A Cloud-Side Decision Offloading Scheme for Mobile Cloud Computing

Hamid Jadad, Abderezak Touzene, Khaled Day, and Nasser Alzeidir

Abstract—**Mobile code offloading is one of the techniques that are used to migrate computation-intensive tasks of mobile applications from mobile devices to the cloud. Many offloading systems make online offloading decisions to ensure that offloading provides significant gain in battery saving and execution time. However, this offloading decision process produces a significant overhead when it is implemented in the mobile device. We call these approaches device-side approaches. In this paper, we propose a new offloading approach that shifts the offloading decision process components to the cloud. In other words, the offloading decision is made in the cloud and we call it cloud-side offloading approach. The evaluation results show that the new cloud—side's approach saves around 20% of energy and execution time compared to the device-side approach.**

Index Terms—**Mobile cloud computing, offloading, cloud-side decision, execution time, energy saving.**

I. Introduction

Mobile cloud computing is a new paradigm that appeared from merging cloud computing and mobility [1]. It allows the mobile users to utilize the cloud services on demand. It is envisioned that this paradigm will help overcoming the limitations of the mobile devices hardware. In [2], the authors have proposed a taxonomy of mobile cloud computing based on the key issues and how they have been tackled in research. One of the key issues is job offloading which consists of migration of jobs (data or code) that takes place from the resource constrained mobile device to the cloud.

Many research studies have explored the process of offloading computation-intensive tasks from mobile to the cloud [3]-[6]. In these works, the offloading decision process components are implemented in the mobile device. For example, in ThinkAir [3] mobile side offloading includes three profilers (hardware, software, and network) which collect the runtime data and feed them into the energy estimation model. The energy estimation model is also implemented inside the ThinkAir energy profiler in the mobile side. It is used to dynamically estimate the energy consumption of each running method. Similar to ThinkAir, many offloading systems [4]-[6] have implemented the profilers; cost prediction models and decisions engine components on the mobile side.

This kind of offloading approaches keeps the offloading decision components running during application execution to make optimal offloading decisions at runtime. However, this process of making an offloading decision produces a significant overhead on the mobile device. This overhead is costly on mobile devices in terms of battery consumption and processing resources [3]. Because the decision is made in the mobile device, we may call these approaches as device-side offloading decision approaches.

In this paper, we address the problem of the overhead of the offloading decision process on the mobile device. We propose a new approach that delegates the offloading decision to the cloud and hence reduces the offloading overhead from the mobile device. Because of the cloud capabilities, we believe that the cloud would perform the offloading in a more efficient way with lower cost.

The rest of this paper is organized as follows: In section II, we discuss related work Section III presents the system architecture. Section IV provides cost prediction models. Section V presents a cloud-side offloading algorithm. Section VI provides the evaluation of the system, followed by a discussion. Finally, we give a conclusion and future work in Section VII.

II. Related Works

The offloading of computation from mobile to the cloud has been investigated in [3]-[6]. They have implemented the offloading decision process components in the mobile device. For instance, ThinkAir [3], implements three profilers in the mobile device to collect data that is forwarded to the energy estimation model which is also located in the mobile side. These profilers and energy estimation models needs to keep running to produce dynamic offloading decisions.

The client side of jade [5] includes the device and program profilers, and an optimizer that uses these profilers to make offloading decisions.

Zhou [7], implements three context profilers and a decision engine in the mobile device.

We call these approaches device-side offloading decision approaches. The offloading decision process components are implemented in the mobile device. The device-side approach produces an overhead on the mobile device in terms of battery consumption and utilizing the hardware resources such as CPU and RAM.

Our approach is different from these approaches by utilizing the cloud capability. Now we can provide more accurate offloading decisions by keeping profilers running without worrying about mobile battery consumption. Moreover, we can run more complex optimizers, which escape the mobile limitations.

Manuscript received May 13, 2018; revised July 7, 2018.

Hamid Jadad, Abderezak Touzene, Khalid Day, Nasser Alziedi are with the Sultan Qaboos University, Muscat, Sultanate of Oman (e-mail: hamidjadad@gmail.com, {touzene, kday, alzidi}@squ.edu.om).

III. System Design Architecture

In this section, we present the design of our system as shown in Fig. 1. The system is divided into two sides: cloud side and mobile side. Each side has its components as follows:

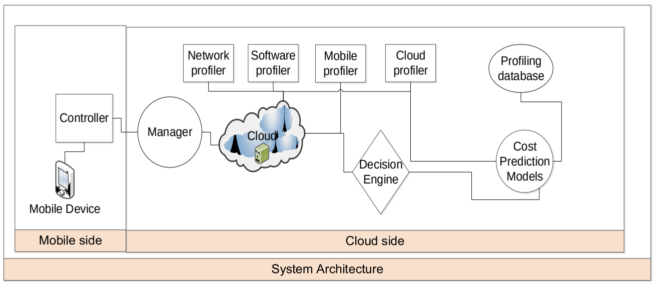

Fig. 1. System architecture.

A. Cloud Side

- Network Profiler: this profiler tracks the connection status, in terms of upload and download throughput between the cloud and the mobile device. The network throughput is used to calculate the time of transferring data in the cost models. This profiler measures the network throughput using the received messages from the mobile that contain the tasks details. This technique will save energy compared to Maui [4] technique which sends a file of 10KB periodically from the mobile to the cloud to measure the network throughput. This technique is costly in terms of energy, especially when the size of the test file is huge. Moreover, it may increase channel congestion in large-scale implementation.
- Software Profiler: this profiler tracks the execution time for each task. This profiler has two main roles. The first one is conducting off-line tests (no load, i.e. background services are stopped and no other applications are running at off-line test) to collect the tasks execution times in both the mobile and in the cloud. For the mobile, the corresponding virtual machine (VM) in the cloud is used to run the off-line test. This VM has similar configurations to the mobile device. Our approach is utilizing the cloud capacity to automate off-line test process, which was manually conducted in MAUI. Then these execution times are stored in the profiling database to be used in the cost estimation models. The second role is estimating the real execution times of tasks during the online execution of the application based on the actual load. We use this gathered data to assess how much our approach is accurate in making offloading decisions that meet design goals, battery saving and low response time.
- Mobile Profiler: the profiler gathers mobile device information such as CPU utilization rate and battery level. For each remotable task, the profiler gets the CPU load and battery level of the mobile. This data is extracted from the mobile request message to offload the task to the cloud. Then, this data is used to calculate the cost to make offloading decisions.
- Cloud Profiler: this profiler collects cloud server's CPU utilization rate. It keeps fresh value of the server CPU load to be used in the cost models. If the load exceeds a defined threshold, it either looks for another server or informs the mobile to execute the task locally. In this case, there is no need to calculate the costs, because the cost of offloading to the cloud in this case will be very high. Because of that, this will reduce the response time.
- Cloud Manager: based on mobile device platform information, the cloud manager creates a corresponding Virtual Machine platform with high processing power as a server in the cloud. This helps our system to be more generic and serves various types of mobile operating systems.
- Profiling Database: to store the execution time of each task during off-line test. The off-line test had been done in both the mobile and the cloud. This will give values that are more real because task-processing time is non-deterministic. The off-line test is done with the assumption that no other loads on the processor. This means no other applications or background services are running during the test. The stored execution time is used in the cost models to calculate the current execution time of the task under the current CPU load.
- Cost Predictor: The runtime data that was collected by the profilers is used here to calculate the total execution cost of the remotable task in both the mobile device and the cloud. Then, the costs will be forwarded to the decision engine to make offloading decision.
- Decision Engine: this engine gets the costs from the cost prediction models and makes a decision to offload the execution of the task or not. The decision is made based on the minimum cost that will reduce the execution time and battery consumption. The decision is send back to the mobile. It is either to offload the task and to send the user input data to the cloud, or to execute the task locally in the mobile.

B. Mobile Side

- Controller: This controller fetches the mobile battery level and CPU utilization rate and forwards them to the cloud. It maintains the connection to the cloud. It also controls the execution of the tasks in both mobile and cloud. This approach implements one controller for all applications, which use this scheme.

C. Fault Tolerance Techniques

One of the main challenges that face distributed systems is fault tolerance. Detecting the system failure and responding to that failure impacts the entire system efficiency. We assume the cloud is available based on the cloud computing definition [3]. Then, the main reason of faults is related to communication link's issues such as network delays. Our approach uses the timeout technique to detect the failure.

When the decision is made to offload the task to the cloud, the mobile controller sets a timer to the expected completion time of the task. If the timer expires before results get back, the task will be rolled back and executed locally on the mobile. Another technique, when the mobile loses the connection to the cloud server after offloading the task, the task will be reassigned back to the mobile. These techniques will save waiting time and energy.

D. Multi-Platforms Support

The heterogeneity of mobile platforms constitutes a challenge for mobile code offloading approaches. The previous offloading systems are platform dependent. For example Jade [5] was built for Android users, and Maui [4] was built for Windows mobile platforms. Our approach uses the cloud services [8] to address this issue. The cloud manger creates a Virtual Machine (VM) in the cloud side as a server for the mobile based on the mobile information received during the initialization phase of the system. For example, if the mobile client system is Android, the manager creates VM for Android platform with high processing capability. This VM contains the application, and the offloading decision components. Thus, our approach is more generic to serve different types of mobile platforms.

IV. Cost Prediction Models

A. Problem Formulation

The aim of this paper is to find the optimal offloading decision that will minimize the execution time and battery consumption using runtime data. This optimization objective is similar to objectives in [7], [9], and [10]. Table I shows the descriptions of notations which are used in the cost models.

TABLE I: Notations

Notation	Description
level	Battery level
BS	Battery Seriousness, Importance of battery to device performance.
$\boldsymbol{EC(T)_m}$	The execution cost in the mobile
$\boldsymbol{EC(T)_c}$	The execution cost in the cloud
$\boldsymbol{ET\ (T)_m}$	The current mobile execution time of task T
$\boldsymbol{ET\ (T)_{m\mathrm{STORED}}}$	the execution time stored in the database of task T.
$\boldsymbol{mload}$	Mobile CPU load.
$\boldsymbol{BC(T)_m}$	The battery consumption on the mobile
$\boldsymbol{B(T)_{proc}}$	The battery consumed by the processor.
$\boldsymbol{B(T)_{Scr}}$	The battery consumed by the mobile screen during the time of executing task *T* locally on the mobile.
$\boldsymbol{BCU_{process}}$	The battery consumption per unit of time during mobile computation
$\boldsymbol{BCU_{transfer}}$	Defines the battery consumption per unit of time while the mobile is transferring data.
$\boldsymbol{t(T)_{tr}}$	Indicates the transmission time of the task *T*.
$\boldsymbol{t(T)_{pro}}$	Defines the processing time for *T* in the cloud.
$\boldsymbol{Data(T)}$	Defines the task data size that will be transferred from the mobile to the cloud.
$\boldsymbol{Result\_size(T)}$	The task result size after processing.
$\boldsymbol{uTP}$	Average uploading throughput.
$\boldsymbol{dTP}$	Average downloading throughput.
$\boldsymbol{RT(T)_c}$	The response time of the task in the cloud.
$\boldsymbol{cLoad}$	Represents the usage percentage of the CPU of the cloud server.
$\boldsymbol{BC(T)_c}$	The battery consumption of task *T* when offloading to the cloud.
$\boldsymbol{BC(T)_{wait}}$	The energy consumed during the waiting time of processing *T* remotely.

The system uses offloading at task level [11]. The task is defined as a consecutive statement segment. For example, C++ program, which consists of functions, the task, will be defined as a function. We assume tasks are being offloaded to be independent. For each task T that is annotated to be executed on the cloud, the system finds the execution cost in the mobile $EC(T)_m$ and the execution cost $EC(T)_c$ at the cloud. The proposed normalized expressions for estimating the costs are given by:

$$EC(T)_m = (1 - BS)\frac{ET(T)_m}{ET(T)_m + ET(T)_c} + BS\ \frac{BC(T)_m}{BC(T)_m + BC(T)_c} \qquad (1)$$

$$EC(T)_c = (1 - BS)\frac{ET(T)_c}{ET(T)_m + ET(T)_c} + BS\ \frac{BC(T)_c}{BC(T)_m + BC(T)_c} \qquad (2)$$

where $ET(T)_m$ defines the execution time of task T in the mobile. $ET(T)_c$ is the execution time of task T in the cloud. BC is the battery consumption to execute T. $BC(T)_m$ defines the battery consumption of the task T in the mobile. $BC(T)_c$ indicates the battery consumption of task T in the cloud. BS indicates the importance of the battery consumption to the device performance. It is set to be inversely proportional to the battery level. In [7] and [12], the authors proposed BS as a static value (0.5), which in reality should be dynamic to take into consideration the battery level. In our approach, we make BS dynamic which is more realistic to reflect the effect of the battery consumption on the mobile performance based on the battery level.

$$BS = \frac{100 - level}{100} \qquad (3)$$

B. Mobile Execution Cost Prediction

To find the mobile execution cost, we retrieve the task execution time from the profiling database. The application was tested offline by the Software Profiler and its execution time was stored in the profiling database. Thus, the current mobile execution time $ET\ (T)_m$ of task T is given by dividing the execution time stored in the database $ET\ (T)_{m\mathrm{STORED}}$ by 1 minus the CPU load of the mobile.

$$ET\ (T)_m\ = ET\ (T)_{m\mathrm{STORED}}\ /(1 - mload) \qquad (4)$$

The prediction of the battery consumption on the mobile device when the task *T* is executed locally is as follows:

$$BC(T)_m\ = B(T)_{proc}\ +\ B(T)_{Scr} \qquad (5)$$

where $B(T)_{proc}$ defines the battery consumed by the processor, and $B(T)_{Scr}$ the battery consumed by the mobile screen during the time of executing task T locally on the mobile. PowerTutor [13] shows that the $B(T)_{Scr}$ stays constant as the screen is ON. This means that $BC(T)_m$ depends mainly on the battery consumed by processing *T*.
The mobile has different battery consumption rates at different states [14]. We define different battery consumption per unit of time (BCU) at different mobile states. We define $BCU_{process}$ as the battery consumption per unit of time during mobile computation; and $BCU_{transfer}$ defines the battery consumption per unit of time while the mobile is transferring data. Therefore, the battery consumption of task T in the mobile is defined as the execution time multiplied by the battery consumption per unit of time as the following formula shows.

$$BC(T)_m\ = BCU_{process}\ \times ET\ (T)_m \qquad (6)$$

The total execution cost of T in the mobile is defined by substituting equations (4) and (6) in equation (1).

C. Cloud Execution Cost Prediction

At the same time of calculating the mobile cost, the system also computes the execution cost of the task *T*, if it is offloaded to the cloud. We start with the response time prediction:

$$RT(T)_c = t(T)_{tr} + t(T)_{pro} \tag{7}$$

where $t(T)_{tr}$ indicates the transmission time of the task T and $t(T)_{pro}$ defines the processing time for T in the cloud.

$$t(T)_{tr} = \frac{Data\ (T)}{uTP} + \frac{Result\_size\ (T)}{dTP} \tag{8}$$

where $Data(T)$ defines the task data size that will be transferred from the mobile to the cloud. $Result\_size(T)$ indicates the task result size after processing. uTP is the average uploading throughput and dTP is the average downloading throughput of the wireless connection. Note that in our approach we do not transfer the task code because we assume that apps codes are already in the cloud. This approach will reduce the offloading network communication time.

Now, the processing time of the current task in the cloud is calculated as follows:

$$t(T)_{pro} = t(T)_{\text{pro\_stored}} / (1 - cLoad) \tag{9}$$

where $t(T)_{\text{pro\_stored}}$ indicates the task processing time retrieved from the database, and $cLoad$ represents the usage percentage of the CPU of the cloud server. The $cLoad$ value is received from the cloud profiler. Based on this information the corresponding stored processing time of the task is divided by 1 minus the CPU load of the server.

From (7), (8) and (9) the response time of the task in the cloud will be as follows:

$$RT(T)_c = \frac{\text{Data } (T)}{uTP} + \frac{\text{Result \_size } (T)}{dTP} + \left(t(T)_{\text{pro\_stored}} / (1 - c\text{Load})\right) \tag{10}$$

The battery consumption of task T when offloading to the cloud depends on the battery consumed to send/receive T to/from the cloud. This is expressed as:

$$BC(T)_c = BC(T)_{tr} \tag{11}$$

As we defined $BCU_{transfer}$ to be the amount of battery consumption per unit of time at transferring state, therefore, the battery consumed during transmitting and receiving the data and the results of T will be:

$$BC(T)_{tr} = BCU_{transfer} \times t(T)_{tr} \tag{12}$$

V. Cloud-Side Offloading Algorithm

When the user starts running the application, at initialization phase, the mobile contacts the cloud and sends its details (mobile ID, platform type, version, Apps ID). The cloud creates the corresponding Virtual Machine (VM), for each remotable task in the corresponding Apps ID; the mobile controller sends the task's details (Apps ID, input data size, battery level, mobile CPU load) to the cloud. The cloud estimates the network throughput using the Round Trip Time (RTT). The cloud sends ICMP massages (ping) to the mobile after receiving mobile's request. The cost models in the cloud get their values from the profilers and the database. These cost models parameters are used to calculate the cost of executing the task in the mobile and the cloud in terms of execution time and battery consumption. The decision engine makes the offloading decision based on the proposed cost model. If the decision is made to offload the task to the cloud, the mobile only sends the user input data as the task code is already in the cloud. Otherwise, the task is executed locally in the mobile.

Algorithm 1 Cloud-side offloading Algorithm

```
$initilization-phase \leftarrow mobile-details(Id, platform, version)$
$VM \leftarrow create-VM(Id, platform, version)$
procedure OFFLOADING-DECISION($Tid, input-size, battery-level, mLoad$)
    $mET-stored \leftarrow fetch-database(Tid)$
    $mEC \leftarrow MobileCostPrediction(Tid, battery-level, mLoad, mET-stored)$

        $throughput \leftarrow network-profiler(T)$

    $cET-stored \leftarrow fetch-database(Tid)$
    $cLoad \leftarrow cloud-profiler(Tid)$
    $cEC \leftarrow CloudCostPrediction(Tid, cLoad, throughput)$
    if $cEC < mEC$ then return $OFFLOAD$
    else return $NOTOFFLOAD$
    end if
end procedure
```

VI. Evaluation

We measured the effectiveness of our new approach by comparing it with our previous work [6] which used a device-side offloading decision approaches.

Simulation settings: There are 1000 tasks where, each task is assigned a random number of instructions between 1 and 10000. The battery level is randomly chosen within the range (5% to 100%) with the assumption that applications cannot run when battery is less than 5%. The size of data to be transferred to the cloud and data returned back are randomly selected from 1 KB to 2MB. This range of data size represents virus scanning, and face recognition applications. Our previous approach [6] and [7] use this range of the data sizes to measure their performance. Thus, we use similar data sizes to compare the performance of our new approach with them. For the WiFi connection, the average upload throughput is randomly selected in the range from 2 Mbps to 5 Mbps, where, the average download throughput is chosen randomly between 5Mbps to 15Mbps. Both the mobile processor load and the cloud server processor load are randomly selected for each task.

As we can see in Fig. 2, the cloud-side approach has lower total task execution time compared to the device-side approach. This reduction is a result of moving the overhead of offloading decision components to the cloud. This illustrates how a more efficient offloading decision operation can further reduce the total execution time.

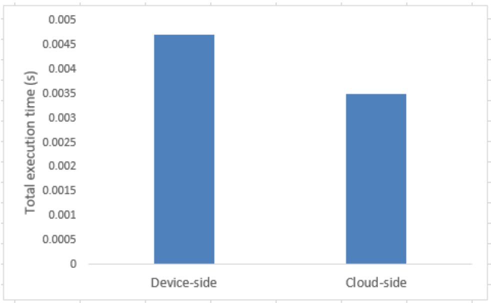

Fig 2. Comparing total execution time between device-side and cloud-side approaches.

With regard to energy saving, Fig. 3 shows that the cloud-side approach saves around 20% of the total energy consumption. This saving is related to moving the overhead of the offloading decision making to the cloud side. Similar to the device-side approach, our approach controls the communication cost between the mobile and the cloud. For example, our approach makes the decision without the need to transfer the user input data. It uses only the size of the input data in the cost models.

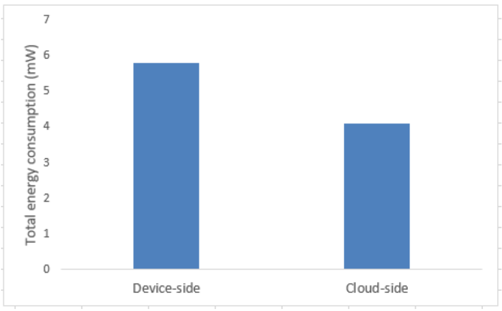

Fig. 3. Comparing total energy consumption between device-side and cloud-side approaches.

VII. Conclusion

The cloud can be used to overcome mobile devices limitations. This paradigm is called mobile cloud computing. Moving the computation of intensive tasks from a mobile to the cloud is known as offloading. The common offloading approach is making an online offloading decision in the mobile device. This approach inflicts an overhead on the mobile device in terms of battery and processing resources in order to make effective offloading decisions. We propose a new approach for code offloading in mobile cloud computing. The approach is cloud-side offloading in-contrast with other approaches, which are device-side based. Our approach is shifting the overhead of the offloading decision process to the cloud. The decision result gets back to the mobile either to offload the task to the cloud or to execute it locally. The evaluation results show significant reduction (around 20%) in the total task execution time and energy consumption. As future work, we plan to improve the approach by adding new optimization techniques using the power of the cloud. These techniques would produce a significant reduction in the execution time and energy consumption.

References

[1] G. Skourletopoulos *et al.*, "Towards mobile cloud computing in 5g mobile networks: Applications, big data services and future opportunities," in *Advances in Mobile Cloud Computing and Big Data in the 5G Era*, C. X. Mavromoustakis, G. Mastorakis, and C. Dobre, Eds. Cham: Springer International Publishing, 2017, pp. 43–62.

[2] G. Lewis and P. Lago, "Architectural tactics for cyber-foraging: Results of a systematic literature review," *J. Syst. Softw.*, vol. 107, pp. 158–186, 2015.

[3] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, 2012, pp. 945–953.

[4] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," *MobiSys'10*, pp. 49–62, 2010.

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

[6] H. Jadad, A. Touzene, N. Alzeidi, K. Day, and B. Arafeh, "Realistic offloading scheme for mobile cloud computing," in *Proc. 13th International Conference on Mobile Web and Intelligent Information Systems*, Vienna, Austria, August 22-24, 2016.

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

[8] M. Lai, J. Wang, T. Song, N. Liu, Z. Qi, and W. Zhou, "VSP: A virtual smartphone platform to enhance the capability of physical smartphone," in *Proc. 10th IEEE International Conference on Big Data Science and Engineering and 14th IEEE International Symposium on Parallel and Distributed Proce*, 2016, pp. 1434–1441.

[9] B. Chun, S. Ihm, and P. Maniatis, "Clonecloud: Elastic execution between mobile device and cloud," *EuroSys '11*, pp. 301–314, 2011.

[10] M. Chen, B. Liang, and M. Dong, "A semidefinite relaxation approach to mobile cloud offloading with computing access point," pp. 1–5.

[11] C. Wang and Z. Li, "Parametric analysis for adaptive computation offloading," *ACM SIGPLAN Not.*, vol. 39, no. 6, p. 119, 2004.

[12] S. Chen, Y. Wang, and M. Pedram, "A semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud," in *Proc. GLOBECOM - IEEE Glob. Telecommun. Conf.*, 2013, pp. 2885–2890.

[13] L. Zhang, R. P. Dick, Z. M. Mao, Z. Wang, and A. Arbor, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *Proc. Eighth IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign Syst. Synth.*, pp. 105–114, 2010.

[14] S. Kafaie, O. Kashefi, and M. Sharifi, "A low-energy fast cyber foraging mechanism for mobile devices," *CoRR*, vol. abs/1111.4, 2011.

Hamid Jadad received B.Sc degree in computer science from Sultan Qaboos University in 2007, M.Sc. degree in internet and computer systems security from University of Bradford in 2013. Now he is a PhD student in Sultan Qaboos University department of computer science. His research interests include mobile cloud computing, mobile applications, cloud security.

Abderezak Touzene received PhD degree in computer science from INPG France in 1992. Now he is an associate professor in Sultan Qaboos University Department of Computer Science. His research interests include mobile networks, network security, and distributed computing.

Khaled Day received the PhD degree in computer science from University of Minnesota, USA in 1992. Now he is a professor in Sultan Qaboos University Department of Computer Science. His research interests include parallel and distributed computing, and networks.

Nasser Alzeidi received the PhD degree in computer science from University of Glasgow, UK in 2007. Now he is an assistant professor in Sultan Qaboos University Department of Computer Science. His research interests include embedded networked and distributed systems, performance evaluation, and mobile Ad hoc networks.