PAPER

An Adaptive Computation Offloading Decision for Energy-Efficient Execution of Mobile Applications in Clouds

Byoung-Dai LEE[†], ***Nonmember***, **Kwang-Ho LIM**[†], ***Student Member***, **Yoon-Ho CHOI**[††], ***and*** **Namgi KIM**[††a)], ***Members***

SUMMARY In recent years, computation offloading, through which applications on a mobile device can offload their computations onto more resource-rich clouds, has emerged as a promising technique to reduce battery consumption as well as augment the devices' limited computation and memory capabilities. In order for computation offloading to be energy-efficient, an accurate estimate of battery consumption is required to decide between local processing and computation offloading. In this paper, we propose a novel technique for estimating battery consumption without requiring detailed information about the mobile application's internal structure or its execution behavior. In our approach, the relationship is derived between variables that affect battery consumption (i.e., the input to the application, the transmitted data, and resource status) and the actual consumed energy from the application's past run history. We evaluated the performance of the proposed technique using two different types of mobile applications over different wireless network environments such as 3G, Wi-Fi, and LTE. The experimental results show that our technique can provide tolerable estimation accuracy and thus make correct decisions between local processing and computation offloading.

key words: *battery consumption, dynamic estimation, linear regression, mobile clouds*

1. Introduction

Since the first-generation iPhone was released in 2007, the popularity of smart devices, such as smart phones and smart tablets, has been increasing continuously. Each year, dozens of new smartphone models are introduced into the market. Worldwide sales of smartphones exceeded those of feature phones in early 2013 [17]. This phenomenon is attributed to significantly improved mobile hardware performance, various high-speed wired/wireless connectivity options, and, consequently, the availability of a wide variety of mobile applications for daily life and business. The vision of "a computer in my hand" has now become a reality.

Unlike stationary computers that have an unlimited power supply, smart devices are battery-operated for portability. Therefore, low power consumption is a critical requirement in mobile hardware and software design. Considerable effort has been expended to address limited battery lifetime, ranging from efficient power management techniques, such as dynamic power management [4] and dynamic voltage and frequency scaling [3], to new battery technologies based on nanomaterials such as fluoride shuttle [1] and grapheme [15]. Recently, cloud computing has emerged as a solution to the inherent resource limitation of mobile devices. In the cloud-based approaches, applications on the mobile device can offload their computation to the cloud to conserve energy, as well as augment the computation and memory capabilities of the mobile device ([5], [6], [9], [11], [13], [14], [16]). Computation offloading reduces the workload on the mobile device but it requires wireless communication to migrate the computation to more resource-rich clouds. Therefore, computation offloading is energy-efficient only if the communication energy does not exceed the computation energy for local processing ([12]).

To determine whether and which portion of the computation to offload, the energy consumption needs to be estimated before execution. Communication energy is a critical factor that determines the energy consumed by computation offloading, and it depends mainly on the size of the data transmitted between the mobile device and the cloud and the network bandwidth. Computation energy depends on the computation time that is affected by the input data and the resource status such as CPU load and memory availability. Many studies ([6], [9], [10], [13], [18]) have been conducted to develop models, methods and algorithms for an accurate estimation of energy consumption, but they are limited for these reasons:

- It is assumed that the size of the results transmitted from the clouds to the mobile device is predefined or known in advance. For some applications, this assumption does not hold. For example, the size of an output video from the video transcoding application is different from that of an input video due to differences in coding efficiency of the corresponding codecs.
- The internal structures of applications or their execution behaviors (e.g., the number of instructions in a function, the number of loop iterations, etc.) can be obtained automatically with the help of dedicated programming tools such as compilers or provided by application developers. Such pieces of information are useful for an accurate estimation of the computation time. However, development of such tools requires significant efforts.

In this paper, we propose a technique for energy consumption estimation to address the abovementioned shortcomings. Mobile applications consist of several functional

Manuscript received December 18, 2013.
Manuscript revised February 24, 2014.

[†]The authors are with the Dept. of Computer Science, Kyonggi Univ., Suwon, 443–760, South Korea.
[††]The authors are with the Dept. of Convergence Security, Kyonggi Univ., Suwon, 443–760, South Korea.
a) E-mail: ngkim@kgu.ac.kr (Corresponding author)

DOI: 10.1587/transinf.E97.D.1804

modules. Some of them must run on mobile devices. Examples include a user interface or codes for handling a peripheral such as a mobile device's camera. For other modules, the decision to offload computation must be made based on energy efficiency. The novel aspect of our technique is that it treats the module of an application as a black box and thus does not require detailed information about how it works internally. Instead, it derives the relationship between variables that affect the energy consumption (e.g., input parameters to the module, transmitted data, and resource status) and the actual consumed energy from the past run history. The integral part of our approach is that various filtering techniques are applied to extract subsets of past run history that are closely related to the module of which energy consumption needs to be estimated. Once having done that, the regression methods are applied. In doing so, irrelevant past history has less influence on the estimation. Furthermore, since it is not known which filtering technique performs best for a given module, our technique selects the best filtering technique dynamically over time based on various error measurement metrics. We evaluated the performance of the proposed technique using two different types of mobile applications: a face recognition application and a video transcoding application. At the same time, we varied the underlying wireless network environments, including high-speed 4G networks such as Long Term Evolution (LTE) [2]. The experimental results showed that our technique delivers tolerable estimation accuracy and can make correct decisions between computation offloading and local processing.

The remainder of the paper is organized as follows: Sect. 2 summarizes related work; Sect. 3 explains the proposed estimation technique in detail; and Sect. 4 presents the experimental results. Finally, we conclude in Sect. 5.

2. Related Works

Significant research [11] has been conducted on offloading decisions for improving performance or saving energy. Chang *et al.* [5] provide a performance/power evaluation model to determine whether offloading a kernel function benefits the application. Their model is based on a simple yet general energy consumption model such as that defined in Kumar *et al.* [12]. Similarly, Kumar *et al.* [13] extend a general energy consumption model of the offloading to be well suited for a content-based image retrieval (CBIR) program. In particular, they decompose a queryprocessing operation in a CBIR session into several steps and are thus able to define a fine-grained energy consumption model.

Cuervo *et al.* [6] use various profiling data as input to a global optimization problem that determines which methods should execute locally and which should execute remotely. Their goal is to find a program partitioning strategy that minimizes the smartphone's energy consumption, subject to latency constraints.

Kovachev *et al.* [10] define the cost function of the offloading that consists of three parts-the data transfer cost, the memory cost, and the CPU cost-and apply integer linear programming to find an optimized partitioning strategy for a given set of offloadable modules of a mobile application, with the objectives of minimizing memory usage, energy usage, and execution time.

Xian *et al.* [18] present a method that uses timeout for offloading decisions. In their approach, the computation is offloaded to servers only if it is not completed after its break-even time, which is the minimum computation time that can benefit from offloading. They use the online statistical information to find the optimal timeout, instead of estimating execution time for each computation instance.

Although our work is similar to others in that past run history is used for predicting computation time, there are important distinguishing differences. First, we do not assume that the size of the results generated from an application module is known before executing it. Second, we assume that the internal structure of an application module and its execution behavior are also unknown. These pieces of information are derived from the past run history. Therefore, our work is generic enough to be applicable to a wide range of mobile applications.

3. Dynamic Estimation of Energy Consumption

For a given module of a mobile application, Fig. 1 depicts the steps to estimate energy consumption for local processing and computation offloading. The *query point* represents the set of information needed to run an instance of the module. This includes the input parameters required by the module and the current resource status of the target system such as CPU load and memory availability. The input parameters are important variables that affect not only the computation time but also the communication time, as the output of the module is correlated with the input to the module, thus influencing the size of data transmitted between the mobile device and the clouds.

When it needs to be determined whether a given application's module must be offloaded, the first step is to extract subsets of past run history that are relevant to the query point by applying a set of filters. In the second step, using the selected datasets, regression methods are applied to predict the output size as well as the computation times on the mobile device and the cloud, respectively (e.g., $T_{local}(i)$, $T_{cloud}(i)$, and $D_{output}(i)$, $i = 1, \ldots, N$). Note that since the resource status affects only the computation time, it is not used by filters for predicting the size of the output of the module. For the third step, the best estimates for T_{local}, T_{cloud}, and D_{output} are selected based on prediction accuracies. The expected energy consumptions for local processing and computation offloading are computed using the selected values and sampling data of consumed energy by the target mobile device as shown in the following equation:

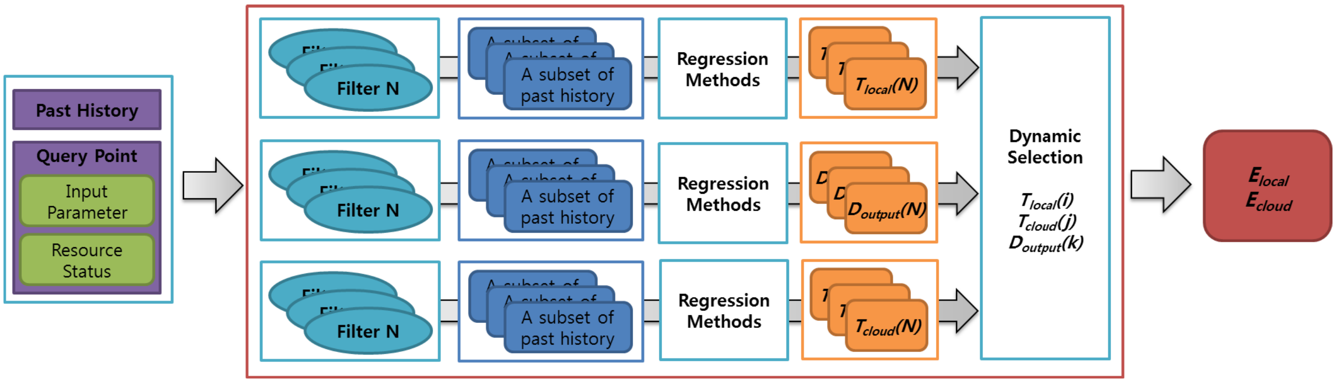


Fig. 1 Sequence of steps to estimate energy consumption. $T_{local}(i)$ and $T_{cloud}(i)$ are estimated computation times for local processing and computation offloading based on the filter i, respectively, whereas $D_{output}(i)$ is the predicted size of the result transmitted from the cloud to the mobile device using the filter i. E_{local} and E_{cloud} are estimated energy consumption for local processing and computation offloading using the best estimations for T_{local}, T_{cloud}, and D_{output} (e.g., $T_{local}(i)$, $T_{cloud}(j)$, $D_{output}(k)$), respectively. N denotes the number of available filters.

$$\begin{aligned} E_{local} &= T_{local}(i) \times J_{comp} \\ E_{cloud} &= E_{tx} + E_{idle} + E_{rx} \\ &= \frac{D_{input}}{BW} \times J_{send} + T_{cloud}(j) \times J_{idle} \\ &\quad + \frac{D_{output}(k)}{BW} \times J_{receive} \end{aligned} \tag{1}$$

where $T_{local}(i)$, $T_{cloud}(j)$, and $D_{output}(k)$ are the best estimations for the computation time on the mobile device, the computation time on the cloud, and the output size using datasets extracted by filters i, j, and k, respectively. The sampling data include the average amount of energy consumed by the mobile device for performing computation for one second (J_{comp}), for sending or receiving data for one second (J_{send} and $J_{receive}$), and for being idle for one second (J_{idle}), respectively. Calculating the energy consumption for local processing is straightforward, whereas the energy needed for computation offloading consists of three components: the energy for sending data to the cloud for the module to execute remotely (E_{tx}), the energy for receiving the execution result from the cloud (E_{rx}), and the energy consumed by the mobile device sitting idle until the remote execution of the module finishes and the result is returned (E_{idle}). E_{tx} and E_{rx} are proportional to communication time and are therefore dependent on the size of transmitted data (D_{input} and $D_{output}(k)$) and the network bandwidth (BW). Note that the size of data to send to the cloud need not be estimated; it can be computed by summing the sizes of the input parameters and code size of the module, if necessary. While the module runs on the cloud, the mobile device must sit idle for T_{cloud}. Therefore, energy consumed during this period must also be taken into account. Finally, when E_{cloud} is found to be less than E_{local}, the module will be offloaded. Otherwise, the module will be processed locally.

When the module finishes its execution, the query point of the module, the actual execution time either on the mobile device or in the cloud, the actual size of the result, and accuracies of individual estimations are stored for later use. Note that we assume that during the lifetime of an application, the mobile device periodically monitors the network bandwidth and resource status such as the CPU load and memory availability of the mobile device and the cloud. In order to reduce monitoring overhead for the cloud, when the module is offloaded, such pieces of information are piggybacked with the execution results and the monitoring period re-starts from that point. Although this process requires extra energy, its impact is relatively small [6]. Detailed descriptions of each of these steps are presented in the next section.

3.1 Regression Methods

Regression analysis is the part of statistics that investigates the relationship between two or more variables in a non-deterministic fashion. In this work, we use the linear regression method because the computational cost is cheaper than for nonlinear regression methods such as quadratic and cubic methods. Note that using linear regression does not mean that we assume that the energy consumption of a given module is a linear function of the input parameters and the resource status. Instead, only the subset of the past run history that is sufficiently close to the query point, which is selected by filters, will be approximated by a linear relationship.

Multiple linear regression models take the form:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k \tag{2}$$

where Y is the dependent variable, the β_i's are regression coefficients, and x_i's are independent variables. In our work, the number of independent variables varies depending on the number of input parameters required by the given module and the type of the dependent variable (e.g., the computation time and the output size). For example, suppose that a module requires two input parameters, p and q, and CPU load and memory availability are considered for the resource status. Then, the computation time is derived by

$Y = \beta_0 + \beta_1 P + \beta_2 Q + \beta_3 CPU + \beta_4 Memory$ and the output size is derived by $Y = \beta_0 + \beta_1 P + \beta_2 Q$, where P, Q, CPU, and $Memory$ are variables representing values of p, q, CPU load, and memory availability, respectively. Due to differences in the units of measurement for individual parameters as well resource status, for multiple regression it is advantageous to carry out standardization of variables before fitting a curve. Let $\overline{x_i}$ and s_i be the sample average and sample standard deviation of observed x_i. In the standardized model, each variable x_i is coded by $x'_i = (x_i - \overline{x_i})/s_i$. The coded variable simply re-expresses any x_i value in units of standard deviation above or below the mean. The benefits of standardization are (1) increased numerical accuracy in all computations and (2) more accurate estimation than for the parameters of the un-standardized model. This is because the individual parameters of the standardized model characterize the behavior of the regression function near the center of the data rather than near the origin [7]. Therefore, given k independent variables, we use the following regression function to predict the expected values of Y, $E(Y)$:

$$\begin{aligned} E(Y) &= \beta_0 + \beta_1\left(\frac{x_1 - \overline{x_1}}{s_1}\right) + \beta_2\left(\frac{x_2 - \overline{x_2}}{s_2}\right) \\ &\quad + \cdots + \beta_k\left(\frac{x_k - \overline{x_k}}{s_k}\right) \\ &= \beta_0 + \beta_1 x'_1 + \beta_2 x'_2 + \cdots + \beta_k x'_k \end{aligned} \tag{3}$$

3.2 Filtering Technique

Linear regression methods attempt to fit a straight line to the available data to minimize the sum of squared deviations of the predicted values from the actual observations. Therefore, if the observations do not show strong linearity, applying the linear regression model will not generate accurate predictions. To address such a limitation, we apply linear regression methods only to subsets of observations that are extracted by a filter in such a way that the selected datasets are close to the query point and show strong linearity. Figure 2 shows the effects of using filters. Suppose that the query point is q_2. The predicted value of Y by the filtering-based method, $h_2(q_2)$, is closer to the actual observation than a prediction without filters, $f(q_2)$. This is because while $h_2(x)$ is generated from data points close to q_2, $f(x)$ is derived from all observed data and, therefore, it minimizes the error for data points not only close to q_2 but also far from q_2.

The closeness of each data point to the query point is defined by a distance function of the filter. Since the range and distribution of independent variables are unknown, the distance function should normalize distances with respect to the query point. For this purpose, we used standardized Euclidean distance. The distance between data point $D = (d_1, d_2, \ldots, d_n)$ and query point $Q = (q_1, q_2, \ldots, q_n)$ is defined as follows:

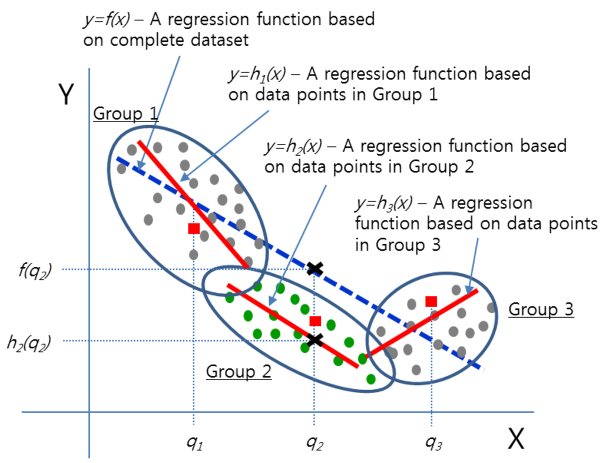


Fig. 2 The effects of using filters. Individual rectangles represent the actual values of Y given X is q_1, q_2, and q_3, respectively. The dashed line, $f(x)$, is generated by a linear regression method using all observed data, whereas the solid lines, $h_i(x)$, are based on subsets of data points close to corresponding query points (q_1, q_2, and q_3).

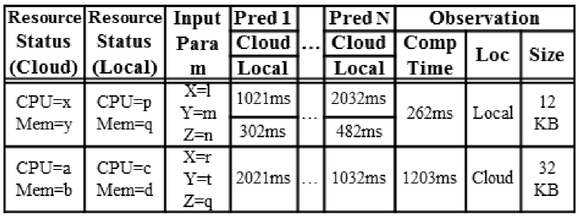
$$\begin{aligned} &Distance(D, Q) = \sqrt{\sum_{i=1}^{n} (d'_i - q'_i)^2} \\ &d'_i = \frac{d_i - \overline{d_i}}{s_i}, q'_i = \frac{q_i - \overline{d_i}}{s_i} \\ &\overline{d_i} : \text{the sample average of variable } i \\ &s_i : \text{the sample standard deviation of variable } i \end{aligned} \tag{4}$$

Once the distance is computed, it is compared with a configurable threshold value, $\alpha(\geq 0)$. If the absolute difference between $Distance(D, Q)$ and α is less than or equal to zero, then D is included for linear regression analysis. If the threshold value is too large, it is more likely that data points irrelevant to the query point are included for analysis, thus decreasing prediction accuracy. On the other hand, using smaller threshold values may cause useful data to be omitted.

3.3 Dynamic Estimator

For a given application module, it is difficult to know in advance which predictor generates the most accurate predictions, because it is highly dependent on the characteristics of the modules. The novel aspect of our approach is to select dynamically a best predictor over time for the target module by using the previous prediction history.

The prediction history database of the dynamic estimator maintains the past prediction history of each predictor, and it is organized in a two-dimensional matrix: columns represent the predictors and the variables that affect the computation time of the module, and each row represents the expected computation time of the predictors and the actual computation time of a module instance (see Fig. 3). When the prediction is needed, the dynamic estimator runs all the registered predictors in parallel. Then it gathers the most recent k prediction history of each predictor and selects the



Resource Status (Cloud)	Resource Status (Local)	Input Param	Pred 1	…	Pred N	Observation		
			Cloud / Local	…	Cloud / Local	Comp Time	Loc	Size
CPU=x Mem=y	CPU=p Mem=q	X=l Y=m Z=n	1021ms / 302ms	…	2032ms / 482ms	262ms	Local	12 KB
CPU=a Mem=b	CPU=c Mem=d	X=r Y=t Z=q	2021ms	…	1032ms	1203ms	Cloud	32 KB

Fig. 3 Structure of prediction history database. "CPU" and "Mem" represent CPU load and memory availability, respectively. X, Y, and Z denote input parameters to a application module. The "Comp Time" column represents the actual computation time and "Loc" indicates the place where the computation was actually performed. "Size" denotes the output size of the computation.

one that shows the most accuracy. We used several well-known fitness functions as metrics for measuring prediction accuracy, such as mean squared error, mean absolute error, relative squared error, and so on. Then, for each predictor, the sum of eight ranks is computed and the one with the smallest sum is selected as the best predictor.

4. Empirical Analysis

4.1 Prototype Applications and Test Environments

For our experiments, we implemented prototype mobile applications by applying the FREEMA framework [14] that enables seamless execution of Android applications in the cloud. The applications are a face recognition application and a video transcoding application, which are frequently used in mobile environments. The face recognition application was implemented so that a module that extracts coordinates of identified faces in a given image would be executed in clouds. As for the video transcoding application, it downloads a video from a repository and converts it to a format the device is able to play. Therefore, video downloading and transcoding were implemented into a module that would be executed in clouds.

In order to investigate the performance of our approach under different mobile communication networks, we used 3G and Wi-Fi networks as well as high-speed 4G networks, such as LTE. Energy consumption of the device was measured with an external power monitoring tool. All energy measurements were collected by supplying the device with a constant voltage of 3.9V and tracking the current with a sampling rate of 5,000Hz. Table 1 shows the main features of commercially available smartphones that we used in the experiments, while Table 2 shows the sampling data of consumed energy for the smartphone. When computing the amount of energy required, energy components (J_{comp}, J_{send}, $J_{receive}$, and J_{idle}) in (1) are replaced with corresponding sampled data. For instance, when the 4G network is used for the mobile communication, J_{comp_lte}, J_{send_lte}, $J_{receive\_lte}$, and J_{idle_lte} should be used. Individual mobile devices have their own energy consumption profiles, and automatic collection of these energy parameters may require support from the underlying framework. For a cloud server, we used a high-performance server computer with a 3.4 GHz quad core CPU and 16GB memory.

Table 1 Device specifications.

Feature	Specifications
3G	HSDPA++
Wi-Fi	IEEE 802.11g
4G	LTE
CPU	1.5GHz Dual Core
Memory	1GB
OS	Android 2.3
Battery	1850mAh

Table 2 Energy consumption profile.

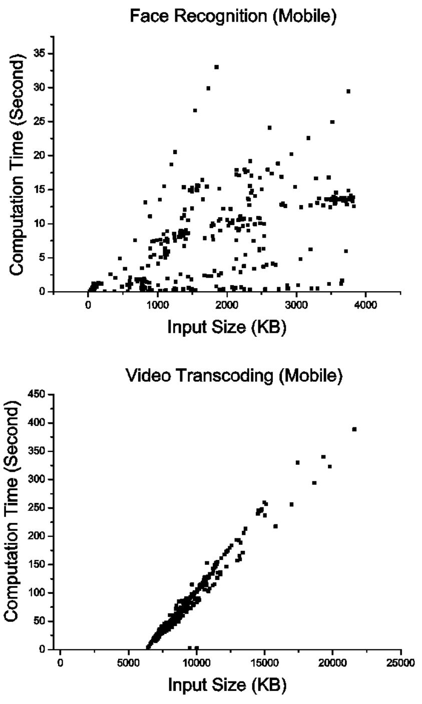
J_{comp_wifi}	Processing with Wi-Fi enabled	15.90
J_{comp_3G}	Processing with 3G enabled	15.71
J_{comp_lte}	Processing with LTE enabled	16.49
J_{idle_wifi}	Idle with Wi-Fi disabled	7.83
J_{idle_3G}	Idle with 3G disabled	7.72
J_{idle_lte}	Idle with LTE disabled	8.21
J_{send_wifi}	Sending data using Wi-Fi	22.68
J_{send_3G}	Sending data using 3G	21.15
J_{send_lte}	Sending data using LTE	30.08
$J_{receive\_wifi}$	Receiving data using Wi-Fi	14.61
$J_{receive\_3G}$	Receiving data using 3G	17.79
$J_{receive\_lte}$	Receiving data using LTE	18.71

4.2 Analysis

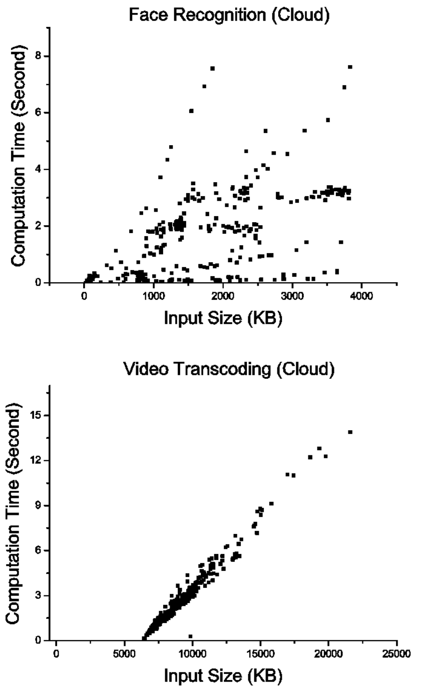
In most cases, the computation time of an application is highly dependent on its input. For example, in the cases of the face recognition and video transcoding applications, the image size and source video size are primary inputs that determine the computation time. For the face recognition application, we used different images with various resolutions between 400×492 and 4800×3600. For the video transcoding application, we used source video files in MP4 format with sizes ranging from 6.4MB to 215MB for conversion to AVI. Figure 4 shows distributions of execution times of individual applications according to their primary input parameters. Given an application, it is not known which value for a threshold used by the filtering technique performs best. Therefore, we used four different values (0.05, 0.1, 0.15, and 0.2) for the threshold and the dynamic estimator eventually selected the best one.

The resource monitoring activity of the mobile device requires both CPU processing and network communications and is an extra cost to enable accurate predictions. Figure 5 shows the amount of energy consumed for resource monitoring. Although the amount of consumed energy for resource monitoring is different for different network types, their magnitude is marginal. However, as the number of target cloud servers to monitor and the monitoring frequency increase, the cost will also increase. One way to compensate this extra cost is to allow several applications supporting offloading to share the resource status information.

According to Fig. 6, the proposed prediction technique performs well in all prototype applications. For the face



(a) Local execution



(b) Cloud execution

Fig. 4 Distributions of execution times of prototype applications.

recognition and video transcoding applications, the output size is indeterministic in that the output is highly dependent on not only the size of the input but also its content. However, for the face recognition application, because the numbers of faces appearing in the experimental input images did not vary significantly, its prediction accuracy was relatively higher than for the video transcoding application.

Figure 7 shows the accuracy of various predictors. In

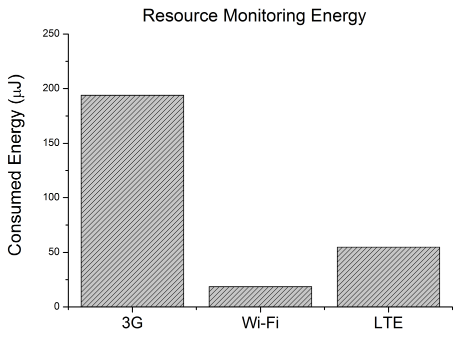


Fig. 5 Resource monitoring energy.

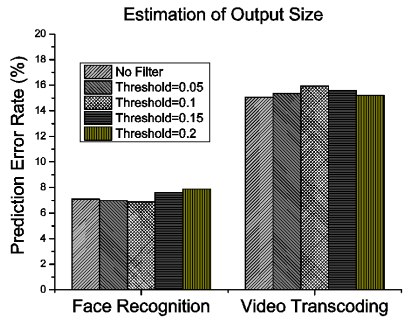


Fig. 6 Prediction accuracy for the output sizes.

the face recognition application, predictors with filters outperformed those with no filter. However, in the case of the video transcoding application, the predictors without filters showed slightly better performance. This can be explained by the distributions of execution times of application instances. As shown in Fig. 4, the distribution of the execution time of the video transcoding application is almost linear. On the other hand, as execution times of the face recognition application does not show strong linearity, applying linear regression analysis using the entire dataset generates more errors than using subsets of history data. Another interesting observation in Fig. 7 is that the threshold values defining the closeness of data point to query point show different performances with different applications as well as with different mobile communication networks. However, since the dynamic estimator considers past prediction accuracy of individual predictors, its accuracy is close to that of the best one under the given circumstances.

Figure 8 shows the energy saved by using offloading. It compares the amount of actual energy consumed when the offloadable modules were executed on the mobile device, on the cloud server, and on the location selected by the offloading decision based on the energy consumption predicted by the dynamic estimator. When high-speed networks such as Wi-Fi and LTE were used, the energy saved by offloading was greater because data transmission time was less, resulting in reduced energy consumed for data communication. However, in the case of the face recognition application, the computation time is very small. Therefore, even if a high-speed network is used, it is more likely that energy saved by

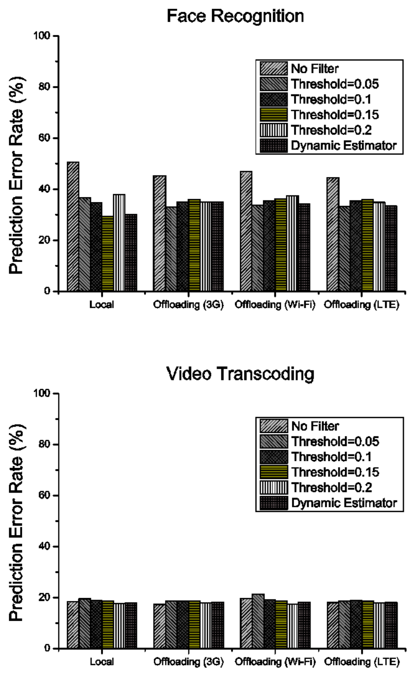


Fig. 7 Comparative prediction accuracy for energy consumption.

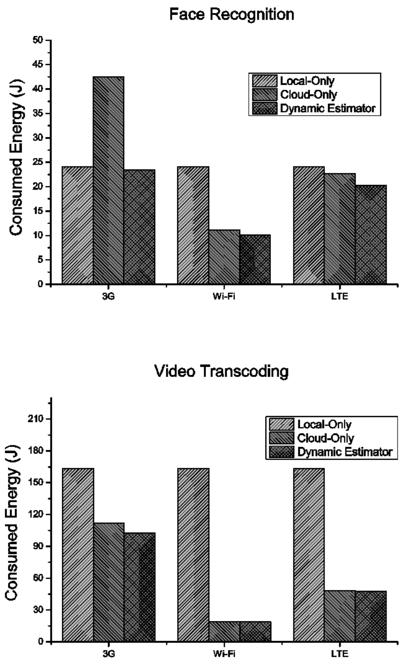


Fig. 8 Comparative energy consumption.

remote execution ($E_{local} - E_{idle}$) is exceeded by the energy consumed by data transmissions between the cloud and the mobile device ($E_{tx} + E_{rx}$). In such cases, it is more profitable to run the modules locally.

Interestingly, although in some cases the prediction error rate of the dynamic estimator is relatively high, energy savings obtained by online offloading as decided by the dynamic estimator was improved. The goal of the offloading decision is to select the best option for executing application modules in terms of energy efficiency. Therefore, even though individual predictions generated by the dynamic estimator may not be accurate, if their predictions are accurate enough to rank the options in terms of energy consumption, the offloading decision maker will make the correct decisions, thus achieving energy savings.

5. Conclusion

In this paper, we proposed a novel technique for battery consumption estimation without requiring detailed information about the internal structure of a mobile application or its execution behavior. In our approach, the relationship is derived between variables that affect the battery consumption (i.e., the input to the application, the transmitted data, and resource status) and the actual consumed energy from the past application run history. In addition, since it is difficult to know which predictor generates the most accurate predictions in advance, we proposed a dynamic estimator that dynamically selects a best predictor over time for the target module by using previous prediction history.

In the current work, we used only independent variables to select relevant observations to the query point. Therefore, we plan to extend the filtering technique so that not only will the independent variables but also the dependent variable be used to select observations that are close to the query point and show strong linearity.

The response time is another important property when choosing a mobile application or service. However, providing fast response time sometimes conflicts with energy savings. Therefore, we plan to investigate trade-offs between user experience and energy consumption and to develop offloading decision algorithms to take into consideration both the response time and the energy consumption, thus being able to provide acceptable delay time while consuming as little energy as possible.

Acknowledgments

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2011-0008552, 2012R1A1A1002133).

References

[1] M.A. Reddy and M. Fichtner, “Batteries based on fluoride shuttle,” J. Materials Chemistry, vol.21, pp.17059–17062, Oct. 2011.
[2] D. Astely, E. Dahlman, A. Furuskar, Y. Jading, M. Lindstrom, and S. Parkvail, “LTE: The evolution of mobile broadband,” IEEE Commun. Mag., vol.47, no.4, pp.44–51, 2009.
[3] L. Benini, A. Bogliolo, and G.D. Micheli, “A survey of design techniques for system-level dynamic power management,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.8, no.3, pp.299–316, June 2000.
[4] B. Brock and K. Rajamani, “Dynamic power management for embedded systems,” Proc. 2003 IEEE International SOC Conference, pp.416–419, Sept. 2003.
[5] Y. Chang and S. Hung, “Developing collaborative applications with

mobile cloud? A case study of speech recognition," J. Internet Service and Information Security, vol.1, no.1, pp.18–36, 2011.
[6] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," Proc. 8th International Conference on Mobile Systems, Applications, and Services, pp.49–62, June 2010.
[7] J. Devore, Probability and Statistics for Engineering and the Sciences, 8th ed., Brooks/Cole, 2010.
[8] J. Furthmuller and O. Waldhorst, "Energy-aware resource sharing with mobile devices," Int. J. Computer and Telecommunication Networking, vol.56, no.7, pp.1920–1934, May 2012.
[9] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud application," Proc. ACM/IFIP/USENIX 10th International Conference on Middleware, pp.83–102, 2009.
[10] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud," Proc. IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, pp.784–791, July 2012.
[11] K. Kumar, J. Liu, Y. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," Mobile Networks and Applications, vol.18, no.1, pp.129–140, Feb. 2013.
[12] K. Kumar and Y. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," Computer, vol.43, no.4, pp.51–56, 2010.
[13] K. Kumar, Y. Nimmagddda, and Y. Lu, "Energy conservation for image retrieval on mobile systems," ACM Trans. Embedded Syst., vol.11, no.3, pp.66–88, Sept. 2012.
[14] B. Lee, "A framework for seamless execution of mobile applications in the cloud," Lecture Notes in Electrical Engineering, vol.126, pp.145–153, 2012.
[15] J. Lin, Z. Peng, C. Xiang, G. Ruan, Z. Yan, D. Natelson, and J.M. Tour, "Graphene nanoribbon and nanostructured SnO_2 composite anodes for lithium ion batteries," ACS Nano, vol.7, no.7, pp.6001–6006, June 2013.
[16] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," IEEE Pervasive Computing, vol.8, no.4, pp.14–23, Oct. 2009.
[17] P. Svensson, "Smartphones now outsell dumb phones," News NZ, April 2013.
[18] C. Xian, Y. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," Proc. 13th International Conference on Parallel and Distributed Systems, pp.1–8, Dec. 2007.

Byoung-Dai Lee is an assistant professor at the department of computer science, Kyonggi University, Korea. He received his B.S. and M.S. degrees in Computer Science from Yonsei University, Korea in 1996 and 1998 respectively. He received his Ph.D. degree in Computer Science and Engineering from University of Minnesota, Minneapolis, U.S.A. in 2003. Before joining the Kyonggi University, he worked at Samsung Electronics, Co., Ltd. as a senior engineer from 2003 to 2010. During the period, he has participated in many commercialization projects related to mobile broadcast systems. His research interests include cloud computing, mobile multimedia platform, and mobile multimedia broadcasting.

Kwang-Ho Lim is a master student at the department of computer science, Kyonggi University, Korea. His current research interest includes sensor networks, cloud resource managements, and mobile platforms.

Yoon-Ho Choi is the assistant professor at department of convergence security in Kyonggi University, Suwon, Korea. He received the M.S. and Ph.D. degrees from school of electrical and computer engineering, Seoul National University, S. Korea, in Sept. 2004 and Sept. 2008, respectively. He was a postdoctoral scholar in Seoul National University, Seoul, S. Korea from Sept. 2008 to Dec. 2008 and in Pennsylvania State University, University Park, PA, USA from Jan. 2009 to Dec. 2009. While working as a senior engineer at Samsung Electronics from May 2010 to Feb. 2012, he had deeply involved in developing commercial LTE CCC (Cloud Communication Center) system. He has served as TPC members in various international conferences and journals. His research interests include Deep Packet Inspection (DPI), high-speed intrusion prevention, mobile computing security, vehicular network security and SNS security.

Namgi Kim is an associate professor at the department of computer science, Kyonggi University, Korea. He received the B.S. degree in Computer Science from Sogang University, Korea, in 1997, and the M.S. degree and the Ph.D. degree in Computer Science from KAIST in 2000 and 2005, respectively. From 2005 to 2007, he was a research staff of the Samsung Electronics. Since 2007, he has been a faculty of the Kyonggi University. His research interests include sensor system, wireless system, and mobile communication