mCloud: A Context-Aware Offloading Framework for Heterogeneous Mobile Cloud

Bowen Zhou, *Student Member, IEEE*, Amir Vahid Dastjerdi, *Member, IEEE*, Rodrigo N. Calheiros, *Member, IEEE*, Satish Narayana Srirama, and Rajkumar Buyya, *Fellow, IEEE*

Abstract—Mobile cloud computing (MCC) has become a significant paradigm for bringing the benefits of cloud computing to mobile devices' proximity. Service availability along with performance enhancement and energy efficiency are primary targets in MCC. This paper proposes a code offloading framework, called mCloud, which consists of mobile devices, nearby cloudlets and public cloud services, to improve the performance and availability of the MCC services. The effect of the mobile device context (e.g., network conditions) on offloading decisions is studied by proposing a context-aware offloading decision algorithm aiming to provide code offloading decisions at runtime on selecting wireless medium and appropriate cloud resources for offloading. We also investigate failure detection and recovery policies for our mCloud system. We explain in details the design and implementation of the mCloud prototype framework. We conduct real experiments on the implemented system to evaluate the performance of the algorithm. Results indicate the system and embedded decision algorithm are able to provide decisions on selecting wireless medium and cloud resources based on different context of the mobile devices, and achieve significant reduction on makespan and energy, with the improved service availability when compared with existing offloading schemes.

Index Terms—Mobile cloud computing, code offloading framework, context-awareness

1 Introduction

For the past decade, the popularity of mobile devices, such as smartphones and tablets, has been increasing rapidly among users because of the advanced functions provided by the enhanced mobile device with faster CPU, substantial memory, and multiple sensors. However, the upgrade in hardware results in higher energy consumption, which leads to a major concern of the insufficient battery lifetime on mobile devices. Furthermore, mobile device users expect improved performance from their mobile devices, which reflects on more endurable battery and shorter processing time of all services provided by the devices. To overcome this obstacle between the user demand and available mobile devices, mobile cloud computing (MCC) [1] is introduced.

Mobile cloud computing provides services by bringing abundant resources in cloud computing [2] to the proximity of mobile devices to improve the mobile applications performance and conserve the battery lifetime. One of the techniques adopted in mobile cloud computing is code offloading [3]. It identifies the computation intensive code of a mobile program, and offloads the task to a cloud service via wireless networks. In the concept of code offloading, cloud resources used for offloading have many different types. First and the most common resource is public cloud services, such as Amazon, Google, and Microsoft Azure, which provide pay-as-you-go services over the Internet. Second, a nearby server named cloudlet [4] is considered as a cloud resource with fast network connection as well as powerful processors. Cloudlet serves as a middle layer between mobile devices and public cloud services to reduce the network delay and accelerates the computing. Last but not the least, a local mobile device ad-hoc network forming a device cloud [5] is another potential cloud resource, especially when there is no access to the Internet.

Code offloading in MCC has been comprehensively studied during the past few years [6], [7], [8], [9], [10], [11]. These works mainly focus on the code partitioning and offloading techniques, assuming a stable network connection and sufficient bandwidth. However, the assumption is rather unrealistic. As in reality, the context of a mobile device, e.g., network conditions and locations, changes continuously as it moves throughout the day. For example, the network connection can be unavailable or the signal strength is low. Since a mobile device usually has multiple wireless mediums, such as WiFi, cellular networks and Bluetooth, and each connection performs differently in terms of speed and energy consumption, the strategy of utilizing wireless interfaces can significantly impact the performance of the mobile cloud system as well as the user experience. Moreover, as we described above, there are multiple options of cloud resources that can be selected for code offloading under different conditions. As an example, in case the Internet connection is inaccessible, a group of mobile device users can still configure a code offloading service by setting up a wireless mobile ad-hoc network. Alternatively, a mobile device user can also connect to a nearby cloudlet to

• *B. Zhou, A.V. Dastjerdi, R.N. Calheiros, and R. Buyya are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratary, Department of Computing and Information Systems, University of Melbourne, Australia. E-mail: bowenz@student.unimelb.edu.au, {amir.vahid, rnc, rbuyya}@unimelb.edu.au.*
• *S.N. Srirama is with the Mobile Cloud Lab, Institute of Computer Science, University of Tartu, Estonia. E-mail: satish.srirama@ut.ee.*

Manuscript received 9 Aug. 2015; revised 9 Nov. 2015; accepted 14 Dec. 2015. Date of publication 22 Dec. 2015; date of current version 6 Oct. 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2015.2511002

outsource the computation intensive mobile tasks when it is infeasible to use mobile data. The heterogeneity of MCC has not been rigorously studied in the literature as previous works only target the public cloud service.

To tackle the issues mentioned above and improve the service performance in mobile cloud computing, we propose a context-aware MCC system, called mCloud, that takes the advantages of the changing context of a mobile device and multiple cloud resources to provide an adaptive and seamless mobile code offloading service. The objective of mCloud is to derive offloading decisions under the context of the mobile device and cloud resources to provide better performance and less battery consumption. This paper is a significant extension of our previous work [12]. The **new contributions** reported in this paper are as follows.

- The related work is updated with some state-of-the-art works for mobile cloud code offloading.
- We redesign the system architecture by investigating and adding the failure detection and recovery module, based on the nature of mobile cloud environment and our proposed framework.
- The process of realizing mCloud offloading framework has been explained in a new design and implementation section. This includes the necessary technologies and approaches adopted in the system.
- The Performance Evaluation section is updated with new set of experiments. The new experiments, which incorporated our updated algorithms (equipped with failure recovery feature), consider multiple mobile cloud resource availability scenarios to capture the dynamic changes in the mobile cloud environment. In addition, these experiments are also conducted under several network condition scenarios to consider the unstable nature of the mobile cloud network. Moreover, the new experiments used heterogeneous pool of public cloud, cloudlet and MANET resources.

The remainder of this paper is organized as follows. We first discuss existing offloading approaches and frameworks for MCC in Section 2, and present an insight of mCloud architecture in Section 3. Then we introduce the system models and propose the context-aware offloading algorithm in Section 4. The approaches for system implementation are introduced in Section 5, followed by a discussion on the system evaluation and numerical results in Section 6. Finally in Section 7, we conclude and propose the future work.

2 Related Work

Frameworks and architectures for code offloading in MCC have been comprehensively studied in previous works.

Flinn et al. [13] proposed a remote execution system called *Spectra* that provides offloading through a set of pre-defined APIs via *Remote Procedure Call*. Similarly, *Chroma* [14] handles offloading with an approach called *tactics*, which are defined as modules of remote calls specified by developers. The application consists of different combinations of the *tactics*. Both of the proposed systems need developers to specify the offloading codes statically, which is considered non-trivial and lack of flexibility. Hence, dynamic offloading schemes have emerged. Cuervo et al. [3] proposed a code offloading framework called *MAUI* that provides method level, energy-aware mobile application offloading for .NET applications. MAUI enables developers to annotate methods and fetches information from a set of profilers to make decisions dynamically on whether to offload. *ThinkAir* [8] enables Android mobile applications to offload the computation intensive jobs to their mobile clones running on the public cloud on the method level. Flores et al. [15] presented an evidence-based offloading framework *EMCO* that extracts knowledge from code offloading traces by applying machine learning algorithms to enhance the decision making process. However it is still in its prototype that needs to be comprehensively studied. These works have focused on offloading to public cloud or a server nearby, which may not be reliable under condition changes, such as cloud availability or connection losses.

Later, several works are proposed considering other types of resources for offloading. Bahl et al. [16] discussed the idea of accelerating the mobile cloud processing time by adding a middle layer called *Cloudlet*. Rahimi et al. [17] proposed a three-tier offloading framework consisting of mobile device, local cloud, and public cloud, with the consideration of mobile device mobility. Xia et al. [18] proposed an online location-aware algorithm in a two-tier MCC environment consisting of local cloudlet and remote cloud service. The objective of the proposed system is to provide equal energy consumption proportion on each local cloudlet. Chen et al. [19] proposed an architecture consisting of wearable devices, mobile devices and cloud for code offloading. It realized the opportunity to run heavy computation applications on wearable devices, by offloading part of the workload onto mobile devices or remote cloud. These existing works investigate the benefits of introducing other types of computing resources than cloud into MCC. However, they have not investigated the impact of wireless medium selection on the offloading performance.

Furthermore, the accuracy of the offloading decision making algorithm can significantly affect the Quality-of-Service of an MCC system. Chen et al. [20] proposed a semi-markovian decision process based approach to decide which part of the program to be offloaded in order to optimize the execution time and energy consumption. The simulation assumes a stable network condition that is rather infeasible in practice. Chen [21] formulated the decision making problem as an offloading game, and provided an offloading scheme that can achieve a Nash equilibrium of the game. Hung et al. [22] proposed a decision making approach by caching the previous offloading plan in the profiles to reduce the decision making overhead. Lin et al. [23] proposed a location based context-aware decision engine for code offloading. The decision engine takes into consideration the geographical location and the time-of-the-day as the history data to make the offloading decisions. All these works have only considered public cloud as their offloading location, and are lack of comprehensive studies on multiple criteria in the device context.

In summary, comparing to existing works, we aim to improve the MCC's performance and service availability in an unstable mobile cloud environment by proposing a new framework that considers different offloading destinations (cloudlets, mobile ad-hoc cloud, and cloud resources) and wireless channels (WiFi, 3G, Bluetooth, WiFi-direct). We also proposed a multi-criteria offloading decision making algorithm. This brings the advantages of considering

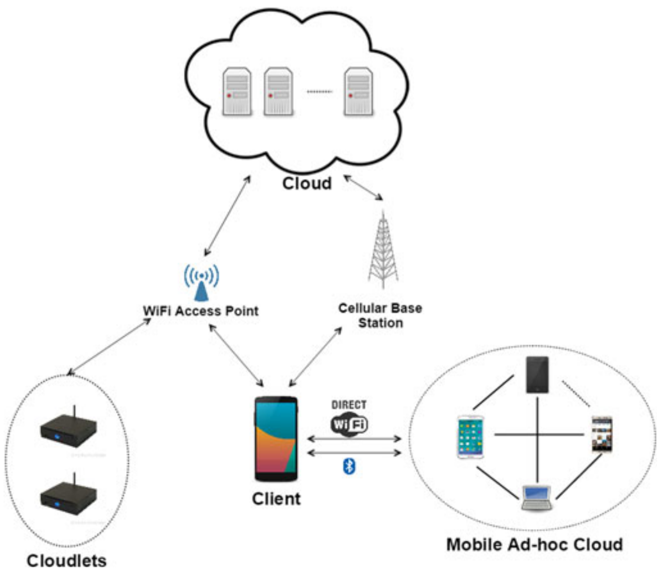


Fig. 1. Overview of mCloud environment.

multiple criteria of energy consumption, execution time reduction, resource availability, network conditions, and user preferences.

3 System Architecture

In this section, we give an insight of mCloud's architecture to address the issue of context aware code offloading in a heterogeneous mobile cloud environment. We first describe an overview of the MCC environment that mCloud fits in. Then, the design of main components is presented in details.

3.1 System Overview

Fig. 1 illustrates the overview of the proposed system for the heterogeneous mobile cloud environment. The device requesting the offloading service is regarded as a *client*. The proposed system leverages three types of mobile cloud resources, namely cloud, cloudlet, and mobile ad-hoc cloud.

First, cloud provides Infrastructure-as-a-Service with scalable computation and storage, which can be connected from mobile clients via WiFi or cellular network. It has the ability to host tasks requiring high computation and communication.

Second, a cloudlet is a local "datacenter in a box", which resembles a cluster of multi-core processors and a high-bandwidth WLAN connection with considerable low power consumption [4]. A task with limited delay-resistance is well-suited in cloudlets.

Third, a mobile ad-hoc cloud that is formed by a group of mobile devices in the client's proximity via short-range communication technologies, e.g., Bluetooth and WiFi Direct.

Due to mobility of the devices, the mobile ad-hoc cloud can be unstable. To ensure the stability of the mobile ad-hoc cloud in the mCloud framework, we adopt a one-hop topology for the mobile ad-hoc network. Moreover, we introduce a fault recovery policy for detecting failures and recovering tasks to obtain the consistency of the results.

3.2 Framework Components

The main components of the mCloud framework belong to two parts: client part and cloud server part. As depicted in Fig. 2, there are five main components on the client side: *Context Monitor*, *Decision Module*, *Task Manager*, *Communication Manager* and *Failure Recovery*. On the cloud server side, main components include *Communication Handler*, *Task Manager* and corresponding mobile cloud infrastructures.

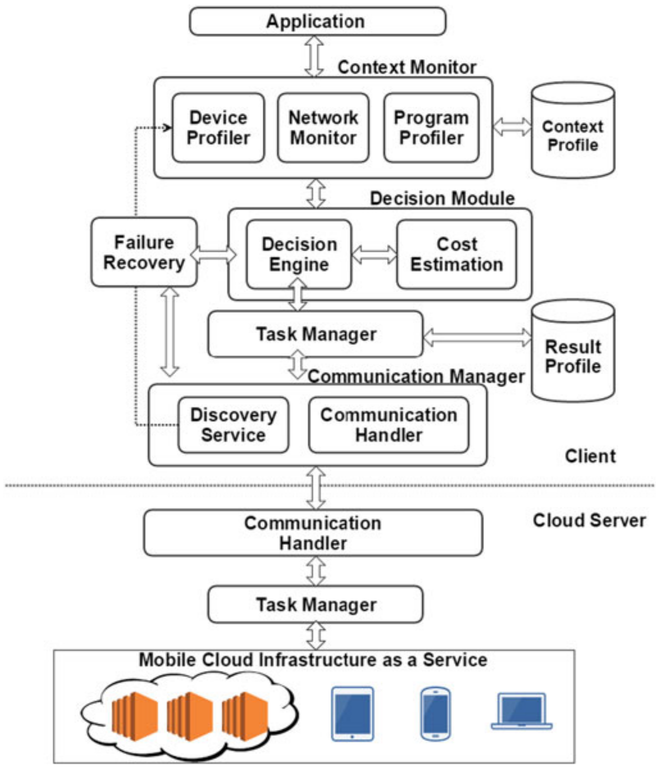


Fig. 2. Main components of the framework.

3.2.1 Context Monitor

The *Context Monitor* offers the context-awareness for the system by profiling multiple context parameters at runtime, and assists the *Decision Engine* when needed. Since the context of the mobile device has significant effect on the decision making accuracy, the system provides three relevant profilers: a program profiler, a device profiler, and a network monitor. However, profiling incurs additional runtime overhead and extra energy consumption. To avoid the high overhead, the profilers adopt the on-demand monitoring strategy, which only fetches context data when the offloadable methods are invoked.

a) *Program profiler*: The program profiler tracks the execution of a program on the method level. The attributes being monitored include:
   - the overall instructions executed,
   - the execution time,
   - the memory allocated,
   - the number of calls of this method,
   - the type of mobile cloud infrastructure resource for the execution (e.g., local, cloud, cloudlet),
   - and the data size of inputs.

The profile is updated at every invocation, and stored in the *Context Profile* database. Later the program profile is passed to *Cost Estimation* module to estimate the execution cost (i.e., running time, energy consumption) for decision making. The details of cost models are discussed in Section 4.2.

b) *Device profiler*: The hardware profiles collected by the profiler represent the operating conditions of the mobile device being monitored. Same as the program profile, the device profile is fed into *Decision Engine* when needed to assist the cost estimation. The profile includes:
   - the average CPU frequency,
   - the average CPU usage,
   - the maximum CPU frequency,
   - and battery level.

c) *Network monitor*: The network monitor collects the network information of the mobile device asynchronously at runtime so that it can record any change in the context. The profile is passed to cost estimation models when needed. The following network conditions are monitored:
   - cell connection state and its bandwidth,
   - WiFi connection state and its bandwidth,
   - Bluetooth state,
   - the congestion level of the connection (RTT) to VMs on the cloud,
   - and the signal strength of cell and WiFi connection.

3.2.2 Decision Module

This component has the responsibility to decide whether and how to offload the mobile task, and dispatch the task to the appropriate mobile cloud infrastructure (i.e., cloud, cloudlet, or MANET) based on the current context. It consists of two main modules: *Cost Estimation* and *Decision Engine*. Based on the context profiles, *Cost Estimation* provides a set of cost estimation models that calculate the execution time of each offloadable task running on three types of mobile cloud infrastructure respectively, with the corresponding energy consumption on the client. *Decision Engine* then applies the cost estimations to the proposed context-aware decision making algorithm to provide the offloading decisions. We give a detailed discussion on the cost models and decision making algorithm in Section 4.

3.2.3 Task Manager

This component works as a middle layer between *Decision Module* and *Communication Manager*. It receives the decision, i.e., method name, offloading location, and network interface, from the upper layer. Then the Manager collects the related information, such as the method inputs, libraries for running the offloaded task, and network address of the offloading location. Last, it persists them into a format called *Task Specs* and passes the information to *Communication Manager*. When receiving the task results, *Task Manager* stores them in the device database.

3.2.4 Communication Manager

The communication manager on both client and server side handles connections between client mobile device and the remote execution in either mobile ad-hoc cloud or remote cloud VMs. It consists of a mobile cloud infrastructure discovery service and a communication handler. Once the decision engine generates the offloading decision, the communication manager takes over the task and executes based on the offloading decisions. Moreover, the *Communication Manager* corporates with *Failure Recovery* service to detect failures and start the recovery process.

a) *Discovery service*: This module is responsible for discovering the mobile cloud infrastructure resources, i.e., the available mobile devices in the MANET, cloudlets, and cloud VMs. The service updates the information of the detected resources, such as network congestion level, IP address, computation capacity, etc., and stores the information in *Device Profiler*. Particularly, for the mobile ad-hoc cloud, the *Discovery Service* detects the available mobile devices in the proximity, forms an mobile ad-hoc cloud, and maintains the network at runtime. Nevertheless, the discovery via network interfaces can potentially incur additional overhead and energy consumption. To avoid the high detection overhead, the *Discovery Service* applies the *periodic detection* strategy, which asynchronously searches for the available devices in certain intervals periodically. We present the detailed implementation of the *Discovery Service* in Section 5.3.

b) *Communication handler*: The Handler operates on both client and mobile cloud infrastructures to handle the communications and data transfer generated in between, which include mobile cloud infrastructure detection, offloading code, state synchronization between client and servers, failure detection, etc. In particular, when offloading a mobile task, the *Communication Handler* on the client serializes the code, related input, and the information of libraries needed to execute the code into the offloading package. Then it dispatches the package to the address provided by *Task Manager*. On the server side, the *Handler* unpacks the package and starts to synchronize the missing libraries described in the package with the client device. Once all the states are synchronised, it passes the deserialized code to the mobile cloud infrastructure for execution, and returns the result to client device upon completion.

3.2.5 Failure Recovery

It works with *Decision Module* and *Communication Manager* to detect the failures and recover the system. The details regarding the types of failures handled by Failure Recovery service, and the detection algorithm are presented in Section 4.4.

4 Cost Estimation Model and Algorithm

We first introduce the system model and the problem formulation. Then we present the details of our cost estimation model and the context-aware offloading algorithm.

4.1 System Model and Problem Formulation

The system considers a heterogeneous mobile cloud environment, consisting of a mobile ad-hoc cloud, the nearby cloudlets and remote public cloud. There are a set of mobile device users that run applications seeking opportunities to offload tasks to the mobile cloud infrastructure.

TABLE 1
Notations

Symbol	Description
t	the mobile task being considered to offload
s	the data size of the offloadable task t that need to be transferred during offloading, in byte
w	the number of instructions for task t to complete
M	a set of mobile device as a mobile ad-hoc cloud
m_k	mobile device k in the mobile ad-hoc cloud
μ	the average CPU speed of the mobile devices or VMs
τ_n	link delay between client and local mobile device cloud
θ	the average CPU usage
r_i	data transferring time for task i
wm_i	the wireless medium used for task offloading
l_i	the execution location of task i
α_1, α_2	weight factors used in the general cost model to adjust the user preference
ρ_d	coefficient of channel energy consumption reflecting on execution performance
$B_{channel}$	the bandwidth of the wireless medium, namely WiFi, 3G, and Bluetooth, in MB/s
$C(t_i)$	general overall cost of task i
$D(t_i)$	execution time of task i running on cloud resource
$E(t_i)$	energy consumption of offloading task i
$\Delta E_{channel}$	the energy consumption of task i under certain bandwidth
$\beta_{channel}$	the estimated channel energy consumption per time unit
β_{tail}	wireless medium tail time energy per time unit
T_{tail}	wireless channel active tail time

4.1.1 Task Modelling

Different mobile applications have different QoS requirements. For example, face detection application requires short processing time while anti-virus applications are usually delay-tolerant. In this model, we model the tasks being offloaded as independent and can be partitioned into subtasks for parallel execution. Thus, let t denote the task generated by the application,

$$t = \langle s, w \rangle. \tag{1}$$

s is the file size of the task, and w denotes the number of instructions of task t to execute.[1] All the symbols used in the models are listed in Table 1.

4.1.2 Mobile Ad-Hoc Cloud Modelling

We apply a one-hop mobile ad-hoc network (MANET) in mCloud to improve the network stability. This is because using multi-hop MANET can cause considerable delay and increase the possibility of node failures. Moreover, when the number of hops is more than two, cloud VMs are the most preferred as they have less communication delay in comparison to MANET [24]. Hence, we only consider a one-hop MANET in mCloud.

Given the one-hop network topology, we assume that the node movement within the signal range does not affect the topology of the MANET and the channel data rate remains the same. Let $M = \{ m_1, m_2, \ldots, m_n \}$ be the set of available mobile devices in the mobile ad-hoc cloud. μ_n denotes the CPU speed of node m_n. τ_n denotes the link congestion level between node m_n and the client device. Then the mobile ad-hoc cloud can be modelled as

$$m_n = \langle \mu_n, \tau_n \rangle, \forall m_n \in M. \tag{2}$$

4.1.3 Cloud and Cloudlet Modelling

As described in Section 3, the client connects to cloudlet via WiFi, and to cloud via WiFi or cellular network. On both cloud and cloudlet, we deploy single-core VMs[2] as the offloading solution. VMs within a cloudlet or cloud are considered homogeneous, while they are heterogeneous across clouds and cloudlets in terms of computation capacity and network delay. Then we model VMs on cloud and cloudlet as follows:

$$v_i = \langle \mu_i, r_i, \theta_i \rangle, \tag{3}$$

where μ_i is the CPU speed, r_i is the network delay from the client to VM, and θ_i is the average CPU usage.

The heterogeneity of VMs between cloudlet and cloud reflects on the different μ, r values in the model.

4.1.4 Problem Formulation

Having presented the models of the system, we formulate the decision making problem as to find a solution of selecting where to execute the task and how to offload so that the overall execution time and energy consumption is the lowest among all the cloud resources in the mobile cloud infrastructure based on the current context of the client device. Specifically, given a set of n tasks T, a set of cloud VMs C, a set of cloudlets CL, and a mobile ad-hoc cloud with h mobile devices M, then the overall cost of executing a set of n tasks is

$$C_{total} = \sum_{i=1}^{n} \Delta C(t_i, l_i, wm_i), \tag{4}$$

where ΔC denotes execution cost of running task t_i, including execution time and energy consumption. l_i represents the execution location for task t_i, which includes local,

1. Our system considers the Android Dalvik bytecode instructions.

2. VMs on cloudlet and public cloud run customized mobile operating systems, e.g., Android x86.

mobile ad-hoc cloud M, cloudlet CL, or cloud C. wm_i is the wireless medium used to offload t_i, including Bluetooth, WiFi, and cellular network. Thus, the problem is to provide an offloading decision $\langle l_i, wm_i \rangle$ for $\forall t_i \in T$ to minimize the overall cost.

4.2 Cost Estimation Models

The cost model consists of two parts, namely the task execution time denoted by D, and the wireless channel energy consumption denoted by E. Then the general model for overall cost of executing tasks t_i is as follows:

$$C(t_i) = \alpha_1 * D(t_i) + \alpha_2 * \rho_d * E(t_i), \tag{5}$$

$$\alpha_1 + \alpha_2 = 1, \alpha_1, \alpha_2 \geq 0, \tag{6}$$

where α_1 and α_2 are weight factors to adjust the portion of time and energy consumption in the overall cost. α_1 and α_2 are considered to capture user preferences on the factors. If the user is not expert, methodologies like AHP can be used to generate the factors from linguistic values provided by users. The value of execution time and energy consumption in the cost model are normalized in a 0-100 scale with the upper and lower bound data profiled from real application results. We adopted ρ_d in our model to represent the effect of hardware settings (DVFS levels and wireless channel rate) on the device performance [20]. ρ_d will be set on a 0-1 scale based on the processor DVFS level and wireless channel rate.

Given the general cost models in Equation (5), we describe the specific cost models for each type of the mobile cloud infrastructure resource.

4.2.1 Mobile Ad-Hoc Cloud Cost Model

First, we model the execution time D for tasks running in the mobile ad-hoc cloud. Given a set of independent tasks $T = \{t_i \mid 1 \leq i \leq n\}$ and a set of heterogeneous mobile devices $M = \{m_k \mid 1 \leq k \leq h\}$, execution time D can be obtained by calculating the earliest finishing time (EFT) among all tasks mapped to the mobile ad-hoc cloud. This independent task scheduling problem is proved to be NP-complete [25]. Therefore the use of heuristics is a suitable approach. For mapping independent tasks to heterogeneous machines, Min-Min heuristic takes less processing time with the result as good as other heuristics [26]. Thus we adopt Min-Min in our cost model.

The Min-min heuristic maps unassigned tasks to available machines. It firstly calculates minimum completion times (MCT):

$$MCT(t_i, m_k) = [min_{1 \leq k \leq h}(CT(t_i, m_k)), \forall t_i \in T], \tag{7}$$

where $CT(t_i, m_k)$ is the completion time for task t_i on device m_k. Then task t_i with MCT is selected and assigned to machine m_k, and t_i is removed from T, and the iterations repeat until all tasks are mapped (i.e., T is empty).

In the next step, we model the energy consumption E of the client device. Based on the results from Min-Min, we calculate the device communication energy cost for transferring data to the MANET and receiving the results. Let $B_{channel}$ denote the channel data rate. $t_i = \langle s_i, w_i \rangle$ represents the task ,where s_i is the data need to be transferred and w_i is the workload. The channel energy consumption for transferring data is given as:

$$\begin{aligned}\Delta E_{channel}(s_i, B_{channel}) &= \beta_{channel} \\ &* \left(\frac{s_i}{B_{channel}} + \frac{s_{result}}{B_{channel}} \right) + \beta_{tail} * T_{tail},\end{aligned} \tag{8}$$

where $\beta_{channel}$ is the power consumption rate related to the transferring time and β_{tail} is the wireless channel tail time power consumption rate.

Let M denote the set of mobile devices in the mobile ad-hoc cloud. μ_k is the CPU speed of device m_k that selected by the MinMin algorithm and θ_k is the average CPU usage. Then the overall execution cost is as follows:

$$\begin{aligned}C_{t_i} = \alpha_1 &\left(\frac{w_i}{\mu_k * \theta_k} + \frac{s_i + s_{i\_result}}{B_{channel}} \right) \\ &+ \alpha_2 * \rho_d * \Delta E_{channel}(s_j, B_{channel}).\end{aligned} \tag{9}$$

4.2.2 Cloud and Cloudlet Cost Models

Let $t_i = \langle w_i, s_i \rangle$ denote a mobile task. There are m VMs available on cloud or cloudlet. μ_{VM} denotes the computing capacity, θ_{VM} denotes the average usage of the VM, and l_{VM} denotes the network latency of the VM. Note that the tasks can be partitioned into subtasks, and VMs on cloud or cloudlet are homogeneous. Hence each task can be evenly partitioned and processed among the machines based on the number of available VMs. Then the cost of running task t_i can be modelled as follows:

$$\begin{aligned}C_{t_i} = \alpha_1 \cdot &\left(\frac{w_i}{m \cdot \mu_{VM} \cdot \theta_{VM}} + \frac{s_i + s_{i\_result}}{B_{channel}} + l_{VM} \right) \\ &+ \alpha_2 \cdot \rho_d \cdot \Delta E_{channel}(s_i, B_{channel}).\end{aligned} \tag{10}$$

This cost model is utilized to estimate the task execution cost on cloud and cloudlet VMs by alternating the parameters $\mu_{VM}, \theta_{VM}, l_{VM}$.

4.2.3 Local Execution Cost

For local cost estimation, we use a history data strategy to reduce the overhead. The local execution time of a method and energy consumption incurred by the device is stored in the database and applied later to the general cost model in Equation (5) for comparison. The estimation costs are then used as the input of the decision making algorithm.

4.3 Context-Aware Decision Making Algorithm

Having shown how to estimate the task execution time and energy consumption with the cost models, we present the algorithm in this section. In order to obtain the lowest execution cost for the offloadable tasks under the context, based on Equation (9) and (10), the context-aware decision algorithm considers a set of context parameters, multiple wireless medium and mobile cloud infrastructure resources to decide when it is beneficial to offload, which wireless medium is used for offloading and which resources to use as the offloading location.

4.3.1 Wireless Medium Selection

Most existing offloading frameworks in the literature only consider network speed and energy consumption when they make offloading decisions. Unlike those, mCloud focuses on utilizing multiple types of mobile cloud resources (i.e., cloud, cloudlet and mobile ad-hoc cloud) and wireless mediums based on device context to improve the offloading service availability and performance. Multiple criteria regarding device context including resource availability, wireless medium availability, network congestion, cost, energy consumption, etc. have been considered for making offloading decisions in mCloud. Therefore, we need a multi-criteria decision making approach (MCDM) in the proposed framework.

Among MCDMs, we apply Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) [27] for wireless medium selection considering abovementioned criteria. TOPSIS offers lightweight processing and shorter response time comparing to other MCDMs [28]. It helps reduce the overhead of the proposed offloading decision making algorithm, considering it is running on mobile devices. Moreover, TOPSIS can be easily modified to consider more criteria if necessary, and the complexity remains the same regardless of the number of criteria. In mCloud, the decision making algorithm considers six criteria related to performance when selecting the wireless interface:

- energy cost of the channel,
- the link speed of the channel,
- the availability of the interface,
- monetary cost (i.e., cost when using mobile data),
- the congestion level of the channel (RTT),
- and the link quality of the channel (signal strength).

Note that for the monetary cost, the algorithm only considers the cost generated by using the mobile data. Other cost such as cloud VM reservation is negligible from the mobile device's perspective as a mobile task generally occupies a negligible time comparing to the cloud VM lifetime.

For the alternatives, the algorithm considers Bluetooth, WiFi, and 3G in this system, but more interfaces can be added if new techniques emerge. Then the process of wireless interface selection is as follows.

First, the relative weights for criteria being considered in TOPSIS are obtained by using analytic hierarchy process (AHP) [29]. The pairwise comparison results are presented in a matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{16} \\ a_{21} & a_{22} & \cdots & a_{26} \\ \vdots & \vdots & \ddots & \vdots \\ a_{61} & a_{62} & \cdots & a_{66} \end{bmatrix}, a_{nn} = 1. a_{mn} = \frac{1}{a_{nm}}. \tag{11}$$

The pairwise comparisons of six criteria are generated based on the standardized comparison scale of nine levels shown in Table 2. Then TOPSIS uses matrix A to calculate the weights of the criteria by obtaining the eigenvector ω related to the largest eigenvalue λ_{max}

$$A\omega = \lambda_{max}\omega. \tag{12}$$

Since the output of AHP is strictly related to the consistency of the pairwise comparison, it is necessary to calculate the consistency index [30]:

TABLE 2
Importance Scale and Definition

Definition	Intensity of importance
Equally important	1
Moderately more important	3
Strongly more important	5
Very strongly more important	7
Extremely more important	9
Intermediate	2, 4, 6, 8

$$\begin{aligned} CI &= \frac{\lambda_{max} - n}{(n-1)}, \\ CR &= \frac{CI}{(n-1) * RamdomIndex}. \end{aligned} \tag{13}$$

CR, which is the consistency ratio of CI, should be less than 0.1 to have a valid relative weight output.

After the weights are generated, a evolution matrix consisting of three alternatives and six criteria is created, denoted by $M = (x_{mn})_{3\times6}$. The values of the criteria are collected at runtime by the *Context Monitor*, and normalized using Equation (14):

$$N_{M_{mn}} = \frac{M_{mn}}{\sum_{n=1}^{6} M_{mn}^2}. \tag{14}$$

Then the weights obtained from AHP method are applied to the normalized matrix $N = (t_{mn})_{3\times6}$

$$M_w = \omega_n * N, \tag{15}$$

where ω_n represents the weight. The best solution and the worst solution are then calculated from the weighted matrix M_w, denoted by $S^+ = \{\langle \min(t_{mn}|m = 1\ldots6)|n \in J^-\rangle, \langle \max(t_{mn}|m = 1\ldots6)|n \in J^+\rangle\}$ and $S^- = \{\langle \max(t_{mn}|m = 1\ldots6)|n \in J^-\rangle, \langle \min(t_{mn}|m = 1\ldots6)|n \in J^+\rangle\}$ respectively, where J^+ is the positive criteria to the cost and J^- is the negative criteria to the cost.

At last, the wireless medium is selected by calculating the Euclidean distance between each alternative and the best and worst solution D_m^+ and D_m^- respectively. and ranking the alternatives by applying a closeness score to the best solution,

$$\begin{aligned} D_m^+ &= \sqrt{\sum_{n=1}^{6}(t_{mn} - t_{mn}^+)^2}, \\ D_m^- &= \sqrt{\sum_{n=1}^{6}(t_{mn} - t_{mn}^-)^2}. \end{aligned} \tag{16}$$

Then rank the alternatives by applying a closeness score R_m to the best solution. The alternative with the highest R_m is selected as the output of the wireless selection algorithm

$$R_m = \frac{D_m^-}{D_m^+ + D_m^-}. \tag{17}$$

However, one drawback of TOPSIS is its sensitiveness to rank reversal, thus in our system, if a new alternative appears, the algorithm will be triggered to generate the new weights and related matrix.

4.3.2 Decision Making

The context-aware decision making algorithm is composed of two main phases, which are summarized in Algorithm 1. The first phase is *estimation* phase (step 2-11), during which the context parameters such as context profiles and wireless interface states are collected from the corresponding modules. Then the cost estimation models calculates the execution cost for each offloading request. In the *selection* phase (step 12-40), the algorithm gives offloading decision based on the available wireless interfaces and the cost estimations. In case there are multiple wireless interfaces available, the algorithm applies TOPSIS model to select the best interface under current context such as data rate, workload size to obtain the best data transfer performance as well as minimum energy consumption. Based on the wireless interface selection result, the algorithm selects the offloading location that has the lowest execution cost. Finally, the algorithm returns the decision pair of $\langle offload\ location, wireless\ medium\rangle$.

Algorithm 1. Context-Aware Decision Algorithm

```
1:  procedure GetDecision $context$,$tasks$
2:      $para[] \leftarrow context$
3:      $task[] \leftarrow tasks$
4:      $programProfile \leftarrow get\ method\ profile$
5:      $start\ discovery\ service\ and\ gather\ resources\ profiles$
6:      $local\_cost \leftarrow estimate\ execution\ cost\ on\ client\ device$
7:      $manet\_cost \leftarrow estimate\ execution\ cost\ on\ mobile\ device$
8:  $cloud\ using\ MinMin\ heuristic$
9:      $cloudlet\_cost \leftarrow estimate\ execution\ cost\ on\ cloudlet$
10:     $cloud\_cost \leftarrow estimate\ execution\ coston\ public\ cloud$
11:     check network interface state
12:     if only cell network is available then
13:         check cloud availability
14:         if cloud is available then
15:             $decision \leftarrow minCost(local, cloud)$
16:             return $decision$
17:         else
18:             return $decision(local\_execution, null)$
19:     else if only WIFI is available then
20:         check cloud, cloudlet and manet availability
21:         $decision \leftarrow minCost(local, cloud, manet, cloudlet)$
22:         return $decision$
23:     else if only Bluetooth is available then
24:         check manet availability
25:         if manet is available then
26:             $decision \leftarrow minCost(local, manet)$
27:             return $decision$
28:         else
29:             return $decision(local\_execution, null)$
30:     else
31:         $interface \leftarrow TOPSIS(context)$
32:         if $interface$ is Wifi then
33:             $decision \leftarrow minCost(local, cloud, manet)$
34:         if $interface$ is 3G then
35:             $decision \leftarrow minCost(local, cloud)$
36:         if $interface$ is Bluetooth then
37:             if manet is available then
38:                 $decision \leftarrow minCost(local, manet)$
39:             else
40:                 return $decision(local\_execution, null)$
```

The complexity of the proposed algorithm is $O(MN\ log\ N)$, where M is the number of devices in mobile ad-hoc cloud and N is the number of tasks. The first phase of the proposed algorithm uses an improved MinMin (in terms of time complexity) [31] (step 7). Each machine maintains a sorted queue of completion time of all tasks on the machine. It takes $O(MN\ logN)$ to construct the queues. In addition, the scheduling takes $O(MN)$ to compare the head of each queue at each scheduling iteration. Thus, the overall complexity is $O(MN\ logN)$. Next, for cloud and cloudlet cost estimation (step 9-10), the time complexity is $O(N)$. Therefore, the time complexity of the first phase in the proposed algorithm is $O(MN\ logN + N) = O(MN\ logN)$. The second phase of the algorithm generates the offloading decision with the time complexity of $O(1)$. Therefore, the time complexity for the proposed algorithm is $O(MN\ logN)$.

4.4 Failure Recovery

The mobility of mobile devices can incur node failures in the proposed mobile cloud environment, especially the mobile ad-hoc cloud. Although mCloud only considers a one-hop network topology, it is necessary to ensure the consistency of results once the failure occurs. Therefore, we apply a pair of failure detection and recovery policy. The failures considered by the policy include remote nodes crash, remote nodes out of communication range, and message omissions.

4.4.1 Failure Detection

The system adopts checkpointing to detect remote node crash and communication lost in Algorithm 2.

Algorithm 2. Failure Detection Policy

```
1:  procedure Detection
2:      $count[] \leftarrow initialized$
3:      $state[] \leftarrow CONNECTED$
4:      $info[] \leftarrow initialized$
5:      for checkpoint i to checkpoint n until target time do
6:          $Send\ out\ confirming\ message\ to\ each\ node$
7:          $counter \leftarrow counter + 1$
8:          for all node i in count do
9:              if Confirmation message received then
10:                 $count[i] \leftarrow count[i] + 1$
11:                 $fetch\ the\ task\ running\ states\ on\ node\ i$
12:                 $info[i] \leftarrow running\ states$
13:         for $node\ i \in count[]$ do
14:             $gap \leftarrow counter - count[i]$
15:             if $gap > \lambda_d$ then
16:                 $state[i] \leftarrow FAILURE$
17:             $gap \leftarrow 0$
18:         $update\ available\ node\ list$
19:         $Recover(state[],\ info[])$
20:     for $node\ i \in state[] == CONNECTED$ do
21:         if no result returned then
22:             $Recover(node\ i,\ info[])$
```

The discovery service inside communication manager periodically broadcasts messages to the remote nodes within the system at each checkpoint and waits for the confirmation message returned by the remote nodes. The discovery service updates a vector that stores the number of confirmation message received from each available node.

Then at the end of each checkpoint period, the discovery service calculates the gap between the number of messages received for each node and its own counter. Nodes with same number as the counter or within the distance of λ_d are considered as working, and are tagged as *CONNECTED*. Nodes that the gap is larger than λ_d are then considered failure, and are tagged as *FAILURE*. Since we only consider one-hop network between the hosting device, MANET devices and cloud VMs, the delivery of the message is not effected by the route change. The failure is mostly likely node crash or dropping out of communication range. In this case, we set λ_d to 3. In case mCloud adopts more complicated network topology in the future, we can adjust the number to fulfil the failure detection requirement.

Moreover, the detecting policy puts the target time on completing the offloading tasks. The target time is related to the estimated execution time provided by the cost models inside decision engine. When the time of checkpoint looping is beyond the target time, the discovery service considers a failure happened in the node that has not returned the result, and proceeds the failure recovery on the tasks from that node.

4.4.2 Recovery

When failures happen, the discovery service fetches the current running states of the tasks on each node at each checkpoint during failure detection. The information includes task ID, the point of failure, and the partial result obtained. If the failure is confirmed by the detection policy, the Discovery Service sends a recovery request to the decision engine. Then the Decision Engine packs all the information of the failed task and choose another available cloud resource node for further execution. If there are no suitable nodes under the current context, the task is then executed locally on the device itself.

5 System Design and Implementation

The implementation of our prototype framework is built based on ThinkAir [8]. The system is implemented on Android operating system. We apply VM migration technology using Java Reflection [32] as our offloading method in mCloud to minimize the modifications of the existing or developing mobile applications. The system and programming APIs are implemented as a library for the Android application developers. Android x86 system[33] is deployed on the cloud and cloudlet VMs. In this section, we explain in details the design and implementation of mCloud and programming APIs.

5.1 Android x86

In order to leverage the cloud resources, the framework needs to offload the mobile tasks from the ARM-based system application to an x86-based cloud VM. Android x86 is an open-source project to port Android operating system to an x86 or AMD host. It provides major functions of an Android device on desktop system like Windows. As a result, our proposed framework can fit into most of the existing and developing Android applications and cloud services without modifications on the program. We deployed the Android x86 system on Virtualbox virtual machines running on commercial public cloud services.

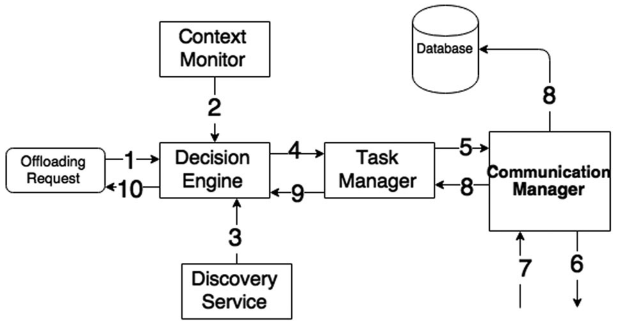


Fig. 3. Execution dataflow: 1) sending offloading request to the decision engine, 2) collecting context parameters from context monitor, 3) get information of available cloud resources, 4) Task Manager starts once the decision is made to offload, 5) Communication Manager divide the jobs into subtasks for parallel processing, 6) Offload to cloud resources for remote execution, 7) pause until receiving the result, 8) aggregate results from parallel processing and store the result of execution time and energy consumption in database, 9) and 10) send result back to device for presentation.

5.2 Offloading Method

We use Java reflection as mCloud's offloading method. Java reflection allows the program to inspect the available Java classes, methods, interfaces and their properties within the system or itself at runtime [34]. We can manipulate programs with certain classes, retrieve related properties like parameter types and fields, and invoke the methods of other programs remotely. Our proposed mCloud framework implements the offloading and remote execution by using Java reflection and Java annotations.

We provide a simple annotation @OFFLOAD for developers to annotate the methods to be considered for offloading. The annotated methods will be processed at the compile time via Java Reflection.Listing 1 shows an example of using Java annotation for offloading. At runtime, the application program can inspect and invoke method *solve* in the offloading example.

Listing 1. Offloading example.

```
public class Example {
  private int expA;
  private int expB;
  @OFFLOAD
  public int solve(){
    return ;
  }
  public void otherMethods(){
  }
}
```

5.3 Execution Environment

The execution dataflow is depicted in Fig. 3. Five main components are implemented, namely Context Monitor, Decision Engine, Task Manager, Communication Manager, and Discovery Service. These components constitute the execution environment for the code offloading tasks.

5.3.1 Discovery Service

The Discovery Service starts when the application is opened on the device. It detects two different types of resources: the available Android x86 VMs in the remote, and mobile devices in the proximity. The Service first contacts a root server on the cloud VM and retrieves a list of available Android x86 VMs and their IP addresses on the public cloud and local cloudlets. Meanwhile, the Discovery Service starts an new thread in the background, which establishes a WiFi hotspot to let the mobile devices nearby connect to the client device. Then the Service can simply ping within a certain IP address range to detect the available mobile devices. The benefit of this approach is that in most cases when the public cloud services are not available, for example at a disaster recovery site, using hotspots to form an ad-hoc network is stable and easy to establish.

The service will search devices in *periodical searching* strategy to keep the available mobile device list updated. Additionally, the Service also considers Bluetooth connection in Discovery Service. It will activate the Bluetooth module on the client device when the service starts at the beginning and initialize a Bluetooth discovery session that detects other Bluetooth capable devices in the range. The hosting mobile device then gets a list of bonded devices and other available devices for connection. All the information of the mobile devices discovered by the system will be stored for the further decision engine evaluation.

5.3.2 Context Monitor

Device profiler, network profiler and program profiler are implemented in the Context Monitor. The Context Monitor is designed to collect context data in an on-demand monitoring strategy in order to reduce the overhead. Hence, the profilers are implemented with BroadcastReceiver[3] to receive the context data only when the context changes. Listing 2 shows an example of the network profiler. The profiler in the example will detect the current active network connection type when it is changed.

Listing 2. Profiler implemented with BroadcastReceiver.

```
public void networkProfiler(){
  networkStateReceiver = new BroadcastReceiver(){
    public void onReceive(){
    netInfo=connectivityManager.getActiveNetworkInfo();
    networkType = netInfo.getTypeName();
    }}}
```

5.3.3 Communication Manager

The communication manager extracts the information of the IP address of the offloading location, offloading task ID, method name, input parameters, and wireless medium type being used generated by task manager. Then it starts a new asyncTask[4] in the background to initialize the connection to the selected cloud resource. The communication manager detects if connected cloud resource has the application files and relevant libraries, and sends missing files. Then the Manager waits until the asyncTask collects the results.

3. An Android class that can receive broadcasts across the applications.

4. An Android class that allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

5.3.4 Decision Engine

Upon receiving the offloading request, the Decision Engine first fetches all the information from Context Monitor and Discovery Service. Then it passes all the context parameters and available device information to the decision algorithm for evaluation. The decision is packed into an array as execution location, machine IP address, offloading method name, input parameters, and wireless channel for communication.

Once the Decision Engine makes the offloading decision, the framework starts the remote execution. Listing 3 gives an example of the remote execution. The remote cloud resource first unpacks and get the parameters regarding the execution environment, such as method name, input parameters, and return types. Then it use Java reflection to create a new instance of the offloaded class. Last, it invokes the annotated method to execute and return the results to the hosting mobile device. During the remote execution, the hosting mobile device is on hold until all the results are returned from the cloud resources.

Listing 3. An example of remote execution

```
private void remoteExecution(InputStream objIn){
  obj = objIn.readObject();
  name = objIn.readObject();
  parameters = objIn.readObject();
  paraTypes = objIn.readObject();
  //Get the class
  class = obj.getClass()
  //construct class in remote host
  constructor = class.getConstructor();
  instance = constructor.newInstance();
  //Get the offloading method
  runMethod = class.getDeclaredMethod(name, paraTypes);
  runMethod.setAccessible (true);
  //invoke the remote method with input parameters
  result = runMethod.invoke ( class, parameters);
}
```

6 Performance Evaluation

6.1 Experiments Settings

To the best we know, there are no available standard testbeds that are suitable for the performance evaluation. Thus, to evaluate the effectiveness of mCloud and offloading scheme on different kinds of mobile applications, we implement two android applications. The applications represent two different types of tasks, one is small file size with high computation, and the other one is big file size with high computation. For the first type, we implement an application that performs math operations based on the input data, and for the second type we implement a face detection application.

We deploy the applications on one HTC G17, one Samsung I997, and one Nexus 5, which form a device cloud for the experiments. One Android x86 clone are installed within VirtualBox on an Intel i5 laptop serving as cloudlet, the emulated CPU speed was adjusted from VirtualBox to match the processing speed of cloudlet. Two Android x86 clones are set up in an Amazon EC2 t2.medium instance. Moreover, we use PowerTutor [35] to monitor the energy consumption of CPU and communication. Unrelated

TABLE 3
Experiment Scenarios

No.	Application	Scenario	User Preference
1	Math application, face detection	stable device context, test performance against baselines	time_sensitive energy_sensitive time_energy
2	Math application	unstable mobile cloud resource availability, stable network (Table 7)	time_energy
3	Math application	stable mobile cloud resource availability, unstable network (Table 8)	time_energy

applications, background services (e.g., GPS, audio, etc.) and screen of the mobile devices are shut down during experiments.

The relative weights for criteria in TOPSIS model is generated based on the system priority. Due to the concern on the processing delay and energy consumption, we assume that the order of priority set for the six criteria under consideration is: resource availability > power consumption > bandwidth > channel congestion level > signal strength > monetary cost. Based on this assumption we calculate the weights from AHP and the results are shown in Table 4. The consistency ratio[5] value is 0.052 (less than 0.1), thus the weights are valid.

We conducted three sets of experiments. A summary of the scenarios is listed in Table 3. In the first scenario, two applications are executed in a stable device context (i.e., all mobile cloud resources and wireless mediums are available and stable) to evaluate the performance of mCloud in terms of time and energy consumption, under three user preference policies. Then we compare them with the baselines of *local_only*. In the second scenario, we conduct experiments under multiple cases of mobile cloud resource availability, while in the third test scenario, mCloud is tested under unstable network conditions. These results are then compared with the existing work ThinkAir. In summary, the second and third set of experiments aim to demonstrate the advantages of mCloud in an unstable mobile cloud context.

6.2 Results and Analysis

We run the two implemented mobile applications with 500 input tasks respectively under three offloading policies, namely *time_sensitive*, *energy_sensitive*, and *time_energy*. Then the results are compared with the baselines that runs workload in offloading policy *local_only*. The characteristics of the generated workloads is listed in Table 5. Workload S_L and S_H are generated by the calculation application, and workload B_H is generated by the face detection application. Each measurement result is calculated by the average of 10 trails. Figs. 4a and 4b compare the execution time and energy consumption respectively of each workload under three offloading policies, which are time sensitive, energy sensitive and time energy combination. Table 6 lists the proportion of the tasks allocated to the multiple cloud resources in mCloud's mobile cloud environment.

5. Calculated by Equation (13).

TABLE 4
Criteria Weights for Topsis

Criteria	Weight	CR
Power Consumption	0.180	
Bandwidth	0.130	
Cloud Resource Availability	0.514	0.052
Congestion Level	0.081	
Signal Strength	0.062	
Cost	0.033	

As shown in Fig. 4a, for workload S_L, the execution time is reduced by around 55 percent under *time_sensitive* policy comparing to *local_only* policy. 75.4 percent of the tasks are scheduled by the decision engine to the cloudlet server (shown in Table 6) that has a much lower network latency than public cloud. In Fig. 4b, the energy consumption for workload S_L is reduced by 55.6 percent under *energy_sensitive* policy, which has the best performance among all policies. 84.2 percent of the tasks are scheduled to the MANET due to the low energy consumption on data transferring via Bluetooth. For the *time_energy* policy, the result shows in Table 6 that 9.6 percent tasks are executed in local, 70.4 percent tasks in cloudlet, and 20 percent in public cloud, with the consideration of network condition and available cloud resources.

We can also observe similar results that, for workload S_H, mCloud gives the best performance by achieving 65 percent of time reduction under *time_sensitive* policy and 70 percent of energy reduction under *energy_sensitive* policy. For workload B_H, mCloud achieves 25 percent of time reduction and 30 percent of energy reduction on average. The experiment results show the mCloud is most beneficial to the tasks that have low data size and high computation.

Figs. 5 and 6 break down time and energy consumption of workload S_H and B_H respectively. For workload S_H in Fig. 5a, the offloading overhead under different policies is around 20 percent on average, while the offloading overhead of workload B_H in Fig. 6a is much larger. The difference of offloading overhead between these two workloads is due to the time consumed in transferring the data. The energy consumption of communication is fairly small since the workload S_H includes tasks with small data sizes (Fig. 5b). On the contrary, workload B_H contains tasks with large data sizes that increase the energy consumption of transferring the data (Fig. 6b). The communication of workload S_H costs around 13.6 percent of the total energy consumption on average, while for workload B_H it costs more than 30 percent on average.

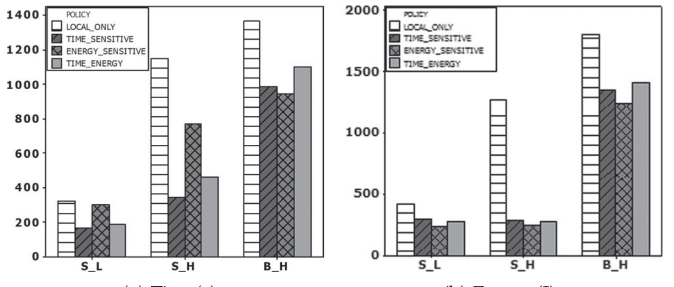
Fig. 6a shows the overall time under *time_sensitive* is greater than *energy_sensitive*, while the execution time under

TABLE 5
Workload for the Experiments

Workload	Average data size(byte)	Average Android bytecode instructions (MI)	Number of tasks
S_L	725	5.8	500
S_H	650	24	500
B_H	3,000	29.5	500

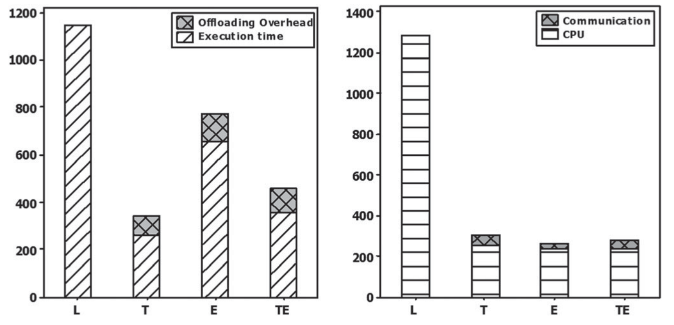
TABLE 6
Proportion of Tasks Mapped in Each Location Under Different Policies (T: *time_sensitive*, E: *energy_sensitive*, TE: *time_energy*)

Workload	Policy	Local	Manet	Cloudlet	cloud
S_L	T	1	0	75.4	23.6
	E	15.8	84.2	0	0
	TE	9.6	0	70.4	20
S_H	T	0	0	31.6	68.4
	E	0	82.1	17.9	0
	TE	0	0	79.3	20.7
B_H	T	41.2	0	58.8	0
	E	60.8	39.2	0	0
	TE	33.5	12.6	53.9	0



(a) Time(s) (b) Energy(J)

Fig. 4. Overall time and energy consumption for each workload under different policies.

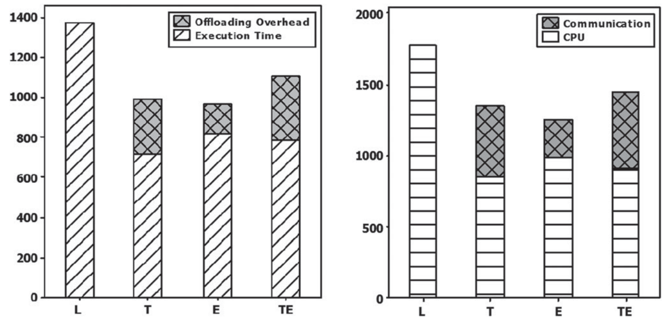


(a) Time(s) (b) Energy(J)

Fig. 5. Task processing time and offloading overhead of workload S_H under local_only(L), time_sensitive(T), energy_sensitive(E) and time_energy(TE) policies.

time_sensitive is smaller. This is because the tasks under *time_sensitive* policy were scheduled among local and cloudlets via a WiFi public access point on our site with long latency, and tasks under *energy_sensitive* policy were scheduled among local and Manet via Bluetooth and WiFi-direct, which has almost no network latency. Consequently, for workload B_H that has large data size to transfer, although the task execution time is lower, more time spent on waiting for results and transmission under *time_sensitive* policy. The overall time will be shorter under *time_sensitive* when using a lower latency access point.

Furthermore, to explore the performance of mCloud in terms of execution time and energy consumption under unstable contexts, we conduct the experiment using workload S_H with different combinations of available cloud resources and network conditions under time and

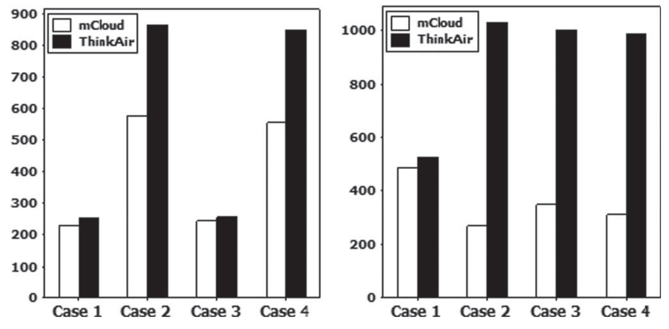


(a) Time(s) (b) Energy(J)

Fig. 6. Task processing time and offloading overhead of workload B_H under different policies.

TABLE 7
Number of Each Type of Cloud Resources

Case	Cloud VM	Cloudlet	MANET
1	2	2	3
2	0	2	3
3	2	2	0
4	0	0	3



(a) Time(s) (b) Energy(J)

Fig. 7. Performance under available resource changing conditions.

energy combined policy, and compare performance with ThinkAir.

First, we evaluate the performances in different available cloud resources conditions while the network condition is stable. The test cases are listed in Table 7.

Case 1 indicates all resources are available while case 2 to case 4 represent the absence of public cloud service, cloudlet service, and nearby wireless mobile ad-hoc network respectively. The results are illustrated in Fig. 7.

Figs. 7a and 7b shows the execution time and overall energy consumption of the workload in each case for our proposed system and ThinkAir respectively. In Fig. 7a, it shows that mCloud outperformed ThinkAir in terms of time saving in all the four cases. Especially in case 2 and case 4, due to the unavailability of public resources, ThinkAir chooses to run all the tasks locally on the device, while mCloud offloaded part of the tasks to the mobile clones on cloudlet and the nearby mobile device cloud, which conserved around one third of the time ThinkAir took for execution. The similar result can be observed for energy consumption of the two systems in Fig. 7b. The results show that our proposed system can provide code offloading services when the mobile cloud environment is short of public cloud

TABLE 8
Average Network Speed for Each Test Case

Case	Wifi (MB/s)	3G (MB/s)	Bluetooth (MB/s)
1	14.3	3.5	2.8
2	0	3.2	3.0
3	0	0	2.2
4	1.7	3.3	2.5

resources and help conserving execution time and battery, unlike many existing mobile cloud frameworks.

Second, we compare mCloud with ThinkAir under unstable network condition. The experiments are conducted by executing workload B_H under changing network context. We alter the bandwidth of Internet connection and 3G connection to stimulate the changing network condition in the real world. Table 8 lists the four test cases. The maximum bandwidth of our testing devices' WiFi connection was 14.3 MBps, the average 3G connection speed was 3 MB/s. The Bluetooth speed was the same throughout the experiment. When the speed was set to 0, it represents the unavailability of the corresponding wireless channel. In order to alter the bandwidth of the network, we set up a virtual access point on the laptop and connect DummyNet [36] to the virtual AP to manage the network bandwidth, latency, etc.

Case 1 represents the scenario where all the wireless channels are available and operating at full speed. Case 2 represents the scenario where WiFi connection is not available. Case 3 represents neither WiFi nor 3G is available. Case 4 represents the WiFi connection speed drops from full speed to low speed that is slower than 3G and Bluetooth. Figs. 8a and 8b show the execution time and overall energy consumption of the workload in each case for our proposed system and ThinkAir respectively.

As illustrated in Figs. 8a and 8b, test case 1 shows that mCloud has close performance as ThinkAir in terms of execution time and energy consumption. For case 2, 3 and 4, mCloud has around 20 percent performance gain comparing to ThinkAir. For test case 2, when only mobile data and Bluetooth are available, mCloud schedules the offloading tasks to either MANET through Bluetooth or local due to the consideration of monetary cost. The result of test case 4 shows that when the network is unstable or slow, mCloud can save more execution time and energy than ThinkAir because of the multiple cloud resources and context being considered in mCloud.

7 Conclusions and Future Work

We proposed a context-aware offloading decision algorithm that takes into consideration context changes (e.g., network conditions and heterogeneous mobile cloud resource) to provide decisions on wireless medium and mobile cloud infrastructure resources to utilize at runtime. We also provide a general cost estimation model for mobile cloud infrastructure resources to estimate the task execution cost including execution time, energy consumption. The models can be easily modified for the new cloud resources. We then designed and implemented a prototype system considering three types of cloud resources (mobile device cloud, cloudlet and public clouds) and a decision engine that runs the

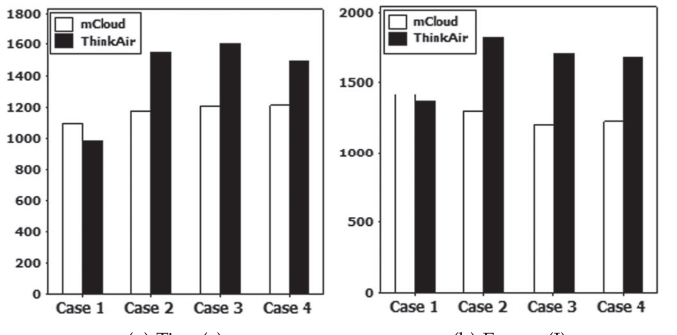


Fig. 8. Performance under available resource changing conditions.

proposed algorithm and related cost estimation models. We presented the evaluation of mCloud, and results showed that the system can provide offloading decisions based on the current context of mobile devices to lower the cost of execution time and energy.

We plan to extend the framework to give the cloud resources ability to intercommunicate with each other, with the handover strategy to perform the failure recovery based on the context changes, so that the current system can be more efficient and reliable.

Moreover, future works can investigate resource management strategies for offloading-as-services cloud providers. Having access to variety of task offloading traces for different cloud instances, providers can utilized machine learning approaches to extract specific offloading policies for offloading tasks in similar conditions.

In addition, the mobility of mobile devices can affect the performance of the mobile ad-hoc cloud. Specifically, the mobile device users may depart and arrive in different patterns and the topology of the mobile cloud environment may change as the result. Therefore, the effect of different user mobility models on system performance need to be investigated in the future.

References

[1] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 369–392, 1st Quarter 2014.

[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[3] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl. Services*, 2010, pp. 49–62.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[5] I. Stojmenovic, *Handbook of Wireless Networks and Mobile Computing*, vol. 27. New York, NY, USA: Wiley, 2003.

[6] M. Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, Mar. 2010, pp. 217–226.

[7] 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.

[8] 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. 31st IEEE Int. Conf. Comput. Commun.*, Mar. 2012, pp. 945–953.

[9] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, “COMET: Code offload by migrating execution transparently,” in *Proc. 10th USENIX Conf. Operating Syst. Des. Implementation*, 2012, pp. 93–106.

[10] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, “Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing,” *Mobile Netw. Appl.*, vol. 16, no. 3, pp. 270–284, 2011.

[11] E. E. Marinelli, “Hyrax: Cloud computing on mobile devices using MapReduce,” DTIC Document, Carnegie Mellon Univ., 2009.

[12] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, “A context sensitive offloading scheme for mobile cloud computing service,” in *Proc. 8th IEEE Int. Conf. Cloud Comput.*, 2015, pp. 869–876.

[13] J. Flinn, S. Y. Park, and M. Satyanarayanan, “Balancing performance, energy, and quality in pervasive computing,” in *Proc. 22nd Int. Conf. Distrib. Comput. Syst.*, 2002, pp. 217–226.

[14] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi, “Tactics-based remote execution for mobile computing,” in *Proc. 1st Int. Conf. Mobile Syst., Appl. Services*, 2003, pp. 273–286.

[15] H. Flores, S. N. Srirama, and R. Buyya, “Computational offloading or data binding? bridging the cloud infrastructure to the proximity of the mobile user,” in *Proc. 2nd IEEE Int. Conf. Mobile Cloud Comput., Services, Eng.*, 2014, pp. 10–18.

[16] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, “Advancing the state of mobile cloud computing,” in *Proc. 3rd ACM Workshop Mobile Cloud Comput. Services*, 2012, pp. 21–28.

[17] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, “MuSIC: Mobility-aware optimal service allocation in mobile cloud computing,” in *Proc. 6th IEEE Int. Conf. Cloud Comput.*, 2013, pp. 75–82.

[18] 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.*, Dec. 2014, pp. 109–116.

[19] Z. Cheng, P. Li, J. Wang, and S. Guo, “Just-in-time code offloading for wearable computing,” *IEEE Trans. Emerging Topics Comput.*, vol. 3, no. 1, pp. 74–83, Mar. 2015.

[20] S. Chen, Y. Wang, and M. Pedram, “A Semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud,” in *Proc. IEEE Global Commun. Conf.*, Dec. 2013, pp. 2885–2890.

[21] X. Chen, “Decentralized computation offloading game for mobile cloud computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, Apr. 1, 2015.

[22] S.-H. Hung, T.-T. Tzeng, G.-D. Wu, and J.-P. Shieh, “A code offloading scheme for big-data processing in android applications,” *Softw.: Practice Experience*, vol. 45, pp. 1087–1101, 2014.

[23] T.-Y. Lin, T.-A. Lin, C.-H. Hsu, and C.-T. King, “Context-aware decision engine for mobile cloud offloading,” in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops*, 2013, pp. 111–116.

[24] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, “Impact of cloudlets on interactive mobile cloud applications,” in *Proc. 16th IEEE Int. Enterprise Distrib. Object Comput. Conf.*, 2012, pp. 123–132.

[25] D. Fernández-Baca, “Allocating modules to processors in a distributed system,” *IEEE Trans. Softw. Eng.*, vol. 15, no. 11, pp. 1427–1436, Nov. 1989.

[26] T. D. Braun, H. J. Siegel, et al., “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems,” *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.

[27] C.-L. Hwang, Y.-J. Lai, and T.-Y. Liu, “A new approach for multiple objective decision making,” *Comput. Operations Res.*, vol. 20, no. 8, pp. 889–899, 1993.

[28] M. Velasquez and P. T. Hester, “An analysis of Multi-criteria decision making methods,” *Int. J. Operations Res.*, vol. 10, no. 2, pp. 56–66, 2013.

[29] T. L. Saaty, *The Analytic Hierarchy Process: Planning, Priority Setting, Resources Allocation* New York, NY, USA: McGraw-Hill, 1980.

[30] H. Wu, Q. Wang, and K. Wolter, “Methods of cloud-path selection for offloading in mobile cloud computing systems,” in *Proc. 4th IEEE Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2012, pp. 443–448.

[31] E. Kartal Tabak, B. Barla Cambazoglu, and C. Aykanat, “Improving the performance of independent task assignment heuristics minmin,maxmin and sufferage,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1244–1256, May 2014.

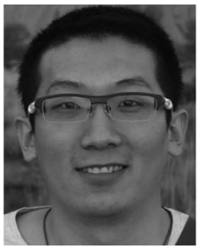
[32] G. McCluskey, “Using java reflection,” *Java Developer Connection*, 1998.

[33] C.-W. Huang, M. Chen, and D. Zavin. (2015). Android-x86 - porting android to x86. [Online]. Available: http://www.android-x86.org/

[34] K. Arnold, J. Gosling, D. Holmes, and D. Holmes, *The Java Programming Language*, vol. 2. Reading, MA, USA: Addison-Wesley, 1996.

[35] L. Zhang, B. Tiwana, R. Dick, Z. Qian, Z. Mao, Z. Wang, and L. Yang, “Accurate online power estimation and automatic battery behavior based power model generation for smartphones,” in *Proc. IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codes. Syst. Synthesis*, Oct. 2010, pp. 105–114.

[36] M. Carbone and L. Rizzo, “Dummynet revisited,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, 2010.



Bowen Zhou received the BS degree from Harbin Institute of Technology, Harbin, China, in 2013. He is currently working toward the PhD degree in the Cloud Computing and Distributed Systems Laboratory, Department of Computing and Information Systems, University of Melbourne. He has been working on the computing augmentation in mobile cloud computing, and task scheduling in mobile cloud systems. He is a student member of the IEEE.

Amir Vahid Dastjerdi is a research fellow with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne. His current research interests include cloud service coordination, scheduling, and resource provisioning using optimization, machine learning, and artificial intelligence techniques. He is a member of the IEEE.

Rodrigo N. Calheiros is a postdoctoral research fellow in the Cloud Computing and Distributed Systems Laboratory (CLOUDS Lab), Department of Computing Information Systems, University of Melbourne, Australia. His research interests include cloud and grid computing and simulation and emulation of distributed systems. He is a member of the IEEE.

Satish Narayana Srirama received the PhD degree in computer science from RWTH Aachen University. He is an associate professor and the head in the Mobile Cloud Lab, Institute of Computer Science, University of Tartu. His current research focuses on mobile web services, cloud computing, mobile cloud, scientific computing, and mobile community support.

Rajkumar Buyya is a professor of computer science and software engineering, future fellow of the Australian Research Council, and the director in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in cloud computing. He has authored over 500 publications and six text books including *Mastering Cloud Computing* published by McGraw Hill, China Machine Press, Morgan Kaufmann, and for Indian, Chinese, and international markets, respectively. He is one of the highly cited authors in computer science and software engineering worldwide. He is a fellow of the IEEE.