A new energy-preserving cloud offloading algorithm for smart mobile devices

Samar A. Said, Sameh A. Salem* and Samir G. Sayed

Faculty of Engineering,
Department of Electronics, Communications
and Computer Engineering,
Helwan University,
Cairo, Egypt
Email: samar_said@h-eng.helwan.edu.eg
Email: sameh_salem@h-eng.helwan.edu.eg
Email: s.sayed@ee.ucl.ac.uk
*Corresponding author

Abstract: The advent of mobile devices becomes the way for various technological developments in mobile communication and information technology. However, mobile users expect to access computational intensive applications through resource constrained mobile devices. Consequently the growing demands for boosting the computations, storage and memory resources became essential for mobile devices. A new trend to incorporate mobile devices and cloud resources with the existence of the network connectivity is named as mobile cloud computing (MCC). MCC is the greatest solution to increase the application processing capabilities on mobile devices by migrating the application to the cloud servers with on demand and unlimited resources. This paper proposes a new energy-preserving cloud offloading algorithm. The proposed algorithm estimates the application computational time and uses multiple weighted parameters to give accurate offloading decisions. Simulation results on various applications clarify that the proposed algorithm is capable of estimating an application's computation time with high correlations compared with the real execution time. This improves the offloading decision which actually preserves the energy and reduces the execution time of mobile applications.

Keywords: mobile cloud; offloading decision; cloud computing; energy preserving.

Reference to this paper should be made as follows: Said, S.A., Salem, S.A. and Sayed, S.G. (2019) 'A new energy-preserving cloud offloading algorithm for smart mobile devices', *Int. J. Hybrid Intelligence*, Vol. 1, No. 1, pp.5–22.

Biographical notes: Samar A. Said is a Demonstrator at the Faculty of Engineering, University of Helwan, Cairo, Egypt. She received her Bachelor of Engineering in Communication and Electronic Engineering from the University of Helwan, Egypt in 2010. Her research interests are cloud computing, mobile cloud computing, algorithms and data structure.

Sameh A. Salem earned his a BSc and MSc degrees in Electronics and Communications Engineering from the Helwan University Egypt in May 1998 and October 2003 respectively. In 2008, he received his of PhD degree in Engineering from the Department of Electrical Engineering and Electronics,

The University of Liverpool, UK. His research interests include clustering algorithms, machine learning, data mining, parallel computing, and cloud computing. In 2014, he is promoted to be an Associate Professor and Honorary Research Fellow at The University of Liverpool, UK. Currently, he is the Department Head of Electronics, Communications, and Computer Engineering at the Helwan University, Egypt.

Samir G. Sayed received his BS and MSc degrees from the Department of Electronics and Engineering, Helwan University Egypt in 1996 and 2003 respectively. He received his PhD degree in Electronic and Electrical Engineering from the University College London (UCL), UK in 2010. Since 2014, he is an Honorary Lecturer at the University College London (UCL), UK. Since 2011, he is the Director of the Malware and Reverse Engineering Department in the Egyptian Computer Emergency and Readiness Team (EG-CERT). His research includes cyber security, malware analysis, and wireless networks.

This paper is a revised and expanded version of a paper entitled 'Energy aware mobile cloud computing algorithm for android smartphones' presented at Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2017, Egypt, 9–11 September 2017.

1 Introduction

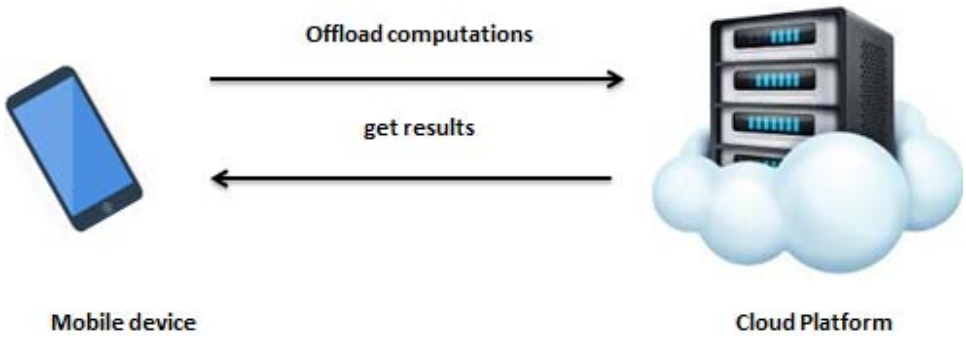
With the huge improvements in the handheld devices as smartphones and tablet computers, the request for these devices to run sophisticated and huge computational applications is growing. These applications such as video processing, object recognition, face recognition, augmented reality, games and natural language processing applications need enormous power and memory. Also, these applications are used in multiple domains as suggested by Zou et al. (2015), Bahrami (2015), Mohammadpour and Tafte (2016), Lo'ai et al. (2016) and Zheng et al. (2016). From the other hand, these applications are not fulfilled by the resource constrained mobile devices due to the limited battery life and processing power. To create the mobile device capable of running these applications with increasing performance uses mobile cloud computing (MCC). MCC is an integration of cloud computing, mobile computing and internet to get huge computational resources for mobile users as proposed by Huang and Wu (2017). One of the prevalent techniques used to enhance the application performance and minimise the energy consumption is a computation offloading. Computation offloading is a procedure in which the intensive computations of the application are migrating from a mobile device to the cloud server as depicted in Figure 1.

The decision for the application execution place is may be locally on mobile device or remotely by offloading the heavy computations to the cloud. Jagtap et al. (2014) suggests that the offloading decision is on basis of the communication technology, application type and the application model used as recommended by. There are many communication technology are used in MCC research as Bluetooth, Wi-Fi, 3G and long term evolution (LTE). Bluetooth used for exchange data through small distance of 10 m range and the application models in Nkosi and Mekuria (2011), Doolan et al. (2008) and Kemp et al. (2010) use the Bluetooth communication technology. Wi-Fi communication technology is faster and provides higher bandwidth. It supports additional devices as compared with

Bluetooth. It works in 100 m range. Kemp et al. (2010), Cuervo et al. (2010), Chun et al. (2011), Satyanarayanan et al. (2009), Huerta-Canepa and Lee (2010) and Wang et al. (2013) use the Wi-Fi communication technology in their application model. Third generation mobile communication (3G) is provide lower bandwidth and have more delay than Wi-Fi communication technology. Previous researches as Kemp et al. (2010), Cuervo et al. (2010), Chun et al. (2011), Altamimi et al. (2012) and Kosta et al. (2012) use 3G communication technology. LTE supports high throughput from 15 Mbps to 100 Mbps with high mobility range and supports higher capacity for mobile users with respect to 3G. Lin et al. (2013) uses LTE communication technology. Consequently, the computation offloading decision can be influenced by the network cost, delay and bandwidth. Furthermore, the nature application is an important metric for offloading decision. There are many applications cannot be offload it because the application access hardware resource as GPS, sensors and camera. As similar, when the application codes build the user interface or handle interaction with a user. But, there are many applications effective in MCC which have intensive computations as gaming applications suggested by Joselli et al. (2012), Chen (2015), Zamith et al. (2011) and Kim (2015). Chen et al. (2011), Huang et al. (2012) and Vecchio et al. (2015) use augmented reality applications, health monitoring applications (Kulkarni et al., 2014; Hanen et al., 2016; Nkosi and Mekuria, 2010), virus scan applications (Kosta et al., 2011) and image recognition applications (Ayad et al., 2014; Somasundaram et al., 2011; Arshad et al., 2017). Also, the application model for mobile cloud may changes based on design and objectives for decision making such as application execution or energy efficiency. Many previous work presented different mobile cloud application model as Java agent development environment (JADE) (Qian and Andresen, 2015) is written in Java and uses Android operating system. It adds an advanced computation offloading to Android application and observes application and device to take the accurate decision for offloading. The communication among the agents is done using Java (RMI) remote method invocation. (Zhang et al., 2010) suggest a model in which partitioned the application into component called weblets. A weblet represent a function that compute independent task. The weblets offloading decision are based on multiple parameters as CPU load, user preference, communication channel and battery level. Also, weblets may be platform dependent or platform independent based on the programming technology used. Mobile assistance using infrastructure (MAUI) (Cuervo et al., 2010) written all its application using C#. MAUI is a model enabled an energy aware offloading of smartphone code to the cloud. The offloaded code to a remote infrastructure is favourable for the purpose of energy conservation. Marinelli (2009) uses Hadoop's mapreduce implementation for distributed data processing proposes a model used to process the extra-large dataset distributed across servers cluster. (March et al., 2011) proposes named µcloud. µcloud is a model focus on the development of applications using heterogeneous components to support reusability and flexibility that may be run on mobile device or cloud server. In µcloud the applications are showed as directed graph. When an application graph executed each node inserts its output into subsequent nodes. Excloud (Ma et al., 2011) proposes VM instance level computation offloading to the cloud. The model make on demand data or code migration and executes computation offloading when mobile device resources are insufficient or reached to a certain level. excloud needs runtime synchronisation VM on cloud server and the mobile device. Chun et al. (2011) suggests clonecloud model that offloads parts of the application execution to the mobile clone in the cloud. In Clonecloud

it's very important the synchronisation. Clonecloud supports thread level migration and requires development of cost model for a particular application under various partitions, where each partition executed separately on mobile device and the cloud server. Kemp et al. (2010) supports offloading parts of applications to the cloud server and uses previously defined tools for application development. It uses an unpretentious decision technique for offloading in case of the server is available. In this paper, the design objectives, the entities that effect on the computation offloading and the implementation details of the proposed algorithm is shown below. The paper is presented as follows: Section 2 describes the system architecture and the implementation details of the proposed offloading algorithm, while Section 3 discusses and shows the results obtained. Finally the paper is concluded in Section 4.

Figure 1 Mobile computation offloading (see online version for colours)



2 System architecture

In this section, the offloading estimator architecture will be introduced. It is built for smart mobile devices that use Android operating system to enhance the applications performance and the energy efficiency. Furthermore, the offloading architecture consists of two parts; the first part is named as the smartphone part while the other is called the cloud part as depicted in Figure 2. This architecture uses an offloading criterion to decide which application modules should be offloaded to the cloud and which modules will remain and run on the mobile device. This criterion uses multiple weighted parameters such as (user waiting time threshold, execution time, energy consumption, remaining battery time, available memory). In addition, the introduced architecture uses an estimator that based on *PI* computation (Super PI, 2017) to estimate and predict the computation time of a module before the real execution of the module. The robustness and reliability of this method is verified by the complexity analysis of the tested modules.

2.1 Offloading estimator architecture

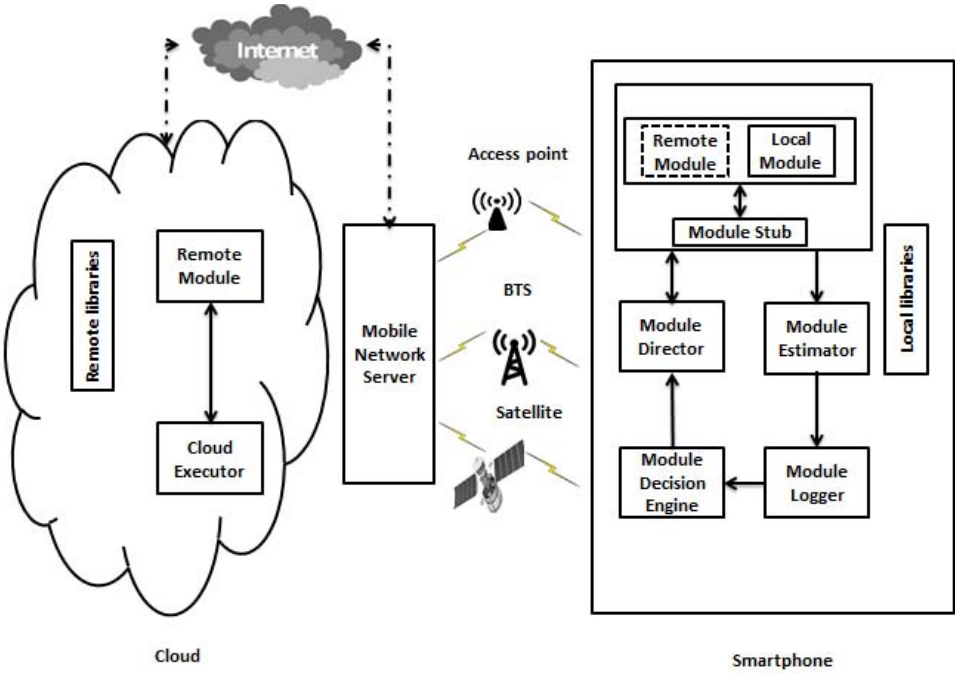
The proposed offloading estimator algorithm can offloads all or part of an application execution on mobile devices or smartphones to a remote cloud or a nearby infrastructure to achieve energy conservation and reduce the application's execution time. This architecture can be applied to any application partitioned into modules where heavy

computational modules can be offloaded into nearby or remote cloud, while other modules can run locally on smartphone using the Android inter process communication (IPC). In this context, the developer develops the application modules as services and redirects the service invocation from the activity to the corresponding service module on the cloud server without altering the application source code.

2.1.1 The smartphone part

As demonstrated in Figure 2, the smartphone part is composed of four components (module estimator, module logger, module decision engine, module offloading director). The detailed function of each component is described as follows.

Figure 2 The architecture of the offloading estimator model (see online version for colours)



2.1.1.1 Module estimator

The module estimator is one of the key parts in the offloading architecure. It is essential to predict the time which is needed to execute an application before the real execution of an application. Therefore, it is required to foresee the execution time to provide the ultimate value of the time needed by an application to run on any computing devices such as a mobile device or a cloud. As a consequence, this can get correct decisions for offloading. Many researches (Engblom et al., 2003; Chen et al., 2005; Kirner et al., 2009; Schoeberl et al., 2010) have been worked on the prediction of execution time. Therefore for simplification purposes, this architecture is augmented with *PI* calculation method for reliable estimation of application module executing time. This estimation criterion is

validated by applying the complexity of the application module. Table 1 show the parameters used in the module estimator and its description.

Table 1 List of parameters used by the module estimator

Parameter	*Description*	*Unit*
T_P^S	The estimated execution time on smartphone	Second
T_P^C	The estimated execution time onto the cloud	Second
I	The number of instructions	Instructions
S_s	The processor speed of smartphone	Instruction per second
S_C	The processor speed of cloud	Instruction per second
S_{PI}	The speed when using *PI* calculation method	Instructions per second
I_{PI}^k	The number of instructions to get the *PI* constant for *k* terms	Instructions
T_{PI}^k	The number of instructions to get the *PI* constant for *k* terms	Instructions
$\propto_{\mathrm{PI}}$	The load factor when using *PI* calculation method	-
$\propto_{\mathrm{C}}$	The load factor when using the complexity method	-

Firstly to illustrate the *PI* calculation method, the predicted time to run an application module on a smartphone or onto the cloud can be computed as in equation (1) and (2)

$$T_P^S = \frac{I}{S_S} \tag{1}$$

$$T_P^C = \frac{I}{S_C} \tag{2}$$

It should be noted that the *PI* constant uses the main dominant operations as addition, subtraction, multiplication and division and is computed using Gregory's series formula for *k* terms of *PI* series (Borwein et al., 2004):

$$PI = \sum_{i=0}^{k} \frac{(-1^i)}{2i+1} \tag{3}$$

The speed of a computing device (S_{PI}) can be obtained by the use of *PI* calculation method as in equation (4):

$$S_{PI} = \frac{\mathrm{I}_{\mathrm{PI}}^{\mathrm{k}}}{\mathrm{T}_{\mathrm{PI}}^{\mathrm{k}}} \tag{4}$$

In this paper, the *PI* method is applied for various application modules such as loops, nested loops, N-Queens, sorting, matrix multiplication, brute force attack and face detection. For particular application module, the execution time is estimated using the *PI* calculation method for different inputs n as depicted in equation (5)

$$T_{PI} = \frac{I_i^n}{S_{PI}} \tag{5}$$

where I_i^n is the number of instructions for the i^{th} module with n inputs. When applying comprehensive tests, a noticeable load parameter factor (α) comes up as in equation (6):

$$\propto_{PI} = \frac{\mathrm{T_{real}}}{\mathrm{T}_{PI}} \tag{6}$$

This load factor for the i^{th} module is almost constant with different inputs. Therefore, this parameter becomes a good indicator for estimating the application module execution time.

For the time complexity method, the focus is on getting the growth rate of the execution time as a function of the input size n taking a 'big-picture' approach. For example, it is regularly sufficient just to know that the execution time of an algorithm rises proportionally to n (Goodrich and Tamassia, 2008). Therefore to estimate the execution time for the i^{th} module, the proposed method use of the unit time execution of the *PI* constant for the computing device and multiply it by the time complexity of the i^{th} module as in equation (7).

$$T_C = t(1) \times t_i(n) \tag{7}$$

where $t(1)$ is unit time execution of the *PI* constant in the computing device, while $t_i(n)$ is the time complexity the i^{th} module.

Similarly, a load factor (α) for each module is calculated as in equation (8):

$$\propto_{PI} = \frac{\mathrm{T_{real}}}{\mathrm{T}_{PI}} \tag{8}$$

For the purpose of validations, the proposed estimation methods are performed on different application modules. Results give that the predicted execution time has high correlation with the real execution time as shown in Table 2.

Table 2 The correlation between the estimated and real execution times

Application	*Loop*	*Nested loop*	*N-Queens*	*Bubble sort*	*Matrix multiplication*	*Brute force attack*	*Face detection*
Using PI	0.99	0.99	0.93	0.99	0.99	0.99	0.99
Using complexity	0.99	0.99	0.99	0.99	0.99	0.98	0.98

2.1.1.2 Module logger

It acts as a profile for each application module. It stores the module's execution time on the smartphone and on the cloud. It should be noted that the module estimator is providing the module logger with such data for future executions of the application module.

2.1.1.3 Module decision engine

The offloading decision engine is the core of the proposed model. Therefore, this module is responsible for deciding whether is beneficial for an application module to run local on

smartphone or to remotely run onto the cloud server. Figure 3 illustrates the workflow of the offloading decision algorithm. In addition, Table 3 shows the parameters used in the workflow and its description. Therefore for a given application module, the following subsequent steps should be applied to get the desirable offloading decision:

Step 1 Initialise the following coefficients C_W, C_E, C_T, C_R, C_M and $D_{offload}$ with zero and then test the server and network availability if OK, go to step 2. Otherwise go to step 9.

Step 2 Find the time to execute the application module on a smart phone $T_{smartphone}$ and T_{Cloud} the total time to execute on cloud as in equations (9) and (10):

$$T_{smartphone} = \frac{I}{S_{PI}^{S}} \tag{9}$$

$$T_{cloud} = \frac{I}{S_{PI}^{C}} + RTT + \frac{D}{BW} \tag{10}$$

Step 3 Find the waiting time threshold $T_{waiting}$ that is specified by a user and then compare $T_{waiting}$ with $T_{smartphone}$. If the user waiting time is greater than the execution time on the smartphone, then the waiting time coefficient is assigned with –1. Otherwise it's assigned with +1.

Step 4 Find the energy consumption of executing the application module on the smartphone and on the cloud server namely $E_{smartphone}$, E_{Cloud} respectively as depicted in equation (11) and (12). Then compare $E_{smartphone}$, E_{Cloud}, if E_{Cloud} is greater than $E_{smartphone}$ the energy coefficient is assigned with –1, else assign it with +1.

$$E_{smartphone} = P_{cpu,\,active} \times T_{smartphone} \tag{11}$$

$$E_{cloud} = P_{network,\,active} \times \left(RTT + \frac{D}{BW} \right) + P_{cpu,\,idle} \times T_{cloud} \tag{12}$$

Step 5 Find the remaining time on smartphone battery $T_{Remaining}$ and then compare it with the execution time on smartphone and the communication time for sending or receiving data to the cloud. If $T_{Remaining}$ is greater than $T_{Communication}$ or lower than $T_{smartphone}$, the remaining battery time coefficient is assigned with +1. Otherwise assign it with –1.

Step 6 Find the available memory on smartphone M_{avail} and the threshold of available memory at which the memory is considered to be low M_{th}. If M_{avail} is lower than M_{th}, the memory coefficient is assigned with +1. Else, assign it with –1.

Step 7 Calculate the final offloading decision $D_{offload}$ using the weights W_0, W_1, W_2, W_3 and W_4 based on priorities of user preferences as in equation (13). Finally, if the offloading decision is greater than or equal zero, go to step 8. Otherwise go to step 9.

$$D_{offload} = W_0 C_W + W_1 C_E + W_2 C_T + W_3 C_R + W_4 C_M \tag{13}$$

Step 8 It is favourable to offload the application module to cloud.

Step 9 It is beneficial to running the application module on smartphone.

Figure 3 The workflow of the proposed offloading decision algorithm

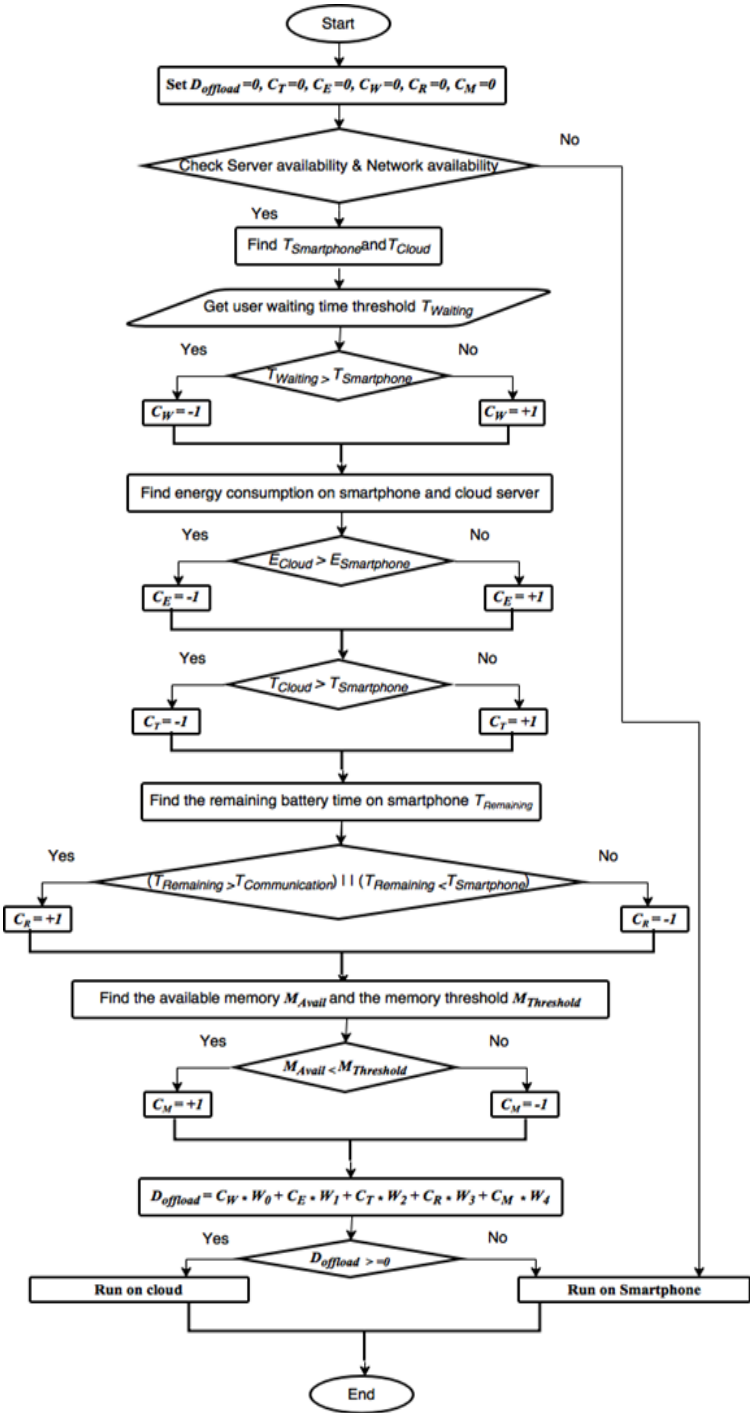


Table 3 List of parameters used in the workflow and its description

Variable	*Description*	*Unit*
$T_{smartphone}$	The application module execution time on smartphone	Second
T_{Cloud}	The application module execution time on cloud	Second
$T_{communication}$	The communication time	Second
$T_{Waiting}$	The waiting time threshold	Second
I	The number of instructions	Instructions
S_{PI}^{S}	Speed of smartphone using PI	Instructions per second
S_{PI}^{C}	Speed of cloud using PI	Instructions per second
RTT	The round trip time	Second
D	The data size to be transferred to/from the cloud	Bytes
BW	Bandwidth of the network	Bytes per second
$E_{smartphone}$	Energy consumed when executing the application on smartphone	Joule
E_{Cloud}	Total Energy consumed when executing the application on cloud	Joule
$P_{cpu,\ idle}$	Smartphone idle power	Watt
$P_{cpu,\ active}$	Active power of CPU on smartphone	Watt
$P_{network,\ active}$	Power consumed when send or receive data	Watt
M_{avail}	Available memory on the smartphone	Megabytes
M_{th}	Threshold of the available memory at which the memory is considered low	Megabytes
C_W	Waiting time coefficient	-
C_E	Energy coefficient	-
C_T	Execution time coefficient	-
C_R	Remaining battery time coefficient	-
C_M	Memory coefficient	-
W_i	Weighting variables $\in$ [0:1] as the user preference such that the $\in$ [0:4], $\Sigma_i W_i = 1$	-
$D_{offload}$	Final offloading decision	-

2.1.1.4 Module director

This module is in charge of executing the application module based on the decision taken by the offloading decision algorithm. If the decision is to execute the application module on the smartphone, the module director calls the local service implementation from the mobile part. But, if the decision is favour to execute onto the cloud, the module director redirects the call to the cloud executor to execute the corresponding service implementation of the module onto the cloud. Finally, the module director will deliver the execution results to the application activity.

2.1.2 The cloud part

The cloud part contains the cloud executor component which is a java application responsible for installing the offloaded modules to the cloud. Afterward, the module service is initialised and invoked when getting a decision from the module offloading director. It should be noted that the module service can be executed on any machine having Java standard edition (SE). When an application is firstly executed, an archived file (JAR File) will be sent and stored at the cloud server for future executions. Whenever the module source code is modified, only the archived file of the module service will be resent. After the execution of the module, the results send back to application activity.

2.2 Implementation details

This section demonstrates the workflow methodology of the offloading estimator architecture. The proposed offloading architecture gives the developer an easy way to offload the application modules to the cloud or to run it on the mobile. This architecture is based on a modular technique. In which, the application developer segments the application into modular parts, where each module act as a service and performs a particular function. The application may have a user interface for user interactions or activities for accessing sensors or cameras. It should be noted that either a user interface or a local activity are not considered as candidate modules for offloading. For a candidate module, two replicas of the same module service are used. The first replica is named as a local service for local executions on the smartphone. The latter contains the same implementation for remote executions on the cloud and is called a remote service which can be updated by the developer. In this paper, the offloading architecture is developed and applied on android platform. As known, the main components of an android application are activities, services, content provider and broadcast receiver (Gargenta, 2011). In this context, the proposed work uses activities and services components. To allow an application activity to connect to a service (module), a predefined interface which is named as Android interface definition language (AIDL) (Developer.android.com, 2017) is used through the inter-process communication (IPC). The communication between AIDL module service and activities is achieved as follows:

- when the application activity requests to invoke a particular method in a service
- the activity directs a call to the identical method in the proxy
- the proxy is in charge of the connection of the service for rendering the needed method
- but, the proxy cannot linked to service directly, it must firstly connects to the stub that call the local service and then return results back to the proxy
- the proxy takes the results and sends them to the caller activity.

It is need to clarify that the android build system is augmented with the offloading model. Consequently, the developer can write the implementation for the module service using AIDL interface. The module service can be run on the smartphone or offloaded onto the cloud based on the offloading decision. Afterward, each time a developer compiles the module, through ant tool (Ant.apache.org, 2017), an archive file (JAR file) for the remote implementations of the cloud is formed and sent the cloud. Then, the cloud executor in

the cloud part is in charge of handling the offloaded modules requests and installing the JAR files. After that, it invokes and initialises the module service.

3 Simulation results

This section shows the simulation results and discusses the performance evaluation of the proposed algorithm. Table 5 shows various applications to examine the proposed algorithm. Furthermore, a real application such as face detection is used. The proposed algorithm allows the user to select an image and then apply the face detection module service on either the mobile device or onto the cloud.

3.1 Experimental setup

The experiments are performed using Samsung galaxy N8100 with android platform as mobile part and a cloud with a core i3 2.3 GHZ CPU. To evaluate the proposed model, seven applications with various input sizes and computational resources are tested. Table 4 explores the specifications of the mobile and cloud parts.

Table 4 Mobile and cloud specifications

Parameters	*Mobile part*	*Cloud part*
Operating system	Android v4.0.3	Windows 7, 64-bit
Processing speed	Dual core 1.5 GHZ	Core i3 2.3 GHZ
RAM	1 GB	4 GB
Storage	16 GB	250 GB
Battery capacity	Li-Ion 1750 mAh	-

Table 5 The application modules description

No.	*Application*	*Description*
1	Loops	Execute loops for n times
2	Nested loops	Execute nested loops for n times
3	Sort	Sort set of n integer numbers using bubble sort
4	Matrix multiplication	Multiply $n \times n$ matrix
5	N-Queens problem	Solve N-Queens problem with different n
6	Brute-force attack	Crack n characters (0–9 and a–z)

3.2 Results and discussion

For the purpose of validation, the proposed algorithm is tested on a real world and commonly used mobile application to examine its suitability and reliability. In this context, the face detection application is used as it requires intensive computations and consumes much of the mobile energy. In this paper, the faces are detected using OpenCV version 2.4.11 which supports Java desktop development compatible to the Android environment. The face detection operation is performed using Haar cascade classifiers

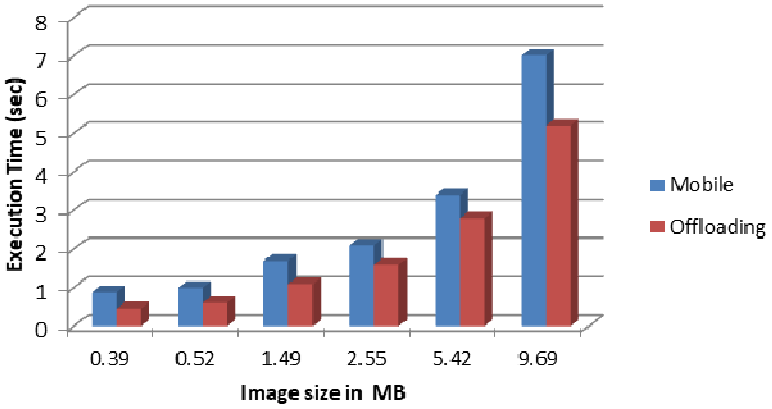
that are introduced by Viola and Jones (2001). Figure 4 depicts a set of tested images before and after applying the face detection application.

Figure 4 (a) The tested images (b) The images after applying the face detection application (see online version for colours)

(a)

(b)

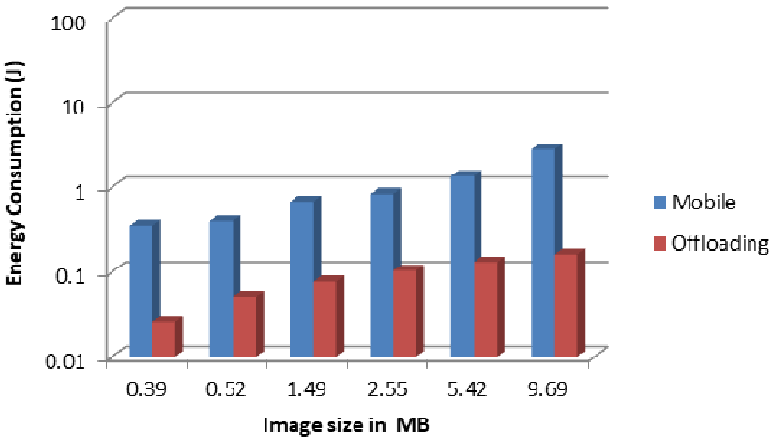
Figure 5 The execution time on mobile and onto the cloud server after offloading (see online version for colours)



To examine the effect of applying the offloading, the energy consumptions and execution times for each module of the face detection application are computed at different images with various sizes. Figure 5 and 6 show the execution time and the energy consumption before and after applying the proposed offloading algorithm. As shown, the offloading gives better energy consumptions specifically for high computational modules. It should be noted that the increase in the image size leads to an increase in the execution time on

the mobile device. Therefore by exploiting the cloud resources, the execution time and energy consumption are improved. The execution time when applying the offloading is the summation of the execution time on cloud server plus the communication time for sending/receiving the application module to/from the cloud server.

Figure 6 The energy consumption on mobile and the cloud server after offloading (see online version for colours)



Further experiments have been carried out to examine the reliability of the proposed offloading algorithm. For this purpose, six different application modules are tested.

Table 5 presents the description of application modules.

Table 6 describes the application modules evaluations with different input with respect to the execution time and energy consumption. As shown, for the loop module, the offloading is always preferred for both the execution time and energy consumption because the input size is very large. While in nested loop module with smaller input sizes, it is beneficial to execute the module locally as the offloading requires large execution times. This is because the communication time exceeds the execution time on the mobile device. In bubble sort and matrix multiplication modules, when the array contains a small set of numbers, the offloading does not give any improvements in energy consumption and execution time. As a consequence it is favourable to execute locally on the mobile device. But for higher sizes, the offloading become a most convenient and suitable solution. Likewise in N-Queens problem, when the input size is less than or equal nine, the offloading is not preferable where the local execution on the mobile is better than the remote execution. For the brute force attack, there are large trials of generated numbers and characters to find the desired user password. Consequently, this application module is considered a high computational module specifically when the password digits are increased. So, it is preferred to offload this module rather than executing it locally on the mobile device.

Table 6 The execution time and energy consumption for the various application modules with different input sizes

Application module	*Input size*	*Execution time in (sec)*		*Energy consumption in (J)*	
		Mobile	*Offloading*	*Mobile*	*Offloading*
Loop (iterations)	2E + 06	1.632	0.53	0.653	0.025
	4E + 06	3.497	0.547	1.399	0.025
	6E + 06	5.201	0.563	2.081	0.025
	8E + 06	6.868	0.578	2.747	0.025
	10E + 06	8.606	0.599	3.442	0.025
Nested loop (iterations)	1E + 04	0.319	0.542	0.128	0.025
	3E + 04	2.297	0.834	0.919	0.026
	5E + 04	6.075	1.385	2.43	0.027
	7E + 04	11.453	2.168	4.581	0.029
	9E + 04	18.543	3.201	7.417	0.031
Bubble sort (numbers)	1E + 03	0.031	1.008	0.012	0.05
	5E + 03	0.344	1.088	0.138	0.051
	10E + 03	1.296	1.264	0.518	0.052
	15E + 03	3.113	1.651	1.245	0.054
	20E + 03	5.36	2.054	2.144	0.056
Matrix multiplication	100 × 100	0.092	1.086	0.037	0.053
	300 × 300	1.366	1.892	0.546	0.082
	500 × 500	7.605	3.892	3.042	0.143
	700 × 700	24.98	7.679	9.992	0.239
	900 × 900	57.698	15.431	23.079	0.361
N-Queens	8	0.054	0.54	0.022	0.025
	9	0.26	0.621	0.104	0.025
	10	0.914	0.824	0.366	0.026
	11	3.13	1.852	1.252	0.027
	12	10.471	8.052	4.189	0.042
Brute-force attack	Crack 3 characters	0.893	0.827	0.357	0.026
	Crack 4 characters	28.916	11.842	11.566	0.051
	Crack 5 characters	954.228	391.251	381.691	0.885

4 Conclusions

This paper has introduced a new energy-preserving mobile cloud offloading algorithm. The proposed algorithm can successfully reduce the energy consumption of mobile devices and enhance the performance of applications. As demonstrated, the application to be executed is partitioned into modules, where each module acts as a service to be run on the mobile device or on a cloud server. Furthermore, the introduced algorithm uses multiple weighted parameters to decide which modules of an application should be executed and offloaded onto the cloud and which modules will reside and run locally on the mobile device. For the purpose of validation, the proposed architecture is tested on real world applications to examine its suitability and reliability. The results showed a remarkable saving on both the energy and the execution time when using the proposed mobile cloud offloading algorithm.

References

Altamimi, M., Palit, R., Naik, K. and Nayak, A. (2012) 'Energy-as-a-service (EaaS): on the efficacy of multimedia cloud computing to save smartphone energy', in *IEEE 5th International Conference on Cloud Computing (CLOUD)*, IEEE, June, pp.764–771.

Ant.apache.org (2017) *Apache Ant – Welcome* [online] http://ant.apache.org/ (accessed 5 October 2017).

Arshad, H., Chun, L.M., Obeidy, W.K. and Yee, T.S. (2017) 'An efficient cloud based image target recognition SDK for mobile applications', *International Journal on Advanced Science, Engineering and Information Technology*, Vol. 7, No. 2, pp.496–502.

Ayad, M., Taher, M. and Salem, A. (2014) 'Real-time mobile cloud computing: a case study in face recognition', in *28th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, May, IEEE, pp.73–78.

Bahrami, M. (2015) 'Cloud computing for emerging mobile cloud apps', in *3rd IEEE International Conference on Mobile Cloud Computing, Services and Engineering (MobileCloud)*, IEEE, March, pp.4–5.

Borwein, J.M., Bailey, D.H. and Bailey, D. (2004) *Mathematics by Experiment: Plausible Reasoning in the 21st Century*, p.103, AK Peters, Natick, MA.

Chen, M., Ling, C. and Zhang, W. (2011) 'Analysis of augmented reality application based on cloud computing, in *4th International Congress on Image and Signal Processing (CISP)*, IEEE, October, Vol. 2, pp. 569-572.

Chen, S., Liu, Y., Gorton, I. and Liu, A., (2005) 'Performance prediction of component-based applications', *Journal of Systems and Software*, Vol. 74, No. 1, pp.35–43.

Chen, X. (2015) 'Decentralized computation offloading game for mobile cloud computing', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, No. 4, pp.974–983.

Chun, B.G., Ihm, S., Maniatis, P., Naik, M. and Patti, A. (2011) 'Clonecloud: elastic execution between mobile device and cloud', in *Proceedings of the Sixth Conference on Computer Systems*, ACM, April, pp.301–314.

Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R. and Bahl, P. (2010) 'MAUI: making smartphones last longer with code offload', in *Proceedings of the 8th International Conference on Mobile Systems, Applications and Services*, ACM, June, pp.49–62.

Developer.android.com (2017) *Android Developers, Android Interface Definition Language (AIDL)* [online] https://developer.android.com/guide/components/aidl.html (accessed 5 October 2017).

Doolan, D.C., Tabirca, S. and Yang, L.T. (2008) 'Mmpi a message passing interface for the mobile environment', in *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, ACM, November, pp. 317–321.

Engblom, J., Ermedahl, A., Sjödin, M., Gustafsson, J. and Hansson, H. (2003) 'Worst-case execution-time analysis for embedded real-time systems', *International Journal on Software Tools for Technology Transfer (STTT)*, Vol. 4, No. 4, pp.437–455.

Gargenta, M. (2011) *Learning Android*, O'Reilly Media, Inc, USA.

Goodrich, M.T. and Tamassia, R. (2008) *Data Structures and Algorithms in Java*, John Wiley & Sons, USA.

Hanen, J., Kechaou, Z. and Ayed, M.B. (2016) 'An enhanced healthcare system in mobile cloud computing environment', *Vietnam Journal of Computer Science*, Vol. 3, No. 4, pp.267–277.

Huang, B.R., Lin, C.H. and Lee, C.H. (2012) 'Mobile augmented reality based on cloud computing', in *International Conference on Anti-Counterfeiting, Security and Identification (ASID)*, IEEE, August, pp.1–5.

Huang, D. and Wu, H. (2017) *Mobile Cloud Computing: Foundations and Service Models*, Morgan Kaufmann, USA.

Huerta-Canepa, G. and Lee, D. (2010) 'A virtual cloud computing provider for mobile devices', in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, ACM, June, p.6.

Jagtap, V.S., Pawar, K.V. and Pathak, A.R. (2014) 'Augmented execution in mobile cloud computing: a survey', in *International Conference on Electronic Systems, Signal Processing and Computing Technologies (ICESC)*, IEEE, January, pp.237–244.

Joselli, M., Zamith, M., Silva, J., Clua, E.W.G., Feijó, B., Leal, R., Valente, L. and Soluri, E. (2012) *An Architecture for Mobile Games with Cloud Computing Module*, SBGames, SBC, Brazil.

Kemp, R., Palmer, N., Kielmann, T. and Bal, H.E. (2010) 'Cuckoo: a computation offloading framework for smartphones', in *MobiCASE*, October, pp. 59–79.

Kim, S. (2015) 'Nested game-based computation offloading scheme for mobile cloud IoT systems', *EURASIP Journal on Wireless Communications and Networking*, No. 1, p.229.

Kirner, R., Kadlec, A. and Puschner, P. (2009) 'Precise worst-case execution time analysis for processors with timing anomalies', in *21st Euromicro Conference on Real-Time Systems, ECRTS'09*, IEEE, July, pp.119–128.

Kosta, S., Aucinas, A., Hui, P., Mortier, R. and Zhang, X. (2011) 'Unleashing the power of mobile cloud computing using thinkair', *Computing Research Repository (CoRR)*, pp.1–17, abs/1105.3232 [online] https://arxiv.org/abs/1105.3232.

Kosta, S., Aucinas, A., Hui, P., Mortier, R. and Zhang, X. (2012) 'Thinkair: dynamic resource allocation and parallel execution in the cloud for mobile code offloading', in *INFOCOM, 2012 Proceedings IEEE*, IEEE, March, pp.945–953.

Kulkarni, G., Shelke, R., Patil, P.B.N., Kulkarni, V. and Mohite, S. (2014) 'Optimization in mobile cloud computing for cloud based health application', in *Fourth International Conference on Communication Systems and Network Technologies (CSNT)*, IEEE, April, pp. 569–572.

Lin, B.S.P., Tsai, W.H., Wu, C.C., Hsu, P.H., Huang, J.Y. and Liu, T.H. (2013) 'The design of cloud-based 4G/LTE for mobile augmented reality with smart mobile devices', in *IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, IEEE, March, pp.561–566.

Lo'ai, A.T., Bakhader, W., Mehmood, R. and Song, H. (2016) 'Cloudlet-based mobile cloud computing for healthcare applications', in *Global Communications Conference (GLOBECOM)*, IEEE, December, pp.1–6.

Ma, R.K., Lam, K.T. and Wang, C.L. (2011) Excloud: transparent runtime support for scaling mobile applications in cloud', in *International Conference on Cloud and Service Computing (CSC)*, IEEE, December, pp.103–110.

March, V., Gu, Y., Leonardi, E., Goh, G., Kirchberg, M. and Lee, B.S. (2011) 'µcloud: towards a new paradigm of rich mobile applications', *Procedia Computer Science*, Vol. 5, pp.618–624.

Marinelli, E.E. (2009) *Hyrax: Cloud Computing on Mobile Devices using MapReduce (No. CMU-CS-09-164)*, School of Computer Science, Carnegie-Mellon University, Pittsburgh PA.

Mohammadpour, S. and Tafte, F. (2016) 'M-commerce: the state of the art, challenges and cloud-based solutions', in *Eighth International Conference on Information and Knowledge Technology (IKT)*, IEEE, September, pp.66–77.

Nkosi, M. and Mekuria, F. (2011) 'Improving the capacity, reliability & life of mobile devices with cloud computing', in *IST-Africa Conference Proceedings*, IEEE, May, pp.1–9.

Nkosi, M.T. and Mekuria, F. (2010) 'Cloud computing for enhanced mobile health applications', in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, November, pp.629–633.

Qian, H. and Andresen, D. (2015) 'Extending mobile device's battery life by offloading computation to cloud', in *2nd ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, IEEE, May, pp.150–151.

Satyanarayanan, M., Bahl, P., Caceres, R. and Davies, N. (2009) 'The case for VM-based cloudlets in mobile computing', *IEEE Pervasive Computing*, Vol. 8, No. 4, pp.14–23.

Schoeberl, M., Puffitsch, W., Pedersen, R.U. and Huber, B. (2010) 'Worst-case execution time analysis for a Java processor', *Software: Practice and Experience*, Vol. 40, No. 6, pp.507–542.

Somasundaram, M., Gitanjali, S., Govardhani, T.C., Priya, G.L. and Sivakumar, R. (2011) 'Medical image data management system in mobile cloud computing environment', *Proceedings of International Conference on Signal, Image Processing and Applications (ICSIPA 2011)*, Academic Press, Kuala Lumpur, Malaysia, pp.11-15.

Super PI (2017) *Wikipedia* [online] https://en.wikipedia.org/wiki/Super_PI (accessed 5 October 2017).

Vecchio, P., Mele, F., De Paolis, L.T., Epicoco, I., Mancini, M. and Aloisio, G. (2015) 'Cloud computing and augmented reality for cultural heritage', in *International Conference on Augmented and Virtual Reality*, Springer International Publishing, August, pp.51–60.

Viola, P. and Jones, M. (2001) 'Rapid object detection using a boosted cascade of simple features', in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001*, IEEE, Vol. 1, p.1.

Wang, Y., Lin, X. and Pedram, M. (2013) A nested two stage game-based optimization framework in mobile cloud computing system, in *IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, IEEE, March, pp.494–502.

Zamith, M., Joselli, M., Clua, E.W.G., Montenegro, A., Leal-Toledo, R.C.P., Valente, L. and Feijo, B. (2011) 'A distributed architecture for mobile digital games based on cloud computing', in *Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, IEEE, November, pp.79–88.

Zhang, X., Jeong, S., Kunjithapatham, A. and Gibbs, S. (2010) 'Towards an elastic application model for augmenting computing capabilities of mobile platform', in *International Conference on Mobile Wireless Middleware, Operating Systems, and Applications (MOBILWARE)*, Springer, pp.161–174.

Zheng, J., Cai, Y., Wu, Y. and Shen, X.S. (2016) 'Stochastic computation offloading game for mobile cloud computing', in *IEEE/CIC International Conference on Communications in China (ICCC)*, IEEE, July, pp.1–6.

Zou, Q., Wang, J. and Chen, X. (2015) 'Secure encrypted image search in mobile cloud computing', in *10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*, IEEE, November, pp.572–575.