# A Context Sensitive Offloading Scheme for Mobile Cloud Computing Service

Bowen Zhou*, Amir Vahid Dastjerdi*, Rodrigo N. Calheiros*, Satish Narayana Srirama†, and Rajkumar Buyya*

*Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
†Mobile Cloud Lab, Institute of Computer Science, University of Tartu, Estonia

***Abstract*—Mobile cloud computing (MCC) has drawn significant research attention as the popularity and capability of mobile devices have been improved in recent years. In this paper, we propose a prototype MCC offloading system that considers multiple cloud resources such as mobile ad-hoc network, cloudlet and public clouds to provide a adaptive MCC service. We propose a context-aware offloading decision algorithm aiming to provide code offloading decisions at runtime on selecting wireless medium and which potential cloud resources as the offloading location based on the device context. We also conduct real experiments on the implemented system to evaluate the performance of the algorithm. Results indicate the system and embedded decision algorithm can select suitable wireless medium and cloud resources based on different context of the mobile devices, and achieve significant performance improvement.**

## I. Introduction

In recent years, mobile devices such as smart phones and tablets have been upgraded into more powerful terminals with faster CPU, substantial memory, and multiple sensors. However, the battery lifetime is still a major concern of the modern mobile devices. From the users' perspective, they need better performance of their mobile devices, which reflects on longer battery life and shorter processing time of any kind of services. To overcome this obstacle, mobile cloud computing [1] is introduced.

Mobile cloud computing (MCC) provides services by bringing the abundant resources in cloud computing [2] to the proximity of mobile devices so as to empower the mobile applications performance and conserve the battery life. One of the techniques adopted in mobile cloud computing is code offloading [3]. It identifies the computing 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 computing services like Amazon, Google and Microsoft Azure that provide pay-as-you-go services over the Internet. Secondly, a nearby server named cloudlet [4] is considered as 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. Third, a local mobile device ad-hoc network forming a device cloud is another potential cloud resource, especially when there is no access to the Internet.

Many works have been done in MCC[5], [6], [7] and [8]. They mainly focus on the code partitioning and offloading techniques, assuming a stable network connection and sufficient bandwidth. However, 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 may not be available or the signal quality is not good. Since a mobile device usually has multiple wireless connections such as WiFi, cellular networks and Bluetooth, and each connection performs differently in terms of speed and energy consumption, the way of utilizing wireless interfaces can significantly impact the performance of the mobile cloud system and user experience. Moreover, as we described above, there are multiple options of cloud resources that can be selected for code offloading under different conditions. For example, when there is no Internet connection available, a group of mobile device users can still perform mobile cloud service by building a local mobile device network, while when a single user who has no access to local mobile device network or Internet but a cloudlet can still outsource the computation-intensive tasks to improve the device performance and save the battery life. This issue has not been rigorously studied in the literature as many works only target the public cloud service as the resource.

To tackle the issues mentioned above and improve the service performance in mobile cloud computing, we propose a context-aware MCC system that takes the advantages of both nearby cloudlet, local mobile device cloud, and public cloud computing services in the remote to provide an adaptive and seamless mobile code offloading service. The objective of the proposed system is to derive an optimal offloading decision under the context of the mobile device and cloud resources to provide better performance and less battery consumption. The **main contributions** of our work are as follows:

- First, we present the design of an MCC architecture with nearby mobile cloud, cloudlet and public cloud VMs.
- Second, we provide cost estimation models for cloud VM, local mobile ad-hoc network, and clone VM running on cloudlet and public cloud, and devise an context-aware

2159-6190/15 $31.00 © 2015 IEEE
DOI 10.1109/CLOUD.2015.119

decision making algorithm taking into consideration the estimation results from the models and the mobile device context to provide offloading policies of where, when and how to offload for the mobile applications. We present implementation and evaluation of the performance of the system and the algorithm. The results show the system can help reduce upto 70% of the execution cost for heavy computation applications.

The remainder of this paper is organized as follows. In Section II, we introduce the design of the system architecture. In Section III, we explain the details of cost estimation model and the context-aware offloading algorithm. In Section IV, we present the implementation of the system and conduct corresponding experiments to evaluate the performance of the system, following by a discussion of related work in section V and finally in section VI we give a conclusion.

## II. System Design

In this section, we explain the design of the system, the cost estimation models and the problem formulation in details.

### A. System Architecture

The proposed framework adopts *client-server communication model*, in which the cloud resources (e.g. mobile device cloud, public cloud) are servers and the mobile device is the client to access the services on servers. On the client side, the framework consists of three components, namely a context monitor, a communication manager and a decision engine. On the server side, it includes a server side communication manager, a program profiler and a task manager. Our system is build on ThinkAir framework[7], which uses Java reflection and annotation to perform the offloading on method level, and ThinkAir also considers scalability of the VMs on the server. The system architecture is shown in Fig.1. In this work, the target cloud resources are mobile devices cloud formed by a wireless ad-hoc network (MANET), and VMs running mobile clone in nearby cloudlet and public cloud. The MANET consists of available mobile devices in proximity that have heterogeneous processing power. It is considered to have limited computing resource and uncertain number of devices. Cloudlet is a local surrogate with considerable processing power and stable connection to the Internet. The public clouds are considered a resource-rich service with homogeneous VM running a customized mobile operating system. The system components design and relations between each component are described below.

*1) Components:*

- **Context Monitor**
  The context monitor profiles multiple context parameters at runtime and passes the values to decision engine. Since the profiling results have significant effect on the decision making accuracy, the system includes a program profiler, a device monitor, and a network monitor.
  **a) Program Profiler**: The program profiler tracks the execution of a program on method level. The information includes 1) the overall instructions executed, 2) the execution time, 3) the memory allocated, 4 )the number of calls of this method and 5) the method execution location(e.g. local, cloud). The profile is updated at every invocation and store in the mobile database. It is used in the cost estimation model for prediction. We will discuss the models in the following section.
  **b) Device Profiler**: The device profiler works with context monitor to collect the hardware information at runtime and then passes it to the estimation model for prediction. The profile includes 1) the average CPU frequency, 2) the average CPU usage 3) memory and its usage and 4) battery level. The profiler gathers the hardware information of all the devices and passes it onto the cost estimation model.
  **c) Network Monitor**: The context monitor collects the context information of the mobile device asynchronously at runtime so that it can record any change in the context. The data is passed to cost estimation models when needed. The context being monitored includes: 1) battery level, 2) cell connection state and its bandwidth, 3) WiFi connection state and its bandwidth, 4) Bluetooth state, 5) the conjunction level of the connection (RTT) to VMs on the cloud, and 6) the signal strength of cell and WiFi connection.

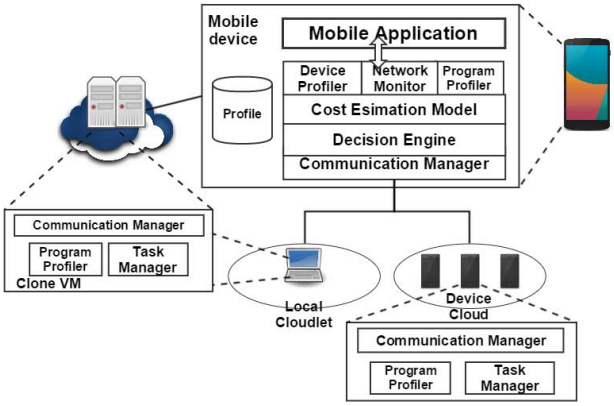


Figure 1. System Architecture

- **Communication Manager**
  The communication manager on both client and server side handles connections between client mobile device and the remote execution in either local mobile device cloud or remote cloud VMs. It consists of a cloud resource discovery service and a communication handler. Once the decision engine produces the offloading policy, the communication manager takes over the task and executes the policy. In case of remote execution failure, the communication manager will fall back and execute the task locally, while reconnecting to the faulty device.
  **a) Discovery Service**: Discovery service at runtime provides the system with the information of MANET and VMs in nearby cloudlets and public clouds. For the cloudlet and public cloud VMs, discovery service updates the availability, network connection quality and

hardware information of the cloud resources periodically. Since we need to reduce the overhead of the decision making process and the conditions of cloudlet and public cloud are more stable, we set the updating time as every one minute. For the MANET, the discovery service is responsible for searching mobile devices in the proximity, forming the ad-hoc network and maintaining the network throughout the application running time. It profiles the device's CPU speed, average CPU utilization and available memory and passes the information to the cost estimation model in the decision engine.

**b) Communication Handler**: Communication handler runs on both the client side and the server side of the system. It provides functions of communication maintaining service between clients and servers, offloaded code execution and dealing with the server scaling request. Particularly, on the client side, the communication handler serializes the offloading code and the remote execution policies, such as how many VMs or devices needed to run the offloading code to the server for execution. On the server side, the communication handler deserializes the request from clients, syncs the state between server operating system and client operating system and starts the execution. Once the communication handler deserializes the request, it checks if the required files and program exist on the server side, if so, it then executes the request and return the result to the server, otherwise, it contacts the client to fetch the file and related libraries for remote execution. In case the server fails to conduct the task, the handler saves the state and passes the task back to the client for further execution.

- **Decision Engine**: This is the core component of the system, consisting of a set of cost estimation models that predict the offloaded tasks' execution cost and a context-aware decision making algorithm to make the decision of when, where and how to offload the task. In details, it takes into account all the data collected from the context monitor and the execution cost estimation from the models and decides at runtime if the annotated method should be offloaded and should it be offloaded to the MANET in proximity, a local cloudlet or a public cloud.
- **Program Profiler (server side)**: The program profiler on the server side tracks the instructions executed and the time taken and passes them back to the client. The information is stored as part of the program profile in the client device database.
- **Task Manager**: The task manager handles the request and executes the offloaded code after the state between client and VM is synced.

## III. Cost Estimation Model and Algorithm

In this section, we first explain the system model and the problem formulation. Then we present the details of our cost prediction model and the context-aware offloading algorithm.

Table I
Notation

| Symbol | Description |
|---|---|
| $t_i$ | the task being offloaded |
| $s_i$ | the data size of the task to be transferred |
| $w_i$ | the number of instructions of the task to complete |
| $v_h$ | virtual machine running on cloud |
| $V$ | available clone VM on cloudlet and public clouds |
| $\mu_{cpu}$ | CPU speed of either device or VM |
| $r_i$ | data transferring time for $task_i$ |
| $\theta_i$ | the average CPU usage of VM$_i$ |
| $M$ | a set of mobile device as a device cloud |
| $m_k$ | mobile device k in the device cloud |
| $P_n$ | CPU speed of mobile device |
| $\tau_n$ | link delay between client and local mobile device cloud |
| $T$ | a set of independent tasks being offloaded |
| $\alpha_1, \alpha_2, \alpha_3$ | weight factors used in the general cost model |
| $\rho_d$ | coefficient of channel energy consumption reflecting on execution performance |
| $B_{channel}$ | wireless channel bandwidth |
| $P(t_i)$ | 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 |
| $\beta_{tail}$ | wireless medium tail time energy per time unit |
| $T_{tail}$ | wireless channel active tail time |

### A. System Model and Problem Formulation

The system considers three tiers, namely the local mobile device cloud, the nearby cloudlet and remote public cloud. These resources are considered as servers in this system. There are a set of mobile device users that run applications seeking opportunities to offload their tasks to the servers. Also, in the system architecture, the mobile device that initiates the offloading requests is considered as the client.

*1) Task modelling:* Different mobile applications have different QoS requirements. Face detection application requires short processing time while anti-virus applications are usually delay-tolerant. In this model, we assume the tasks being offloaded are independent and can be partitioned into subtasks for parallel execution. Thus, let $t_i$ denote the task generated at time i, $s_i$ is the file size of the task, and $c_i$ denotes the number of instructions of task $t_i$ to execute. Then $t_i = \langle s_i, c_i \rangle$. All the symbols used in the models are listed in Table I.

*2) Clone VM modelling:* Since the local cloudlet and VMs in the remote public clouds are similar in terms of the processing power and the offloaded task is executed inside the VM, we model both the cloudlet VM and cloud VM as a general VM provides computing resources with different communication delay. In this work we assume that VMs are homogeneous for analysing. Since in practice the cloud services providers have multiple types of VMs, our model can be easily modified to meet the requirement. Thus, let $V = \{ v_1, v_2, ..., v_h \}$ denote the set of h VMs on the cloud and nearby cloudlet. For each VM $v_i$, $\mu_i$ is the CPU speed, $r_i$ is the round trip time from the client to VM and $\theta_i$ is the average CPU usage.

*3) Mobile device cloud modelling:* The mobile device cloud is formed in a mobile ad-hoc network manner. A mobile ad-hoc network (MANET) is a networking technology that does not rely on a fixed network infrastructure but a rapid

configuration of a temporary network formed by devices themselves [9]. The topology of a MANET is either one-hop network or multi-hop network with protocols. However, using multi-hop MANET can cause considerable delay and unstable network topology and reduce the task completion rate due to the node mobility. we only consider a one-hop MANET in our system to obtain short delay and easy management because the cloud VM outperformed MANET in terms of communication delay when the number of hops is more than 2 [10]. We assume that the devices in the network are heterogeneous in terms of hardware. When considering the node failure caused by the mobility, 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 $M$ = { $m_1$, $m_2$,..., $m_n$ } be the set of mobile devices in the MANET. $p_n$ denotes the CPU speed of node $m_n$. $\tau_n$ denotes the link conjunction level (e.g. RTT) between node $m_n$ and the client device. Then the mobile device cloud can be modelled as an undirected weighted complete graph *G(V, E, P, $\tau$)*, where V is a finite set of vertices representing the mobile nodes in the network, the weight *P* of the vertices is the computing power of that node. *E* is a finite set of edges representing the links between nodes. The weight $\tau$ of the edge represents network delay.

*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 including mobile device cloud (MANET), cloudlet and public clouds under current context of the client device. Specifically, given a set of n tasks $T$ = $\{t_i \mid 1 \leq i \leq n\}$, a set of m cloud VMs $V$ = $\{v_j \mid 1 \leq j \leq m\}$ and a local MANET with h mobile devices $M$ = $\{m_k \mid 1 \leq k \leq h\}$, then the overall cost of executing a set of n tasks is

$$C_{total} = \sum_{i=1}^{n} \Delta E(t_i, l_i, w_i) \tag{1}$$

where $l_i$ represents the execution location for task $t_i$ and $w_i$ is the wireless channel used to offload $t_i$. Thus the problem is to provide a offloading policy of which cloud resource and which wireless channel can be utilized when offloading needed for each task to minimize the overall cost.

### B. Cost Estimation Models

The cost model consists of three parts, namely the task execution time denoted by *D*, wireless channel energy consumption denoted by *E* and monetary cost denoted by *M* when related. Then the total cost of executing task $t_i$ is as follows:

$$P(t_i) = \alpha_1 * D(t_i) + \alpha_2 * \rho_d * E(t_i) \tag{2}$$

where $\alpha_1$, $\alpha_2$ and $\alpha_3$ are weight factors that the system can adjust the portion of the cost to different scenarios. $\rho_d$ is a coefficient reflecting how serious the power consumption on battery life affects the device performance [11].

*1) Mobile device cloud:* On the MANET side, given a set of independent tasks $T$ = $\{t_i \mid 1 \leq i \leq n\}$ and a set of mobile devices $M$ = $\{m_k \mid 1 \leq k \leq h\}$, the objective is to find a task scheduling policy to estimate the earliest finish time (EFT) of *T*. Since the tasks are independent and the workload of each tasks are not uniformed, this problem of optimally mapping tasks to distributed heterogeneous machines is shown to be NP-complete. Thus, we adopt Min-Min heuristic to model the EFT for the set of task *T*. We choose this heuristic because under our task model and computing environment condition, Min-Min gives excellent performance and has a very small execution time [12]. Then the EFT is modelled as follows:

the Min-min heuristic maps unassigned tasks to available machines. It firstly calculates minimum completion times.

$$MCT = [min_{1 \leq k \leq h}(CT(t_i, m_k)), \forall t_i \in T] \tag{3}$$

Then the task that has the smallest MCT is selected and assigned to that machine, and it is removed from *T*, and the iterations repeat until all tasks are mapped (i.e., *T* is empty).

Let $B_{channel}$ denote the channel data rate, $t_i = \langle s_i, w_i \rangle$ denote 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:

$$\begin{aligned} \Delta E_{channel}(s_i, B_{channel}) = \beta_{channel} * (\frac{s_i}{B_{channel}} + \\ \frac{s_{result}}{B_{channel}}) + \beta_{tail} * T_{tail} \end{aligned} \tag{4}$$

where $\beta_{channel}$ is the power consumption rate related to the transferring time and $\beta_{tail}$ is the wireless channel tail time power consumption rate. Let $M_k = \langle \mu_k, r_k \rangle$ denote the device property information where $\mu_i$ is the CPU speed and $r_k$ is the time for transferring data. $T_k = (t_i | 1 \leq i \leq n)$ denotes tasks scheduled to device *k*. Then the overall execution cost on the MANET is as follows:

$$\begin{aligned} C_{MANET} = \alpha_1 \max_{k \in M}(\sum_{i=1}^{n} \frac{w_i}{\mu_k * \theta_k} + r_k) + \\ \alpha_2 * \rho_d \sum_{j=1}^{h} \Delta E_{channel}(s_j, B_{channel}) \end{aligned} \tag{5}$$

where the first half represents EFT and the second half represents the overall data transfer energy consumption.

*2) Cloud VM cost model:* Similar as the MANET cost model, let $T$ = $\{t_i \mid 1 \leq i \leq n\}$ denote a set of *n* independent tasks, $V$ = $\{v_j \mid 1 \leq j \leq m\}$ be a set of remote servers, where $v_i = \langle \mu_{cpu}, \theta_{util}, r_i \rangle, \forall v_i \in V$. $\mu_{cpu}$ is the CPU speed of remote servers, $\theta_{util}$ is the CPU usage, and $r_i$ is the RTT between client and each remote server. Since we assume all the VMs are homogeneous in this model, the tasks are mapped to the servers evenly. The cloud VM cost model is as follows:

$$\begin{aligned} C_{cloud} = \alpha_1 \max_{i \in V}(\frac{\sum_{j=1}^{n} w_j}{m * \mu_i * \theta_i} + \frac{\sum_{j=1}^{n} s_j}{B_{channel}} + \frac{\sum_{j=1}^{n} s_{j\_result}}{B_{channel}}) \\ + \alpha_2 * \rho_d \sum_{j=1}^{h} \Delta E_{channel}(s_j, B_{channel}) \end{aligned} \tag{6}$$

Table II
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 |

where the last part of the model represent the monetary cost of using wireless channel and public cloud services. This depends on the type of services used in practice.

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 for comparison. The estimation costs are then used as the input of the decision making algorithm.

### C. Context-aware Decision Making Algorithm

Having shown how to estimate the task execution time and energy consumption with the related models, we present the algorithm in this section. From Equation (4), (5) and (6), it shows that to optimize the overall cost of the model, the execution time and the data transferring cost both need to be minimized. The context-aware decision algorithm consider a set of context parameters, multiple wireless medium and different offloading cloud resources to decide when it is beneficial to offload, which wireless medium is used for offloading and which cloud resources to use as the offloading location.

*1) Wireless medium selection:* The channel used for offloading data should be chosen carefully based on the different context conditions. An interface selection mechanism is therefore needed to select the best available access network that satisfies QoS requirements at the lowest cost and energy use. In the proposed context-aware offloading algorithm, we apply Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) for wireless medium selection. The criteria weight used in TOPSIS is generated by the analytic hierarchy process (AHP) [4]. TOPSIS provides lightweight online processing cost and short response time, which helps to reduce the overhead of the proposed algorithm. Moreover, the criteria in TOPSIS can be modified and its relative weight can be set to suit different scenarios to make decision algorithm scalable. For the proposed framework, the decision making algorithm considers six criteria related to the performance when selecting the wireless interface, which are 1) energy cost of the channel 2) the link speed of the channel 3) the availability of the interface 4) monetary cost of the channel 5) the conjunction level of the channel (RTT) 6) the link quality of the channel (signal strength). For the alternatives, the algorithm considers Bluetooth, WIFI, and 3G in this system, but more interfaces can be added if new techniques emerge.

First, the relative weights for criteria being considered in TOPSIS are obtained by using AHP. The pairwise comparison results are presented in a matrix.

**Algorithm 1** Context-aware decision algorithm

```
1: procedure GETDECISION(context,tasks)
2:     para[] ← context
3:     task[] ← tasks
4:     programProfile ← get method profile
5:     start discovery service and gather cloud resources profiles
6:     local_cost ← estimate execution cost on client device
7:     manet_cost ← estimate execution cost on mobile device
8: cloud using MinMin heuristic
9:     cloudlet_cost ← estimate execution coston cloudlet
10:    cloud_cost ← 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 ← 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 ← 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 ← minCost(local, manet)
27:            return decision
28:        else
29:            return decision(local_execution, null)
30:    else
31:        interface ← TOPSIS(context)
32:        if interface is Wifi then
33:            decision ← minCost(local, cloud, manet)
34:        if interface is 3G then
35:            decision ← minCost(local, cloud)
36:        if interface is Bluetooth then
37:            if manet is available then
38:                decision ← minCost(local, manet)
39:            else
40:                return decision(local_execution, null)
```

The pairwise comparison of N criteria is generated based on a standardized comparison scale of nine levels shown in Table II. Then TOPSIS uses $A$ to calculate the weights of the criteria by obtaining the eigenvector related to the largest eigenvalue. Since the output of AHP is strictly related to the consistency of the pairwise comparison, we need to calculate the consistency index [13]:

$$CR = \frac{\lambda_{max} - n}{(n-1) * RamdomIndex} \tag{7}$$

the CR value should be less than 0.1 to have an valid relative weights output. Then the process of wireless interface selection is as follows.

After the weights are generated, a evolution matrix consisting of three alternatives and six criteria is created, denoted by $M = (x_{mn})_{3\times 6}$. The values of the criteria are collected at runtime by the Context Monitor , and normalized using Equation 9.

$$N_{M_{mn}} = \frac{M_{mn}}{\sum_{n=1}^{6} M_{mn}^2} \tag{8}$$

Then the weights obtained from AHP method are applied to the normalized matrix $N = (t_{mn})_{3\times 6}$.

$$M_w = \omega_n * N \tag{9}$$

where $\omega_n$ is the weights, which is shown in Table III. The best solution and the worst solution are then calculated from the weighted matrix, denoted by $S^+ = \{\langle \min(t_{mn}|m = 1\dots 6)|n \in J^-\rangle, \langle \max(t_{mn}|m = 1\dots 6)|n \in J^+\rangle\}$ and $S^- = \{\langle \max(t_{mn}|m = 1\dots 6)|n \in J^-\rangle, \langle \min(t_{mn}|m = 1\dots 6)|n \in J^+\rangle\}$, where $J^+$ is the positive criteria to the cost and $J^-$ is the negative criteria to the cost.

At last, TOPSIS chooses the interface 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,

$$R_m = \frac{{D^-}_m}{{D^+}_m + {D^-}_m} \tag{10}$$

to make the selection. 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.

*2) Decision making:* In the proposed decision making algorithm, all the context parameters and profiles are collected from the system components at runtime to provide offloading decision making policies. The algorithm first check the availability of the cloud resources and the wireless interfaces, then the cost models gives the prediction of tasks execution on different cloud resources. If there are multiple wireless interface available, then we apply 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. According to the wireless medium selection result, the algorithm then compares the estimation cost of the corresponding cloud resources and the cloud resource with the lowest cost is chosen as the offloading location. Then the algorithm returns the decision pair of $\langle offloadlocation, wirelessmedium\rangle$. The detail of the proposed context-aware algorithm is given in Algorithm 1. In the next section, we present the evaluation of our proposed cost estimation models and related algorithm performance.

## IV. Performance Evaluation

In this section, we evaluate the performance of our system and algorithm by conducting real experiments on multiple mobile devices. We implement the decision engine, related profilers, context monitor and communication manager into a library on Android operating system, which can be added in Android application for development. Developers can just simply annotate the potential method being considered to offload and our decision engine will automatically provide an offloading policy for this application.

### A. Experiments Settings

To the best we know, there are no available workload benchmarks that are suitable for our system. Thus, to get insights to the effectiveness of the proposed architecture and offloading scheme on different kinds of mobile applications, we implement two android applications for the workload used in the experiments. 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 a calculation application that performs math operations according to the input data, and for the second type we implement a face detection application.

Table III
WORKLOAD FOR THE EXPERIMENTS

| Workload | Average data size(byte) | Average VM instructions (MI) | Number of tasks |
|---|---|---|---|
| S_L | 725 | 5.8 | 500 |
| S_H | 650 | 24 | 500 |
| B_H | 3000 | 29.5 | 500 |

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. We use PowerTutor [14] to monitor the energy consumption. All the mobile devices are running in a standalone environment that unrelated applications and background services are shut down.

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

Table IV
CRITERIA WEIGHTS FOR TOPSIS

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

In first scenario, we conduct two sets of experiments under two different scenarios. In the first scenario, the two applications are executed in a environment that all the wireless mediums available and the connection is stable throughout the experiment, in order to show the performance of the system in terms of energy saving and execution time. And then compare them with the baseline of executing in local only, and cloud only that is performed by plenty of existing work. In the second scenario, the two applications are executed in unstable context condition that the wireless mediums are unstable, to

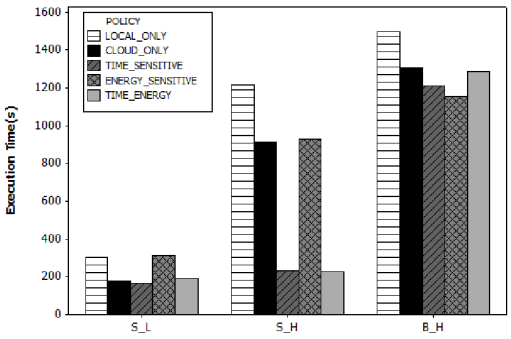


Figure 2. Total execution time for each workload under different policies

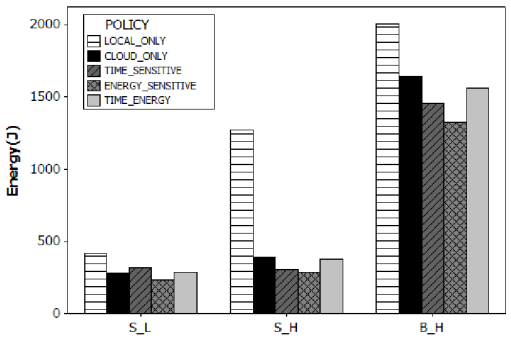


Figure 3. Total energy consumption for each workload under different policies

demonstrate the necessity and advantages of the proposed architecture and context-aware offloading scheme.

### *B. System Evaluation Results and Analysis*

We run the two implemented mobile applications with 500 input tasks respectively under three priority policies, namely time sensitive, energy sensitive, and time energy combination. Then the results are compared with the baseline that runs workload in local only and cloud only. The characteristics of the generated workloads is listed in Table III. 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. Figure 2 and Figure 3 compare the execution time and energy consumption respectively of each workload under three user preference policies, which are time sensitive, energy sensitive and time energy combination, with two baseline policy local only and cloud only. Table V lists the proportion of the tasks allocated to the multiple cloud resources in the proposed mobile cloud environment.

As shown in Figure 2, for workload S_L, the execution time is reduced by 54.9% under time sensitive policy comparing to it under local only policy. It also slightly outperformed the cloud only policy execution time due to the result that 75.4% of the tasks are scheduled by the decision engine to the cloudlet server (shown in Table V) that has a much lower network latency than public cloud. In Figure 3, the energy

Table V
PROPORTION OF TASKS MAPPED IN EACH LOCATION UNDER DIFFERENT POLICIES

| 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 |

Table VI
COMPARISON BETWEEN PROPOSED SYSTEM AND THINKAIR

| System | Execution Time(s) | Energy Consumption (J) |
|---|---|---|
| Proposed | 841 | 552 |
| ThinkAir | 1152 | 902 |

consumption is reduced by 55.6% under energy sensitive policy, which has the best result among all five policies. 84.2% of the tasks are scheduled to the MANET due to the low energy consumption on data transferring via Bluetooth. It also outperformed the policy that only offloads tasks to the cloud. For the time energy policy, the result shows in Table V that 9.6% tasks are executed in local, 70.4% tasks in cloudlet, and 20% in public cloud, with the consideration of network condition and available cloud resources. We can also observe similar results that workload S_H gives the best result under time energy policy by achieving 75% of time reduction and 70% of energy reduction. For workload B_H, the proposed system achieves 25% of time reduction and 30% of energy reduction on average. The experiment results show the proposed system is most beneficial to the tasks that have low data size and high computation. The system can also outperform the cloud only policy that applied by many existing works in terms of time and energy saving in mobile devices by considering the multiple cloud resources and context sensitive decision engine within the proposed architecture.

We take a further step to analyse the performance when the Internet is not available. We conduct the experiment using workload S_H in the unstable network condition that the Internet is unavailable, and we compare the performance with ThinkAir. The results are listed in Table VI. Due to the assistance of mobile device cloud and cloudlets, the proposed system can still provide offloading services when the network connection is not available.

## V. RELATED WORK

Many related works in code offloading for MCC only considered architecture and offloading decision between mobile devices and public cloud services. Kosta et al.[7] proposed a smartphone virtualization framework ThinkAir that provides method-level computation offloading to the cloud. ThinkAir focuses on the offloading between the mobile devices and VM located in public cloud and the scalability of the cloud.

Developers use the program API provided by the framework to annotate the potential method to be offloaded. Flores et al. [15] presented a evidence based offloading framework EMCO that extract knowledge from code offloading traces to enhance the decision making process. Ravi and Peddoju [16] proposed a system architecture that is similar to our work. The system has three cloud resources: MANET, cloudlet and cloud. It provides an offloading decision algorithm to select cloud resources using multi criteria decision making methodology, and a handoff strategy move the tasks between different resource based on the energy consumption and the remaining connection time with the wireless medium. But the decision is made only based on few parameters. It did not fully investigate the influence and relationship between wireless channel and cloud resources. Lin et al. [17] proposed a context-aware decision engine that can work with various mobile cloud offloading system in the literature. The decision engine takes into consideration signal strength, transmission time, geographical location and the time slots of the offloading opportunities [17] to make the offloading decisions. However, it does not consider the multiple cloud resources can be utilized and the wireless medium availability, and the geo-information usually is inaccurate when the device is indoor.

Compared to related works, our system considers the importance of device context such as wireless medium conditions, and the relationship between wireless medium and multiple cloud resources and other device context to make the offloading decisions and provide seamless mobile cloud services.

## VI. Conclusions and Future Work

We proposed a context-aware offloading decision algorithm that takes into consideration context changes such as network condition, device information and the availability of multiple types of cloud resources to provide decisions on the selection of wireless medium to utilize when performing code offloading and the cloud resources to offload at runtime. We also provide a general cost estimation model for cloud 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 design and implement a prototype system considering three types of cloud resources (mobile device cloud, cloudlet and public clouds) and a decision engine that runs the proposed algorithm and related cost estimation models. We presented the evaluation of the proposed system, and results showed that the system can provide suitable offloading decisions based on the current context of mobile devices to provide the offloading services and lower the cost of execution time and energy.

As part of our future work, we plan to explore the intercommunication between different cloud resources, and improve the performance of the prototype system by study handover policies for decision making in terms of device failure tolerance. Also we aims to improve the performance of the decision making algorithm for future work by considering more context parameters, e.g. context of public cloud to provide an optimal solution for code offloading decision making process.

## References


[1] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: taxonomy and open challenges," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 369–392, 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 Generation computer systems*, 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 *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. New York, NY, USA: ACM, 2010.

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

[5] M. Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *Proceedings of 2010 IEEE International Conference on Pervasive Computing and Communications*, March 2010.

[6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*. New York, NY, USA: ACM, 2011.

[7] 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 *Proceedings of 31st IEEE International Conference on Computer Communications*, March 2012.

[8] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "COMET: Code offload by migrating execution transparently," in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*. Hollywood, CA: USENIX, 2012.

[9] G. Aggelou, *Wireless Mesh Networking*. McGraw-Hill Professional, 2008.

[10] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of cloudlets on interactive mobile cloud applications," in *Proceedings of the 2012 IEEE 16th International Enterprise Distributed Object Computing Conference*. IEEE, 2012, pp. 123–132.

[11] 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 *Proceedings of 2013 IEEE Global Communications Conference*, Dec 2013.

[12] 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," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[13] H. Wu, Q. Wang, and K. Wolter, "Methods of cloud-path selection for offloading in mobile cloud computing systems," in *Proceedings of 2012 IEEE 4th International Conference on Cloud Computing Technology and Science*, Dec 2012.

[14] 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. of Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, Oct 2010, pp. 105–114.

[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 *Proceedings of 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE, 2014.

[16] A. Ravi and S. K. Peddoju, "Handoff strategy for improving energy efficiency and cloud service availability for mobile devices," *Wireless Personal Communications*, pp. 1–32, 2014.

[17] T.-Y. Lin, T.-A. Lin, C.-H. Hsu, and C.-T. King, "Context-aware decision engine for mobile cloud offloading," in *Proceedings of 2013 IEEE Wireless Communications and Networking Conference Workshops*. IEEE, 2013.