < R_mut then
            mutate(child);
        genepool' ← genepool' ∪ {child};
    genepool ← genepool';
```

We select one of the cloudlets in the network as the supervisor to direct and manage the islands. For each cloudlet $p \in \{1,\ldots,K\}$, the supervisor partitions the cloudlets into overloaded and underloaded cloudlets according to p. It then instantiates an island on each cloudlet to manage and evolve a genepool. At the end of each generation, each island asynchronously pushes the fittest gene in the previous generation to the supervisor as an update. This can be seen in Fig. 5 where the nodes i, j, and k represent islands and s represents the supervisor. Island i invokes the push-update method, and the supervisor saves the gene in a results table according to the current partition index and the island's id. If an island is exporting migrant genes, the supervisor will select an island destination and handle the transfer of migrants to the destination. The island destination is selected via roulette selection based on the fitness of the gene from each island to ensure that the island with the least-fit genes is most likely to receive migrant genes. The migrant genes are then cached with the destination island as the key, and when the destination island polls the supervisor for migrants, the supervisor transfers the migrants to the destination. This is shown in Fig. 5 where island j migrates a set of genes to the supervisor, island k polls the supervisor for migrants, and the supervisor transfers the



(a) Cloudlet Response Time Distribution (b) Network Link Weight (c) Task Arrival Rate

Fig. 6. Impact of network conditions on performance of algorithms `Heuristic` and `Distributed`.

migrant genes to k. Eventually all the islands will be terminated after reaching N generations, and the fittest gene is saved by the supervisor. After searching the partitions according to all K cloudlets, the supervisor returns the fittest gene of those selected from each partition. The detailed algorithm is given in Algorithm 2.

Algorithm 2. *CLBP-DGA-Supervisor Algorithm*

Distributed algorithm for cloudlet load balancing problem.
Input: $NET, S, P, R_{mut}, R_{mig}, N$
Output: $f(i,j)$, $i,j \in \{1,2,\ldots,K\}$.

for p $\leftarrow 1$ to $K-1$ **do**
  $V_s \leftarrow \{\, i \mid T_i(\lambda_i) > T_p(\lambda_p) \}$;
  $V_t \leftarrow \{\, j \mid T_j(\lambda_j) \leq T_p(\lambda_p) \}$;
  for id$\leftarrow 1$ to K **do**
    /* Create an island on each cloudlet. */
    Invoke `Procedure` CLBP-DGA-Island(id, V_s, V_t);
Search $results$ table for the fittest gene;
return the fittest gene;

/* Functions invoked by islands */
function PUSH-UPDATE (id, $gene$)
  /* Save gene in $results$ table where p is the current partition */
  $results$[p][id] $\leftarrow$ $gene$;

function PUSH-MIGRANTS ($id, migrants$)
  Select an island id using roulette selection;
  Cache migrants using id as the key;

function POLL-MIGRANTS (id)
  if cache-has-key(id) **then**
    $migrants \leftarrow$ cache-get(id);
    cache-remove-key(id);
    return $migrants$;
  else
    return $\varnothing$;

6 Simulation

In this section we evaluate the performance of the proposed algorithm in simulation environments through experiments. We begin by explaining the simulation settings. We then evaluate the performance of the proposed algorithms in different networks. We finally investigate the impact of important parameters on the performance of the proposed algorithms.

6.1 Simulation Environment

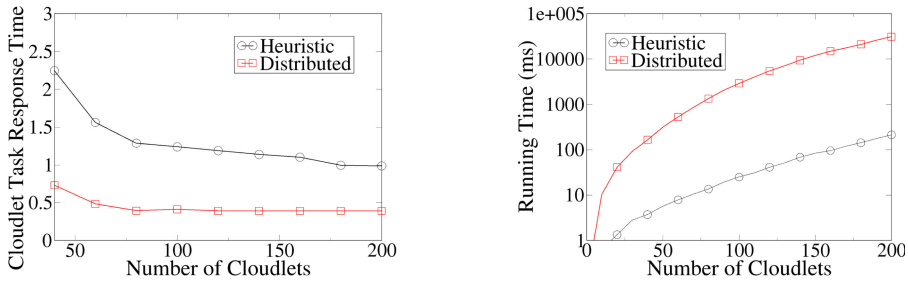
We adopt a similar method of network generation as used in [16] to generate an WMAN. We assume that the positions of cloudlets in an WMAN follows a scale-free distribution, and we generate random scale-free cloudlet network topologies using the Barabasi-Albert Model [2]. Similar to [16], we assume that network delay is proportional to the physical distance between APs. 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 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 n_i 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 $0 < N(15,6) < \mu_i \cdot n_i - 0.25$. Notice that arrival rate λ_i does not exceed $\mu_i \cdot n_i$, otherwise, it would cause the queue time at cloudlet i to be infinite, according to Eq. (1). Unless otherwise stated, the default number of cloudlets in the network $K = 40$.

We refer to the proposed heuristic `Algorithm 1` and the distributed algorithm `Algorithm 2` as algorithms `Heuristic` and `Distributed`, respectively. For algorithm `Heuristic`, let $\theta = 0.1$ and $\epsilon = 0.05$. For algorithm `Distributed`, let the maximum number of generations $N = 1{,}000$, the population size of each island $P = 100$, the rate of gene mutation $R_{mut} = 0.02$, the rate of gene migration $R_{mig} = 0.04$, the percentage population of migrant genes $P_{mig} = 0.02$, and the number of survivors per generation $S = 10$. All data points in simulations are the average of 100 trials with independent randomly generated networks unless otherwise stated.

6.2 Performance Evaluation of Different Algorithms

Fig. 6 studies the impact of network conditions on the performance of the proposed algorithms. In Fig. 6a, we first compare the probability distribution of task response times at cloudlets when no tasks are redirected (isolated cloudlet) and when tasks at cloudlets are redirected, according to the proposed heuristic and distributed algorithms. As can be seen, the heuristic algorithm narrows the range of task response times between 0.2 and 1.0 seconds, with a median task response time of 0.5 seconds. The range is larger than our accuracy bound $\theta = 0.1$, which is due in part to network delay causing a mismatch between outgoing task demand at overloaded cloudlets, and incoming task demand at underloaded cloudlets. The distributed algorithm has the same median task response time of 0.5 seconds, however it has a tighter range with task response times ranging between 0.3 and 0.8.



(a) Number of Cloudlets (b) Running Time

Fig. 7. Impact of network size K on the performance of algorithms `Heuristic` and `Distributed`.

Fig. 6b investigates the maximum task response time of the proposed algorithms when increasing the network delays on network edges. Network delay $c_{i,j}$ between cloudlet i and cloudlet j is generated by randomly sampling the normal distribution $x - 0.05 \leq \mathcal{N}(x, 0.05) \leq x + 0.05$, where x is the given average network delay between any two cloudlets in the network. Both plots have a roughly linear relationship to increased network delay. However, the maximum task response time of the heuristic algorithm out-paces the growth of network delay, i.e., after increasing network delay by 0.35 seconds, the maximum task response time delivered by algorithm `Heuristic` increased by 1 second. In contrast, the maximum task response time in the solution delivered by algorithm `Distributed` only increases by 0.19 seconds. This slow growth is a result of the genetic optimization performed by algorithm `Distributed` to balance the cost of task flow against the cost of locally processing tasks. It can be seen that algorithm `Distributed` is more robust compared to algorithm `Heuristic` in a heavily congested network.

Fig. 6c examines the maximum task response times of both proposed algorithms when increasing the average arrival rate of tasks. Task arrival rate λ_i for cloudlet i is generated by sampling the Normal distribution $0 < N(x, 6)$ where x is the given average task arrival rate. In this experiment, we do not put a limit on the cloudlet task arrival rate, allowing some cloudlets to have an infinite task response time initially. As can be seen, both algorithms deliver the maximum task response times that dramatically increase with the growth of cloudlet task arrival rates. The growth of the maximum task response time is observed to accelerate when task arrival rate exceeds 12.5 as cloudlets begin approaching the limit of their processing capability. When the average task arrival rate exceeds 17.5, both algorithms deliver the maximum task response times of infinity as the cloudlets become overwhelmed by the task arrival rate. It can be seen that the gap between the two proposed algorithms does not increase significantly.

6.3 Impact of Important Parameters on the Performance of Algorithms

In the following we first investigate the impact of network size K on the performance of the proposed algorithms. We then study the impacts of parameters θ and ϵ on the performance of algorithm `Heuristic`. We finally evaluate the impact of parameters such as the numbers of partitions and iterations on algorithm `Distributed`.

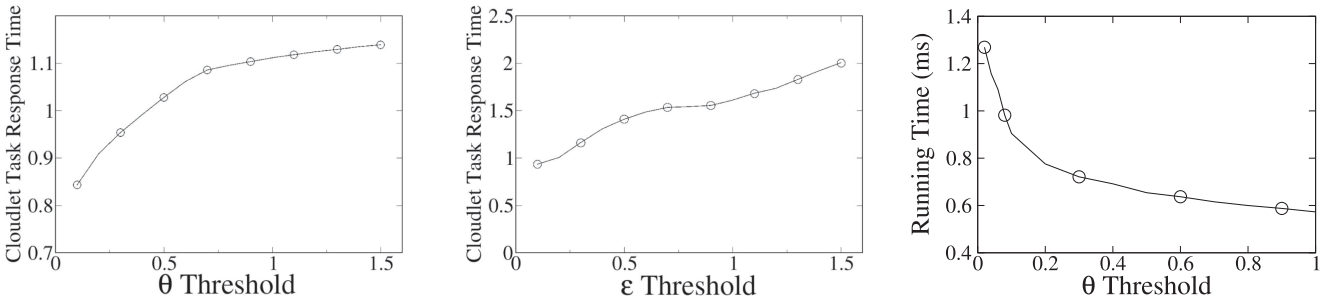
Fig. 7 examines the impact of the number of cloudlets K on the performance of the proposed algorithms. Fig. 7a studies the maximum task response time delivered by different algorithms. As can be seen, the maximum task response time by both algorithms decrease sharply with the growth on the number of cloudlets K. However, algorithm `Distributed` quickly reaches a plateau at around 80 cloudlets. After that, any further addition does not result in further improvement to the maximum task response time. It is observed that after the network has a sufficient number of cloudlets, the queuing time at cloudlets become negligible and the cost of network delay outweighs the benefits of offloading tasks to new cloudlets. Fig. 7b shows the running time of the proposed algorithms, by varying the number of cloudlets. Notice that the plot uses a logarithmic scale for the running time, where the running time of an algorithm was obtained based on a machine with a 3.40 GHz Intel i7 Quadcore CPU and 16 GiB RAM. As algorithm `Distributed` makes use of concurrent processing, we measure its running time by taking the sum of the longest running genetic island (simulated using Java threads) in each partition. We conclude from these comparisons that although the distributed algorithm consistently delivers better solutions than that of the heuristic algorithm especially in congested networks, its running time makes it impractical for many networks.

Fig. 8 plot the two accuracy measures adopted in algorithm `Heuristic`: θ and ϵ, where θ determines how close the maximum task response time of underloaded cloudlets (after network flow) is to the chosen $\overline{D}$ task response time before the solution is acceptable, while ϵ controls the accuracy of the cloudlet task demand ϕ_i for each cloudlet i.

Fig. 8a illustrates the maximum task response time delivered by algorithm `Heuristic`, by varying the value of the accuracy threshold θ. As θ increases, the termination condition of the algorithm becomes more lax, and the number of incremental refinements of $\overline{D}$ is reduced. This results in the increase of the maximum task response time.

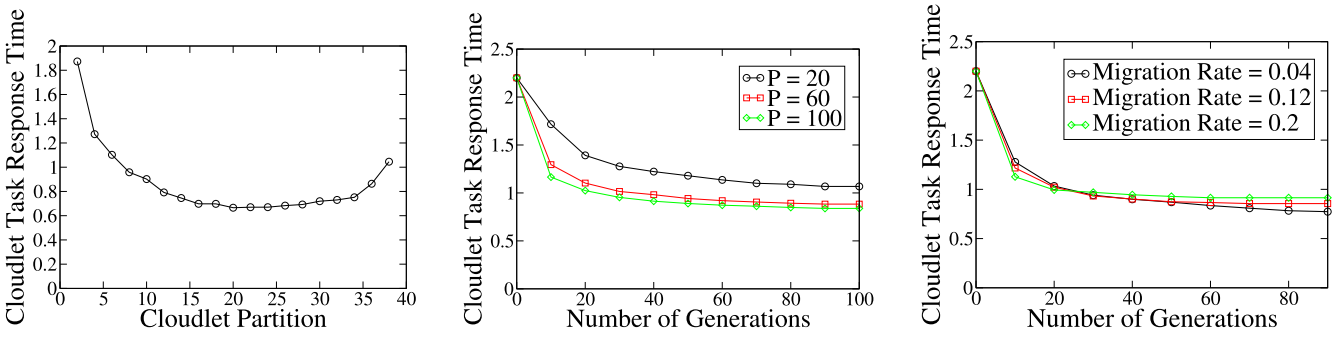
Fig. 8b studies the maximum task response time delivered by algorithm `Heuristic` by varying the accuracy bound ϵ. As ϵ increases, the error margin of task demand at each cloudlet also increases, resulting in the algorithm redirecting the correct amount of flow for a given balanced task response time $\overline{D}$.

Fig. 8c shows the running time of the algorithm increases with the growth of θ. 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 tradeoff



(a) θ threshold impact on task response time (b) ϵ threshold impact on task response time (c) θ threshold impact on running time

Fig. 8. The impact of important parameters on the performance of algorithm `Heuristic`.



(a) Population vs Task Response Time (b) Population vs Task Response Time (c) Migration vs Task Response Time

Fig. 9. The impact of important parameters on the performance of algorithm `Distributed`.

between the running time and the accuracy of the maximum task response time delivered by the algorithm.

Fig. 9 investigates the impact of cloudlet partitions, the effects of population size and migration rate on the performance of algorithm `Distributed`. Fig. 9a shows the maximum task response time given by the fittest gene generated from each cloudlet partition in algorithm `Distributed`. The cloudlet partitions are numbered according to the size of overloaded cloudlets V_s in the partition. When the cloudlet partition has a low number of overloaded cloudlets, task flow is limited and optimal load balancing cannot be fully achieved resulting in a high maximum task response time. Cloudlet partitions with more overloaded cloudlets produce results with a lower maximum task response time, until we reach the optimal partition at 20 overloaded cloudlets. After this point, an increase in overloaded cloudlets within a partition results in a higher maximum task response time, as there are fewer underloaded cloudlets to offload tasks to. Between cloudlet partitions 15 and 30, the variation in the maximum task response time is minor, and a near optimal maximum task response time can be found by optimizing on any cloudlet partition found in this range.

Fig. 9b examines the maximum task response time for three different population sizes per island. We can see that a larger population size leads to a faster convergence as well as a lower task response time. However, there appears to be diminished returns as we increase the population. The improvement gained between a population size of 20 and 60 is much greater than between a population size of 60 and 100.

Fig. 9c examines the maximum task response time for different migration rates. We can see that a lower migration rate leads to a lower task response time, but a slower convergence. The lower migration rate allows each genetic island to develop new solutions independently, leading to larger diversity of genes in the global population. The higher migration rate converges significantly faster but ultimately delivers a higher task response time, compared to lower migration rates.

To conclude, the solution delivered by algorithm `Distributed` can reduce the maximum task response time significantly, while its running time is much longer in comparison with that by algorithm `Heuristic`. Although the results of algorithm `Heuristic` are promising, a key issue is that algorithm `Heuristic` cannot accurately estimate the network delay for an underloaded cloudlet making it difficult for underloaded cloudlets to meet the target balanced task response time $\overline{D}$. This is particularly the case when network delay between cloudlets is highly variable due to congestion. Because of this, algorithm `Heuristic` may terminate prematurely before an accurate solution is reached. On the other hand, while algorithm `Distributed` consistently delivers a more accurate solution, its running time is several orders of magnitude larger than that of algorithm `Heuristic`, which can be a significant issue for large-scale networks.

Clearly, algorithms `Distributed` and `Heuristic` have complimentary qualities, and there are some potential approaches to combining their strengths. An example of this is to encode the solution generated by algorithm `Heuristic` as a gene, and include it in the initial gene population of algorithm `Distributed`. As a result, the optimization process can be speeded up and converges towards an acceptable solution in significantly fewer generations than if algorithm `Distributed` were to start with a completely randomly generated gene population, significantly reducing the running time of algorithm `Distributed`.

7 Conclusion

Cloudlets are an important technology that provides performance improvements to mobile applications. As wireless Internet availability continues to grow, public available and easy-to-access cloudlets will be vital to the future of mobile computing. In this paper we studied the problem of QoS-aware cloudlet load balancing in an WMAN. We proposed two efficient algorithms for balancing the workload among cloudlets to minimize the maximum task response time. We also evaluate the performance of the proposed algorithms through experimental simulation. The simulation results demonstrated that the proposed algorithms are promising.

Acknowledgments

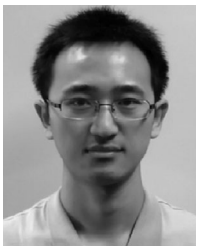
We really appreciate the four anonymous referees and the associate editor for their expertise comments and constructive suggestions, which have helped us improve the quality and presentation of the paper greatly.

References

[1] E. Alba and J. M. Troya, "A survey of parallel distributed genetic algorithms," *Complexity*, vol. 4, no. 4, pp. 31–52, 1999.
[2] R. Albert, H. Jeong, and A.-L. Barabási, "Internet: Diameter of the world-wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.
[3] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, pp. 19–19.
[4] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *Proc. 10th Workshop ACM SIGOPS Eur. Workshop*, 2002, pp. 87–92.
[5] E. Cant-Paz, "A summary of research on parallel genetic algorithms," Illinois Genetic Algorithms Lab., Univ. Illinois at Urbana-Champaign, Champaign, IL, USA, IlliGAL Rep. no. 95007, 1995.
[6] V. Cardellini, et al., "A game-theoretic approach to computation offloading in mobile cloud computing," 2013. [Online]. Available: http://www.optimizationonline.org/DB HTML/2013/08/3981.html
[7] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1818–1831, Jun. 2017.
[8] E. Y. Chen and M. Itoh, "Virtual smartphone over IP," in *Proc. IEEE Int. Symp. World Wireless Mobile Multimedia Netw.*, 2010, pp. 1–6.
[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 301–314.
[10] Cisco systems: Data center: Load balancing data center services. 2008. [Online]. Available: https://learningnetwork.cisco.com/docs/DOC-3438
[11] 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," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2012, pp. 122–127.
[12] E. Cuervo, et al., "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Serv.*, 2010, pp. 49–62.
[13] A. Dolan and J. Aldous, *Networks and Algorithms: An Introductory Approach*, Hoboken, NJ, USA: Wiley, 1993.
[14] L. R. Ford and D. R. Fulkerson, *A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problem*, RAND Corporation, Santa Monica, P-743, Sep. 1955.
[15] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan, "Just-in-time provisioning for cyber foraging," in *Proc. 11th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2013, pp. 153–166.
[16] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 725–737, Oct.–Dec. 2017.
[17] M. Jia, W. Liang, and Z. Xu, "QoS-aware task offloading in distributed cloudlets with virtual network function services," in *Proc. 20th ACM Int. Conf. Model. Anal. Simul. Wireless Mobile Syst.*, Nov., 2017, pp. 109–116.
[18] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet load balancing in wireless metropolitan area networks," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[19] L. Kleinrock, *Queueing Systems: Theory*, Volume I. Hoboken, NJ, USA: Wiley, 1975, pp. 101–103.
[20] 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," in *Proc. IEEE INFOCOM*, 2012, pp. 945–953.
[21] C. B. Pettey, M. R. Leuze, and J. J. Grefenstette, "Parallel genetic algorithm," in *Proc. 2nd Int. Conf. Genetic Algorithms Genetic Algorithms Appl.*, 1987, pp. 155–161.
[22] J. L. R. Filho, P. C. Treleaven, and C. Alippi, "Genetic-algorithm programming environments," *Comput.*, vol. 27, no. 6, pp. 28–43, 1994.
[23] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *Pers. Commun.*, vol. 8, no. 4, pp. 10–17, 2001.
[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, 2009.
[25] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proc. 3rd ACM Workshop Mobile Cloud Comput. Serv.*, 2012, pp. 29–36.
[26] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Leveraging cloudlets for immersive collaborative applications," *IEEE Pervasive Comput.*, vol. 12, no. 4, pp. 30–38, Oct.–Dec. 2013.
[27] G. A. Vignaux and Z. Michalewicz, "A genetic algorithm for the linear transportation problem," *IEEE Trans. Syst. Man Cybern.*, vol. 21, no. 2, pp. 445–452, Mar./Apr. 1991.
[28] J. Wan, D. Zhang, Y. Sun, K. Lin, C. Zou, and H. Cai, "VCMIA: A novel architecture for integrating vehicular cyber-physical systems and mobile cloud computing," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 153–160, 2014.
[29] H. Wu, K. Wolter, and A. Grazioli, "Cloudlet-based mobile offloading systems: A performance analysis," in *Proc. 31st Int. Symp. Comput. Perform. Model. Meas. Eval. Student Poster Abstracts*, Sep. 24–26, Vienna, Austria, pp. 3–4, 2013.
[30] Q. Xia, W. Liang, Z. Xu, and B. Zhou, "Online algorithms for location-aware task offloading in two-tiered mobile cloud environments," in *Proc. 7th IEEE/ACM Int. Conf. Utility Cloud Comput.*, 2014, pp. 109–116.
[31] Q. Xia, W. Liang, and W. Xu, "Throughput maximization for online request admissions in mobile cloudlets," in *Proc. 38th Annu. IEEE Conf. Local Comput. Netw.*, 2013, pp. 589–596.
[32] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Capacitated cloudlet placements in wireless metropolitan area networks," in *Proc. IEEE 40th Conf. Local Comput. Netw.*, 2015, pp. 570–578.
[33] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2866–2880, Oct. 2016.
[34] W. Tian, Y. Zhao, Y. Zhong, M. Xu, and C. Jing, "A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters," in *Proc. IEEE Int. Conf. Cloud Comput. Intell. Syst.*, 2011, pp. 311–315.

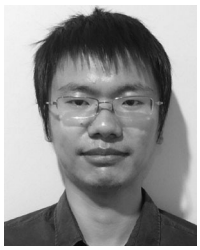
Mike Jia received the BSc degree in mathematics and computer science from Imperial College London in United Kingdom, in 2013, and the honours degree with the first class honours in computer science from the Australian National University, in 2015. He is currently working toward the PhD degree in computer science at the Australian National University. His research interests include mobile cloud computing and software defined networks.

Weifa Liang (M'99-SM'01) received the BSc degree from Wuhan University, China, in 1984, the ME degree from the University of Science and Technology of China, in 1989, and the PhD degree from the Australian National University, in 1998, all in computer science. He is currently a full professor in the Research School of Computer Science, Australian National University. His research interests include design and analysis of energy-efficient routing protocols for wireless ad hoc and sensor networks, cloud computing, software-defined networking, online social networks, design and analysis of parallel and distributed algorithms, approximation algorithms, combinatorial optimization, and graph theory. He is a senior member of the IEEE.



Zichuan Xu (M'17) received the BSc and ME degrees from Dalian University of Technology, in China, in 2008 and 2011, and the PhD degree from the Australian National University, in 2016, all in computer science. He was a research associate in the Department of Electronic and Electrical Engineering, University College London, United Kingdom. He currently is an associate professor in the School of Software, Dalian University of Technology, China. His research interests include cloud computing, software-defined networking, network function virtualization, wireless sensor networks, routing protocol design for wireless networks, algorithmic game theory, and optimization problems. He is a member of the IEEE.

Meitian Huang received the BSc degree with the first class honours in computer science from the Australian National University, in 2015. He is currently working toward the PhD degree in computer science at the Australian National University. His research interests include software-defined networking and cloud computing.



Yu Ma received the BSc degree with the first class honours in computer science from the Australian National University, in 2015. He is currently working toward the PhD degree in computer science at the Australian National University. His research interests include software defined networking, Internet of Things (IoT), and social networking.