An Online Algorithm for Task Offloading in Heterogeneous Mobile Clouds

BOWEN ZHOU, AMIR VAHID DASTJERDI, RODRIGO N. CALHEIROS, and RAJKUMAR BUYYA, The University of Melbourne, Australia

Mobile cloud computing is emerging as a promising approach to enrich user experiences at the mobile device end. Computation offloading in a heterogeneous mobile cloud environment has recently drawn increasing attention in research. The computation offloading decision making and tasks scheduling among heterogeneous shared resources in mobile clouds are becoming challenging problems in terms of providing global optimal task response time and energy efficiency. In this article, we address these two problems together in a heterogeneous mobile cloud environment as an optimization problem. Different from conventional distributed computing system scheduling problems, our joint offloading and scheduling optimization problem considers unique contexts of mobile clouds such as wireless network connections and mobile device mobility, which makes the problem more complex. We propose a context-aware mixed integer programming model to provide off-line optimal solutions for making the offloading decisions and scheduling the offloaded tasks among the shared computing resources in heterogeneous mobile clouds. The objective is to minimize the global task completion time (i.e., makespan). To solve the problem in real time, we further propose a deterministic online algorithm—the Online Code Offloading and Scheduling (OCOS) algorithm—based on the *rent/buy* problem and prove the algorithm is 2-competitive. Performance evaluation results show that the OCOS algorithm can generate schedules that have around two times shorter makespan than conventional independent task scheduling algorithms. Also, it can save around 30% more on makespan of task execution schedules than conventional offloading strategies, and scales well as the number of users grows.

CCS Concepts: • **Computer systems organization → Cloud computing**; • **Human-centered computing → Mobile computing**; • **Theory of computation → Scheduling algorithms**; *Integer programming*;

Additional Key Words and Phrases: Mobile cloud computing, code offloading, mixed integer programming, online scheduling

ACM Reference format:
Bowen Zhou, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya. 2018. An Online Algorithm for Task Offloading in Heterogeneous Mobile Clouds. *ACM Trans. Internet Technol.* 18, 2, Article 23 (January 2018), 25 pages.
https://doi.org/10.1145/3122981

1 INTRODUCTION

Mobile computing technologies developed enormously in recent years and mobile devices (e.g., smartphones, tablets, and wearable devices) have been involved as part of people's daily activities

Authors' addresses: B. Zhou and R. Buyya, The Cloud Computing and Distributed Systems Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia; emails: bowenzhou1990@gmail.com, rbuyya@unimelb.edu.au; A. V. Dastjerdi, PricewaterhouseCoopers; email: vahid.av@gmail.com; R. N. Calheiros, School of Computing, Engineering and Mathematics, Western Sydney University; email: r.calheiros@westernsydney.edu.au.

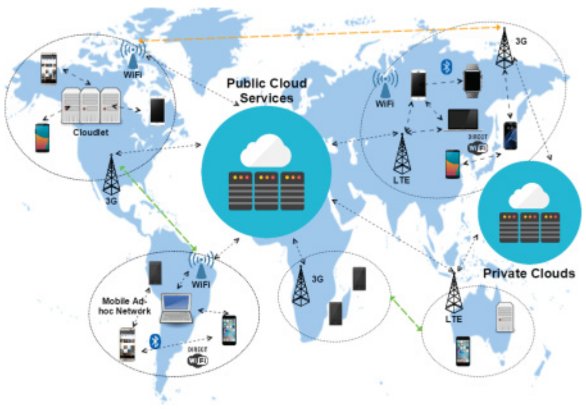


Fig. 1. A typical heterogeneous mobile cloud environment.

(Dinh et al. 2013). Mobile applications such as cognitive applications (e.g., optical character recognition, face detection) and augmented reality are gaining popularity among mobile device users. These applications typically require intensive computation as well as considerable energy consumption. However, compared to desktop computers, mobile devices still provide a relatively inferior performance in terms of processing capacity, memory, storage capacity, and battery life to support long-time services. As a result, the gap between resource-constrained mobile devices and computing intensive applications has posed a significant challenge.

Cloud computing is a promising solution to this challenge. It enables users to run applications and services on various cloud service models (i.e., IaaS, PaaS, SaaS) on demand. Recently, mobile cloud computing is introduced by bringing the benefits of cloud computing to augment resource-constrained mobile devices. Code offloading provides an approach to migrate computing intensive tasks to other computing resources in order to enhance the processing capacity and reduce the energy consumption of the mobile device.

Previous works (Kumar and Lu 2010; Chen et al. 2015) have seen limits of only offloading to the cloud since wireless networks can cause performance bottleneck. To overcome this challenge, it is of interest to provide mobile cloud users with alternative computing paradigms. Due to the improvement in short range wireless networks and mobile devices, mobile cloud computing has evolved beyond the onefold resource into a heterogeneous network of computing resources.

As illustrated in Figure 1, the heterogeneous mobile cloud (HMC) environment contains different types of computing resources such as public clouds, private clouds, cloudlets (Satyanarayanan et al. 2009), and mobile ad-hoc networks (MANET) in proximity that can be utilized to offload computing intensive mobile tasks. Computing resources are considered machines that are interconnected by multiple wireless technologies, namely WiFi, cellular networks, and Bluetooth to form a shared pool of resources for mobile devices. Each mobile device within the shared environment is able to offload tasks to other available machines, and vice versa. For the individual mobile device, it is important to make the offloading decision based on its context such as network conditions, load of other machines, and mobile device's own constraints (e.g., mobility and battery). Moreover, to achieve a global optimal task completion time for tasks from all the mobile devices, it is necessary to devise a task scheduling solution that schedules offloaded tasks in real time.

In this article, we jointly investigate the mobile code offloading problem and task scheduling problem for offloaded tasks as a mobile code offloading and scheduling problem (MCOSP). Unlike the existing code offloading approaches, MCOSP not only makes decisions of whether, when, and how to offload tasks for each mobile device, but also aims to schedule offloaded tasks

Table 1. List of Acronyms

Acronyms	Description
HMC	Heterogeneous Mobile Clouds
MCOSP	Mobile Code Offloading and Scheduling Problem
OCOS	Online Code Offloading and Scheduling algorithm
LCSD	Low Computation Small Data Size workload
HCSD	High Computation Small Data Size workload
HCLD	High Computation Large Data Size workload
VM	Virtual Machine
OCR	Optical Character Recognition
OLB	Opportunistic Load Balancing algorithm
LO	Mobile Local Only
CO	CO Offloading to Cloud Only

among shared machines in heterogeneous mobile clouds. MCOSP takes into consideration the constraints of the shared computing resources (i.e., load of the resources, network conditions, and limits of battery lifetime) when scheduling, such that the overall task completion time is minimized. Our work considers a common task scheduling model where the mobile device within the heterogeneous mobile cloud environment generates independent tasks at arbitrary time, and one machine may only process one task at a time. The heterogeneity in both tasks and machines makes MCOSP an NP-hard problem (Garey and Johnson 1990). We study the optimization of MCOSP in both off-line and online cases. Table 1 lists the acronyms used in this article. The **key contributions** of this work are the following:

—First, we propose the models of computation, energy consumption, monetary cost, and time-to-failure for mobile devices that represent the heterogeneity of the proposed heterogeneous mobile cloud environment.

—Second, we formulate MCOSP as a mixed integer non-linear programming problem based on the models of the proposed heterogeneous mobile clouds and provide an analysis on its hardness. Then, we transform it into a mixed integer linear programming formulation using linearization and solve the problem using the branch-and-bound (BB) algorithm.

—Finally, to provide a practical solution for the problem, an online real-time scheduling algorithm for MCOSP based on a generalization of rent/buy problem framework is proposed to obtain competitive schedules for large workloads. Experimental results demonstrate that the proposed online algorithm OCOS generates 2-competitive makespan on average comparing to off-line optimal solution in terms of scheduling performance, and scales well in terms of offloading gains when the number of mobile users in the system increases.

The remaining article is organized as follows. The related work in mobile code offloading is reviewed in Section 2. In Section 3, the system models for task scheduling are introduced, followed by a description on MCOSP and the MIP-based formulation in Section 4. Then, we proposed the online solutions and its competitive analysis in Section 5. The experiment settings are explained and the performance evaluation results are discussed in Section 6. Finally, we conclude our work and present the future work in Section 7.

2 RELATED WORK

Mobile code offloading has been widely studied in the literature. Most previous work has focused on the application partitioning and task offloading from a single mobile device user point of view.

However, the lack of consideration for the load of other machines can significantly affect the task completion time. Additionally, the scalability of these offloading strategies would suffer when the number of mobile device users grows.

Ma et al. (2013) presented a review of computation offloading works in mobile cloud computing and identified key issues. Wen et al. (2012) studied the optimization of configuring clock frequency of mobile processors and data transmission rate for offloading to minimize the energy consumption subject to deadlines. *ThinkAir* (Kosta et al. 2012) is an Android platform for code offloading that aims to lower the execution time and energy consumption together. Huang et al. (2012) provided a Lyapunov optimization-based solution for saving energy while satisfying application deadlines, device mobility, and network conditions. Kovachev et al. (2012) designed mobile cloud offloading middleware MACS for Android devices, which aims to minimize the task execution cost of CPU, memory, and data transmission. Chen et al. (2013) presented an offloading algorithm based on a semi-Markovian decision process to optimize execution time and energy consumption. Lai et al. (2013) investigated the communication latency issue as a mobile cloud offloading bottleneck for time critical applications. The *SpotCloud* was proposed to shorten the latency by offloading tasks to local home PCs. Zhang et al. (2013) devised a threshold policy to dynamically set CPU frequency and radio transmission rates for energy optimization under stochastic wireless channels based on energy consumption rates and wireless channel states. Barbera et al. (2013) studied the feasibility of both mobile computation offloading and mobile data backups using cloud resources. The influence of wireless network availability and quality on the offloading performance is evaluated. Deng et al. (2015) investigated the effect of user mobility and fault tolerance on code offloading decisions. They proposed a genetic algorithm to solve offloading problems with consideration of random walk models and failure recovery time. Krishna et al. (2016) proposed a model for task offloading using a learning automata based decision-making algorithm. It compares the execution time and energy consumption of a task on a local device and the cloud to make offloading decisions. Noticeably, all the above-mentioned work only focused on the localized offloading benefits between a single user and cloud servers, assuming cloud resources are unlimited. However, the potential cost incurred by renting the resources is of concern in reality.

Several works study the offloading problem under the concept of multiple users sharing the limited cloud resources. Chen (2015) adopted a game theory approach to address the challenge of devising efficient offloading coordination among mobile device users considering the wireless access efficiency. Rahimi et al. (2013) proposed a multi-tier offloading framework that uses a greedy heuristic to make the offloading decisions considering device mobility. Barbarossa et al. (2013) introduced a joint optimization problem of computation resource and transmission power allocation among multiple users under application delay constraints. Liu et al. (2016) formulated the energy consumption optimization of a multi-device task offloading problem as a binary non-linear integer programming problem with acceptable time delay and communication quality. The proposed problem is converted into a continuous convex optimization and solved by an iterative decoupling algorithm. Wang et al. (2016) took a further step to study the QoS-aware mobile data traffic offloading in vehicular ad-hoc networks (VANET). They proposed a combinatorial optimization problem to minimize the mobile data traffic and maximize resource utilization while meeting the QoS guarantee. Nan et al. (2016) studied the tradeoff issue between average response time and average cost by scheduling tasks among resources in fog computing systems. The solution can achieve near optimal results by using a Lyapunov optimization-based online algorithm. However, these works have not studied the benefit of utilizing the heterogeneity of computing resources under multiple types of tasks and different context of machines and network conditions.

Table 2. Notation

Symbol	Description	Symbol	Description
M	the set of heterogeneous machines such as cloud VMs, cloudlets, and MANET	$T_{i,m}^{exec}$	the processing time of task i on machine m
S	the set of independent tasks to be scheduled	$T_{i,ch}^{trans}$	the data transferring time of task i using wireless channel ch
m	machine m in HMC environment	$E_{i,m}^{exec}$	the energy consumption for executing task i on machine m
i	mobile task i	$E_{i,m}^{trans}$	the energy consumption for transferring task i to machine m using wireless channel ch
f_i	the data size of task i	$x_{i,m}$	the binary variable indicating task i is assigned to machine m in HMC
ϖ_i	the computation workload of task i	w_i, c_i, b_i	the binary variables indicating the wireless medium used to offload task i
t_i^{arrive}	the arrival time of task i	$o_{i,j,m}$	the binary variable indicating whether task i is scheduled before task j on node m
μ	the processing speed of the machines	$t_{i,m}^{start}$	the real variable representing the start time of task i on machine m
l	network latency of the HMC network	$t_{i,m}^{finish}$	the real variable representing the finishing time of task i on machine m
r	the charge rate per time unit for cloud instance in M	$\overline{T}$	a constant greater than the worst-case makespan
θ	the proportion of battery energy provided by mobile device in M	R_i	renting cost of task i
v_m	the energy consumption rate of CPU on machine m	$B_{i,m}$	buying cost of dispatching task i to machine m
PR^{active}	energy consumption per time unit when the machine is in the active mode	C_m^{budget}	the monetary budget of machine m
PR^{idle}	energy consumption per time unit when the machine is in the idle mode	C_i^{trans}	the monetary cost of transferring data of task i via mobile data
E_m^{total}	the total battery energy of mobile device m	C_i^{VM}	the monetary cost of executing task i on cloud VMs
t_m^{join}	the time when machine m joins the HMC network		
t_m^{leave}	the time when machine m leaves the HMC network		

In this article, we study the multi-user mobile offloading task scheduling, under the constraints of HMC resources (i.e., limited computing capacity, battery lifetime, network conditions, user mobility, and monetary cost), to minimize the makespan.

3 HMC SYSTEM MODEL

3.1 Motivation Example

A group of oversea tourists are travelling at a local museum. Since they do not speak the local language, an optical character recognition (OCR) application is installed on smartphones and tablets to help them read museum exhibit descriptions. The OCR application takes photos of the descriptions, processes to extract the text from the photos, and sends extracted texts over the Internet to its back-end services in the cloud for translation. However, some mobile devices do not have enough computing capacity to perform the text extraction, and some other mobile devices do not have an Internet connection to get the translation. In order to enable every group member to use the OCR application on his or her device, mobile devices, including smartphones, tablets, and laptops, and public cloud virtual machines (VMs) running back-end services, form a network of shared computing resources (i.e., HMC). The individual mobile device that lacks resources is able to utilize the shared computing and network resources by offloading tasks to other devices or the cloud.

The challenge in the scenario is deciding whether, where, and how to offload a mobile task, as well as scheduling the offloaded tasks with load balance among the shared resources to achieve optimal task completion time.

3.2 Heterogeneous Mobile Clouds

The system models of the environment are formally presented in this section. The definition of the proposed heterogeneous mobile cloud is given as follows.

Definition 3.1 (Heterogeneous mobile cloud). The heterogeneous mobile cloud is a computing paradigm that consists of multiple types of machines including mobile devices, cloudlets, and public cloud services. The machines are interconnected via wireless communications technologies such as WiFi, WiFi-direct, Bluetooth, and mobile cellular networks. Each machine can execute its tasks either on its own or offload tasks to other machines via available wireless networks. In order to utilize the shared resources, each machine needs to provide its computation time as well as a portion of battery.

Suppose a set of heterogeneous machines $m \in M$ are connected via different wireless technologies. M consists of three types of computing resources: virtual machines on a single public cloud service, nearby cloudlets, and mobile devices in proximity.

The heterogeneity of each machine m is modeled in terms of computing and network capability. The model is given as $m \triangleq \langle \mu, l, r, \theta, PR^{active}, PR^{idle}, E^{total}, t^{join}, t^{leave}, C^{budget} \rangle, \forall m \in M$, where μ is the processing speed of machine m, l is the network latency between the client device and m, r is the charge rate per time unit for the cloud instance, and θ denotes the proportion of energy machine m will provide to execute offloaded tasks. M represents the set of machines that form the shared resource pool. Note that θ is an exclusive property for mobile devices. The value of this metric for other types of computing resources is set to 0. PR^{active} represents the energy consumption rate per time unit of the mobile device in active mode, and PR^{idle} represents the energy consumption per time unit in idle mode. E^{total} is the total battery energy of machine m. The mobility of mobile devices are represented by the joining time and leaving time, respectively, as t^{join} and t^{leave}. C^{budget} denotes the monetary budget of machine m for using the mobile cloud service and cellular network data usage.

Mobile devices are not always available due to the mobility of users, and the available time of the machines in HMC has a vital impact when scheduling mobile tasks. To enable a tractable analysis, we first assume that network settings (i.e., signal strength, network speed, and latency) remain unchanged throughout the time when the mobile device is within the HMC environment. This is reasonable considering a group-based environment such as offices or group travelling.

In each proposed HMC network (e.g., one of the circled networks in Figure 1), a task scheduler is deployed to receive task offloading requests from mobile devices within the network and schedule the tasks based on the proposed scheduling algorithms. To reduce scheduling overhead, the scheduler is placed on the nearby cloudlet to reduce network latency. In case the cloudlet is not available or in failure, the scheduler is placed on the VM on the public cloud as a backup. The proposed HMC environment is enabled by our previously proposed mobile cloud offloading framework mCloud (Zhou et al. 2015).

3.3 Workload Model

Only tasks with high computation and small data input size are beneficial from mobile cloud offloading (e.g., OCR and GPS-based applications), and we call it offloading tasks (Zhou et al. 2015). Consider an application is composed of offloading tasks and other tasks. The proposed algorithm only considers offloading tasks for offloading scheduling. Note that each application can only submit the offloading tasks to the scheduler (this is enabled by the framework in Zhou et al. (2015) with Java Reflection). Hence, we assume that the tasks evaluated at the scheduler are independent and non-preemptive of each other. We define a 3-tuple for offloading tasks $i \triangleq \langle f_i, \varpi_i, t_i^{arrive} \rangle$, where f_i is the data size of the task i to be offloaded, ϖ_i denotes CPU cycles needed to complete task i, and t_i^{arrive} represents the arrival time of task i at the scheduler. The mobile device user can adopt the methods proposed by Cuervo et al. (2010) and Chun et al. [2011] to obtain the value of f and ϖ. Tasks are generated at an arbitrary time. Due to the concern of communication overhead, once offloading tasks are generated, only the information of the 3-tuple model is sent to the central scheduler at the stage of scheduling.

3.4 Computation and Communication Models

The execution time and energy consumption required to execute a task are major factors considered in the process of making offloading decisions. CPU and the wireless transmitter are two major energy consumption sources for mobile devices in mobile cloud computing. Typically, the power consumption of these hardware components depends on their operating state (i.e., utilization, data rate, etc.). Without loss of generality, we assume that the CPU operates at full utilization when executing mobile tasks and the data rate of the wireless channel is constant and stable. In our model, the scheduler takes into account two metrics: the computation time and the energy consumption of processors and transmitters.

Suppose that an offloading task i is waiting to be scheduled to M. The computation execution time of task i on machine m is composed of CPU computation time and memory I/O access time. Since memory access is tightly coupled with the type of application and instructions executed and can vary on different hardware architectures, the model takes on a high-level abstraction of memory access based on a cache miss model proposed by Fan et al. (2005).

$$T_{i,m}^{exec} = \frac{\varpi_i}{\mu_m} + t_{access} * N_{miss},$$

where t_{access} and N_{miss} represent memory access time and the number of cache miss, respectively. These two parameters can either be obtained from hardware specifications or from existing performance counters on many modern processors (Fan et al. 2005). If machine m is not the one that generates task i, a task offloading occurs with extra cost in terms of time and energy required for data transmitting. The data transmission time is given as

$$T_{i,m}^{trans} = \frac{f_i}{BW_{ch}},$$

where BW_{ch} denotes the network speed of the wireless channel used to offload data. The computation execution time for an offloaded task is the sum of $T_{i,m}^{exec}$ and $T_{i,m}^{trans}$. Note that the time and energy cost for sending computation results back to the mobile device user are neglected, due to the fact that the data size of results for the considered applications (i.e., cognitive applications) is much smaller compared to the offloaded data (i.e., input data and application code).

Similar to computation models, the energy consumption model consists of the execution of offloaded tasks, and energy consumed by data transmission if offloading occurs. The energy consumption model for task i executing on machine m is given as

$$E_{i,m}^{exec} = \nu_m \frac{\varpi_i}{\mu_m},$$

where ν_m represents the energy consumption rate of m executing tasks based on the processor's frequency (Wen et al. 2012). $E_{i,src}^{exec}$ is the energy consumption for machine m if the scheduler decides to execute task i locally. For the computation offloading, the amount of energy incurred by data transfer, $E_{i,m}^{trans}$, depends on the size of the data and the wireless medium that carries the transmission.

$$E_{i,m}^{trans} = \rho_{ch} \frac{f_i}{BW_{ch}},$$

where ρ_{ch} denotes the energy consumption rate of the wireless medium applied and BW_{ch} is the network speed. Accordingly, the energy consumption of task i executing locally is $E_{i,src}^{trans}$.

3.5 Monetary Cost

We assume that each individual mobile device has a monetary cost budget set by the mobile user for using the proposed mobile cloud services. The monetary cost is generated by transferring data via cellular networks and using public cloud services. The scheduler needs to make offloading decisions without violating budget constraints. In our system model, the data transferring cost of offloading task i is calculated as $C_i^{trans} = \kappa(f_i + f_{res})$, where κ is the cost per megabyte of cellular networks, and f_i, f_{res} denotes the data size of task i and computation results, respectively.

The cost of using public cloud services is based on the type of service and the usage. The spending on running task i on cloud VM m is calculated as $C_i^{VM} = r_{vm}\frac{\varpi_i}{\mu_{vm}}$, where r_m is the cost per time unit of cloud instance m.

3.6 Failure Model

Mobile devices are unreliable due to their mobility and unstable wireless network links. The disconnections of mobile devices from the mobile cloud environment can cause failures in execution of offloaded tasks. Therefore, considering the availability of machines is important for making offloading decisions. Note that the study of mobility patterns of mobile devices is not the focus of this article. Instead, the system model adopts the time-to-failure metric to represent the availability of the machines. Time-to-failure represents the time between the machine joining (i.e., t_m^{join}) and being disconnected from the shared resource pool (i.e., t_m^{leave}).

To obtain the time-to-failure of mobile devices, Weibull distribution is adopted since it can well-represent the case where failure rate is constant as well as the case where failure rate increases or decreases as time goes by (see Lee et al. (2010)), and is shown to be useful for modeling lifetime data in engineering sciences (Lawless 2011). The probability density function (PDF) of Weibull distribution is given by:

$$f(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} e^{-(\frac{t}{\eta})^{\beta}},$$

where β is the shape parameter and η is the slope parameter of the distribution. Time-to-failure of each machine can be randomly obtained from the inverse function of Weibull distribution. t_m^{leave} of machine m can be calculated by padding the time-to-failure after t_m^{join} of the machine.

4 MOBILE OFFLOADING SCHEDULING PROBLEM FORMULATION

In our system model, mobile tasks, machines, and wireless networks are all considered heterogeneous. To guarantee a consistent user experience for all mobile cloud service users, we target on the balanced utilization of the shared computing resources with the shortest possible tasks response time for all mobile device users, i.e., the overall task completion time is minimized considering the heterogeneity and constraints of the HMC environment. In such case, task completion time for an individual user may not be optimal, but for all the service users, the task completion time of all mobile tasks submitted by mobile devices is minimized to ensure the balanced and fair use of the shared computing resources.

4.1 Mixed Integer Programming Based Off-line Optimal Formulation

The MCOSP aims to decide on which machine to execute each generated mobile task, as well as which wireless medium to use if an offloading is required. Furthermore, it needs to devise an optimal schedule for the mobile task execution with minimum makespan, subject to constraints from the proposed HMC environment.

Based on proposed system models, we presented a mixed integer programming (MIP) based problem formulation for MCOSP. In the MIP formulation, five binary variables and two continuous variables are defined to devise the optimal solution. Note that the term *machine* is used to represent any types of computing resources in the proposed HMC environment.

Let $t_{i,m}^{start}$ and $t_{i,m}^{finish}$ be the starting time and finishing time of task i executing on machine m, respectively. The binary variable $x_{i,m}$ represents whether task i is scheduled on machine m.

$$x_{i,m} = \begin{cases} 1, & \text{if task } i \in S \text{ is assigned to machine } m \in M \\ 0, & \text{otherwise} \end{cases}$$

Then, the variables for the wireless medium used for data offloading are defined as binary variable w_i, c_i, and b_i. The variables represent whether the data is offloaded via WiFi, cellular network, or Bluetooth, respectively. The variable that represents the utilized wireless medium for task i is set to 1, otherwise it is set to 0. As an example, the definition of w_i is given below.

$$w_i = \begin{cases} 1, & \text{if task } i \text{ is offloaded via WiFi} \\ 0, & \text{otherwise} \end{cases}$$

In order to simplify the calculation of task completion time, binary variable $o_{i,j,k}$ is defined to represent the order of tasks scheduled on each machine.

$$o_{i,j,m} = \begin{cases} 1, & \text{if task } i \text{ is scheduled before task } j \text{ on machine } m \\ 0, & \text{otherwise} \end{cases}$$

The MIP formulation is given as follows.

$$\text{Minimize: } \max\{t_{i,m}^{finish}\}, \forall i \in S, \forall m \in M \tag{1}$$

$$\text{Subject to: } \forall i \in S, \sum_{m \in M} x_{i,m} = 1 \tag{2}$$

$$w_i + c_i + b_i + x_{i,src} = 1 \tag{3}$$

$$\overline{T}(o_{i,j,m} - 1) \leqslant t_{j,m}^{finish} - t_{i,m}^{start} \leqslant \overline{T} * o_{i,j,m} \tag{4}$$

$$x_{i,m} + x_{j,m} + o_{i,j,m} + o_{j,i,m} \leqslant 3 \tag{5}$$

$$o_{i,j,m} + o_{j,i,m} \leqslant 1 \tag{6}$$

$$t_{i,m}^{start} \geqslant t_i^{arrive} \tag{7}$$

$$t_{i,m}^{start} \geqslant t_m^{join} + \overline{T}(x_{i,m} - 1) \tag{8}$$

$$t_{i,m}^{start} \geqslant t_{j,m}^{finish} - \overline{T}(1 - o_{j,i,m}) - \overline{T}(2 - x_{i,m} - x_{j,m}) \tag{9}$$

$$t_{i,m}^{finish} \leqslant t_m^{leave} \tag{10}$$

$$t_{i,m}^{finish} = t_{i,m}^{start} + x_{i,m} * (T_{i,m}^{exec} + T_{i,m}^{trans}) \tag{11}$$

$$t_{i,m}^{trans} = w_i \cdot T_{i,WiFi}^{trans} \cdot I_{wifi}^{m} + c_i \cdot T_{i,3g}^{trans} \cdot I_{3g}^{m} + b_i \cdot T_{i,bl}^{trans} \cdot I_{bl}^{m}) \tag{12}$$

$$T_{i,src}^{exec} \geqslant x_{i,m} * (T_{i,m}^{exec} + T_{i,m}^{trans}) \tag{13}$$

$$\sum_{i \in S} x_{i,m} \cdot E_{i,m}^{exec} \leqslant E_m^{total} \cdot \theta_m \tag{14}$$

$$v_{src} \cdot E_{i,src}^{exec} > E_i^{trans} \tag{15}$$

$$\sum_{i \in S} c_i \cdot C_i^{trans} + \sum_{m \in M} x_{i,m} \cdot C_i^{VM} \leqslant C_m^{budget} \tag{16}$$

$$\forall i, j \in S, \forall m \in M \tag{17}$$

The objective function of Equation (1) minimizes the maximum of the task completion time from all the machines (i.e., makespan of the scheduled tasks). Constraints (2)–(11) ensure a valid schedule of the tasks. Constraint (2) ensures that each task needs to be assigned to one and only one machine (either local execution or offloading to another machine). Constraint (3) ensures that for each task being scheduled, either it is executed locally ($x_{i,src}$) or offloaded via one and only one of the wireless networks (w_i, c_i, b_i). Different tasks are scheduled with one of the different types of wireless networks and it can be different for each task based on its execution conditions (Equation(12)). $x_{i,src}$ is one of the binary variables $x_{i,m}$.

Constraint (4) assigns values to the order variable $o_{i,j,m}$ for each pair of tasks scheduled on the same machine based on the difference between the finish and start time of each task. $\overline{T}$ is the constant greater than the worst case makespan. Constraints (5) and (6) restrict the tasks scheduled on the same machine from overlapping; that is, each machine can only run one task at a time. Since the tasks are generated randomly at times, Constraint (7) ensures a valid schedule of tasks starting after the task arriving time.

Additionally, the starting and finishing time of a task being scheduled to another mobile device should be within the range of that device's joining and leaving time. This is guaranteed by Constraints (8) and (10). If the task i is not assigned to machine m, $t_{i,m}^{start}$ will be set to a value greater than the worst case makespan by Constraint (8). Constraint (9) indicates that the task is scheduled to an available time slot of the machine. Constraint (10) calculates the finish of task i on machine m. The finish time of task i on machine m is equal to the sum of its start time $t_{i,m}^{start}$, the execution time $T_{i,m}^{exec}$, and the data transferring time $T_{i,m}^{trans}$, if $x_{i,m}$ is equal to 1.

Constraints (13)–(16) describe the unique constraints of code offloading in proposed heterogeneous mobile clouds. Constraint (13) specifies that tasks that are offloaded should have a shorter execution time than that of local execution. $I_{channel}^m$ is a binary indicator that represents the availability to access machine m via each type of wireless interfaces. It is set by the machine when it is initially connected to the network. In the system model, each mobile device is only contributing θ_m proportion of its battery energy for running the offloaded tasks from other mobile device users. This is enforced by Constraint (14) for each mobile device. When a task is scheduled to offload to another machine, the energy consumption of the data communication should be less than that of executing the task locally. The energy consumption constraint is described in Constraint (15). The left part of the inequation represents the energy consumption of executing the task i on a local processor, and the right side is the energy of computing it on machine m. v_{src} is the energy consumption rate of the original machine of task i. Finally, Constraint (16) guarantees that the monetary cost of each machine by using the public cloud services and mobile data does not exceed its limit C_m^{budget}.

We can observe that the formulation is a non-linear model because of Constraints (1), (11), and (13). Generally, the mixed integer non-linear programming model is not solvable with the optimization software. Therefore, the model needs to be linearized for further optimization.

The objective function is a min-max function that can be linearized by minimizing a continuous variable T and adding the following constraint:

$$T \geq t_{i,m}^{finish}, \forall i \in S, \forall m \in M. \tag{18}$$

To linearize Constraints (11) and (13), three new binary variables $\delta_{i,m}^{medium}$ are defined as follows.

$$\delta_{i,m}^{medium} = x_{i,m} \cdot medium_i, \tag{19}$$

where $medium_i$ is w_i, c_i, or b_i. All the quadratic terms in Constraints (11) and (13) can be equivalently replaced by $\delta_{i,m}^{medium}$, and three constraints are added:

$$\delta_{i,m}^{medium} \leqslant x_{i,m} \tag{20}$$

$$\delta_{i,m}^{medium} \leqslant medium_i \tag{21}$$

$$\delta_{i,m}^{medium} \geqslant x_{i,m} + medium_i - 1 \tag{22}$$

Therefore, the mixed integer linear programming formulation is given as:

$$\begin{aligned} \text{Min} \ &: T \\ \text{s.t.} \ &: (2) - (18), (20) - (22). \end{aligned} \tag{23}$$

4.2 Complexity Analysis

MCOSP is based on a task scheduling problem for heterogeneous computing (Topcuoglu et al. 2002). Given a heterogeneous computing environment that has processors with different processing speeds, and a set of tasks modeled by a directed acyclic graph waiting to be scheduled onto the machines, the objective of the heterogeneous task scheduling problem is to minimize the maximum task completion time while the schedule satisfies the task precedence. The task scheduling in heterogeneous systems is proven to be an NP-hard problem (Hong and Prasanna 2007; Ibarra and Kim 1977). The main differences of MCOSP are that it considers a set of independent, non-preemptive tasks. Moreover, our proposed problem considers the unique constraints of the shared computing resources to offload in HMC in terms of computing capacity, battery limits, the availability of mobile devices, and network conditions. Therefore, based on the proposed mixed integer linear programming (MILP) formulation, the MCOSP is an NP-hard problem.

Generally, an MILP problem is solved by the BB method (Alisa Land 1960). As a result, there exists a large search space when being solved by the BB method due to the number of variables, which is related to the size of the task set and machines. Hence, the optimal results can only be devised for a small set of problem instances. The efficiency of the proposed MILP solution is evaluated in the experiments (Section 6). Therefore, an online, lightweight scheduling algorithm for MCOSP is of interest.

5 ONLINE CODE OFFLOADING AND SCHEDULING ALGORITHM

In the proposed heterogeneous mobile cloud environment, the central scheduler has no knowledge of future task arrivals. Hence, the scheduler has to make decisions of task offloading and scheduling in a real-time manner without knowing the entire input sequence (i.e., task arrival times). An online optimization framework is therefore needed to solve the MCOSP in real time. We proposed the OCOS algorithm based on the *rent/buy* problem to tackle this challenge.

5.1 Mobile Code Offloading and Scheduling Problem

Many online problems involve a sub-problem called *rent/buy* problem. One has to decide whether to stay in current state with a certain amount of cost per time unit, or pay some fixed cost to move to another state.

Ski rental (Karlin et al. 1990) is a classic example. Suppose a person is skiing for an unknown number of days. He needs to either buy the equipment or rent from the shop. Renting costs r per day while buying costs B, where $B > r$. The objective is to make the renting or buying decision online in order to minimize the total cost on skiing.

A well-known generalization of this classical *rent/buy* problem is the transmission control protocol (TCP) acknowledgement problem (Karlin et al. 1990). The system needs to decide how long a packet waits before sending the acknowledgement. Waiting incurs delay cost while acknowledging incurs some cost that is more than delay cost. It can be beneficial as multiple packets can potentially be acknowledged together.

Consider the MCOSP problem in the online manner. Similar to the TCP acknowledgement problem, MCOSP can be considered as a generalization of *rent/buy* problems. Multiple jobs can be submitted to the scheduler at the same point of time. Hence, as the service continues, there will be several mobile tasks waiting on the scheduler to be dispatched. In particular, consider the problem as to whether to hold a task waiting in the scheduler or to offload the tasks to one of the machines in HMC to execute. The tasks may finish execution earlier if they wait in the scheduler for some time and execute on another machine that is available later other than the currently available ones. A task waiting to be processed sometime in the future generates a waiting cost of 1 for each time unit. Otherwise, it can pay an additional cost B to execute on one of the machines immediately. Obviously, if the scheduler somehow knows the task needs to wait for at least B time periods, it is optimal to pay a higher cost and run the task immediately in one of the machines at the beginning. However, in reality, the scheduler has no knowledge of that. Therefore, for each task that awaits in the queue, the scheduler needs to decide at each time period whether to keep the task waiting or offload to a machine, considering the scheduler does not have any knowledge of the next task arrival. We define the following entities in the context of MCOSP:

(1) **Rent:** The scheduler holds the task awaiting in the queue for one time period.
(2) **Buy:** The scheduler dispatches the task to one of the machines for execution.
(3) **Renting cost R_i:** Renting incurs one cost per time unit. Thus, the renting cost R_i of task i refers to the time period task i waited before being dispatched. R_i is defined as:

$$R_i = T_i^{wait}$$

(4) **Buying cost $B_{i,m}$:** Buying cost refers to the time consumed for the task to finish execution. Assume task i is assigned to machine m for execution. $B_{i,m}$ is defined as:

$$B_{i,m} = T_m^{avail} + T_{i,m}^{exec} + T_{i,m}^{trans} - T_i^{arrival},$$

where T_m^{avail} is the earliest available time of machine m, $T_{i,m}^{exec}$ is the execution time of task i on machine m, and $T_i^{arrival}$ is the arriving time of task i. To calculate the task execution time of task i on machine m, if there is any task offloading required, the data transmission time for offloading is calculated based on the wireless channel availability of machine m (i.e., $I_{wifi}^m, I_{3g}^m, I_{bt}^m$) in Equation (12). If multiple channels are available, then the channel with minimum transmission time is selected. The leaving time of a machine is considered when calculating the cost. That is, if the leaving time of a machine m is before a task i can finish computation, $T_{i,m}^{exec}$ is set to infinity so that the overall buying cost would be infinity.

ALGORITHM 1: Online Code Offloading and Scheduling Algorithm

```
Initialize : waitlist $L, n \leftarrow 0$
for $\forall m \in M$ do
    initialize machine properties $\mu_m, r_m, \theta_m, v_m, PR^{active}, PR^{idle}$
end for
while $n \leqslant D$ do
    Upon receiving a task $i$: $L \leftarrow i$
    $R_i \leftarrow 0$
    for $\forall m \in M$ do
        update $T^m_{avail}$
        $B_{i,m} \leftarrow 0$
    end for
    for $\forall i \in L$ do
        for machine $m \in M$ do
            calculate $B_{i,m}$
        end for
    end for
    $B_{i,m*} \leftarrow min_{m \in M} B_{i,m}$
    for $\forall i \in L$ do
        $R_{min} \leftarrow$ MAX_VALUE
        if $R_i \geqslant B_{i,m*}$ and $R_i < R_{min}$ then
            $R_{min} \leftarrow R_i$
            $tag \leftarrow i$
        end if
    end for
    if $C_{tag,m^*} + C^{current}_{m^*} \leqslant C^{budget}_{m^*}$ and $E_{tag,m^*} + E^{current}_{m^*} \leqslant E^{total}_{m^*} \cdot \theta_{m^*}$ then
        Schedule task $tag$ to machine $m^*$
        Remove task $tag$ from $L$
    end if
    for $\forall i \in L$ do
        $R_i \leftarrow R_i + 1$
        Update $R_i$
    end for
    $n \leftarrow n + 1$
end while
```

5.2 Online Code Offloading and Scheduling Algorithm

Based on the problem definition above, we proposed an online code offloading and scheduling algorithm (OCOS) based on the break-even algorithm. The break-even algorithm (Karlin et al. 1986) has been widely used to design the online algorithms. A break-even point refers to the time when the renting cost equals the buying cost. The algorithm decides to buy the resources after the break-even point. It has been proven to be 2-competitive (Karlin et al. 1986).

A discrete time horizon of D epochs is considered, where D could possibly be infinite. Starting from time 0, the tasks arrive at the scheduler at arbitrary epoch. The pseudo code of the proposed algorithm is given in Algorithm 1. Firstly, the scheduler initializes a waitlist L and updates all the information of the machines available (e.g., earliest available time, processing speed, energy rate, and so on) by sending a request to each machine (Steps 1–4). If there is any change of the information, it will be sent to the scheduler periodically. When a task is generated, its information is sent to the scheduler, it is added into the waitlist, and its renting cost is set to 0 (Steps 6–7). At the beginning of each scheduling epoch, the scheduler updates the earliest available time of each machine (Steps 12–16). The machine m^* that has the minimum buying cost is selected (Step 17). Then, it calculates the renting cost and buying cost on every machine for each task in the waiting list. Tasks meeting the following condition will be selected and scheduled to the corresponding machine m with the least buying cost $B_{i,m}$:

$$R_i \geqslant \min_{m \in M} B_{i,m}. \tag{24}$$

If there is more than one task meeting the above-mentioned condition for the same machine, the task with the lowest R will be scheduled (Steps 18–24). Note that to follow the energy and monetary cost constraints, the algorithm maintains two variables — $C_m^{current}$ for the current total monetary cost of machine m (i.e., mobile user m) and $E_m^{current}$ for the current overall energy consumption for machine m — to reject the schedule if the conditions are not met (Steps 25–28). Scheduled tasks will be removed from the waiting list. Then, the scheduler updates the renting cost of tasks in the waitlist by adding one time unit (Steps 29–32).

The complexity of the proposed algorithm is $O(|M| \cdot |S|)$, given that $|S|$ tasks arrived during the time period D and there are $|M|$ machines available. Step 6 takes $O(|M|)$ to update the available time at each epoch. Updating buy cost for each task in the waiting list takes $O(|M| \cdot |S|)$ (Steps 6–10). Step 11 takes $O(|S| log |M|)$ to check each task's condition. Therefore, the overall algorithm takes $O(|M| \cdot |S| + |M| + |S| log |M|)$ at each epoch, which can be summed as $O(|M| \cdot |S|)$. We then present a **competitive analysis** on the online algorithm as follows.

Definition 5.1 (Competitive Ratio). An online algorithm *ALG* is **c-competitive** if for all finite input sequences I,

$$ALG(I) \leq c \cdot OPT(I) + \alpha,$$

where $ALG(I)$ is the cost of the online algorithm and $OPT(I)$ is the cost of the off-line optimal (Borodin and El-Yaniv 1998). A c-competitive online algorithm ALG is also a c-approximation algorithm with the restriction that ALG must compute online.

The competitive ratio is evaluated to show the performance of the proposed online algorithm by calculating the total completion time of the tasks in a time period for the off-line optimal solution and the online algorithm.

Let $T_{\sigma,m}^{ALG}$ and $T_{\sigma,m}^{OPT}$ denote the makespan of tasks scheduled on machine m by the proposed OCOS algorithm and off-line optimal, respectively, where σ denotes the set of tasks scheduled to machine m by OCOS or off-line optimal. Given a heterogeneous set of machines M, and an arbitrary task sequence σ to schedule among machines in M, we obtain the following:

Lemma 5.2. *For $\forall m \in M, T_{\sigma_A,m}^{ALG} \leqslant 2\, T_{\sigma_O,m}^{OPT}$, where $\sigma_A \in \sigma, \sigma_O \in \sigma$ are the task sequences scheduled to machine m by the OCOS algorithm and off-line optimal algorithm, respectively.*

Proof. For task set σ_A and σ_O, there are two cases to discuss where $\sigma_A = \sigma_O$ and $\sigma_A \neq \sigma_O$.

First, if $\sigma_A = \sigma_O$, based on the scheduling policy of the OCOS algorithm (Equation (24)),

$$T_{\sigma_A,m}^{ALG} = \sum_{i \in \sigma_A} (R_i + B_{i,m}) = 2 \sum_{i \in \sigma_A} B_{i,m}$$

$$T_{\sigma_O,m}^{OPT} = \sum_{i \in \sigma_O} B_{i,m} = \sum_{i \in \sigma_A} B_{i,m} = \frac{1}{2}\, T_{\sigma_A,m}^{ALG}.$$

Thus, for $\sigma_A = \sigma_O$, $T_{\sigma_A,m}^{ALG} = 2\, T_{\sigma_O,m}^{OPT}$.

Second, if $\sigma_A \neq \sigma_O$, the possible relations between σ_A and σ_O are either $\sigma_A \subset \sigma_O$ or $\sigma_A \not\subset \sigma_O$. We prove by contradiction that the case of $\sigma_A \not\subset \sigma_O$ does not exist.

Assume $\sigma_A \not\subset \sigma_O$ exists. Then, let $\sigma' = \sigma_A - \sigma_O$ so that σ' is scheduled to machine m by the OCOS algorithm. Based on OCOS algorithm's scheduling policy that dispatches task i only when the renting cost R_i is equal to the minimum of buying cost $B_{i,m}$ among M (Equation (24)), $\sum_{i \in \sigma'} B_{i,m}$ is the minimum. On the other hand, since σ' is scheduled to another machine by the off-line optimal algorithm, for instance, machine n, we have $\sum_{i \in \sigma'} B_{i,n} \leqslant \sum_{i \in \sigma'} B_{i,m}$, which is contradicted to the case, for which $\sum_{i \in \sigma'} B_{i,m}$ is the minimum. Therefore, $\sigma_A \not\subset \sigma_O$ does not exist.

When $\sigma_A \subset \sigma_O$,

$$\frac{T^{ALG}_{\sigma_A,m}}{T^{OPT}_{\sigma_O,m}} = \frac{T^{ALG}_{\sigma_A,m}}{T^{OPT}_{\sigma_A,m} + T^{OPT}_{(\sigma_O - \sigma_A),m}} \leqslant \frac{T^{ALG}_{\sigma_A,m}}{T^{OPT}_{\sigma_A,m}} = 2.$$

Therefore, for $\forall m \in M$, $T^{ALG}_{\sigma_A,m} \leqslant 2\, T^{OPT}_{\sigma_O,m}$. This completes the proof. □

Theorem 5.3. *The proposed OCOS algorithm for task scheduling on heterogeneous mobile cloud environment is 2-competitive.*

Proof. Let ALG denote the proposed OCOS algorithm and OPT denote the off-line optimal algorithm. Let $\sigma = (t_1, t_2, \ldots, t_n)$ be an arbitrary tasks sequence submitted for scheduling, and $\forall m \in M$ be a machine in the environment to execute tasks. $ALG(\sigma)$ and $OPT(\sigma)$ denote the makespan of the schedule generated by the OCOS algorithm and the off-line optimal schedule, respectively, on the input sequence σ.

Without loss of generality, we assume that $\sigma = \sigma_1 \cup \sigma_2 \cup \cdots \cup \sigma_m$ is the task sequence scheduled on each machine by algorithm OPT, and

$$OPT(\sigma_1) \leqslant OPT(\sigma_2) \leqslant \cdots \leqslant OPT(\sigma_m).$$

Let $\sigma = \sigma'_1 \cup \sigma'_2 \cup \cdots \cup \sigma'_m$ be the task sequence scheduled by algorithm ALG and

$$ALG(\sigma'_1) \leqslant ALG(\sigma'_2) \leqslant \cdots \leqslant ALG(\sigma'_m),$$

where σ_m and σ'_m are tasks scheduled to machine m by OPT and ALG, respectively. Thus, the makespan $OPT(\sigma) = OPT(\sigma_M)$ and $ALG(\sigma) = ALG(\sigma'_m)$.

Based on the definition of competitive ratio and Lemma 1,

$$\frac{ALG(\sigma)}{OPT(\sigma)} = \frac{ALG(\sigma'_m)}{OPT(\sigma_m)} \leqslant 2.$$

Hence, the competitive ratio of the OCOS algorithm is 2. □

6 PERFORMANCE EVALUATION

In this section, we evaluate the proposed algorithms in two aspects, namely the scheduling performance (i.e., the makespan of the tasks) and the offloading performance (i.e., total execution time and energy saved). Six sets of experiments are conducted to evaluate the proposed off-line and online scheduling solutions in terms of scheduling performances, algorithm efficiency, and the effect of the parameters considered by the algorithms. For the scheduling performance, the proposed algorithms are compared with the online mode independent task scheduling heuristic opportunistic load balancing (OLB) (Braun et al. 2001). For the offloading performance, the proposed algorithms are compared with the offloading policies in ThinkAir (Kosta et al. 2012).

6.1 Experiment Settings

For the off-line mixed integer linear programming model proposed, the off-line optimal algorithm is implemented in Gurobi Optimizer 6.5 (Optimization et al. 2015) to provide the off-line optimal solutions of the model. The optimization is undertaken on an m1.xlarge instance on Nectar Cloud

Table 3. Characteristics of Workloads

Type	Data Size (MB)	CPU Cycle (Giga)
LCSD	[0.05, 0.5]	[1, 10]
HCSD	[1, 5]	[50, 100]
HCLD	[1, 5]	[50, 100]

Table 4. Parameter Settings for Evaluation

Parameter	Value
Task data size (f)	[0.05,0.5],[1,5]
Task computation (ϖ)	[1,10],[50,100]
Mobile device CPU frequency (μ)	{0.2,0.5,0.8,1.0,1.2}
Cloudlet CPU frequency	2.5
Cloud VM CPU frequency	3.6
Cloud VM charge rate	0.84
Battery limit fraction (θ)	Uniform(0.2,0.5)
Active CPU power consumption rate (PR^{active})	0.07,0.34,0.48,0.56,0.6
Idle CPU power consumption rate	0.002
WiFi data rate (BW_{wifi})	1
Bluetooth data rate (BW_{bt})	0.26
Cellular data rate (BW_{cell})	0.85
WiFi power rate (ρ_{wifi})	1.94
Bluetooth power rate (ρ_{bt})	0.28
Cellular power rate (ρ_{cell})	5.56
Weibull parameters (α, β)	(1.9543,326.87),(1.2861,178.56), (0.8712,276.87), (1,163)

(Nectar 2015) with 8vCPUs and 32GB RAM. For the online algorithm evaluation, we develop our own experiment environment to evaluate the performance of the proposed algorithms. The scheduler runs on a local computer that has an Intel Core i7 CPU with 3.4GHz and 8GB RAM. Experiment settings and workload parameters are listed in Table 4.

Due to the lack of real-world traces that are suitable for the HMC environment and mobile device constraints, workloads are obtained by profiling the cognitive applications running on mobile devices in the experimental environment. The workload consists of three types of task sets that represent the diversity of mobile application tasks, namely low computation small data size (LCSD), high computation small data size (HCSD), and high computation large data size (HCLD).

In order to profile values of the parameters of task sets, an open-source OCR application[1] is executed on Android with Android Studio performance analysis tools to profile the OCR application on the method level. The profile, which consists of data size of the captured picture frame and inclusive CPU running time of each operation (i.e., methods for Android application) for processing, is then classified into the three workload types. In workload LCSD, the range of data size of tasks lies in the interval [0.05, 0.5] megabytes, and the computation of completing each task distributes in the interval [1, 10] giga CPU cycles. Similarly, in workload HCSD and HCLD, data size of the tasks are in the interval of [0.05, 0.5] and [1, 5] megabytes, respectively, and the computation of both task sets is in an interval of [50, 100] giga CPU cycles. A summary of the workloads used in the experiment is listed in Table 3.

Regarding the experiment environment, it includes two identical cloudlets and three identical VMs in the public cloud. The number of mobile devices varies based on different experiments conducted. The hardware information is profiled from a number of Android smartphones. For each mobile device, the CPU speed μ is randomly assigned from the set {0.2, 0.5, 0.8, 1.0, 1.2}GHz for the mobile devices. These CPU frequencies are obtained from different mobile devices such as Nexus 4, HTC G13, HTC G3, and Samsung I997. The CPU speed of the cloudlet is 2.5GHz, and cloud VM CPU speed is 3.6GHz following the Amazon EC2 C3 instances. The monetary cost of cloud VM is \$0.84 per offloading request. θ_i is uniformly selected from the interval [0.2, 0.5]. For the energy aspect of the mobile devices, the parameters given in the energy models proposed by Ali et al. (2016) are adopted in the experiments. The active CPU energy consumption rate PR^{active}

[1] Available at https://github.com/rmtheis/android-ocr.

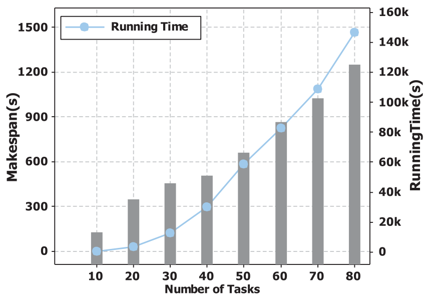


Fig. 2. Makespan and running time of the off-line optimal approach.

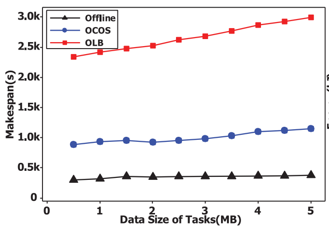
is set from {0.07,0.34,0.48,0.56,0.6} based on the above-mentioned CPU frequency, accordingly. The idle power consumption rate PR_{idle} is set to 0.002 for all devices.

The network of the experiment environment consists of three types of wireless mediums, namely mobile cellular network, Bluetooth, and WiFi. Network parameters are obtained by profiling the network conditions in our experiment environment using file transferring applications on the mobile device. WiFi network speed BW_{wifi} is set to 1MBps, Bluetooth transmission speed BW_{bl} is set to 0.26MBps, and cellular network speed BW_{cell} is set to 0.85MBps. The network latency to public cloud service is set to 0.1s. It is profiled by testing the delay to the Amazon EC2 service in the Sydney region. Moreover, the energy consumption parameters of WiFi, Bluetooth, and cellular network are set to $\rho_{wifi} = 1.94\ W$, $\rho_{bt} = 0.28\ W$, and $\rho_{cell} = 5.56\ W$, respectively, based on the energy model proposed by Balasubramanian et al. (2009).

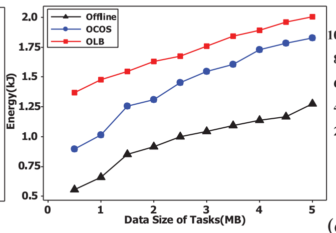
A 2-parameter Weibull distribution is used to obtain the leaving time of the mobile device from HMC. To capture the mobility patterns of real mobile devices in the wireless network, the CRAWDAD tracesets (Kotz et al. 2009) are used to obtain valid values for the shape parameter α and the slope parameter β in the Weibull distribution. The traceset is composed of the system logs of WiFi access points on the Dartmouth campus from September 1, 2005 to October 4, 2006. The number of sessions in every minute is extracted from the trace to generate the probability function using rank regression (Rinne 2008). Then, the time-to-failure of each machine is randomly obtained from the inverse function of Weibull distribution in order to generate the t^{leave}.

6.2 Experiment Results

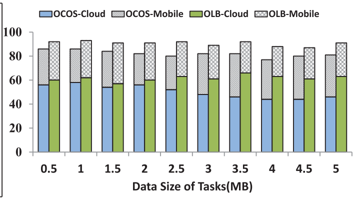
6.2.1 Off-line Approach Performance Evaluation. Figure 2 shows the time taken to generate the solution and the makespan of tasks in the workload by using the off-line approach. The task set for the experiment is obtained from workload HCLD. Note that since the off-line approach uses the BB algorithm to solve the MILP model formulated, the search space of the BB algorithm is only based on the number of tasks and machines rather than the heterogeneity of the tasks and machines. Hence, either workload can be used for the evaluation of the off-line approach. Results of makespan for the off-line optimal are used as a benchmark for the online algorithms' evaluation. As shown in Figure 2, the processing time for producing the optimal solution increases expeditiously as the number of tasks grows. This is because the complexity of our proposed MILP model is a product of the number of tasks, the number of machines, and the number of variables defined. Hence, solving an off-line optimal schedule that involves large data sets would require an unreasonably large amount of time for the mobile cloud offloading systems, which demands lightweight and timely offloading decisions. Therefore, the results show necessity of the online scheduling algorithm to solve the proposed mobile cloud offloading and scheduling problem with low processing delay.



(a) Makespan



(b) Energy consumption



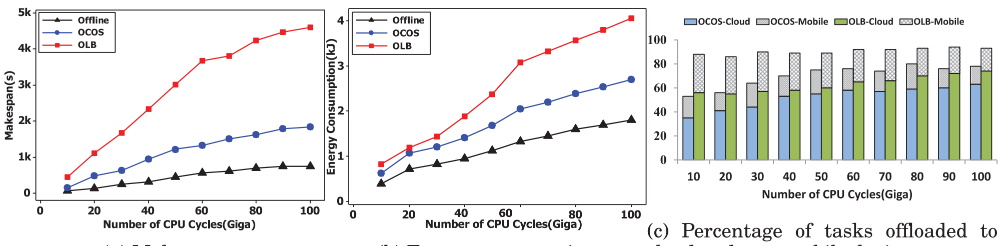
(c) Percentage of tasks offloaded to cloud and peer mobile devices

Fig. 3. Performance of OCOS algorithm with different task data size.

6.2.2 OCOS Algorithm Performance Evaluation. The performance of the proposed OCOS algorithm is investigated. The online solutions under all three types of workloads are evaluated and compared with the non-preemptive task scheduling heuristic OLB as well as the benchmark offline optimal schedule. The OLB heuristic schedules tasks to the machine with the earliest available time when tasks arrive at the scheduler. Note that the OLB heuristic is different from the OCOS algorithm in that it does not consider the offloading benefits (e.g., whether the offloading would shorten the task completion time) when scheduling the tasks. Two metrics are considered, namely makespan and the overall energy consumption. The makespan represents the maximum task completion time of the set of scheduled tasks. The overall energy consumption is the sum of each mobile device's energy consumption for running the scheduled tasks.

To explore the impact of different offloading data sizes on the performance of the OCOS algorithm, tasks from workload HCSD and HCLD are categorized into 10 task sets based on the offloading data size (from 0.5MB to 5MB) and CPU cycles from 50 to 60 giga cycles. Results are averaged by 50 runs. Figure 3(a) depicts that the makespan of the OCOS algorithm and off-line approach algorithm only have a small increase as the offloading data size of the tasks increases. OLB heuristic generates a much higher makespan. This is caused by its scheduling strategy that only considers the earliest available time of the machines, which makes the load of machines unbalanced. Also, as the data size increases, there is a much faster growth in makespan than the OCOS algorithm and off-line approach, since mobile devices take a longer time to transfer the offloaded data. The results indicate that, for the cognitive application with offloading data size less than 5MB, the performance of OCOS is not affected by the data size. It can also be observed that makespans generated by the online algorithm are around 2.6 times as much as the off-line approach on average, while the OLB heuristic is five times longer. Different from the 2x makespan shown in Theorem 5.3, the 2.6x makespan yielded by the experiment is caused by the difference of modeling and hardware in machines. The models for task execution time have a high level of hardware abstraction to eliminate the complexity lying in hardware (which is not the focus of this work). However, in reality, machines have multi-core CPU and multi-level memory hierarchy I/O that will affect the task execution time. Moreover, in reality, mobile devices and HPC servers run many background activities that will affect and lengthen the execution time of tasks in the proposed work.

The overall energy consumption of the mobile devices in the experiments (Figure 3(b)) increases as the data size grows. For the OLB heuristic, the percentage of tasks scheduled to cloud and peer mobile devices does not have much fluctuation (shown in Figure 3(c)) with the growth of data size, due to the earliest-available-time-only scheduling strategy. The increase in the energy consumption of the schedule made by the OLB heuristic is caused by the larger data size and the higher makespan as the data size increases. For the OCOS algorithm, more tasks are scheduled to



(a) Makespan (b) Energy consumption (c) Percentage of tasks offloaded to cloud and peer mobile devices

Fig. 4. Performance of OCOS algorithm for tasks with different computing requirements.

peer mobile devices (shown in Figure 3(c)) rather than the cloud as the data size of tasks grows. This is because OCOS considers the task completion time as well as the energy consumption of executing tasks. Tasks with larger data size tend to be offloaded to peer mobile devices rather than to the cloud to reduce the data offloading energy consumption as well as the transferring time and to generate balanced schedules. As a result, the overall energy consumption of all mobile devices increases. In addition, the growth in data size also increases the energy consumption of the wireless transmitters.

Furthermore, for tasks with different computing resource requirements, the impact on the performance of the proposed algorithms is evaluated. For each run, the tasks from workload type LCSD and HCSD are categorized into the 10 task sets based on the CPU cycles required (10 to 100). Results are averaged by 50 runs. The makespan of schedules, total energy consumption, and portion of tasks offloaded to the cloud as well as peer mobile devices are shown in Figure 4. In Figure 4(a), the makespans generated by all three algorithms increase with the increase in computation load. The makespan generated by the OLB heuristic grows much faster than that of the OCOS algorithm due to its offloading strategy that tasks with high computation requirements are scheduled to low computing capacity mobile devices. Eventually, the makespan of the unbalanced schedule becomes higher. On the other hand, the growth of makespan generated by the OCOS algorithm is much slower than that of the OLB heuristic. This is because the OCOS algorithm compares the current renting cost (waiting time) of the task with its buying cost (task completion time) on different machines when making the scheduling decisions, and thus, generates a more load-balanced schedule compared to OLB. In terms of energy consumption, Figure 4(b) shows that the differences of growth trend between three algorithms are similar to those of the makespan in Figure 4(a). This is because the longer the execution time of the task, the higher the energy mobile devices consume. Therefore, as the computation requirements of the tasks grow, the overall energy consumption of the mobile devices increases. On average, the energy consumed executing the tasks of the schedule generated by the OCOS algorithm is around two times less than that of OLB, and around 1.5 times more than that of the off-line approach.

Figure 4(c) shows the percentage of tasks scheduled to offload to the cloud and peer mobile devices by the OCOS algorithm and OLB heuristic, respectively, under different computing requirements. For the OCOS algorithm, the portion of tasks offloaded to the cloud and the total portion of tasks offloaded both increase with the increase of computing requirements of tasks. This is because OCOS considers the task completion time as well as how much time and energy can be saved when deciding the schedule. Therefore, as the computing requirements of the tasks increase, OCOS schedules more tasks to the cloud to reduce the makespan. On the contrary, the OLB heuristic only considers the earliest available time of the machine when scheduling tasks. Thus, most of the tasks (90% on average) are offloaded to the cloud and other mobile devices regardless

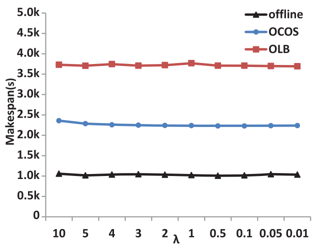


Fig. 5. Makespan with different task arriving rates (λ).

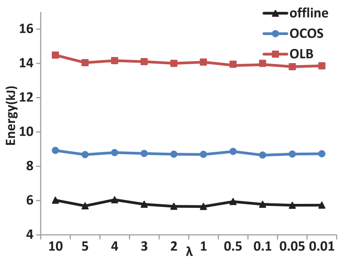


Fig. 6. Energy consumption with different task arriving rates.

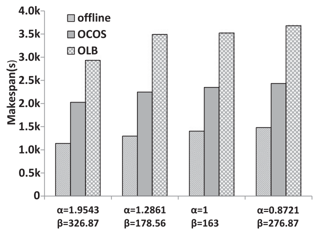


Fig. 7. Makespan with different Weibull Distribution (α: shape, β: scale).

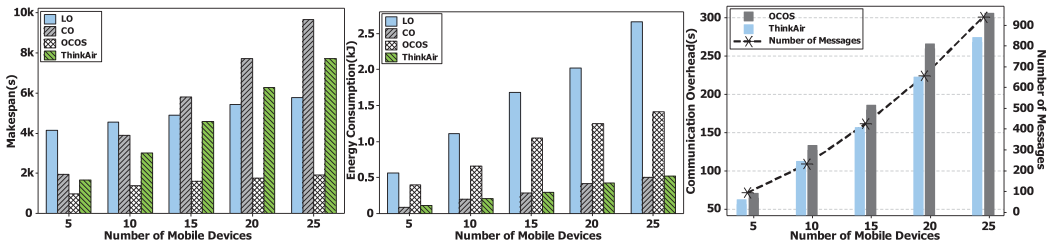
of the offloading benefits. This scheduling strategy also causes unbalanced schedules among tasks (shown in Figure 4(a)), compared to OCOS that takes machine load balance into consideration.

Based on the results obtained from the two sets of experiments conducted above, compared to OLB, the OCOS algorithm has a steady performance on makespan and energy consumption for cognitive applications with task data size between 1MB and 5MB. The performance ratio of the online algorithm over the off-line approach is around two for makespan.

To observe the performance of the algorithm in terms of mobile device reliability, two sets of experiments are conducted. The first set compares the makespan and energy consumption of the three algorithms with different task arriving rates (i.e., λ). The second experiment tests the effect of different mobile device reliability (i.e., device leaving time) on makespans of scheduled tasks. Tasks from workload HCSD are used for both sets of experiments. Results are shown in Figures 5–7. From Figure 5 and Figure 6, we can observe that the task arriving rate does not have much influence on makespan and overall mobile device energy consumption since the results do not have fluctuations over different arriving rates. This is due to the fact that all three algorithms aim to schedule multiple tasks that match their scheduling policies at the same time. Moreover, makespan and energy consumption generated by the OCOS algorithm are around 65% and 57% of the OLB algorithm respectively, and 2.2 and 1.5 times of the off-line algorithm, respectively.

To test the performance of proposed algorithms on mobile device reliability, multiple cases are tested to represent different device mobility as device leaving time. The device leaving time is obtained from the above-mentioned Weibull distributions in Section 6.1 using CRAWDAD tracesets. The shape (α) and scale (β) parameters for the Weibull distributions are listed in Table 4. The same task set from workload HCSD is used for all the cases. α, β for the first two cases are extracted from the data between 1 and 8 p.m. in tracesets, the fourth case is obtained from data between 5 a.m. and 1 p.m., and the third case is a synthetic pair of parameters for comparison. $\alpha < 1$ indicates the device failure rate decreases with time; that is, mobile devices have short connection time. $\alpha > 1$ indicates the device failure rate increases with time and mobile devices have a longer connection time, while $\alpha = 1$ represents a constant failure rate. In Figure 7, it shows that as the device failure rate increases (i.e., mobile device connection time decreases), the makespan generated by OCOS has an approximate 10% increase throughout four cases, while makespans generated by OLB have a 20% increase. This is due to the fact that more tasks are scheduled onto cloud servers instead of mobile devices that left the environment. The makespans of the OCOS algorithm are around 37% less of those generated by the OLB algorithm since OCOS considers the load balancing of machines during scheduling.

Moreover, the performance of the OCOS algorithm in terms of offloading benefits (e.g., execution time and energy saved) against conventional offloading strategies is evaluated. Upon comparison,



(a) Makespan (b) Energy consumption (c) Communication time overhead

Fig. 8. Performance comparison of different offloading strategies.

three baselines are implemented. The first baseline, offloading to the cloud only (CO), always offloads the submitted mobile tasks to the cloud VMs. The second base online, Mobile Local Only (LO), always executes mobile tasks on its own mobile device locally. The third baseline algorithm is the mobile code offloading strategies proposed in ThinkAir with the execution time priority policy (Kosta et al. 2012). Note that the offloading decision making algorithm in ThinkAir only considers the context of a single mobile device to the cloud offloading model. All algorithms are evaluated with the same task sets from workload type HCLD. The size of the task set is 20 tasks generated per mobile device on average. Results are shown in Figure 8 with a different number of mobile devices.

As shown in Figure 8(a), the makespan generated by all four algorithms increases as the number of mobile device users increases. For baseline LO and CO, the growth of CO is much faster than that of the LO. Similarly, the makespan generated by ThinkAir increases faster than the LO baseline. The results indicate that as the number of mobile device users increases, the benefits of offloading to fixed public cloud resources decreases. This is because both baseline algorithms make individual offloading decisions without considering the overall computation load of the cloud resources. As a result, the task offloaded by one mobile device may be delayed in the cloud VM, which ends up with a longer makespan. On the contrary, the makespan generated by the OCOS algorithm increases much slower compared to the other three baselines. This is because OCOS uses the peer mobile devices in the HMC environment and considers the computation load of each machine in the environment when making the offloading decisions. It can also be observed that the ratio of the makespan of ThinkAir over that of the OCOS algorithm increases as the number of users increases. The growth shows that the OCOS algorithm outperforms conventional offloading strategies (e.g., CO, LO, and ThinkAir) when the number of mobile users is large.

Furthermore, the performance of the OCOS algorithm in terms of overall mobile device energy consumption is illustrated in Figure 8(b). It shows that the overall mobile device energy consumption in HMC for all the algorithms increases with growth in the number of mobile users. The energy consumption of algorithm CO and ThinkAir is relatively low compared to the other two algorithms since the mobile devices consume much less energy in the idle mode when offloading tasks to the cloud. Compared to that, for the OCOS algorithm, the overall energy consumption for all the mobile devices is around 50% of LO on average, and twice as much as that of CO and ThinkAir. This is because the objective of the OCOS algorithm is to minimize the makespan with the constraints of mobile devices such as battery lifetime and mobility. Thus, although the energy consumption is higher than CO and ThinkAir, the makespan generated by the OCOS algorithm is one fourth and half of the makespan generated by CO and ThinkAir, respectively, and can get much lower when the number of mobile users scales up (as shown in Figure 8(a)).

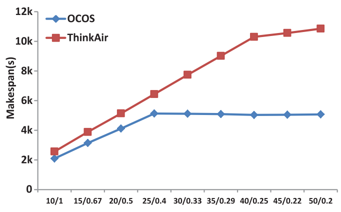


Fig. 9. Makespan with different WiFi bandwidth usages.

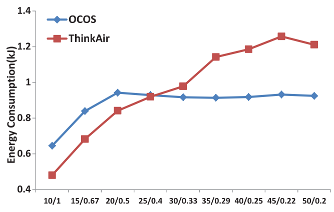


Fig. 10. Average energy consumption per machine with different WiFi bandwidth usages.

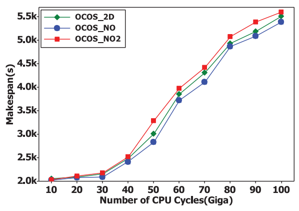


Fig. 11. Makespan with different Weibull Distribution (α: shape, β: scale).

In addition, the communication time overhead for operations related to communication such as data offloading and messages passing are evaluated, and results are shown in Figure 8(c). The communication overhead of OCOS is slightly larger than that of ThinkAir due to the use of Bluetooth when offloading to a peer mobile device, which incurs a longer communication time comparing to WiFi. The average communication time overhead for the OCOS algorithm is around 12% of the makespan. Although it is slightly larger than ThinkAir, the offloading gain (i.e., makespan saved) is much more than ThinkAir. Furthermore, as a centralized scheduling algorithm, it is necessary to measure the behavior of the number of control messages sent by the scheduler to update the hardware information of the mobile devices and cloud. The results are shown in Figure 8(c) with the dashed line. The number of messages sent by the scheduler increases as more mobile users join the environment. Since ThinkAir makes offloading decisions locally on each mobile devices, it has less communication overhead than OCOS. However, the difference of the communication overhead is rather small compared to the shorter makespan enabled by OCOS.

In reality, there are usually limited numbers of WiFi access points available in a mobile cloud network; therefore, the limited network bandwidth may affect the performance of the proposed algorithm. A set of experiments are conducted to analyze the influence of network bandwidth usages. The effects of cellular network, WiFi-direct, and Bluetooth are not considered here as their bandwidths are not affected by the number of machines in the network. The experiment is set with one available WiFi access point and it provides the fixed bandwidth of 10Mbps. All the machines share the network bandwidth evenly. The task set containing a number of 500 tasks obtained from workload type HCLD are used for the experiments. The results are averaged by 50 runs and shown in Figures 9 and 10 in comparison with results of ThinkAir. In Figure 9, as the average bandwidth per machine decreases from 1Mbps, the makespan of OCOS increases from around 2,000 seconds to 5,000 seconds and becomes stable after 0.4Mbps since the bandwidth does not change too much after that, while the makespan of ThinkAir increases more rapidly. The increase of makespan is due to the competition of bandwidth between devices leading to longer data transferring time over WiFi to public clouds. Also, OCOS schedules some tasks with large data size to run local instead, which generates longer makespan. The similar results also reflect on the average energy consumption per mobile device. The shorter makespan and less energy consumption generated by OCOS, especially with more devices, is because of the use of a mobile device cloud that is able to load balance the computation among the entire network.

Since machines, especially mobile devices, may drop out of the network before their estimated leaving time, the robustness of the OCOS algorithm is evaluated. The makespans generated over different task computation workloads are compared in three cases: (1) no machine dropouts, (2) two randomly chosen machines dropping out at random time before its estimated leaving time, and (3) no machine dropouts without the two machines chosen in case 2 in the network. The results are averaged by 50 runs and depicted in Figure 11. *OCOS_NO* represents case 1, *OCOS_2D* represents case 2, and *OCOS_NO2* represents case 3. As shown in Figure 11, the machine dropouts do not have a significant effect on the makespan generated by the OCOS algorithm. Compared with the results of *OCOS_NO*, the makespan has around 0.05% increase for different amounts of computation workload with machines accidentally dropping out of the network. This is because the OCOS algorithm receives updates on the earliest available time from each machine in the network at the beginning of each scheduling epoch. If a machine drops out before its estimated leaving time, the scheduler is able to detect it and considers the rest of the machines when scheduling tasks. Moreover, the makespans of the OCOS algorithm with machine dropouts (*OCOS_2D*) are lower than those generated by the OCOS algorithm with two less machines in the network from the beginning (*OCOS_NO2*) due to the load balancing of the OCOS algorithm.

7 CONCLUSIONS AND FUTURE WORK

In this article, we discussed the MCOSP in the heterogeneous mobile cloud environment. We formally defined the problem with a mixed integer programming model that considered a shared resources pool of heterogeneous machines including peer mobile devices, nearby private cloud, remote public cloud services and various network conditions. In order to provide an online optimal scheduling solution for our proposed mobile code offloading and scheduling problem, we further designed a centralized, real-time scheduling algorithm based on the ski-rental framework. We show that the proposed online scheduling algorithm OCOS is 2-competitive of the off-line optimal solution. The experimental results show that the OCOS algorithm is consistent with the off-line optimal solution in terms of competitiveness. The algorithm generates shorter makespans and less energy consumption with the cognitive applications such as OCR and face detection applications comparing to the OLB heuristic, and also scales well comparing to other previous proposed mobile code offloading strategies when the number of mobile device users increases in the system.

For the future work, we plan to investigate the real-time scheduling solutions for mobile cloud services that combine both mobile code offloading and mobile task delegation (Flores and Srirama 2014) in which the dependency of the mobile tasks within a mobile application as well as the application deadlines should be considered when scheduling. Moreover, as energy constrain is vital for mobile devices, a joint optimization of makespan and energy consumption for mobile cloud offloading will be studied in the future work. The malicious users that only submit tasks while rejecting task offloading requests from the HMC network can affect the performance of the scheduling results. The fault detection and tolerance mechanism needs to be studied to prevent such malicious activities.

REFERENCES

Farhan Azmat Ali, Pieter Simoens, Tim Verbelen, Piet Demeester, and Bart Dhoedt. 2016. Mobile device power models for energy efficient dynamic offloading at runtime. *Journal of Systems and Software* 113 (2016), 173–187.

Alison Doig Alisa Land. 1960. An automatic method of solving discrete programming problems. *Econometrica* 28, 3 (1960), 497–520.

Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. 2009. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC'09)*. ACM, New York, NY, 280–293.

Sergio Barbarossa, Stefania Sardellitti, and Paolo Di Lorenzo. 2013. Joint allocation of computation and communication resources in multiuser mobile cloud computing. In *Proceedings of the IEEE 14th Workshop on Signal Processing Advances in Wireless Communications*. IEEE, 26–30.

Marco Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. 2013. To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In *Proceedings of IEEE INFOCOM Student Poster Session*. 1285–1293.

Allan Borodin and Ran El-Yaniv. 1998. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, NY.

Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Blni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel and Distrib. Comput.* 61, 6 (2001), 810–837.

Chao Chen, Weidong Bao, Xiaomin Zhu, Haoran Ji, Wenhua Xiao, and Jianhong Wu. 2015. AGILE: A terminal energy efficient scheduling method in mobile cloud computing. *Transactions on Emerging Telecommunications Technologies* 26, 12 (2015), 1323–1336.

Shuang Chen, Yanzhi Wang, and M. Pedram. 2013. A semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud. In *Proceedings of the IEEE Global Communications Conference*.

Xu Chen. 2015. Decentralized computation offloading game for mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems* 26, 4 (2015), 974–983.

Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. CloneCloud: Elastic execution between mobile device and cloud. In *Proceedings of 6th Conference on Computer Systems*. ACM, New York, NY.

Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. ACM, New York, NY, 14.

Shuiguang Deng, Longtao Huang, J. Taheri, and A. Y. Zomaya. 2015. Computation offloading for service workflow in mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems* 26, 12 (Dec 2015), 3317–3329.

Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. 2013. A survey of mobile cloud computing: Architecture, applications, and approaches. *Wireless Communications and Mobile Computing* 13, 18 (2013), 1587–1611.

Xiaobo Fan, Carla S. Ellis, and Alvin R. Lebeck. 2005. *The Synergy Between Power-Aware Memory Systems and Processor Voltage Scaling*. Springer, Berlin, Germany, 164–179.

Huber Flores and Satish Srirama. 2014. Mobile cloud middleware. *Journal of Systems and Software* 92 (2014), 82–94.

Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY.

Bo Hong and Viktor Prasanna. 2007. Adaptive allocation of independent tasks to maximize throughput. *IEEE Transactions on Parallel and Distributed Systems* 18, 10 (Oct. 2007), 1420–1435.

Dong Huang, Ping Wang, and Dusit Niyato. 2012. A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications* 11, 6 (June 2012), 1991–1995.

Oscar H. Ibarra and Chul E. Kim. 1977. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM (JACM)* 24, 2 (1977), 280–289.

Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. 1990. Competitive randomized algorithms for non-uniform problems. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 301–309.

Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. 1986. Competitive snoopy caching. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*. 244–254.

Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. 2012. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings of the 31st IEEE International Conference on Computer Communications*.

David Kotz, Tristan Henderson, Ilya Abyzov, and Jihwang Yeo. 2009. CRAWDAD dataset dartmouth/campus (v. 2009-09-09). Retrieved from http://crawdad.org/dartmouth/campus/20090909/syslog.

Dejan Kovachev, Tian Yu, and Ralf Klamma. 2012. Adaptive computation offloading from mobile devices into the cloud. In *Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*. 784–791.

P. Venkata Krishna, Sudip Misra, D. Nagaraju, V. Saritha, and Mohammad S. Obaidat. 2016. Learning automata based decision making algorithm for task offloading in mobile cloud. In *Proceedings of the 2016 International Conference on Computer, Information and Telecommunication Systems (CITS)*. 1–6.

Karthik Kumar and Yung-Hsiang Lu. 2010. Cloud computing for mobile users: Can offloading computation save energy? *Computer* 43, 4 (April 2010), 51–56.

Kunfeng Lai, Hong Tang, Haiyang Wang, Shengyong Ding, and Dan Wang. 2013. Cloud offloading on customer-provided resources. In *Proceedings of the 2013 IEEE Wireless Communications and Networking Conference (WCNC)*. 4695–4700.

Jerald F. Lawless. 2011. *Statistical Models and Methods for Lifetime Data*. Vol. 362. John Wiley & Sons.

JongHyuk Lee, SungJin Choi, Taeweon Suh, HeonChang Yu, and JoonMin Gil. 2010. Group-based scheduling algorithm for fault tolerance in mobile grid. In *Security-Enriched Urban Computing and Smart Grid*. Springer, 394–403.

Kaiyang Liu, Jun Peng, Heng Li, Xiaoyong Zhang, and Weirong Liu. 2016. Multi-device task offloading with time-constraints for energy efficiency in mobile cloud computing. *Future Generation Computer Systems* 64 (2016), 1–14.

Xiaoqiang Ma, Yuan Zhao, Lei Zhang, Haiyang Wang, and Limei Peng. 2013. When mobile terminals meet the cloud: Computation offloading as the bridge. *IEEE Network* 27, 5 (September 2013), 28–33.

Yucen Nan, Wei Li, Wei Bao, Flavia C. Delicato, Paulo F. Pires, and Albert Y. Zomaya. 2016. Cost-effective processing for delay-sensitive applications in cloud of things systems. In *Proceedings of the IEEE 15th International Symposium on Network Computing and Applications (NCA)*. 162–169.

Nectar. 2015. *URL: https://nectar.org.au/research-cloud/* (2015).

Gurobi Optimization and others. 2015. Gurobi optimizer reference manual. *URL: http://www. gurobi. com* (2015).

MReza Rahimi, Nalini Venkatasubramanian, and Athanasios Vasilakos. 2013. MuSIC: Mobility-aware optimal service allocation in mobile cloud computing. In *Proceedings of 6th IEEE International Conference on Cloud Computing*. 75–82.

Horst Rinne. 2008. *The Weibull Distribution: A Handbook*. CRC Press.

Mahadev Satyanarayanan, Paramvir Bahl, Ramn Caceres, and Nigel Davies. 2009. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing* 8, 4 (Oct. 2009), 14–23.

Haluk Topcuoglu, Salim Hariri, and Min-you Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13, 3 (2002), 260–274.

Shangguang Wang, Tao Lei, Lingyan Zhang, Ching-Hsien Hsu, and Fangchun Yang. 2016. Offloading mobile data traffic for QoS-aware service provision in vehicular cyber-physical systems. *Future Generation Computer Systems* 61 (2016), 118–127.

Yonggang Wen, Weiwen Zhang, and Haiyun Luo. 2012. Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM)*.

Weiwen Zhang, Yonggang Wen, Kyle Guan, Dan Kilper, Haiyun Luo, and Dapeng Oliver Wu. 2013. Energy-optimal mobile cloud computing under stochastic wireless channel. *IEEE Transactions on Wireless Communications* 12, 9 (2013), 4569–4581.

Bowen Zhou, Amir Vahid Dastjerdi, Rodirgo Calheiros, Satish Srirama, and Rajkumar Buyya. 2015. mCloud: A context-aware offloading framework for heterogeneous mobile cloud. *IEEE Transactions on Services Computing* 10, 5 (2015), 797–810. DOI: http://dx.doi.org/10.1109/TSC.2015.2511002

Received November 2016; revised June 2017; accepted July 2017