A Code Offloading Framework for Mobile Cloud Computing: ICEMobile

Emre Çalışır
Computer Engineering Department
Galatasaray University
Istanbul, Turkey
e-mail: emrecalisir@gmail.com

Gülfem Işıklar Alptekin
Computer Engineering Department
Galatasaray University
Istanbul, Turkey
e-mail: gisiklar@gsu.edu.tr

B. Atay Özgövde
Computer Engineering Department
Galatasaray University
Istanbul, Turkey
e-mail: aozgovde@gsu.edu.tr

***Abstract*— Smartphones have become a crucial part of our life with their high performance data processing features and ability to access information from anywhere at any time. However, they tend to become inadequate to meet computation intensive operations with their limited battery life and processing capabilities. Mobile cloud computing may be a solution, but Wide Area Network (WAN) latencies and unstable response times of cloud services negatively affect user experiences. In this paper, the cloudlet approach, which offers the cloud services with Local Area Network (LAN) bandwidth, is presented as a possible solution to this problem. A framework, called ICEMobile, is introduced that brings the computation offloading capability to mobile applications. The aim of the ICEMobile framework is to direct application developers in determining which methods need to be offloaded in order to save energy during the mobile execution. The applicability and efficiency of the framework and related optimization model are shown via real life scenarios. The test results reveal that it is possible to save energy up to 98% on the mobile device by using the proposed framework.**

Keywords—mobile cloud computing; cloudlet; code offloading; energy efficiency

I. INTRODUCTION

The evolution of digital world is correlated with the fulfillment of people's expectations that can be summarized as accessing technology from anywhere and anytime, called as ubiquitous computing. It is the method of enhancing computer usage by making many computers available throughout the physical environment, but making them effectively invisible to the user [1]. With the increasing mobility of people and the rise of social media, people's behaviors are changing towards using small and portable personal computers to benefit from vast resources through Internet. On the other hand, the need for intense computation is ever increasing. However, current smartphones are unable to respond these needs because of their limited battery life and CPU power. At this point, cloud computing comes into the scene since it enables retrieving on-demand services from a shared pool of configurable computing resources [2]. Combining ubiquitous computing, mobile computing and cloud computing, a new model, called Mobile Cloud Computing, is developed to bring cloud computing services into the edge to extend resource-rich services on the mobile devices [1].

A cloudlet is a small-scale cloud-like infrastructure, which is located in one hop distance to the mobile user and connected with a high network speed. Cloudlets are fully dedicated to the mobile devices with the aim of executing their resource intensive but latency-sensitive tasks [3]. A mobile device that is connected to a cloudlet benefits from much powerful computing capabilities and unlimited power. These benefits are also valid for distant cloud-based computation; however, in that case it may occur serious time losses due to ambiguous service response times of the cloud and WAN latencies. Therefore, offloading to a distant cloud is not always a solution. The computation or code offloading paradigm means transferring complex tasks to more powerful environments.

In this paper, we develop a framework having code offloading capability and examine three use cases containing computation-intensive operations. The framework involves a decision making engine that analyzes efficiency of the given application. Doing so, we generate the call graph of the program and analyze in detail the time cost of each node and edge. We then test this mechanism with an Android application and a Java based web server, which are connected with the RESTful web services during the execution of three computation-intensive use cases.

The remaining part of the paper is structured as follows: Section 2 presents related works in literature. Section 3 introduces the proposed ICEMobile architecture with its optimization model. In Section 4, experiments and results are given. Finally, the last section presents concluding remarks driven from the test results.

II. LITERATURE REVIEW

The studies in literature that consider the idea of moving the cloud closer have started with the 'cloudlet' concept of Satyanarayanan et al. [3]. With the impact of this novel approach, it has recently become a hot topic in mobile cloud computing related research. In this cloudlet model, the cloudlets are at one hop distant to the mobile device, having high bandwidth wireless access. They may be located in coffees, airports like wireless hotspots to deliver instant services to the mobile clients. The proposed computation offloading techniques in [3] are virtual machine (VM) migration and dynamic VM synthesis. In VM migration, the entire snapshot of the mobile device is transferred to the cloudlet, while in the second technique, there is a dynamic VM synthesis to reduce the size of the VM snapshot by receiving the delta with the previous VM state. The dynamic VM synthesis technique provides offloading capability for each mobile device, since it transfers the VM of the mobile device rather than platform specific objects. On the other hand, it contains many problems especially related with user experience and lack of a decision engine.

 ISBN: 978-1-61208-503-6

Another research, entitled as MAUI, targets to offload only the computation intensive parts of the mobile application with method-level offloading based on the .NET framework [4]. In this research, a system is proposed, which is capable of making the decision of whether offloading will save energy for each offloadable part of the application. The proposed method-level offloading type is a more fine-grained approach than the VM migration or dynamic VM synthesis. In addition to reducing size of the transferred objects, MAUI also provides an intelligent mechanism targeting to reduce the energy consumption on the mobile device by optimally deciding to offload subject to device and server capabilities and network conditions. However, some of the processes in MAUI framework prevent creating dynamic code offloading environments. The first issue is having the necessity of modification on the application since it seems as a disadvantage comparing to [3]. Secondly, there is not any mechanism for automatically identifying the computation intensive parts of the application.

Another research entitled as CloneCloud, focuses on the application partitioning and thread-level offloading [5]. The VM migration mechanism is used in the background of this study by offloading execution blocks of applications from smartphones to their mirror image running on the server. This framework works as a middleware in Android OS, on the top of Dalvik VM. An advantage of the CloneCloud is that it does not require developers' modification.

In addition to the previous studies, a research, called Cuckoo, focuses creating a dynamic offloading decision making tool in runtime in Android OS installed smartphones [6]. It presents a system to offload mobile device applications onto a cloud using a Java stub/proxy model. Cuckoo can be offloaded onto any resource that runs the Java VM. In order to use Cuckoo, the applications need to be re-written such that the application supports remote execution as well as local execution. Moreover, it does not contain any optimization when deciding to offload; instead it always offloads when it connects to the cloud. In a recent related work [1], Zhou et al. propose a context-aware offloading decision algorithm that works on a mobile cloud computing offloading system with multiple cloud resources. Their algorithm takes into account the context changes to select the wireless medium to utilize.

The offloading granularity and optimization mechanism of ICEMobile framework are similar to MAUI's model. However, our approach is typical client/server architecture rather than being a middleware of the device operating system.

III. ICEMOBILE ARCHITECTURE

The focus of our proposed framework ICEMobile is to minimize the energy consumption of the mobile device when computation-intensive functions need to be executed. Doing so, it transfers the resource-intensive code partitions of mobile application to the cloudlet for remote execution. The main purpose is to extend battery life of the mobile device. The energy consumption of the cloudlet is not the concern in this paper, since it is assumed to be continuously fed from the energy sources. The ICEMobile framework architecture is depicted in Fig. 1. It involves Remote Method Invocation (RMI) framework both in the mobile platform and in the cloudlet. In case of cloud-based offloading, it is possible to integrate this framework into the cloud configuration. The advantage of using the nearby cloud in LAN is that it provides higher bandwidth compared to the one in WAN. Besides, it does not require having Internet connection since the cloudlet is ready to satisfy all client needs. In case of need to extra resources, the distant cloud services may be used, but in that case the user experience may be decreased, especially when real-time computation-intensive operations are executed. The essential part of the ICEMobile offloading framework is in the server side, where there is not any limitation on the operating system with the help of JVM technology. The environmental profiling and program analyzing efforts together with the optimization component in the server side enable optimal decision making for offloading. In this architecture, the Profiler and the Analyzer are not executable programs. The developer manually operates them and their outputs are transmitted to Optimization Solver as input. This process is realized automatically in [5]. In addition to these components, the same offloadable methods of the mobile application are also present on the cloudlet to be executed when necessary.

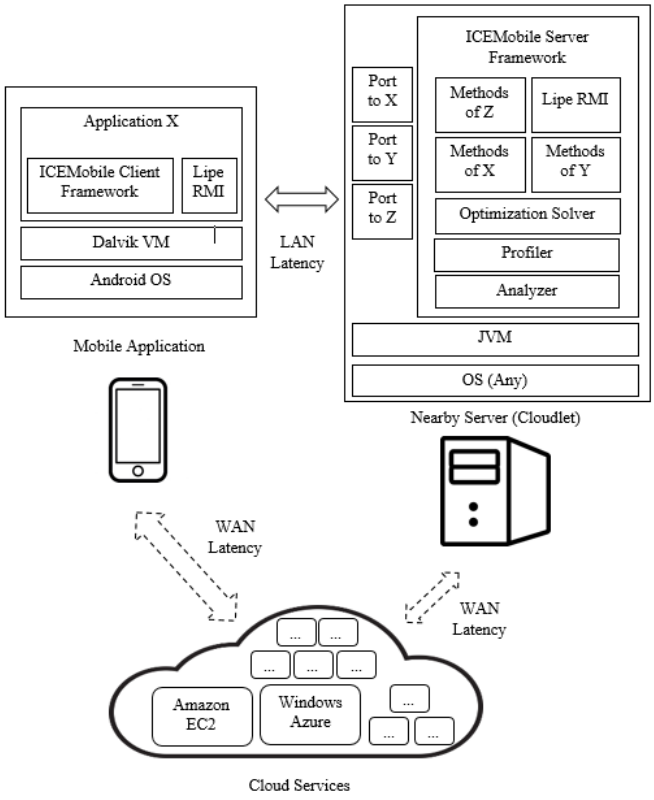

Figure 1. ICEMobile System Architecture

In the client side, any mobile device can benefit from the ICEMobile task offloading mechanism. For mobile devices with higher computational capabilities, the need for

offloading tends to decrease. In order to promote offloading, the application code needs to be modified with the ICEMobile client framework codes. In the server side, a port is specifically reserved for each client. In order to initiate the server, the following two steps are proceeded:

i. A *CallHandler* is identified. The interface file that keeps the method signatures and the class file that contains the client methods are globally registered to *CallHandler*.

ii. *CallHandler* is bounded by the available port to start listening the environment and respond to the incoming requests.

A. Making the Optimal Offloading Decision

The optimization solver in the ICEMobile framework aims at determining which portions of the application code are better to be offloaded to the remote server in order to use resources more effectively. The decision making procedure is based on the model proposed in [4]. The advantage of offloading in method-level granularity is the ability to transfer only the computation-intensive parts of the application, rather than transferring the full VM snapshot, as in [3]. In the first step of the ICEMobile optimization process, the application is analyzed via its call graph. In order to extract the call graph, static or dynamic analyzers can be used. Next, each node and edge in the call graph is attached with a time and energy cost. The measurements are generated for each of the following three cases:

- Perform all methods on the mobile device and measure the time and energy cost,
- Perform all offloadable methods on the cloudlet and measure the time cost,
- Offload all offloadable methods and measure the time and energy cost.

The energy cost is measured using the Android application called PowerTutor. At the end of the optimization process, the nodes that are better to be offloaded is determined. The offloading variable in the objective function is a binary one that has two values: 0 for not offloading, and 1 for offloading. The given maximization problem is solved by *lpsolve* solver of the R language.

For a given graph $G=(N, E)$, N represents *node* and E represents *edge*. Each node $n \in N$ represents a method and edge $e=(m, n)$ represents an invocation of method n from m. We annotated each node $n \in N$ with the energy it takes to execute the method locally E_n^l. The energy consumption of the cloudlet is not considered. The time that a node takes to execute the method locally is shown by T_n^l, and the time that a node takes to execute the method remotely is specified by T_n^r. Each edge is annotated as $e=(m, n)$. The time it takes to transfer the necessary program state is given by $B_{m,n}$ when m calls n and the energy cost of transferring that state is annotated as $C_{m,n}$. The binary parameter r_n indicates whether the node n is offloadable or not.

The 0-1 integer programming function of ICEMobile is shown in the equations (1), (2) and (3) [4]. The objective is to maximize the energy savings in the mobile device (1). The optimization solver determines the offloading variables (I_n) that is the indicator variable of offloading decision for each node. As a result of the optimization, if I_n is equal to 0, it means that the method will not be offloaded and if I_n is equal to 1, the method will be offloaded and it will result in saving energy.

$$\max \sum_{n \in N} I_n . E_n^l - \sum_{(m,n) \in E} \left| I_m - I_n \right| . C_{m,n} \quad (1)$$

$$s.t. \sum_{n \in N} \left((1 - I_n) T_n^l \right) + \left(I_n . T_n^r \right) + \sum_{(m,n) \in E} \left| I_m - I_n \right| B_{m,n} \leq L \quad (2)$$

$$I_n \leq r_n, \forall n \in N \quad (3)$$

The ICEMobile client framework contains the RMI interfaces presenting the signatures of the offloadable methods and the necessary codes building a connection with the cloudlet. It requires adding the necessary code at the beginning of each offloadable method. The *ICEMobileDecisionMap* object contains the offloading decision of each offloadable method. It is obtained from the optimization result text file, kept into the mobile device memory, and in the form of a Java Hash Map object having <*key*, *value*> pairs.

IV. Experiments and Results

A. Hardware and Software Specifications of the Implementation Environment

The backend server is a computer with 4 cores of Intel i7 4th generation x64 microprocessor and a RAM with 8GB DDR3 capacity. Apache Tomcat provides the web server functionality by listening incoming requests. As the mobile client device, we used an LG G3 smartphone having Android Kitkat 4.4.2 OS, a Qualcomm Snapdragon 801 microprocessor with quad core processors hardware and a 3GB RAM. In order to create client/server communication, we included the Lipe-RMI for RMI-based offloading. In all of our three implementations, the mobile device and the server are connected to the same LAN. The bandwidth measurement tests reveal that LAN bandwidth is approximately 64% higher than the WAN bandwidth.

B. Scenarios with Mathematical Calculations

As the first demonstrative numerical example, in the client side, we implemented several matrix operations, including matrix creation with random values, addition, multiplication, division and inverse operations at matrices of different sizes. These mathematical operations use JScience mathematical library. In the server side, we developed backend server software in Java, which contains the same methods of the Android application with the same JScience library.

Fig. 2 depicts the call graph of these implementations. The first node initiates the application flow after getting an input from the user. The vertical flows are differentiated based on the matrix size. Among all the operations, matrix

 ISBN: 978-1-61208-503-6

creation and matrix addition were found as the easiest operations to be completed by the device. On the other hand, there is a significant challenge when mobile devices perform multiplication, division and inverse operations.

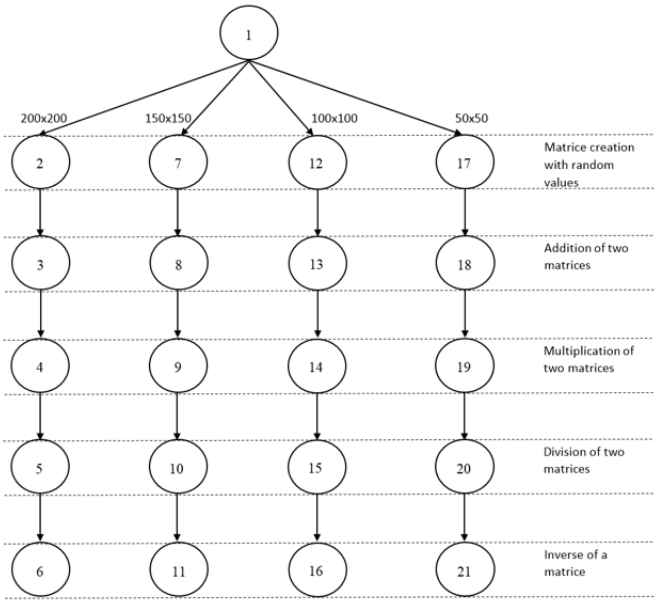

Figure 2. Call Graph of Matrices Calculations

TABLE I. MEASURED TIME AND ENERGY COSTS OF THE COMPARISON OF OBTAINED ENERGY EFFICIENCY

node id	T_n^l	T_n^r	$B_{m,n}$	E_n^l	$C_{m,n}$
1	-	-	-	-	-
2	50	52	695	35,8	720
3	4	1	2020	2,8	680
4	328	25	2100	114,6	700
5	72699	700	2515	44556	700
6	73206	614	1832	42350	800
7	35	21	520	20	700
8	4	10	1010	7,9	700
9	162	15	1210	58,1	700
10	30260	284	1400	19200	700
11	29539	270	1170	15950	700
12	15	12	320	5	700
13	1	2	585	3,2	700
14	80	6	450	6,9	700
15	8643	89	575	4600	780
16	8889	89	685	4700	700
17	2	2	242	5,4	700
18	1	2	124	0,6	760
19	14	2	107	29,8	700
20	1107	14	128	801	700
21	1076	9	97	450	700

For each scenario with different matrix size, we measured time and energy costs of each component of the call graph (Table 1). T_n^l represents the time that a node takes to execute the method locally, while T_n^r identifies the time that a node takes to execute the method remotely. $B_{m,n}$ describes the time that is required to transfer the necessary program state when *m* calls *n*. E_n^l shows the energy required to execute the method locally for each node *n*, $n \in N$. $C_{m,n}$ represents the energy that is required to transfer the necessary program state when *m* calls *n*. We generated two different scenarios. In the first one, all of the nodes are marked as offloadable except the root node; while in the second one several nodes are marked as non-offloadable.

Scenario 1: Unconstrained Offloading

For this case, we mark all the nodes as offloadable except the first one. If a node requires taking user input or accessing to a native device component, that node cannot be offloaded. For this scenario, the optimization solver resulted that it is better to offload 16 of 21 nodes to have maximum energy saving on mobile device. In that case, the nodes 3, 8, 13 and 18 will be offloaded to the cloudlet, together with their consecutive invocations. Since the objective is to maximize the energy saving in mobile device; even though the time performances of mobile device and cloudlet are equal to each other for any node, the optimization solver prefers offloading.

TABLE II. COMPARISON OF OBTAINED ENERGY EFFICIENCY

	Total Energy Cost (mJ)	The Energy Gain (mJ)	Percentage of Gain
Without Offloading	132.906	0	0%
Unconstrained Offloading	2.874	130.032	98%
Constrained Offloading	72.826	60.080	45%

The optimization function starts to offload with the lightweight nodes instead of the nodes having high energy cost while transferring. After transferring the node with lowest cost, the consecutive ones will not consume any energy, since they are already on the cloudlet, so they will be operated on the cloudlet. This example shows that the optimization function examines all of the nodes in high level, rather than considering it node by node. Since the set of mathematical operations are the same in each vertical flow with different size of matrices, the pattern is occurred in the same way. We can conclude that the position of the nodes and their invocations have significant effects on the offloading decision.

Scenario 2: Constrained Offloading

In order to discover the effect of node offloadability, we modified the first scenario by marking the nodes of 5, 10, 15 and 20 as not-offloadable in addition to the node 1. In this case, since the former methods of non-offloadable nodes are not resource-intensive, the optimization function decides offloading their consecutive nodes (6, 11, 16 and 21), which are relatively heavy tasks for the mobile device.

For these two scenarios, the total costs and obtained energy gains are summarized in Table 2. The total cost represents the sum of energy weights, when there is a mobile processing. The energy gain describes obtained energy saving via offloading. As a result, we observe that it is possible to achieve an energy saving up to 98% in

 ISBN: 978-1-61208-503-6

unconstrained offloading and 45% in *constrained offloading* by integrating ICEMobile framework on the mobile device.

C. Scenarios with Face Detection

This is one of the most commonly applied scenarios for the mobile cloud computing, that identifies human faces in a given photo. Even though it seems as a quite simple image processing operation, it may become a highly computation-intensive operation proportional to the image resolution. As image processing library, we use the native Android FFTE face detection API. Our reason for choosing FFTE rather than OpenCV Android SDK is that the APIs of OpenCV do not detect faces of a static image. Instead, they make the face detection when there is an actual streaming on the camera [7]. Fig. 3 shows the screenshot of our developed Android application. The user selects a photo from the hard drive and then selects face detection function. The operation is performed on mobile device or on synchronized cloudlet. Then, the faces are identified with green rectangles based on their coordinate values. In the right side of the screen, the time cost is shown together with the detailed information about the call graph.

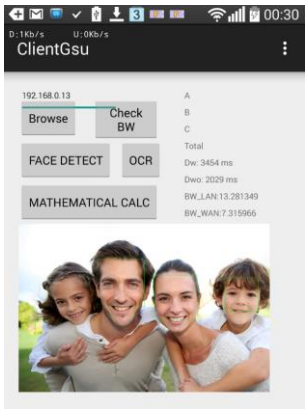

Figure 3. Screenshot of Face Detection Application

As the example, an image containing four human faces is chosen and it is resized to obtain multiple images of different size of pixels. The main reason of duplicating an image in different sizes is to eliminate other parameters, such as RGB values of the pixels, which could affect the test performance. Table 3 summarizes general properties of the images. In the table, the total number of pixels describes the multiplication of the width and height values.

The proposed offloading capability is integrated into the face detection application (Fig. 4). The figure represents the flow of on-device and cloudlet-based processing. The first and last nodes of two processing types are the same. However, in the second node of cloudlet-based processing (B^1), there is a conversion to raw data with Base64 encoding before transmitting the image data. Then in node C^1, the necessary initializations for socket, REST or RMI are completed. The node C^2 is the only node that is executed on the cloudlet and performs decoding and face detection operation. At the end of its execution, node C^2 sends obtained results back to the mobile device.

TABLE III. IMAGES WITH DIFFERENT DIMENSIONS FOR FACE DETECTION

	Width	Height	Total # of Pixels	Size on Disk
Image1	720	480	0.3 M	102 KB
Image2	1440	960	1.3 M	321 KB
Image3	2880	1920	5.5 M	980 KB
Image4	5760	3840	22.1 M	3410 MB

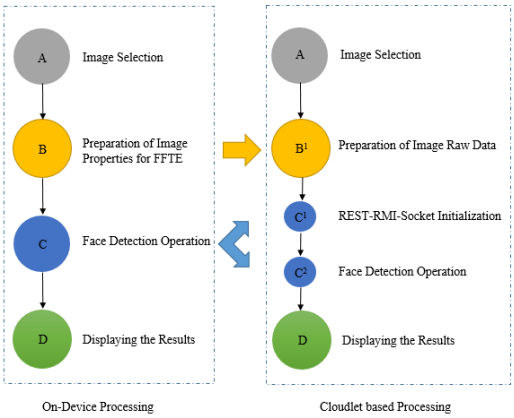

Figure 4. Call Graph Transformation of Face Detection

The tests of on-device processing reveal that the execution of native Android FFTE method takes more than 80% of total energy consumption. Since it is a native function, we are unable to partition this function to obtain a balanced distribution. As a result, we concluded that this use case is not ideal for optimization-based offloading. Fig. 5 shows that offloading will save energy for the images that are larger than 0.3 megapixels. We made use of this scenario to analyze different client/server communication techniques including Java sockets, RESTful Web Services and LipeRMI to explore the most efficient offloading model.

REST, socket and RMI-based communications are amongst the well-known offloading techniques. In order to compare the time and energy consumption of these communication types, we isolated the data transmission process of the use case, which starts by sending the data packet from client to server and finishes by receiving it back from server. (i.e., the edges $B^1 \rightarrow C^1$ and $C^2 \rightarrow D$ in Fig. 4).

We executed the operations for 10 times, and calculated the average of measured values. The energy consumption is measured in mW, and the time is measured in ms. As a result, as shown in Fig. 6 and Fig. 7, we observe that REST and socket-based offloading consume nearly the same amount of time and energy. Lipe-RMI is found more costly than REST and socket-based communications, besides its advantages of offloading.

 ISBN: 978-1-61208-503-6

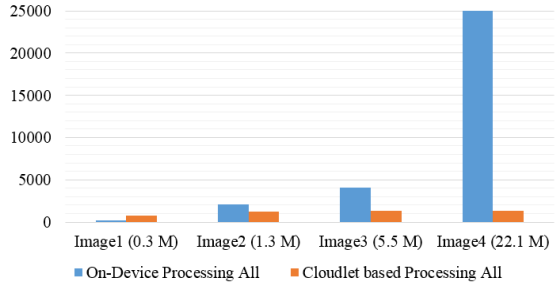

Figure 5. Comparison of Energy Consumption of Face Detection Operation

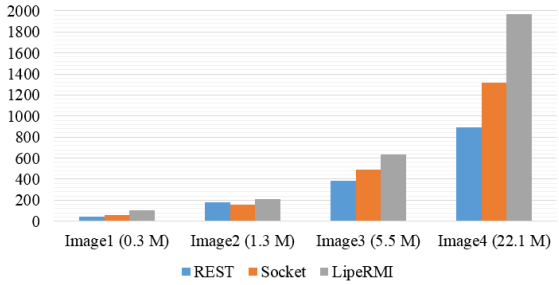

Figure 6. Comparison Based on Time Consumption

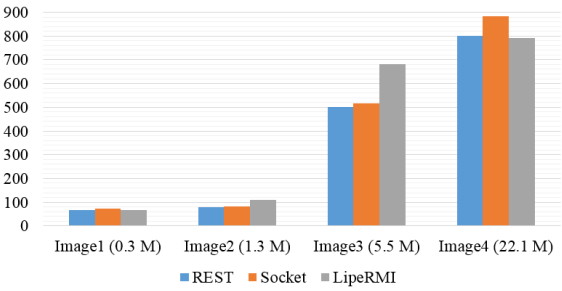

Figure 7. Comparison Based on Energy Consumption

D. *Scenarios with OCR*

As another use case, the ICEMobile framework is integrated into the Optical Character Recognition (OCR) application. The face detection and OCR are similar scenarios due to the fact that there is an image processing in each use case by using native functions. Hence, their call graph transformation is generated similarly (Fig. 4). The tests of on-device processing reveal that the execution of the *nativeGetUTF8Text* operation takes nearly 90% of total time and energy cost. It is much higher than the face detection operation, because there is not any painting operation on the image in OCR. Since Tesseract is not a Java-based library, we are unable to get into the structure of its API (*nativeGetUTF8Text*), and it becomes impossible to obtain a balanced distribution. We concluded that, the optimization model is not needed for this use case. Fig. 8 shows that offloading will save energy for images having more than 600 characters.

V. CONCLUSION

In this research, we first presented the challenges in mobile cloud computing and then focused on the usage of cloudlets as a solution for increasing energy efficiency of mobile devices. Cloudlets enable time and energy efficiency compared to distant clouds during the execution of computation-intensive tasks. In this paper, we implemented a lightweight RMI-based computation offloading in Java-based client and server. In order to examine the applicability of the proposed framework, we created three groups of synthetic test scenarios. Related call graphs are generated for each scenario. Detailed energy consumption analysis is done using a mobile application, PowerTutor. These values are utilized as the input of the optimization function. The results show that ICEMobile allows saving up to 98% of energy on mobile devices in specific cases, together with keeping the level of mobile user experience.

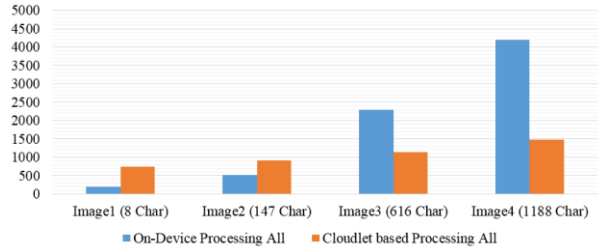

Figure 8. Comparison of on Energy Consumption of OCR Operation

ACKNOWLEDGMENTS

This work is supported by Galatasaray University Research Fund, under the grant number 15.401.005.

REFERENCES

[1] B. Zhou, A.V. Dastjerdi, R.N. Calheiros, S.N. Srirama and R Buyya, "A Context Sensitive Offloading Schme for Mobile Cloud Computing Service", The Eighth International Conference on Cloud Computing (CLOUD), New York, USA, pp. 869-876, 2015, ISSN: 2159-6182.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing", US National Institute of Science and Technology, 2011, URL: http://dx.doi.org/10.6028/NIST.SP.800-145 (accessed on 21st December)

[3] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing", IEEE Pervasive Computing, vol.8(4):3, pp.14-23, Dec. 2009, doi: 10.1109/MPRV.2009.82.

[4] E. Cuervo, et al., "MAUI: Making Smartphones Last Longer with Code Offload", The Eighth ACM MobiSys, pp.49-62, 2010, ISBN: 978-1-60558-985-5.

[5] B. Chun and P. Maniatis, "Augmented Smartphone Applications Through Clone Cloud Execution", The Eighth Workshop on Hot Topics in Operating Systems (HotOS), pp.8-8, 2009. URL: https://www.usenix.org/legacy/event/hotos09/tech/full_papers/chun/chun.pdf (accessed on 15th November)

[6] R. Kemp, N. Palmer, T. Kielmann and H. Bal, "Cuckoo: a computation offloading framework for smartphones", The Second International Conference on Mobile Computing, Applications, and Services, MobiCASE., pp.59-79, 2010, ISSN: 1867-8211, ISBN: 978-1-4673-7286-2.

[7] OpenCV4Android SDK, http://opencv.org/platforms/android.html (accessed on 27th December 2015).

 ISBN: 978-1-61208-503-6