An Efficient and Secured Framework for Mobile Cloud Computing

Ibrahim A. Elgendy, Wei-Zhe Zhang, *Member, IEEE*, Chuan-Yi Liu, *Member, IEEE*, and Ching-Hsien Hsu, *Senior Member, IEEE*

Abstract—Smartphone devices are widely used in our daily lives. However, these devices exhibit limitations, such as short battery lifetime, limited computation power, small memory size and unpredictable network connectivity. Therefore, numerous solutions have been proposed to mitigate these limitations and extend the battery lifetime with the use of the offloading technique. In this paper, a novel framework is proposed to offload intensive computation tasks from the mobile device to the cloud. This framework uses an optimization model to determine the offloading decision dynamically based on four main parameters, namely, energy consumption, CPU utilization, execution time, and memory usage. In addition, a new security layer is provided to protect the transferred data in the cloud from any attack. The experimental results showed that the framework can select a suitable offloading decision for different types of mobile application tasks while achieving significant performance improvement. Moreover, different from previous techniques, the framework can protect application data from any threat.

Index Terms—Smartphones, mobile cloud computing, computation offloading, security, particle swarm optimization

1 Introduction

Smartphones provide a broad range of applications, such as face detection, augmented reality, image and video processing, and video gaming and speech recognition. These applications are complex, and the demand for computing resources is increasing. However, despite the advancements in smartphones, the extent of battery life has remained as one of the main challenges in improving computational requirements through battery upgrade, [1].

Cloud computing [2], [3] allows access to unlimited resource over the internet. Cloud computing provides several advantages, such as self-service provisioning, elasticity, broad network access, resource pooling, low costs, and ease of utilization, among others.

Thus, mobile cloud computing [4] is introduced to overcome the limitations of smartphone devices. Mobile cloud computing is a new paradigm that integrates cloud computing technology and mobile devices to extend the battery lifetime and increase application performance.

Recent studies have proposed to offload all or part of the mobile applications from mobile device to the cloud for remote execution [5], [6]. These frameworks are designed to make a trade-off between one or more constraints, such as energy consumption of the mobile device, CPU utilization, execution time, remaining battery life, and data transmission amount on the network, in the offloading decision. However, most of these models do not consider memory usage as a constraint in the offloading decision. Memory usage is one of the main resources consumed by mobile applications. In addition, security techniques are not applied in the protection of offloaded data from attacks. Therefore, this work mainly focuses on building a new model that combines most of the mentioned constraints to improve the performance of mobile applications and to protect the application data from any attack.

We specifically proposed a novel framework that uses computation offloading to offload only the intensive tasks of mobile applications. We formulated an optimization model responsible for determining the offloading decision. The main results and contributions of this paper are as follows:

- This work proposes a novel framework that offloads only intensive tasks instead of offloading all applications, thereby requiring less network communication.
- An optimization model is formulated to determine the offloading decision dynamically at runtime based on four main constraints, namely, execution time of the task, CPU utilization, memory usage and energy consumption.
- A new security layer is added to encrypt the data of the task before transferring to the cloud side by using AES encryption technique.
- Three different types of mobile applications are used in the experimental studies to test this framework and to show the selection of a proper offloading decision for improved application performance.

The rest of the paper is organized as follows. Section 2 discusses the state-of-the-art computational offloading frameworks and their drawbacks. Section 3 presents the framework architecture and its design goals. Section 4

- *I. A. Elgendy, W.-Z Zhang, and C.-Y Liu is with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. E-mail: {ibrahim.elgendy, wzzhang}@hit.edu.cn, cy-liu04@mails.tsinghua.edu.cn.*
- *C.-H. Hsu is with the Department of Medical Research, China Medical University Hospital, China Medical University, Taichung, Taiwan. E-mail: robertchh@gmail.com.*

Manuscript received 3 Feb. 2018; revised 18 Apr. 2018; accepted 10 June 2018. Date of publication 18 June 2018; date of current version 5 Mar. 2021. (Corresponding author: Wei-Zhe Zhang.) Recommended for acceptance by B. Di Martino. Digital Object Identifier no. 10.1109/TCC.2018.2847347

shows the experimental studies and discusses experimental findings. Finally, Section 5 reviews the conclusion with a recommendation for future research.

2 Related Work

Numerous approaches have been recently proposed to address the challenges of mobile devices by offloading the computation tasks to the cloud resources for remote execution [5], [6]. Some of these approaches migrate only a process from the mobile device to the cloned virtual machine (VM) on the cloud [7], [8]. In [7], a combination of static analysis and dynamic profiling modules is used to partition the application and determine which process is migrated to the cloud. Kosta et al. [8] created VMs of a complete smartphone system on the cloud and used a profiler module to monitor the remote execution of the methods using an execution controller. The main drawback of [7] and [8] is the energy-consuming requirement of basic synchronization with the clone VM on the cloud [9]. Furthermore, application data are not protected from attacks during transfer to the cloud. In [10], the synchronization problem is handled by offloading only the intensive services and not the full process to the cloud. In addition, the authors build a model to determine the offloading decision for these services. However, this model is extremely simple and static and always prefers remote execution. In certain cases, executing services on the mobile is superior to offloading to the cloud. The transferred data must be protected by applying any security technique.

Other frameworks involving the partition of the application and the offload of intensive methods are proposed in [11], [12] and [13]. These frameworks also use an integer linear programming model like our framework in making offloading decisions. Total response time, remaining battery life, and energy consumption constraints [14] are considered in making the offloading decision without adding any memory usage consideration and security to the offloading model. By contrast, in [15], the full Android application is offloaded from the mobile device to the cloud, which is resource consuming owing to the large amount of transferred data over the network. In addition, the application sent to the cloud must be safe, so any security technique should be protected.

The minimization of the data transmission and the energy consumption are the main goals of [16], which offloads only the resource-intensive services and exploits from Software-as-a-Service model for the configuration of intensive services on the cloud server. Similar to [7], [16] needs basic synchronization between the mobile device and the cloud server node, which consumes additional battery power and makes the offloaded data vulnerable to attacks.

A context-aware mobile cloud computing system with an estimation model was built in [17], providing a dynamic decision about where, when, and how to offload the tasks of the mobile application. However, this framework used a discovery service to obtain the hardware information of the cloud resources every minute, thereby consuming additional energy. In addition, the transferred data were not protected from attacks.

An iterative algorithm is proposed in [18], integrating resource scheduling policy and dynamic offloading to minimize the energy efficiency cost by the mobile device for completing the application. The authors considered completion time deadline and task-precedence as the main constraints in its model. This algorithm comprised three main parts, namely, computation offloading selection, CPU clock frequency control in local computing, and transmission power allocation in cloud computing. However, this framework did not consider memory usage [19] as a constraint in the offloading decision and apply any security technique to protect the transferred data from attacks.

Recently, the reduction of the total energy consumption while satisfying the reliability and time constraints are explored in [20]. The study proposed an energy-aware dynamic task scheduling algorithm, which used directed acyclic graph (show the task precedence and its communication cost) and critical path assignment approach to acquire the optimal execution order of each task that minimized the overall energy consumption. However, this model focused only on energy consumption metric and did not address other major metrics, such as memory usage, CPU utilization, and remaining battery life, which are considered as important metrics.

Taking into consideration all of the mentioned work, other works considered memory usage constraints in their models. However, no security technique was applied to protect the offloaded data to the cloud. In this paper, we formulate a model that handles four different constraints, namely, memory usage, execution time, CPU utilization, and energy consumption, in the offloading decision. This model made the offloading decision dynamically at runtime. In addition, we provided this framework with a new security layer to protect the offloaded data to the cloud. The proposed framework is tested with three different types of mobile applications that were developed using Android [21], [22].

3 Framework Architecture and Design

In this section, we explain the architecture of the framework and show how its modules can communicate to achieve the design goals of the system. In addition, the linear optimization model is defined. A detailed determnation of the offloading decision is also showed. Then, we provide an algorithm that clarifies how this framework works. Finally, we identify the required steps to add this framework during development.

3.1 Framework Architecture

As shown in Fig. 1, the framework architecture consists of six modules, namely, estimator, profile, network and bandwidth monitor, decision maker, mobile manager, and cloud manager.

First, the framework works at the method level, where the developers need to add an annotation (@Remote) above all intensive methods at the developing step. These methods should require additional computation and can be offloaded to the cloud for remote execution. These methods must not a) *depend on the user interface* or b) *use any I/O mobile device such as GPS, camera, or accelerometer*. Thereafter, at the installation step, a binary file containing these method codes and its related libraries is sent to the cloud. This section will be discussed later.

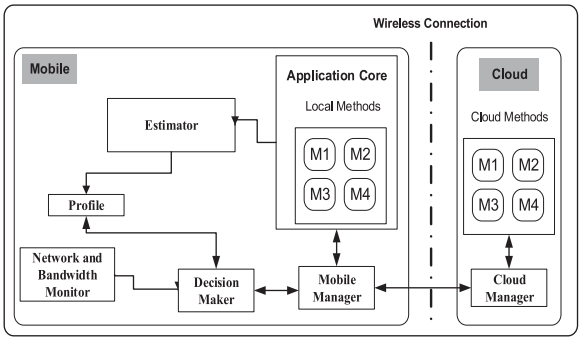

Fig. 1. Framework architecture.

Estimator: The estimator module is responsible for identifying these methods for local execution on the mobile device and remote execution on the cloud with different input sizes (stored as a sample) at the installation step. Then, the module obtains the values of execution time, memory usage, CPU utilization, and energy consumption for each annotated method for these different input sizes (minimal eye application is used to measure the energy consumption and CPU utilization). Finally, the values are communicated and sent to the profile module.

Profile: The profile module obtains the values of execution time, memory usage, CPU utilization, and energy consumption from estimator module for each annotated method. Then, the module creates a new file for each method and stores these values into the file. These files are updated after each running process and used by the decision maker module as a history-based file in the offloading decision.

Network and Bandwidth Monitor: This module only monitors the current status of the network and gathers cell connection state and its bandwidth, Wi-Fi connection state and its bandwidth, and signal strength of cell and Wi-Fi connection (get this information using programming code). Then, this information is sent to the decision maker module to support the determination the offloading decision.

Decision Maker: The decision make, that is, the core module of the proposed framework, contains an integer linear programming model and decision-making algorithm that predicts at runtime where the annotated methods are executed. The goal of the model is to find an application partitioning strategy that minimizes the energy consumption, transfer data, memory usage, and CPU utilization, in smartphones, subject to certain constraints. The decision maker also takes into account all data collected from the profile and network and bandwidth monitor modules.

Let us assume that we have n number of annotated methods that may be offloaded to the cloud for remote execution, that is, $M1, M2, \ldots, Mn$. Each method i consists of a set of parameters, namely, input size ($input_i$), memory usage ($memo_i$), CPU utilization (CPU_i), and battery consumption ($power_i$), for local execution. Other parameters, such as memory used for security ($memo\_sec_i$), battery consumption used for security ($power\_sec_i$), and CPU used for security (CPU_sec_i) are also considered if the method is offloaded for remote execution. In this model, x_i is introduced for each method i, which indicates whether the method is executed locally on the mobile device ($x_i = 0$) or offloaded for remote execution ($x_i = 1$).

The objective function is represented as follows:

$$\min_{x\in 0,1}(c_{transfer} * w_{tr} + c_{memory} * w_{mem} + c_{CPU} * w_{CPU} + c_{power} * w_{power}),$$

Where:

$$\begin{aligned} C_{transfer} &= \sum_{i=1}^{n} input_i * x_i \\ C_{memory} &= \sum_{i=1}^{n} memo_i * (1 - x_i) + \sum_{i=1}^{n} memo\_sec_i * x_i \\ C_{CPU} &= \sum_{i=1}^{n} CPU_i * (1 - x_i) + \sum_{i=1}^{n} CPU\_Sec_i * x_i \\ C_{power} &= \sum_{i=1}^{n} power_i * (1 - x_i) + \sum_{i=1}^{n} power\_sec_i * x_i, \end{aligned} \tag{1}$$

$C_{trensfer}$, C_{memory}, C_{CPU}, and C_{power} represent cost for transferring the input size, memory used, CPU used, and power consumed for method i respectively. w_{tr}, w_{memo}, w_{CPU}, and w_{power} are the weights for each these costs, which lead to different objectives.

According to the objective function in (1), the three constraints that must be handled with care are as follows:

Minimize the memory used by the application methods on the mobile device. Memory cost is divided into two parts. The first part is the memory used when the method of the application is executed locally on the mobile device, whereas the second part is used to encrypt the data before transferring to the cloud in the offloading case. This memory cost must not exceed the available memory on the mobile device. The method is restricted to a threshold value, which is determined based on the fraction between number of running applications and the total size of memory. This constraint can be written as follows:

$$\sum_{i=1}^{n} memo_i * (1 - x_i) + \sum_{i=1}^{n} memo\_sec_i * x_i \leq M_{th}, \tag{2}$$

where M_{th} is the memory threshold.

Minimize the total execution time, that is, the second constraint, for the application. The total time for executing the application methods remotely on the cloud must be less than the total time for executing the methods of the application locally on the mobile device. Let S_M and S_C be the processor speeds (instruction per second) of the mobile and the cloud, respectively, and C be the number of instructions involved in the method invocation. C_{sec} represents the number of instruction to encrypt the data prior to transfer to the cloud. If the amount of data required for the method execution is D and the network bandwidth is B, then the time required to transfer this data is D/B. Therefore, the total time for executing methods on the cloud is divided into the following three parts: the time consumed by encrypting the data, data transmission, and the time execution on the cloud. This constraint can be represented as follows:

$$Exe\_time\_local > Exe\_time\_cloud, \tag{3}$$

where *Exe_time_local* is total time for the local execution of the method, as calculated as follows:

$$Exe\_time\_local = \frac{C}{S_M} * x_i. \quad (4)$$

Exe_time_cloud is total time for offloading the method for remote execution, as calculated as follows:

$$Exe\_time\_cloud = \left(\frac{C_{\text{sec}}}{S_M} + \frac{C}{S_C} + \frac{D}{B} + t_{overhead}\right) * x_i. \quad (5)$$

Where $t_{overhead}$ is the overhead time of our framework.

Minimize the total energy consumption, that is, the last constraint for the objective function. This constraint deals with the energy consumed by executing the application method. Therefore, the energy consumed by offloading the application methods for remote execution must be less than the energy consumed by executing the application method locally on the mobile device. This constraint can be represented as follows:

$$Energy\_con\_local > Energy\_con\_cloud. \quad (6)$$

If the mobile consumes P_M Watts (W) for method computation locally, P_d W for being idle, P_{sec} W for encrypting the data of the method, and P_r W for transferring to the cloud, then the total energy consumed locally on the mobile device can be calculated as follows:

$$Energy\_con\_local = P_M * \frac{C}{S_M} * x_i. \quad (7)$$

The total energy consumed for remote execution on the cloud can be calculated as follows:

$$Energy\_con\_cloud = \left(P_{Sec} * \frac{C_{sec}}{S_M}\right) + \left(P_d * \frac{C}{S_C}\right) + \left(P_r * \frac{D}{B}\right) * x_i. \quad (8)$$

Finally, after we solve this formulation, each method x_i can be determined whether for local execution ($x_i = 0$) or offloading to the cloud ($x_i = 1$). In the experiments, particle swarm optimization (PSO) java code is used to implement the linear model. Particle swarm optimization is a heuristic global optimization method and an optimization algorithm based on swarm intelligence [23]. PSO is similar to the Genetic Algorithm (GA) and Ant Colony Optimization (ACO) in the sense where they are population-based search methods. But GA and ACO are considered as expensive computational cost compared with PSO [24], [25] and we need to minimize the overall consumption, therefore, PSO is used as best optimization algorithm with significantly better computational efficiency to solve our optimization problem. PSO provides a solution with acceptable speed in the order of tens of milliseconds.

Mobile Manager: The mobile manager module is responsible for sending a binary file containing the method code and its required libraries at the installation step. The mobile manager handles the execution of the method based on the model decision. If the method is executed locally on the mobile device, the files are updated with new values through the profile module. However, if the decision is to offload the method, then the mobile manager encrypts the offloaded data by using AES technique and communicate with the cloud manager module to transfer this data with the method name. Finally, the mobile manager receives and delivers the results to the application. In case of remote execution failure, the mobile manager module executes the method locally.

Cloud Manager: The cloud manager module is the only module deployed on the cloud side. This module is written purely in Java. Therefore, any application can benefit from the proposed framework to offload its computation to any resource that runs the Java Virtual Machine (JVM). Communication between the cloud manager and the mobile manager modules is managed by Ibis communication middleware [26]. In the first communication, at the installation step, a binary file containing the method code and its required libraries are sent to the cloud. Then, the cloud manager receives the methods data and decrypt them in the following run. Then, the manager executes the method remotely and sends the result back to the mobile manager module with the new values to be updated by the profile module.

Algorithm 1. Framework execution flow

Input: input size, memory usage, CPU utilization and energy consumption for each annotated method.
Output: execution place and result for each method.

```
Read annotated methods name.
Check the current network status using Network & Bandwidth monitor Module.
if there isn't connection then
    Execute the method locally on the mobile device.
else
    for each method i do
        Read C_trensfer, C_memory, C_CPU and C_power through the profile module.
        Solve the optimization model in and determine the offloading decision.
        if the decision is offloading then
            Encrypt the method data using AES Algorithm.
            Send it to the cloud for remote execution.
            Return Result back to the mobile device (communication managed using Mobile Manager & Cloud Manger Modules).
        else
            Execute the method locally on the mobile device.
        end if
    end for
end if
Update the profile file with new values.
```

3.2 Framework Execution Flow

This section discusses the flow of execution of the proposed framework. Algorithm 1 illustrates the detailed processes of the framework and how the offloading decision for the annotated method is made. The time complexity for this algorithm is represented by **O(n)** and does not consume additional resources from the mobile device. First, at the developing step, the mobile application is partitioned the methods into two types. The first type is the computing-intensive methods which are annotated by developer and requires more computation resources. While the second one is the methods which depend on the device hardware and must be executed locally as discussed above. Then, at the execution step, the decision maker module reads the

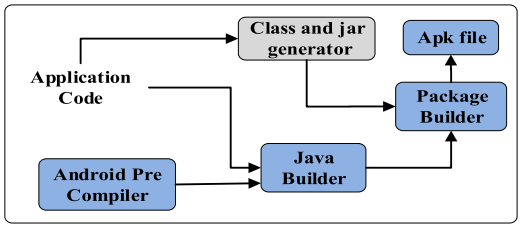

Fig. 2. Integration between builders in the building process.

annotated method name (first type of methods) and checks the network status using network and bandwidth monitor module while the application is running. When no connection or failed connection occurs, then all of the methods are executed locally on the mobile device; otherwise, the decision maker module reads the transfer data size, memory usage, CPU utilization, and energy consumption through the profile module, where these values are stored at the installation step and updated during each run. Then, the optimization model is solved. The optimization model determines which method is executed locally on the mobile device and which method is offloaded for remote execution. In the offloading case, the data of the methods are encrypted using AES algorithm and then transferred to the cloud for execution through mobile manager and cloud manager communication. Finally, whether the method is run locally or remotely, the profile file is updated with new values according to the running process.

3.3 Integrating the Framework in Mobile Application

In this section, we clarify the required steps to integrate the proposed framework in the mobile application. Our framework works on the method level and uses a Java reflection and annotation to identify the methods which can be offloaded.

First, at the developing step, the developer needs to add an annotation (@Remote) above each intensive method that can be offloaded for remote execution. Thereafter, each Android application goes through three main builders to generate the APK installation file as shown in Fig. 2. The first builder is the Android Pre-Compiler, which generates the Java source files from your Android resources and Java source files for any service interface. Second, the Java Builder compiles the generated files from the first builder. Last, the package builder obtains all compiled files and packages them in an APK file. Our framework adds a new additional builder, called the Class and Jar Generator. This builder creates a class that contains all annotated methods code that may be executed remotely and the related libraries required to execute these methods on the cloud. Then, the builder generates a binary file from created class and libraries, which is sent to the cloud at the installation step. The result from this builder is combined with the result from the Java Builder and embodied in an APK file through the Package Builder.

4 Evaluation and Analysis

The proposed framework is evaluated using three different types of mobile applications, as shown in Table 1. The experimental results measure four parameters for running the application methods locally on a mobile device and when offloading the methods to the cloud by using the framework. These parameters include processing Time, CPU utilization, battery consumption, and memory usage. The evaluation shows how these applications can benefit from the proposed framework for performance improvement.

TABLE 1
Application used in experimental

Application	Description
Face Detection	Detect faces from an image and draw a rectangle on each face detected.
Gaussian Blur	Blurring an image by a Gaussian function.
Quick Sort	Sort a given set of integer array elements by using Quicksort

4.1 Experimental Setup

The experimental setup for testing the proposed framework is composed of a Samsung Galaxy S Plus mobile device, a server node, and a Wi-Fi wireless network of radio type 802.11g. The Samsung Galaxy S plus GT-I9001 runs on Android platform 4.4.2 with Qualcomm MSM8255T CPU, 512 MB memory, a battery capacity of 1650 mAh at 3.7 volts integrated with a Wi-Fi interface. The server node runs Microsoft Windows 7 Ultimate 64-bit operating system with Intel Core(TM) i5-2500 CPU with 2.4 GHz frequency, 4 GB RAM capacity, 600 GB hard disk, and 100 Mbps network interface. The mobile device accesses the wireless network via Wi-Fi wireless network connection of radio type 802.11g, with the available physical layer data rate of 54 Mbps. In the evaluation, little eye V2.4[1] software is used to measure processing time, CPU utilization, battery consumption, and memory usage.

4.2 Experimental and Evaluation Results

The application methods may need data as input for execution. Data are transferred over the network and stored on the cloud side if this method is offloaded for execution. Therefore, these data are vulnerable to attacks. Cryptographic algorithms are needed to ensure the security of data and communications [27]. Cryptographic algorithms are classified as symmetric key algorithms and asymmetric key algorithms. Symmetric key algorithms [28], [29], also known as single key, use a private (shared secret) key to execute encryption and decryption process, whereas asymmetric key algorithms [28], [29], also known as public key, use a public (shared) key to execute encryption and uses other private key decryption processes. The most well-known symmetric algorithms are DES, TDES, AES, RC6, Twofish, Blowfish, Serpent, and MARS, whereas RSA, DSA, PGP, SSH, and SSL are the well-known asymmetric algorithms.

Given that symmetric algorithms are faster and consume less energy compared with asymmetric algorithms, symmetric key algorithms are selected to ensure the security of this data and consumption of less energy. The AES algorithm is selected and used as encryption technique to protect the transferred data of the applications [30].

In the evaluation, the proposed linear programming model of the framework is applied to the three applications, which select the correct decision for all applications. In the

1. Little eye: http://www.littleeye.co/

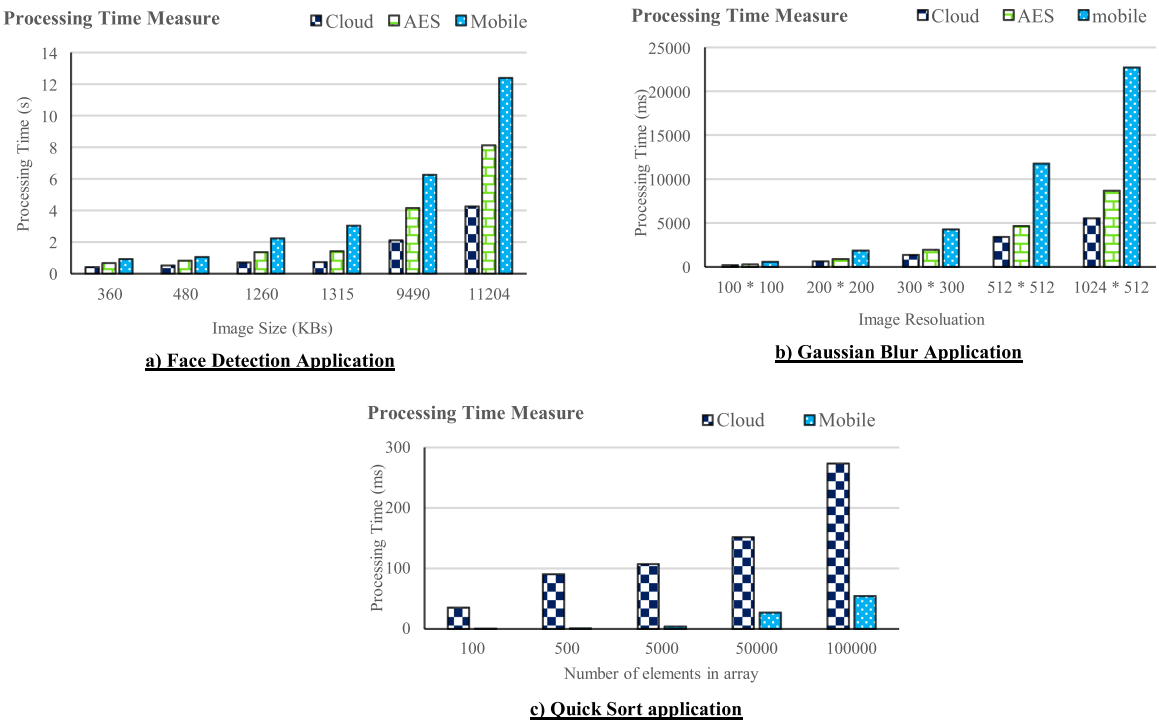

Fig. 3. Processing time measure for running the three applications.

experimental work, three different scenarios are used to prove this result. In the first scenario, the application methods are executed locally on the mobile device. In the second scenario, the application methods are offloaded for execution on the cloud by using the proposed framework model without applying any security technique. In the last scenario, the application methods are offloaded for execution on the cloud by using AES algorithm to protect the transferred data over the network and to show the effects of adding this layer on the framework.

In the proposed framework, the linear model is used to determine the offloading decision and selects a correct decision for the three applications when solving for the parameters of the methods. However, the framework is modified to allow the quick-sort application to run remotely on the cloud to prove our result.

In the experimental work for the face detection application, six images with 360, 480, 1260, 1315 KB and 9 and 11 MB sizes are used. Five images with 100×100, 200×200, 300×300, 512×512, and 1024×512 resolutions are used for the Gaussian blur application. Five arrays of 100, 500, 5000, 50000, and 100000 elements are used in the quick-sort application. Each application is executed 20 times for each input, and the average values are obtained.

Fig. 3 shows the results of the processing time measure for running the three applications. Face detection and Gaussian blur applications are executed in three different scenarios. The quick sort application is executed using only two scenarios because additional time is needed for offloading to the cloud. Therefore, the addition of a security layer is not needed when a local execution decision is advantageous. Figs. 3a and 3b show the results of processing time for face detection and Gaussian Bluer applications. Fig. 3c shows the results of processing time for the quick-sort application. In Fig. 3a, image size in KBs is represented in the x-axis, and processing time in seconds is represented in the y-axis. In Fig. 3b, image resolution is represented by the x-axis, and processing time in milliseconds are represented by the y-axis. Finally in Fig. 3c, the number of elements in the array is represented by the x-axis, and processing time in milliseconds are represented in the y-axis. As shown in Figs. 3a and 3b, the average processing time for executing the face detection method or Gaussian blur method in the absence of the proposed framework is 5–8.5 s. The average processing time when the applications are executed using the proposed framework is 1.5–2.5 s without any security and about 2.8–3.5 s after using AES as security algorithm for encrypting the transferred data to the cloud. Therefore, the framework can save 65–73 percent time for running the applications on the cloud by using the proposed framework. However, Fig. 3c, local execution of the quick-sort method on the mobile device requires the shortest processing time because face detection and Gaussian blur applications need time for data computation and task completion. Therefore, use of the cloud to execute these tasks is an advantage, whereas a quick-sort application that involves a simple task can be advantageously run locally on the mobile device.

Fig. 4 shows the results of CPU utilization measure for running the three different applications. As shown in the figure, if we execute the application locally without using our framework, the smartphone applications utilizes an 30 percent of the CPU on average (except for the quick-sort application because it is a simple task that takes more processing for offloading than executing locally). If the proposed framework is used, the smartphone applications utilize an average of 7 percent of the CPU without any security layer and approximately 12 percent with AES security layer is added. The variation in CPU utilization increases for the application that requires large computation and small data for transfer.

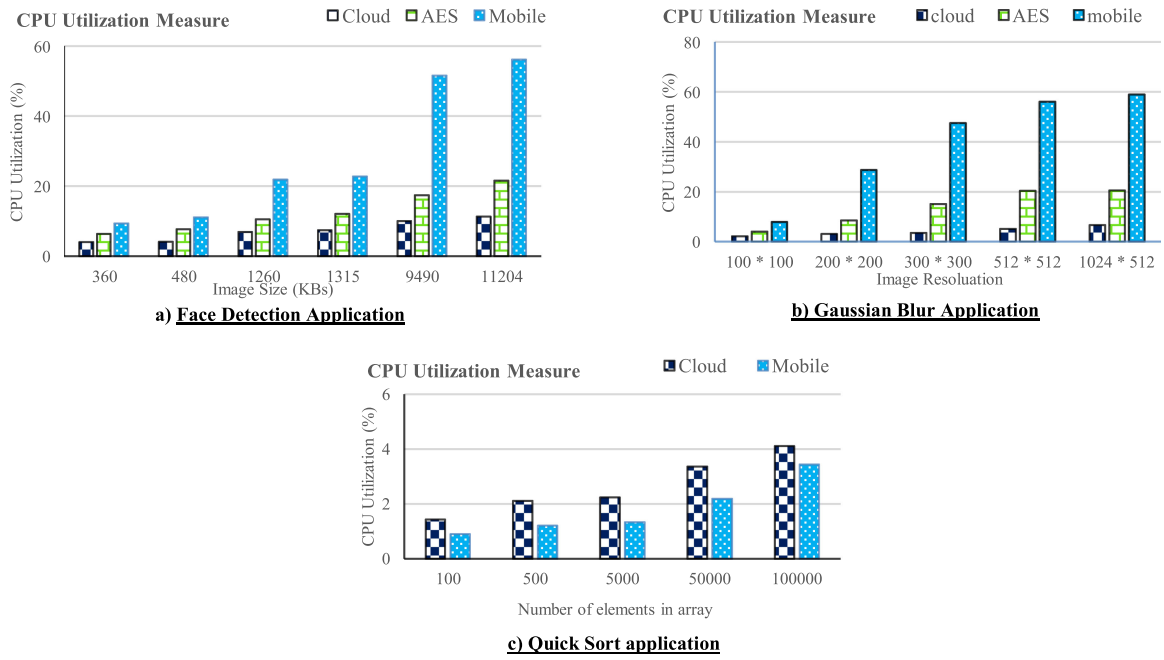

Fig. 4. CPU utilization measure for running the three applications.

Similarly, Fig. 5 shows the results of memory usage measurement for executing the three different applications. As shown in the figure, the average memory usage for executing face detection and Gaussian blur applications are 50 and 26 MB respectively. However, the ratios decrease to 35 and 20 MB without security and 38 and 24 MB when using AES as a security layer to protect the transferred data. Therefore, the proposed framework can improve more than 30 percent of the memory usage ratio for face detection application and 10 percent for executing Gaussian Blur on the cloud. However, in Fig. 5c, the mobile memory is conserved if the quick-sort application runs locally on the mobile device.

Finally, Fig. 6 shows the results of battery consumption of the mobile for running the three different applications. According to the results in Figs. 6a and 6b, face detection and Gaussian blur applications consume less energy when executed with our framework because resource-intensive computational tasks are offloaded to the cloud, thereby reducing computational overhead on the smartphone.

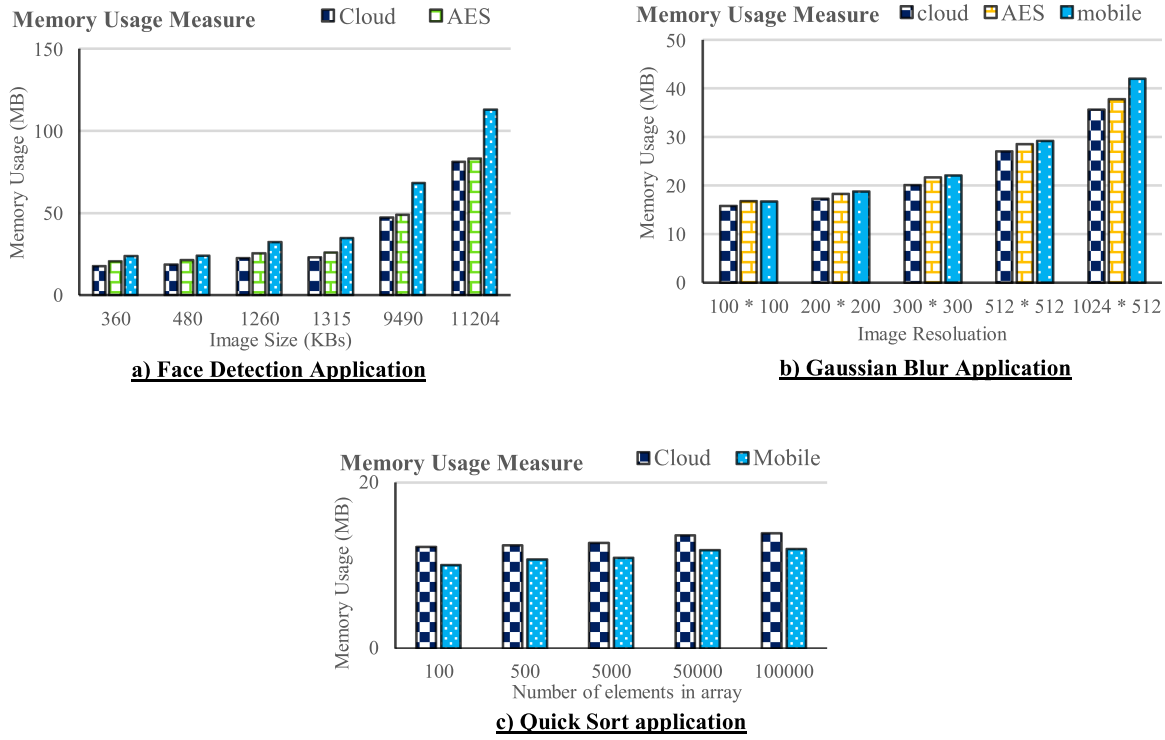

Fig. 5. Memory usage measure for running the three applications.

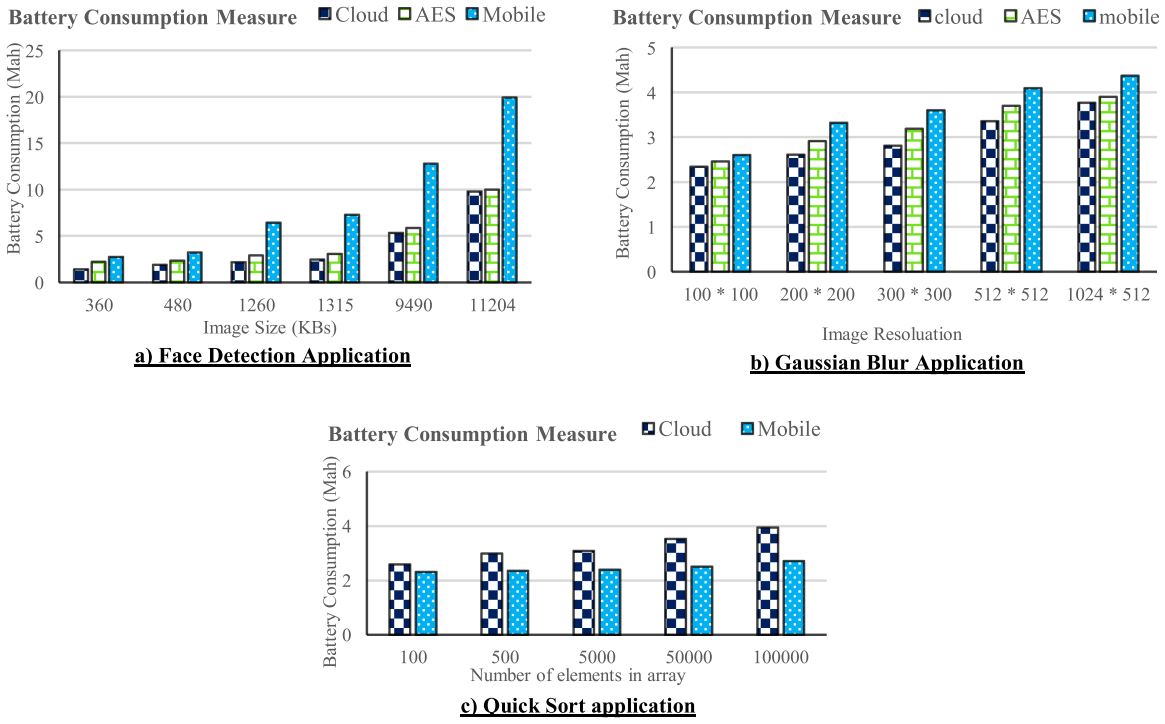

Fig. 6. Battery consumption measure for running the three applications.

However, for improved energy efficiency, the applications must not consume high energy in terms of communications (for offloading and transmitting data); otherwise, computation offloading may not be beneficial. For instance, offloading the quick-sort task is not energy efficient because the task overconsumes the smartphone energy, as shown in Fig. 6c.

In general, the results show that the proposed framework saves considerable mobile resources, such as processing time, CPU utilization, memory usage, and battery consumption.

5 Conclusion

In this paper, a novel secured, optimized framework is proposed to improve the efficiency of offloading computation from the mobile device to the cloud. This framework can offload only the application methods that consume substantial mobile resources. The offloading decision is made using a formulated 0–1 integer linear programming model. This decision is made dynamically at runtime based on four constraints, namely, memory usage, CPU utilization, energy consumption, and execution time. The framework also adds a new security layer, which uses an AES technique to protect the methods data before transferring to the cloud in the offloading case. The evaluation results proved that the proposed framework can improve mobile application performance by reducing consumption in mobile resource, such as processing time, battery consumption, CPU utilization, and memory usage. This study also shows how the proposed algorithm can select suitable offloading decisions. Finally, we conclude that executing intensive methods of mobile applications remotely on the cloud by using the proposed framework conserves mobile resources, especially if the application needs high computation and few data to transfer. In the future, we need to apply the same model on the edge computing server instead of central cloud computing to reduce latency. In addition, we need to enable parallelization for the execution of the method on the cloud to potentially reduce execution time and energy consumption.

Acknowledgments

This work is supported by the National Key Research and Development Plan under grant No. 2016YFB0800801, the National Science Foundation of China (NSFC) under grant No.61672186, 61472108. Professor Zhang is the corresponding author.

References

[1] N. Vallina-Rodriguez and J. Crowcroft, "Energy management techniques in modern mobile handsets," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 179–198, Jan-Mar. 2013.
[2] B. Sosinsky, *Cloud Computing Bible*. Hoboken, NJ: Wiley, 2010.
[3] G. Motta, N. Sfondrini, and D. Sacco, "Cloud computing: An architectural and technological overview," in *Proc. Int. Joint Conf. Serv. Sci.*, vol. 3, 2012, pp. 23–27.
[4] D. Kovachev, Y. Cao, and R. Klamma, "Mobile cloud computing: A comparison of application models," *CoRR*, vol. abs/1107.4940, 2011.
[5] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 393–413, 2014.
[6] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1294–1313, Jul.-Sep. 2013.
[7] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proc. Conf. Comput. Syst.*, 2011, pp. 301–314.
[8] S. Kosta, A. Aucinas, P. Hui, and R. Mortier, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, 2012, pp. 945–953.
[9] W. Zhang, S. Han, H. He, and H. Chen, "Network-aware virtual machine migration in an overcommitted cloud," *Future Generation Comput. Syst.*, vol. 76, pp. 428–442, 2016.
[10] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Proc. Int. Conf. Mobile Comput. Appl.*, vol. 76, 2010, pp. 59–79.

[11] 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. Int. Conf. Mobile Syst. Appl. Serv.*, 2010, pp. 49–62.
[12] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl.*, 2012, pp. 784–791.
[13] F. Xia, F. Ding, J. Li, X. Kong, L. T. Yang, and J. Ma, "Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing," *Inf. Syst. Frontiers*, vol. 16, no. 1, pp. 95–111, 2014.
[14] W. Zhang, Y. Wen, K. Guan, K. Dan, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.
[15] M. Shiraz and A. Gani, "A lightweight active service migration framework for computational offloading in mobile cloud computing," *J. Supercomputing*, vol. 68, no. 2, pp. 978–995, 2014.
[16] M. Shiraz, A. Gani, A. Shamim, S. Khan, and R. W. Ahmad, "Energy efficient computational offloading framework for mobile cloud computing," *J. Grid Comput.*, vol. 13, no. 1, pp. 1–18, 2015.
[17] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, and S. N. Srirama, "A context sensitive offloading scheme for mobile cloud computing service," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2015, pp. 869–876.
[18] S. Guo, B. Xiao, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.
[19] W. Z. Zhang, H. C. Xie, and C. H. Hsu, "Automatic memory control of multiple virtual machines on a consolidated server," *IEEE Trans. Cloud Comput.*, vol. 5, no. 1, pp. 2–14, Jan.-Mar. 2017.
[20] Y. Li, M. Chen, W. Dai, and M. Qiu, "Energy optimization with dynamic task scheduling mobile cloud computing," *IEEE Syst. J.*, vol. 11, no. 1, pp. 96–105, Mar. 2017.
[21] M. Burton and D. Felker, *Android Application Development for Dummies*. Hoboken, NJ, USA: Wiley, 2012.
[22] M. Murphy, *Beginning Android 2*. New York, NY, USA: Apress, 2010.
[23] D. P. Rini and S. M. Shamsuddin, "Particle swarm optimization: Technique, system and challenges," *Int. J. Comput. Appl.*, vol. 1, no. 1, pp. 33–45, 2011.
[24] B. Booba and T. V. Gopal, "Comparison of ant colony optimization and particle swarm optimization in grid scheduling," *Australian J. Basic Appl. Sci.*, vol. 297, no. 4, pp. 230–235, 2014.
[25] W. O. H. Rania and C. Babak, "A comparison of particle swarm optimization and the genetic algorithm," in *Proc. Struct. Dyn. Mater. Conf.*, 2004, pp. 833–843.
[26] R. V. V. Nieuwpoort, J. Maassen, G.;ska, R. F. H. Hofman, C. J. H. Jacobs, T. Kielmann, and H. E. Bal, "Ibis: A flexible and efficient java? based grid programming environment," *Concurrency Comput. Practice Experience*, vol. 17, no. 7/8, pp. 1079–1107, 2005.
[27] J. Toldinas, R. Damasevicius, A. Venckauskas, T. Blazauskas, and J. Ceponis, "Energy consumption of cryptographic algorithms in mobile devices," *Elektronika Ir Elektrotechnika*, vol. 20, no. 5, pp. 158–161, 2014.
[28] A. Jeeva, D. V. Palanisamy, and K. Kanagaram, "Comparative analysis of performance efficiency and security measures of some encryption algorithms," *Int. J. Eng. Res. Appl.*, vol. 2, no. 3, pp. 3033–3037, 2014.
[29] M. Ebrahim, S. Khan, and U. B. Khalid, "Symmetric algorithm survey: A comparative analysis," *Int. J. Comput. Appl.*, vol. 61, no. 20, pp. 12–19, 2014.
[30] I. A. Elgendy, M. El-Kawkagy, and A. Keshk, "An efficient framework to improve the performance of mobile applications," *Int. J. Digit. Content Technol. Appl.*, vol. 9, no. 5, pp. 43–54, 2015.

Ibrahim Elgendy received the MSc degree from Computer Science Department, the Faculty of Computers and Information, Menoufia University, Egypt, in 2016. He is working toward the PhD degree from the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. He worked as a demonstrator and assistant lecturer in the Faculty of Computers and Information, Menoufia University, Egypt since April 2012 till now. His research interests include cloud computing, mobile edge computing and distributed computing.

Wei-Zhe Zhang is currently a professor with the School of Computer Science and Technology, Harbin Institute of Technology, China. His research interests include primarily in parallel computing, distributed computing, cloud and grid computing, and computer network. He has published more than 100 academic papers in journals, books, and conference proceedings. He is a member of the IEEE.

Chuanyi Liu received the PhD degree in computer science and technology from Tsinghua University, Beijing, China, in 2009. His research interests include computer architecture, cloud security, and data protection. He spent one year as a visiting scholar with the Digital Technology Center of the University of Minnesota, Minneapolis, USA. He is now associate professor of the Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China. He has published more than 30 papers in related conferences and journals. He is a member of the IEEE.

Ching-Hsien Hsu is currently a professor with the Department of Computer Science and Information Engineering, Chung Hua University, Taiwan. His research interests are primarily in parallel and distributed computing, cloud and Grid computing, P2P computing, RFID, services computing, and smart homes. He has published more than 150 academic papers in journals, books, and conference proceedings. He was awarded the annual outstanding research award in 2005, 2006, 2007, and 2010, and a distinguished award in 2008 for excellence in research from Chung Hua University. He serves on a number of journal editorial boards. He has edited more than 20 international journal special issues as a guest editor and has served many international conferences as a chair or committee member. He serves on the executive committee of the IEEE Technical Committee on Scalable Computing (TCSC) and is a senior member of the IEEE.