On Exploiting Dynamic Execution Patterns for Workload Offloading in Mobile Cloud Applications

Wei Gao*, Yong Li*, Haoyang Lu*, Ting Wang† and Cong Liu‡
*Department of Electrical Engineering and Computer Science, University of Tennessee at Knoxville
†IBM T. J. Watson Research Center
‡Department of Computer Science, University of Texas at Dallas
*{weigao, yli118, hlu9}@utk.edu, †tingwang@us.ibm.com, ‡cong@utdallas.edu

***Abstract*—Mobile Cloud Computing (MCC) bridges the gap between limited capabilities of mobile devices and the increasing users' demand of mobile multimedia applications, by offloading the computational workloads from local devices to the remote cloud. Current MCC research focuses on making offloading decisions over different methods of a MCC application, but may inappropriately increase the energy consumption if having transmitted a large amount of program states over expensive wireless channels. Limited research has been done on avoiding such energy waste by exploiting the dynamic patterns of applications' run-time execution for workload offloading. In this paper, we adaptively offload the local computational workload with respect to the run-time application dynamics. Our basic idea is to formulate the dynamic executions of user applications using a semi-Markov model, and to further make offloading decisions based on probabilistic estimations of the offloading operation's energy saving. Such estimation is motivated by experimental investigations over practical smartphone applications, and then builds on analytical modeling of methods' execution times and offloading expenses. Systematic evaluations show that our scheme significantly improves the efficiency of workload offloading compared to existing schemes over various smartphone applications.**

I. Introduction

Smartphones nowadays are designated to execute computationally expensive applications such as gaming, speech recognition, and video playback. These applications increase the requirements on smartphones' capabilities in computation, communication, and storage, and seriously reduce the smartphones' battery lifetime. A viable solution to support these applications is Mobile Cloud Computing (MCC) [24], which offloads the computation workloads from local smartphones to the remote cloud. Remote workload execution is then supported by various techniques such as code migration [5], [16] and Virtual Machine (VM) synthesis [2], [10], [11].

The major challenge of workload offloading in MCC is that wireless communication between smartphones and the remote cloud through 3G and WiFi is expensive. Hence, unconscious migration of full application processes with large datasets does not necessarily reduce the local energy consumption [17]. Instead, workloads are offloaded at a more fine-grained level of different application methods [9], [16], and an application is adaptively partitioned to ensure that the amount of energy saved by remote execution overwhelms the expense of wirelessly transmitting the relevant program states [21], [7].

This work was supported in part by the DOD Grant W911NF-14-1-0262.

However, limited research has been done on exploiting the dynamic patterns of applications' run-time execution for workload offloading. Most existing schemes assume fixed execution path among different application methods, and make workload offloading decisions through either empirical heuristics [21], [16] or solving deterministic optimization problems [22], [5]. The run-time heterogeneity of application execution paths due to different input datasets, user operations and system contexts are completely ignored. Moreover, the impact of this heterogeneity is also excluded from estimating the elapsed time and energy consumption of method execution. Such estimations in current schemes are only based on system profiling over individual methods themselves, resulting in inaccurate estimation of the practical cost of application execution.

The key to efficient exploitation of dynamic execution patterns is to develop an analytical framework which appropriately formulates the stochastic characteristics of application execution paths among different methods. Development of such framework is challenging in two aspects. First, a method invocation may relate to previous executions of multiple methods with heterogeneous system contexts, and it is difficult to characterize the invocation interdependency. Second, user applications are usually executed with heterogeneous system and network conditions, which complicate the quantitative analysis on the energy saving of workload offloading.

In this paper, we propose a novel scheme to address the aforementioned challenges and to make decisions on workload offloading with respect to the run-time application dynamics. Our basic idea is to formulate the interdependency among method invocations using an order-k semi-Markov model, and further characterize the transitions among application methods based on the sojourn time distributions of Markovian states. The appropriateness of workload offloading is then judged by a probabilistic decision framework which incorporates the knowledge about both method execution times and energy consumption. Parameters about method transitions, execution times, and energy consumptions are dynamically re-estimated at runtime during application executions by various online system profilers [15], [16]. The computational efficiency of the decisions of workload offloading is also ensured by restricting such decisions within a local scope of application execution. To the best of our knowledge, our work is the first which analytically formulates and exploits the stochastic characteristics of run-time application execution for workload offloading.

978-1-4799-6204-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ICNP.2014.22

Our detailed contributions are as follows:

- We experimentally investigate the run-time execution patterns of various smartphone applications. We observe significant heterogeneity in the run-time execution paths of these applications, as well as the execution times and energy consumption of different application methods.
- Based on the investigation results, we develop a semi-Markov framework which formulates the stochastic transitions among application methods, and provide quantitative guidelines on offloading decisions through probabilistic estimations of the offloading effectiveness.
- We propose analytical modeling on various aspects of system dynamics during application executions, including the execution times and energy consumption of application methods. These models are then integrated into decisions of workload offloading.

The rest of this paper is organized as follows. Section II reviews the existing work. Section III presents our experimental investigations on practical smartphone applications and motivates our idea of the stochastic offloading framework. Section IV describes the details of our framework, and Section V presents analytical modeling on system dynamics. Section VI evaluates the performance of our proposed approach. Section VII discusses and Section VIII concludes the paper.

II. Related Work

MCC integrates cloud computing into the mobile environment [6] and allows mobile users to efficiently utilize the cloud resources. Cloudlets, which are local resource-rich servers providing prompt cloud access to nearby mobile users, have been suggested to avoid the wireless transmission latency between smartphones and the remote cloud [24]. Other designs adopt various cloud computing techniques such as virtualization [12] and Service-Oriented Architecture (SOA) [26].

Workload offloading in MCC focuses on *how* to offload and *what* to offload. Research has been done to support efficient remote application execution. MAUI [5] relies on developers to specify the application partitioning by annotating remoteable methods under the specific application framework, i.e., Microsoft .NET. Later solutions improve the efficiency and reliability of workload offloading through synthesis and migration of VMs [4], [29]. CloneCloud [3], [2] creates an augmented clone of the local application on the cloud, and ThinkAir [16] enforces on-demand VM creation and resource allocation. Other schemes further improve the offloading generality by supporting multi-threaded [10] and interactive applications [21].

Appropriate decisions of application partitioning, on the other hand, are the prerequisite to efficient workload offloading. Such decisions are based on the profiling data about application execution and system context, such as the CPU usage, energy consumption, and network latency. Some schemes such as CloneCloud [2] and MAUI [5] implement offline profiling with various benchmark testing, and other schemes [15], [16], [27] use online application profilers to monitor application executions. Based on profiling data, most schemes partition user applications using empirical heuristics with specific assumptions. Odessa [21] assumes linear speedup over consecutive frames in a face recognition application. ThinkAir [16] defines multiple static offloading policies, each of which focuses on a sole aspect of system performance. The efficiency of offloading decisions in practice, however, are left unexamined and questionable. In contrast, our proposed approach considers the most generic MCC scenario, and the proposed stochastic offloading framework builds on experimental investigations of the execution patterns of practical smartphone applications.

Analytical offloading framework has also been studied [22], [9], [5], [7], [19], but they assume the method invocations to be represented by a stationary calling graph. Offloading decisions are formulated as a graph cut problem [22], [9], integer optimization [5], or fuzzy logic decision process [7], but all the offloading operations are deterministic regardless of the run-time dynamics of application execution. Comparatively, in this paper we propose to make decisions of workload offloading based on probabilistic transitions among application methods that are formulated by a stochastic framework, and hence ensure the practical applicability of workload offloading.

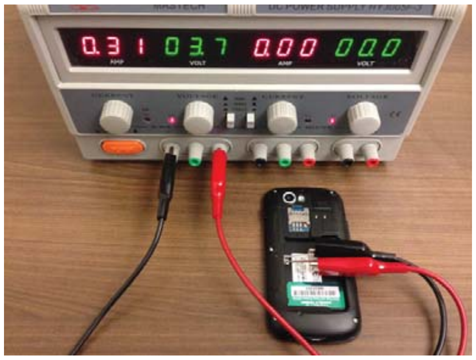


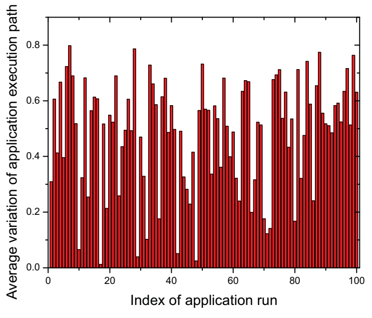
Fig. 1. Power metering system for energy measurement

III. Experimental Investigation

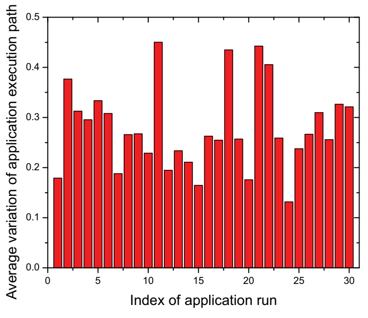
In this section, we motivate the development of a stochastic framework depicting the dynamic execution patterns of user applications, through experimental investigations of the run-time executions of the following smartphone applications:

- **Firefox for Mobile** with version 24.0 is one of the most popular web browsers on smartphones[1]. It supports various types of interactive media contents such as Flash and JavaScript, and hence has heterogeneous execution patterns in both computation and communication.
- **Chess-Walk** is a chess game which supports either offline or online gaming[2], and is computationally expensive according to different configurations of game difficulty levels.

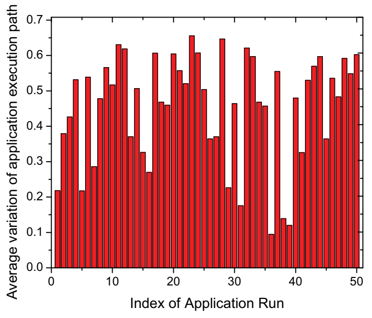
[1]http://hg.mozilla.org/mozilla-central/archive/tip.tar.gz
[2]https://gitorious.org/chesswalk



(a) Firefox

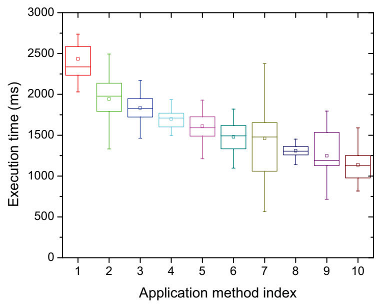


(b) Chess-walk

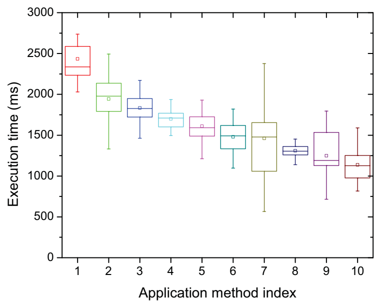


(c) Barcode scanner

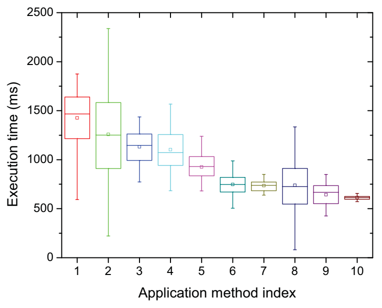
Fig. 2. Run-time dynamics of application execution paths



(a) Firefox



(b) Chess-walk



(c) Barcode scanner

Fig. 3. Run-time heterogeneity of methods' execution times

- **Barcode Scanner** is an application that scans products' barcodes and then searches online for product prices and reviews[3]. It is usually communication intensive due to the functionality of online search.

Our experiments use the Android Operating System (OS) as the software platform. We analyze the source codes of each user application, and embed the specialized monitoring codes, which are connected to our backend software monitor, into the application. Such software monitor is implemented as an Android system service, and undertakes the application profiling action every time when an application method starts or finishes. The methods' execution times are recorded based on the smartphones' local time measurement, and their power consumptions are measured using a hardware power meter as shown in Figure 1. All the experiments are performed on a Samsung Nexus S smartphone with a 1GHz Cortex-A8 CPU, 512MB RAM, and a 100-Mbps wireless LAN. Each application is executed several times with different input datasets and operations. For Firefox, we randomly load 100 different webpages with various page layouts and interactive contents. For Chess-walk, we play 30 games with different difficulty levels. For Barcode scanner, we randomly scanned and searched 50 different electronic products in a local BestBuy store.

[3]http://code.google.com/p/zxing/

TABLE I
RUN-TIME DYNAMICS OF APPLICATION EXECUTION PATHS

Application	Firefox	Chess-walk	Barcode
Avg No. of methods involved	93.7	34.7	57.9
Avg No. of methods executed per run	122.4	52.5	76.3
Avg method execution time (ms)	378.8	897.3	577.5
Avg variation of execution path	0.484	0.278	0.455

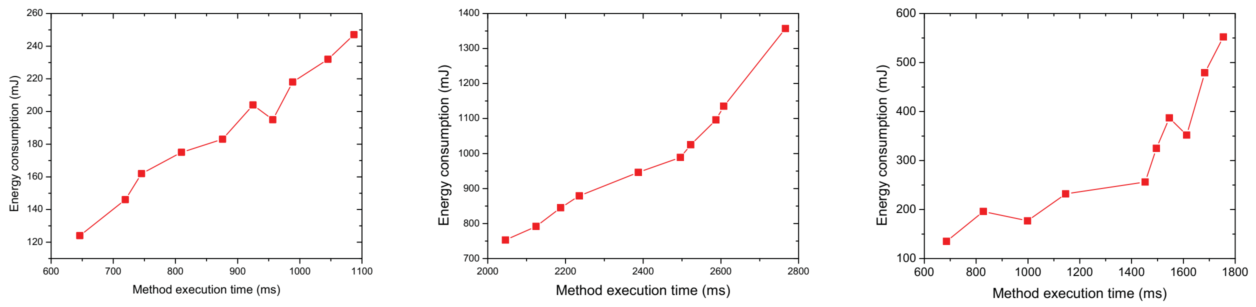
A. Run-time Dynamics of Application Execution Paths

We first focus on the variations of method execution paths in different application runs. For each application, we index all the application methods and record the application execution path in each application run. The sequence of method executions in each application run with the specific input dataset S is recorded as a time series $E_S = \{m_1^S, m_2^S, ..., mL_S{}^S\}$, where m_i^S is the i-th application method being executed. Then, we calculate the difference between two time series E_{S_i} and E_{S_j} as $D_{ij} = \sum_{k=1}^{N} d_k/N$ where $N = \max\{L_i, L_j\}$ and

$$d_k = \begin{cases} 0 & \text{if } m_{ik} = m_{jk} \\ 1 & \text{otherwise} \end{cases}$$

In case that $L_i < L_j$, S_i will be padded with 0s so that the corresponding values of d_k are all 1s, and vice versa.

The variations of method execution paths among different application runs are illustrated in Figure 2. For the i-th application run, Figure 2 shows the average variance between E_{S_i} and all the other time series, i.e., $\sum_{j=1,j\neq i}^{K} D_{ij}/(K-1)$,



(a) Firefox (b) Chess-walk (c) Barcode scanner

Fig. 4. Run-time heterogeneity of methods' energy consumption

where K is the total number of application runs. The statistical results are then summarized in Table I. From these results, we see that the executed methods in multiple application runs are highly possible to be different from each other, and such difference is usually determined by the application contexts and input datasets. For applications handling complicated and interactive media contents such as Firefox, Figure 2(a) shows that the execution variation can be up to 0.8. Applications with specialized functionality exhibit lower variation, but Figure 2(b) shows that the variation for the Chess-walk application could still be as high as 0.4.

B. Run-time Heterogeneity of Application Characteristics

Second, we investigate the run-time heterogeneity of application characteristics, including the execution times and energy consumption of different application methods. For each smartphone application, we investigate its top 10 methods with the maximum average execution times, and demonstrate the variations of their execution times in different runs in Figure 3. We observe that the run-time method execution times exhibit significant variances, which could be over 50% in the Chess-Walk and Barcode Scanner applications. Such variances are usually method-specific and determined by the specific contexts of method execution, such as the input dataset and application execution path. In particular, we observe that the execution times of a specific method will be significantly varied by the next method being invoked.

Similarly, for the method with the maximum average execution time in each application, we show the fluctuations of its run-time energy consumption along with its execution times in different runs in Figure 4. We see that the energy consumption of method execution increases along with the method execution time, but the increasing speed is application-specific. Computation-intensive applications such as Chess-Walk have smaller fluctuation of methods' energy consumption, as shown in Figure 4(b). Moreover, randomness and irregularity could also be observed. In Figures 4(a) and 4(c), the energy consumption of a method execution with longer time could also be possibly lower.

C. Motivation of Stochastic Offloading Framework

The above results show that the run-time transitions between application methods are highly dynamic and therefore should be formulated as a continuous-time stochastic process. Markov Decision Processes (MDPs), in both discrete-time [13] and continuous-time domains [23], have been exploited to formulate the run-time behavior patterns of mobile applications, in which the application components are mapped to Markovian states in different ways. In [23], the Markovian states correspond to the application context information, and the states in [13] are vectors of variables indicating the system status.

However, an ordinary Markov model is insufficient for fine-grained formulation of the transitions between application methods, whose characteristics of transition times and invocation independency could be heterogeneous. This is our motivation of adopting an order-k semi-Markov model. Since a semi-Markov model supports arbitrary distributions of sojourn times between Markovian states, we can associate each application method with a Markovian state and characterize the transition times between methods by re-estimating the parameters of state sojourn time distributions. Moreover, an order-k Markov model enables appropriate formulation of the impact of previous method invocations on the future application execution, and flexibly balancing between the formulation accuracy and overhead by adjusting the value of k.

IV. Stochastic Offloading Framework

In this section, motivated by the experimental results in Section III, we develop our stochastic framework for efficient workload offloading decisions considering the dynamic execution patterns of smartphone applications. Our basic idea is to formulate the dynamic transitions of different methods of a specific user application as a stochastic process, more specifically, a order-k semi-Markov model, and further make the workload offloading decisions based on probabilistic estimations of the offloading effectiveness. Such estimations build on the variations of methods' execution times and offloading expenses, with respect to the notations in Table II, which are used throughout the rest of this paper.

TABLE II
NOTATION SUMMARY

Notation	Explanation
$\mathbb{M}$	The set of application methods being executed
M_n	The n-th application method being executed
T_n	The time when M_n is invoked
$\tilde{M}_n$	The composite Markovian state $\{M_{n-1}, ..., M_{n-k}\}$
$\mathbf{T}$	State transition probability matrix for the embedded order-k Markov chain
$H_{ij}(t)$	Sojourn time distribution transiting from $i \in \mathbb{M}^k$ to $j \in \mathbb{M}$
C_L^j, C_R^j	Method j's probabilities of being executed locally and remotely, respectively
$E_{M_n}(t)$	Energy consumed by M_n with local execution time t
$C_{M_n}(t)$	Energy consumed by transmitting M_n's program states with local execution time t

A. System Model

The computational workloads of different methods of a user application are adaptively offloaded from the local smartphone to the remote cloud, so as to minimize the smartphone's local resource consumption. Since an application method may request access to various local system resources, in this paper we consider that the application programmer is responsible to specify whether a method is remoteable before the deployment and execution of the application. Similar assumption is also made by the existing work [5], [16].

We consider a generic scenario of MCC which is also used in [5]. We assume that the remote execution of an application method causes negligible computational overhead to the remote cloud, but may incur the smartphone's local resource consumption of transmitting the program states before and/or after this remote execution. As shown in Figure 5, such transmission is necessary when other parts of the application either invoking this method or being invoked by this method are executed locally, and could only be ignored between consecutive remote executions of application methods.

Modern smartphone OS enables multiple user applications to be running simultaneously. However, their executions are generally independent from each other. For example, Android OS encapsulates the run-time binaries of each user application into an instance of Davlik Java VM, so that the resource space of this application is isolated from the rest of the system. Most existing schemes realize workload offloading based on the techniques of VM migration and remote synthesis. Therefore, the rest of this paper will focus on ensuring the workload offloading efficiency of a single user application. The consideration of the mutual impact between different user applications is orthogonal to the major focus of this paper.

B. Semi-Markov Chain Formulation

An order-k Markov renewal process representing the transitions between different application methods is a set of random variables $\{(M_n, T_n) : n \geq 0\}$, where state space $\mathbb{M} = \{m_1, m_2, ..., m_N\}$ is the set of user application methods. The random variable $T_{n+1} - T_n$ thus indicates the execution time of M_n. Without loss of generality, we develop our stochastic offloading framework with discrete-time, and various time

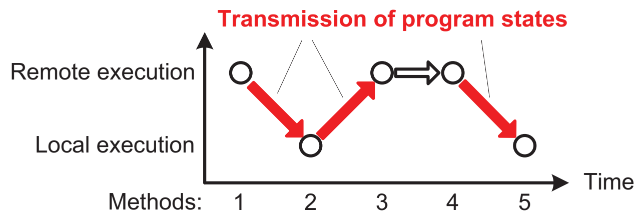


Fig. 5. Model of transmitting program states between method executions

metrics, such as the number of CPU cycles, can be adopted to quantify the values of T_n.

We consider that the invocation of a method M_{n+1} is determined by the previous k method invocations that are indicated by the "composite state" $\tilde{M}_n = \{M_{n-k+1}, ..., M_{n-1}, M_n\}$. Correspondingly, we use an extended state transition probability matrix $\mathbf{T} \in \mathbb{R}^{N^k \times N}$ for the embedded order-k Markov chain, such that for any $p_{ij} \in \mathbf{T}$, $j \in \mathbb{M}$ represents a regular system state, and $i \in \mathbb{M}^k$ corresponds to a k-tuple of values in the original state space $\mathbb{M}$. In practice, the values of p_{ij} are dynamically updated by the profiled execution history of user application methods.

As a result, the associated order-k semi-Markov kernel Q is defined by

$$\begin{aligned} Q_{ij}(t) &= \mathbb{P}(M_{n+1} = j, T_{n+1} - T_n \leq t | \tilde{M}_n = i) \\ &= p_{ij} H_{ij}(t), \end{aligned} \tag{1}$$

where $H_{ij}(t)$ indicates the probability that the method j will be invoked by the composite state i at or before time t. Based on $H_{ij}(t)$, we have the distribution $D_i(t)$ indicating the average holding time of the composite state i as

$$D_i(t) = \mathbb{P}(T_{n+1} - T_n \leq t | \tilde{M}_n = i) = \sum_{j=1}^{N} Q_{ij}(t)$$

Without loss of generality, we consider that $H_{ij}(t)$ covers both cases of j's local and remote executions, and hence

$$H_{ij}(t) = c_L^{ij} h_L^{ij}(t) + c_R^{ij} h_R^{ij}(t), \tag{2}$$

where $h_L^{ij}(t)$ and $h_R^{ij}(t)$ correspond to j's sojourn time distribution when it is executed locally and remotely, respectively. The exponents c_L^{ij} and c_R^{ij} indicate the method j's probability of local and remote execution, respectively, when j is invoked from the composite state i. The cumulative probability for the method j to be executed locally and remotely can then be calculated as $C_L^j = \sum_{i \in \mathbb{M}^k} c_L^{ij}$ and $C_R^j = \sum_{i \in \mathbb{M}^k} c_R^{ij}$. As a result, we will make our offloading decisions based on such knowledge about the sojourn time distributions indicating the method execution times.

C. Offloading Decisions

At time T_n when the method M_n is about to be invoked, our basic idea of determining whether to offload the execution of M_n to the remote cloud is to adaptively evaluate the effectiveness of M_n's remote execution on saving the smartphone's

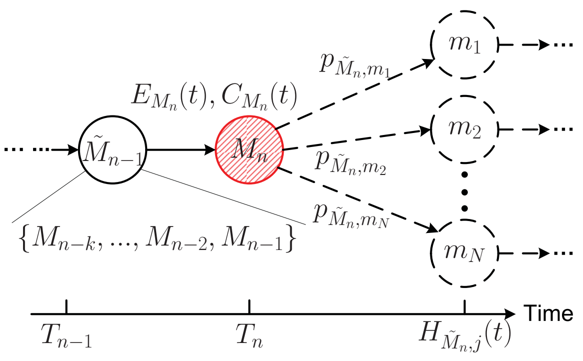


Fig. 6. Local Invocation Graph (LIG) of the current method M_n

local energy, and only offload the execution of M_n if the amount (G_{M_n}) of energy saving is non-negative.

The calculation of G_{M_n} takes into account the stochastic characteristics of M_n's execution times and energy consumption, and is restricted within the scope of the Local Invocation Graph (LIG) of M_n, so as to ensure the computational efficiency of such offloading decision without sacrificing the decision correctness. The LIG of a method M_n, as illustrated in Figure 6, is defined to include the composite state $\tilde{M}_{n-1}$ determining the invocation of M_n, and the other application methods that may be invoked by M_n in the future.

Without loss of generality, we consider the possibility for M_n to invoke every other application method $m_j \in \mathbb{M}$, which is indicated by the state transition probability $p_{\tilde{M}_n,j}$. If M_n will invoke m_j after completing M_n's own execution, G_{M_n} equals to the difference between the energy consumption of M_n's local execution and the transmission of M_n's program states after its remote execution. According to Section IV-A, such transmission is only necessary when m_j is executed locally. As a result, at time T_n when M_n is about to be invoked by M_{n-1}, we compute G_{M_n} as

$$G_{M_n} = \sum_{j=1}^{N} p_{\tilde{M}_n,j} \cdot \left(\int_0^\infty h_L^{\tilde{M}_n,j}(t) \cdot (E_{M_n}(t) - C_L^j C_{M_n}(t))dt \right) - \begin{cases} 0 & , M_{n-1} \text{ is executed remotely} \\ -C_{M_{n-1}}(T_n - T_{n-1}) & , M_{n-1} \text{ is executed locally,} \end{cases} \tag{3}$$

where $h_L^{\tilde{M}_n,j}(t)$ and C_L^j are defined in Eq. (2), and we consider that both $E_{M_n}(t)$ and $C_{M_n}(t)$ vary along with M_n's local execution time t. Since the transmission of program states is only necessary when m_j will be executed locally, Eq. (3) essentially provides a lower bound of energy saving that happens when both M_{n-1} invoking M_n and M_{n+1} being invoked by M_n are executed locally. Note that energy consumption of transmitting M_{n-1}'s program states, which is indicated by $C_{M_{n-1}}(T_n - T_{n-1})$, is fixed by T_n and T_{n-1}.

From Eq. (3), we see that the offloading decision on M_n highly depends on whether other application methods invoking M_n or being invoked by M_n are offloaded or not. As illustrated by Figure 5, the "ping-pong" operations on workload offloading, i.e., frequent switches between the local and remote method executions, simply waste the majority of consumed energy on unnecessary transmission of program states between local smartphones and the remote cloud, and lead to severe penalties on the system's energy efficiency. Such penalties are reflected in Eq. (3) by incorporating the quantity $C_{M_n}(t)$.

V. System Dynamics of Application Execution

In this section, we further substantiate our stochastic offloading framework presented in Section IV by providing detailed formulations on various aspects of system dynamics of application executions. These formulations enable quantified guidelines for the workload offloading operations in practice. For each aspect of the system dynamics, we also provide analytical methods to adaptively re-estimate the parameters describing such dynamics, using the runtime profiled information about the real-time status of application execution. These information could be efficiently provided by various online application profilers [15], [16] and is then used to ensure the practical appropriateness of workload offloading decisions.

A. Sojourn Time Distribution

The sojourn time distributions of Markovian states representing the user application methods are the key to the efficiency and correctness of workload offloading decisions. In this section, we focus on determining the appropriate characteristics of such distribution with respect to the context of mobile workload offloading, and further develop algorithms to efficiently re-estimate the parameters of such distribution at real-time.

The characteristics of sojourn time distributions, which determine the transitions among Markovian states representing application methods, mainly depend on whether an application method is executed locally or remotely. According to our experimental investigations in Section III-B, when a method is executed locally, its execution time is determined by the instantaneous computational capability of the local smartphone, which could be highly dynamic due to the current system workload (e.g., CPU utilization, system memory usage) and resource availability. When an application method is executed remotely, the dominant factor determining its execution time, on the other hand, is the network bandwidth and data transmission latency. As suggested by MAUI [5], such wireless network conditions are also dynamic, and would significantly affect the method execution times when the amount of data being transmitted for remote execution is large. Due to such dynamic characteristics of method executions, we consider that both the local and remote method execution times follow Gaussian distributions, and formulate the sojourn time distribution $H_{ij}(t)$ as a Gaussian mixture, such that

$$\begin{aligned} H_{ij}(t) &= c_L^{ij} h_L^{ij}(t) + c_R^{ij} h_R^{ij}(t) \\ &= c_L^{ij} \mathbb{N}(t|\mu_L^{ij}, (\sigma_L^{ij})^2) + c_R^{ij} \mathbb{N}(t|\mu_R^{ij}, (\sigma_R^{ij})^2), \end{aligned} \tag{4}$$

where $\mathbb{N}(\cdot)$ is Gaussian density form. Such Gaussian mixture is illustrated in Figure 7. The decision of workload offloading in Eq. (3) uses quantities $h_L^{ij}(t)$ and C_L^j from Eq. (4).

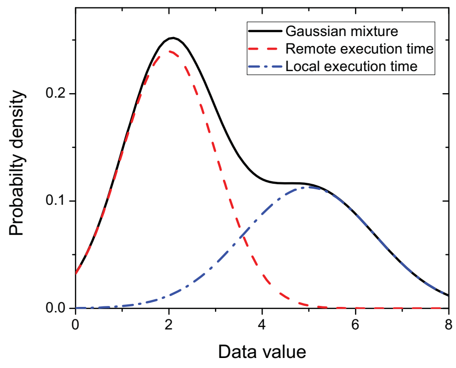


Fig. 7. Gaussian mixture model of method execution times

The most common way to re-estimate the parameters of $H_{ij}(t)$ is to apply the Expectation-Maximization (EM) algorithm [28] over the profiling information about the past method execution times. However, the accuracy of such re-estimation under the Maximum Likelihood (ML) criterion, which depends on large training datasets and multiple rounds of iteration, would be seriously impaired by the limited availability about such profiling information.

Instead, we adopt the Maximum A Posteriori (MAP) method [8] for the limited training data. Given a prior parameter set λ_0 of $H_{ij}(t)$ and the training data set $\mathbf{X} = \{x_1, ..., x_T\}$, we compute the *a posteriori* probability for the data sample x_t and the m-th Gaussian component of $H_{ij}(t)$ as

$$\mathbb{P}(m|x_t, \lambda_0) = \frac{c_m^{ij}\mathbb{N}(x_t|\mu_m^{ij}, (\sigma_m^{ij})^2)}{\sum_{p\in\{L,R\}} c_p^{ij}\mathbb{N}(x_t|\mu_p^{ij}, (\sigma_p^{ij})^2)}$$

where $m = \{L, R\}$. Based on such probabilities, the parameters of $H_{ij}(t)$ are re-estimated as

$$\hat{c}_m^{ij} = \sum_{t=1}^{T} \mathbb{P}(m|x_t, \lambda_0),$$
$$\hat{\mu}_m^{ij} = \frac{1}{\hat{c}_m^{ij}} \sum_{t=1}^{T} \mathbb{P}(m|x_t, \lambda_0)x_t,$$
$$(\hat{\sigma}_m^{ij})^2 = \frac{1}{\hat{c}_m^{ij}} \sum_{t=1}^{T} \mathbb{P}(m|x_t, \lambda_0)x_t^2 - (\hat{\mu}_i^{ij})^2.$$

In practice, the computational complexity of parameter re-estimation with T training data samples is $O(T)$. Such re-estimation overhead hence could be efficiently controlled by varying the size of training dataset and achieving different tradeoffs between the responsiveness and accuracy of parameter re-estimation. Adopting a smaller dataset enables the Gaussian mixture model to promptly capture the up-to-date dynamics of method execution times, but reduces the accuracy of parameter re-estimation due to the limited training data. A larger dataset, on the other hand, increases such accuracy at the risk of missing important variations of system characteristics.

B. Energy Consumption of Local Execution

The energy consumption $E_j(t)$ of the method j's local execution, as described by Section IV and experimentally validated by Section III-B, is determined by j's local execution time t. Although various energy models such as JouleMeter [14] have been proposed to measure the relationship between the method execution time and energy consumption, they are all empirical models based on experimental data and ignore the probabilistic uncertainties of method executions.

Our experimental investigation results in Figure 4 have shown that such energy consumption generally increases along with t. Therefore, our basic idea is to formulate the relationship between the energy consumption and local execution time of an application method using a m-order polynomial, i.e.,

$$E_j(t) = \sum_{i=1}^{m} a_i^j t^i, \tag{5}$$

where $E_j(0) = 0$ and the coefficients a_i^j can be adaptively re-estimated through real-time linear regression analysis [25] over the profiled information about method execution times and energy consumption, which can be written as tuples $\{(t_j^1, E_j^1(t_1)), (t_j^2, E_j^2(t_1)), ...\}$. Since the local method execution times are specified in Section V-A to be normally distributed, we know that $E_j(t)$, as a random variable, follows chi-square distribution with m degrees of freedom.

It is obvious that the value of m is vital to the effectiveness of such modeling, and an inappropriate value of m would lead to the problem of either "under-fitting" or "over-fitting". In practice, a larger value of m is adopted to depict the randomness of energy consumption characteristics. As shown in Figure 4, such randomness is mainly exhibited by the non-monotonicity between the variations of energy consumption and method execution times. Therefore, our framework adaptively selects the value of m at real-time according to the current level of system dynamics. Such value is determined as proportional to the variances of local execution time distribution, i.e., $(\sigma_L^{ij})^2$ as specified in Eq. (4). The larger variance exists among the local execution times of method j, the more likely that the energy consumption of these executions to be heterogeneous, and hence a larger m should be adopted.

To better depict such randomness of energy consumption characteristics, we also extend the above model to consider each polynomial coefficient a_i^j as a Gaussian random variable, i.e., to use $\widetilde{a}_i^j \sim \mathbb{N}(a_i^j, (\sigma_i^j)^2)$ replacing a_i^j in Eq. (5). As a result, for any given t of the method j's local execution time, we probabilistically predict the energy consumption of j's local execution in form of a Gaussian distribution. The variances of $\widetilde{a}_i^j$ are determined, being similar to the choice of m, according to the dynamics of j's local executions.

C. Energy Consumption of Transmitting Program States

The transmission of program states, as illustrated by Figure 5, only happens when application methods are invoked consecutively but executed at different network locations (i.e., local devices or the remote cloud). Previous studies [5], [30] have shown that the energy consumption of such data transmission is determined by the wireless network conditions, rather than the amount of data being transmitted. The majority of energy is consumed on establishing the wireless network connections instead of actual data transfer. The more unstable the wireless connection is, the more frequent the end-to-end data transmission sessions will be disconnected and resumed, and the higher energy is consumed on data transmission. Therefore, the energy consumption $C_j(t)$ of transmitting method j's program states is also formulated as a function of j's local execution time t. The longer a method is executed, the data transmission is more likely to last longer and hence there is higher chance for the wireless network to be disconnected. More specifically,

$$C_j(t) = \sum_{i=1}^{m} b_i^j t^i, \tag{6}$$

and the same order of m is used in $C_j(t)$ for simplicity.

However, being different from Section V-B which considers the polynomial coefficients as normally distributed random variables, we believe that the probabilistic features of such energy consumption of data transmission would be highly skewed. Therefore, b_i^j are extended to be heavy-tailed distributions. We consider that the distributions of b_i^j could be either exponential or power-law, and refer the detailed investigation of such probabilistic characteristics to be the future work.

D. Quantified Operations of Workload Offloading

By combining Eqs. (4), (5), and (6), we derive the analytical solution of the workload offloading decision described in Eq. (3). In particular, we are able to simplify the integration specified by Eq. (3) by exploiting the similarity of Eqs. (5) and (6), both of which are m-order polynomials.

We focus on the non-deterministic part of Eq. (3), which is

$$\tilde{G}_{M_n} = \sum_{j=1}^{N} p_{\tilde{M}_n,j} \cdot \left(\int_0^\infty h_L^{\tilde{M}_n,j}(t) \cdot \sum_{s=1}^{m} p_s^j t^s dt \right),$$

where $p_s^j = a_s^j - C_L^j \cdot b_s^j$.

Since the value of each p_s^j is independent from each other and the value of t, we have

$$\begin{aligned}
\tilde{G}_{M_n} &= \sum_{j=1}^{N}\sum_{s=1}^{m} p_s^j \cdot \int_0^\infty h_L^{\tilde{M}_n,j}(t) \cdot t^s dt \\
&= \sum_{j=1}^{N}\sum_{s=1}^{m} \frac{p_s^j}{\sqrt{2\pi}\sigma_L^{ij}} \cdot \int_0^\infty \exp(-\frac{(t-\mu_L^{ij})^2}{2(\sigma_L^{ij})^2}) \cdot t^s dt \\
&= \sum_{j=1}^{N}\sum_{s=1}^{m} \frac{2^{(s-3)/2} \cdot p_s^j}{\pi(\sigma_L^{ij})^{(1-s)}} \cdot \Gamma(\frac{1+s}{2}) \cdot \\
&\left(\frac{\mu_L^{ij}(1+s)}{\sqrt{2}} M(\frac{1-s}{2}, \frac{3}{2}, \frac{(\mu_L^{ij})^2}{2(\sigma_L^{ij})^2}) + \sigma_L^{ij} M(-\frac{s}{2}, \frac{1}{2}, \frac{(\mu_L^{ij})^2}{2(\sigma_L^{ij})^2}) \right),
\end{aligned} \tag{7}$$

where $\Gamma(t)$ is the Gamma function and $M(a,b,z) = \sum_{n=0}^{\infty} \frac{a(a+1)\ldots(a+n-1)\cdot z^n}{b(b+1)\ldots(b+n-1)\cdot n!}$ is the confluent hypergeometric function, a.k.a., Kummer's function [1]. By substituting Eq. (7) to Eq. (3), we calculate the the amount of energy saved by offloading method M_n as

$$G_{M_n} = \tilde{G}_{M_n} - \begin{cases} 0 & \text{if } m_{ik} = m_{jk} \\ 1 & \text{otherwise.} \end{cases} \tag{8}$$

In practice, both Gamma and Kummer functions could be efficiently computed by non-iterative numerical methods, such as Lanczos approximation [18] or rational approximation [20]. Therefore, the practical computational complexity of calculating Eq. (7) is $O(m \cdot N)$. Therefore, compared to existing schemes which require solving complicated graph-cut [22], [9] or integer optimization [5] problems for the offloading decision for each application method, our proposed method is far more computationally efficient, and is able to ensure prompt offloading operations in computational-intensive scenarios.

VI. Performance Evaluation

In this section, we evaluate the effectiveness of our proposed framework on reducing the local devices' resource consumption. Our proposed framework is compared with the following schemes which make decisions of workload offloading based on static execution patterns of user applications:

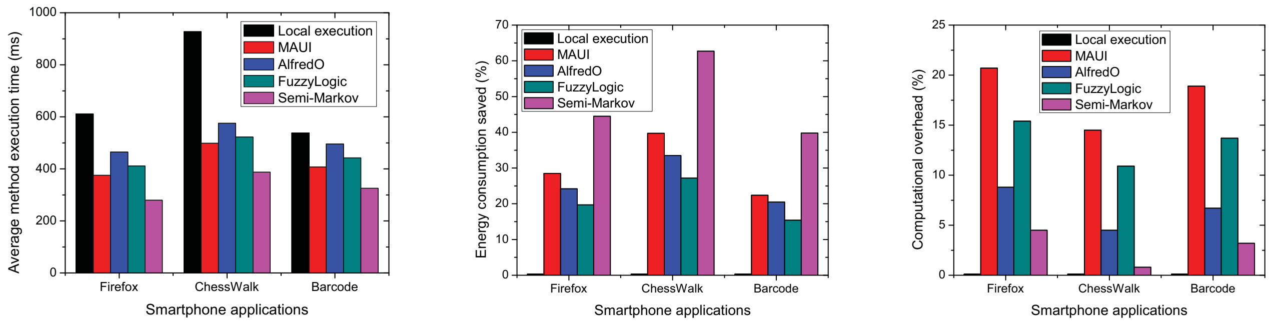
- **MAUI** [5]: the decision of workload offloading is made by solving an integer optimization problem based on a global calling graph among application methods.
- **AlfredO** [22], [9]: the offloading decision is based a calling graph similar to MAUI, but aims to find an optimal graph cut at the local scope of application execution.
- **FuzzyLogic** [7]: The decision of workload offloading is made by a fuzzy logic engine to adapt to the continuous changes of application executions. Each input parameter of method execution is transformed into respective fuzzy sets, to which various offloading rules are applied.

The following metrics are used in our evaluations. Each experiment is repeated multiple times with random input datasets or user operations for statistical convergence.

- **Method execution time,** the average elapsed time of method executions over multiple experiment runs.
- **Amount of energy saved,** the percentage of local energy consumption saved by workload offloading.
- **Computational overhead,** the smartphones' local energy consumption for decisions of workload offloading. Particularly, such overhead is measured as the percentage of the energy consumption of application executions.

A. Evaluation Setup

We evaluate the efficiency of workload offloading over the three smartphone applications being used for experimental investigation in Section III. We embed the implementation of our offloading framework into the binaries of each user application, and our offloading operations follow the method of CloneCloud [2], such that a clone VM is maintained at our cloud server for each smartphone application running foreground. Each time before an application method is invoked, our offloading engine is invoked first and evaluate



(a) Method execution time (b) Amount of energy saved (c) Computational overhead

Fig. 8. Effectiveness of workload offloading

the practical effectiveness of the method's remote execution according to Eq. (7). An application method is only offloaded for remote execution if such operation could actually reduce the local energy consumption, and our offloading engine is responsible for triggering the remote execution by transmitting the method's program states to the cloud server.

Our evaluations use Samsung Nexus S smartphones with Android v4.2, and a Dell OptiPlex 9010 PC with an Intel i7-3770@3.4GHz CPU and 8GB RAM as the cloud server. The smartphones are connected to the cloud server via 100Mbps campus WiFi. We adopt the techniques proposed in MAUI [5] for method profiling, and collect information about the execution times and energy consumption of method executions. The power consumptions of method execution and network data transmission are measured in the similar way as described in Section III, and the remote method execution times being measured include the time of wirelessly transmitting the program states between the local mobile devices and the remote cloud. Such profiling information is then used to re-estimate the parameters of semi-Markov models and functions describing system dynamics.

Each experiment lasts for 10 minutes, during which the smartphone applications are operated by multiple human beings with heterogeneous operational behavior patterns and real-world application scenario contexts. Such evaluation methodology, being different from the randomly generated application workloads, is able to better depict the practical run-time dynamics of user applications in real-world scenarios. More specifically, for Firefox, heterogeneous webpages with different types of interactive contents are browsed. For ChessWalk, users play games with their specified difficulty levels. For Barcode Scanner, users scan different product barcodes at random intervals during the 10-minute experiment period. Offloading decisions are then made independently for each application method being invoked during the experiment.

B. Offloading Effectiveness

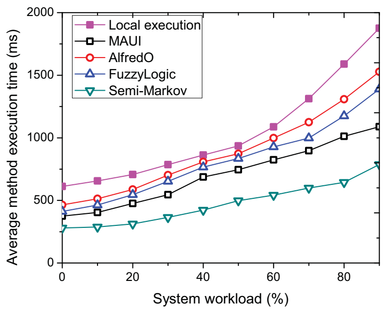
We first compare the offloading effectiveness of our proposed framework with other schemes. Smartphone applications are executed with idle phone status[4], and an order-3 semi-Markov model is applied. The results are shown in Figure 8. By selectively offloading the appropriate workload to the cloud server, our framework dramatically reduces the application method execution times. As shown in Figure 8(a), the reduction of method execution time achieved by our framework exceeds 50% in all applications. Comparatively, such reductions achieved by the existing schemes are generally below 25%, mainly due to their ignorance of the dynamic patterns of application execution when making decisions of workload offloading. In particular, MAUI has the lowest offloading effectiveness because it considers all the execution paths in user applications to be fixed.

Meanwhile, our framework also reduces the energy consumption of method execution. As shown in Figure 8(b), such energy saving is application-dependent and determined by the level of run-time dynamics of application executions. In ChessWalk where most application operations are logical calculations, our framework saves over 60% of the smartphone energy. Comparatively, such energy savings in Firefox and Barcode Scanner are relatively lower, but our proposed framework generally outperforms other existing schemes by over 20%. The major reason of such advantage is that our scheme is able to achieve more accurate estimations about the actual energy consumed by method executions, through investigation and integration of dynamic application execution patterns.

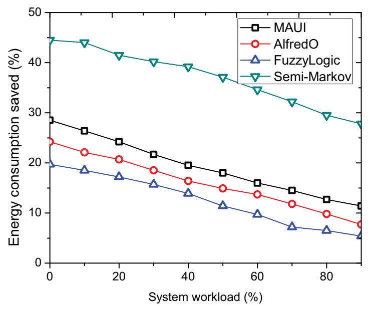
We also evaluate the computational overhead of making workload offloading decisions. As shown in Figure 8(c), such overhead of our proposed framework is lower than 5% of the actual application execution in all cases. Comparatively, decisions of workload offloading on MAUI takes up to 20% of the system resources, which are mainly used to solve the integer optimization problems with respect to every method invocation. The overhead of AlfredO is lower due to its focus on a local optimum of the graph-cut problem, but is still around 50% higher than our scheme.

We further evaluate the effectiveness of workload offloading with different system workloads. Such evaluation is conducted

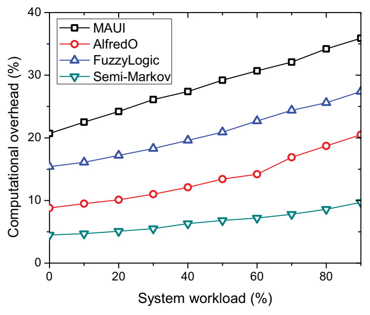
[4]The current system workload is minimized to avoid the impact of other concurrent user applications on the workload offloading decisions.



(a) Method execution time

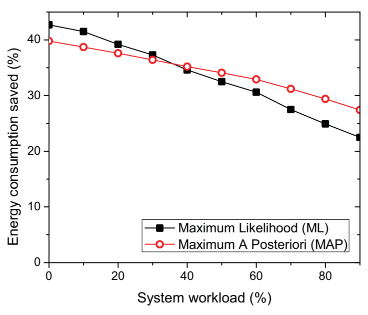


(b) Amount of energy saved

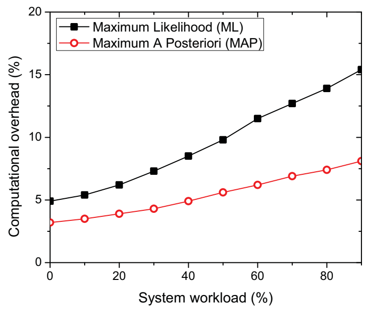


(c) Computational overhead

Fig. 9. Effectiveness of workload offloading with different levels of system workload



(a) Amount of energy saved



(b) Computational overhead

Fig. 10. Effectiveness of workload offloading with different methods estimating the parameters of sojourn time distribution

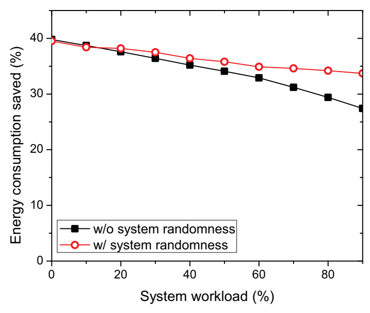
based on the Firefox application which has heterogeneous runtime occupation of CPU when browsing different webpages. The variations of system workloads are simulated by running articulated dumb calculations with different levels of complexity at background. As demonstrated by Figure 9, the effectiveness of workload offloading on reducing the smartphones' local resource consumption generally decreases with higher system workload, because the smartphone applications used for our experiments have to compete for the system resources with other concurrent system processes. Nevertheless, our proposed framework is still able to minimize such degradation of offloading effectiveness in such cases. As shown in Figure 9(a), when the system workload increases from 10% to 90%, the increase of the average method execution time in our scheme is less than 500ms, while such increase in AlfredO and FuzzyLogic can be up to 1000ms. Similar difference is shown in Figure 9(b), where our scheme can still save over 30% of the energy consumption when the system workload is higher than 70%.

C. Different Modeling of Application Dynamics

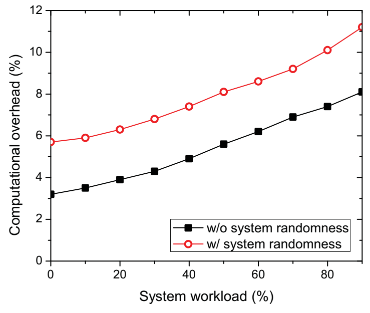
Section V describes various methods to analytically formulate the system dynamics during application executions. Such formulations are then used to enable quantified operations of workload offloading as specified by Eq. (7). It is therefore interesting to evaluate the impact of these methods on the effectiveness of workload offloading, which is highly dependent on the actual characteristics of system dynamics.

We evaluate such impacts using the Barcode Scanner application and an order-3 semi-Markov model. We first evaluate different methods estimating the parameters of sojourn time distribution, including the Maximum Likelihood (ML) and Maximum A Posteriori (MAP) methods. As shown in Figure 10(a), both methods are able to significantly reduce the energy consumption of local smartphones. When the system workload is lower than 30%, the ML method works better, but its effectiveness of workload offloading decreases with higher system workload. On the other hand, Figure 10(b) shows that the ML method incurs higher computational overhead. Especially when the system workload is higher than 50%, the iterative computations needed by the ML method leads to over 40% overhead increase. Such increase hence validates our choice of the MAP method in Section V-A.

In Sections V-B and V-C, we proposed to better depict the randomness of energy consumption characteristics of method execution by considering each polynomial coefficient in Eqs. (5) and (6) as a random variable. The impact of this approach is shown in Figure 11. When the system workload is lower than 50%, considering such system randomness only has negligible improvement on energy saving, but consumes extra 30% computational overhead as illustrated in Figure 11(b). Only when the system workload is higher than 70%, there



(a) Amount of energy saved



(b) Computational overhead

Fig. 11. Impact of considering system execution randomness on workload offloading

is noticeable improvement on the effectiveness of workload offloading, as shown in Figure 11(a), due to the increasing level of system dynamics and uncertainty. Therefore, we advise to only incorporate such probabilistic formulation of system randomness in computational-intensive scenarios.

D. The Order of Semi-Markov Model

In this section, we investigate the impact of different orders (k) of the semi-Markov model, which is the key of our proposed framework to formulating the invocation relationship among application methods, on workload offloading. We maintain the system workload as 0% for such evaluation. The results are shown in Figure 12. When the value of k is small, the semi-Markov model is generally incapable of precisely characterizing the impacts of previous application methods on subsequent method executions, and hence the effectiveness of workload offloading is reduced. This reduction is particularly significant when $k < 3$. As shown in Figures 12(a) and 12(b), when k is reduced from 3 to 1, the average method execution time increases by over 50%, and the amount of energy saved decreases by 30%. In contrast, when k is large, further increase of k will not improve the effectiveness of workload offloading, but unnecessarily incur higher computational overhead for maintaining the Markovian state space and transition probability matrix. The computational complexity of making offloading decisions also increases. As shown in Figure 12(c), when k increases from 3 to 5, the computational overhead increases by over 300%, only leading to less than 10% further improvement of offloading effectiveness.

To summarize, increasing k helps better characterize the dynamic execution patterns of user applications, but may incur additional or even unnecessary system overhead. From Figure 12, we generally conclude that $k = 3$ is the best choice for the smartphone applications we used in our evaluations.

VII. Discussions

A. Timing Constraints

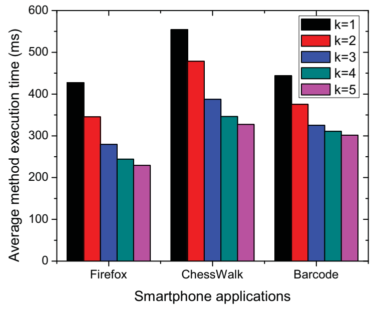
Timing constraints are found in many user applications such as navigation and gaming, where the execution of an application method must be completed within a given deadline. Being different from existing schemes which explicitly consider these constraints [5], [16], we implicitly satisfy these constraints by incorporating the sojourn time distributions of Markovian states into the decisions of workload offloading. In Eq. (3), the offloading effectiveness is only evaluated during the sojourn time of the current Markovian state. A timing constraint can be simply applied by replacing the upper integral limit in Eq. (3).

B. Temporal Dependency of Offloading Decisions

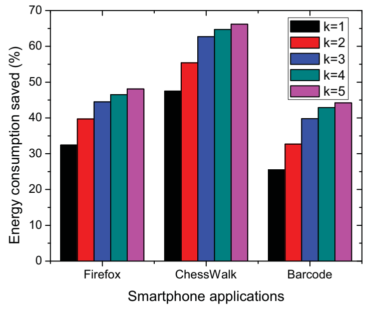
The offloading decision of an application method is affected by such decisions on previous methods. Existing schemes consider such temporal dependency at the global scale of application execution based on static execution patterns [9], [5], but the computational complexity of formulating such dependency with dynamic execution patterns would increase exponentially with the scale of application methods. Our proposed framework, as shown in Figure 6, adaptively balances between the offloading effectiveness and computational overhead by constructing a Local Invocation Graph (LIG). In practice, such temporal dependency could be easily considered in larger scales by expanding the LIGs and incorporating more methods in the subsequent application execution path.

C. Mutual Impacts between Different Applications

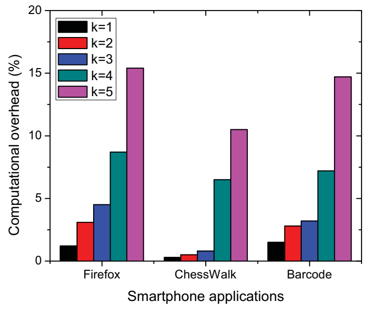
A modern mobile device is able to simultaneously run multiple user applications. The offloading decisions and operations of different smartphone applications may affect each other when these applications are running simultaneously, especially in cases of resource sharing or inter-process communication. Although the consideration of such mutual impact is out of the scope of this paper, the offloading framework proposed in this paper could be extended for the co-existence of multiple applications. For example, the difference of run-time execution patterns among user applications could be quantified by the Kullback-Leibler (KL) distance measure between the corresponding semi-Markov models. The smaller such difference is, the more likely that applications will compete for the limited system resources at the same time, and the larger impact they will have on the offloading operations of each other. The co-existence of multiple user applications, on the other hand, may also affect the interdependence of application method



(a) Method execution time



(b) Amount of energy saved



(c) Computational overhead

Fig. 12. Impact of the order of semi-Markov model on the effectiveness of workload offloading

invocations due to the mutual disturbance of the memory access patterns among multiple applications. We refer the studies of such mutual impacts among multiple applications to our future work.

VIII. Conclusion

In this paper, we investigate and exploit the dynamic run-time execution patterns of MCC applications for workload offloading. Our basic idea is to formulate these patterns using an order-k semi-Markov model, and characterize the method transitions based on the sojourn time distributions of Markovian states. Based on experimental investigations over various smartphone applications, we probabilistically evaluate the effectiveness of offloading operations, and ensure that the offloading decisions reflect various aspects of system dynamics. The effectiveness of our proposed scheme is evaluated by extensive experiments over realistic smartphone applications.

References

[1] H. Buchholz, H. Lichtblau, and K. Wetzel. *The confluent hypergeometric function: with special emphasis on its applications*. Springer-Verlag Berlin, 1969.

[2] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the 6th Conference on Computer Systems*, pages 301–314, 2011.

[3] B.-G. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM MCS Workshop*, pages 7–14, 2010.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of USENIX NSDI*, pages 273–286, 2005.

[5] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of ACM MobiSys*, pages 49–62, 2010.

[6] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 2011.

[7] H. R. Flores Macario and S. Srirama. Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning. In *Proceeding of the 4th ACM MCS workshop*, 2013.

[8] J.-L. Gauvain and C.-H. Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *IEEE Transactions on Speech and Audio Processing*, 2(2):291–298, 1994.

[9] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso. Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In *Proceedings of the 10th International Middleware Conference*. 2009.

[10] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen. Comet: Code offload by migrating execution transparently. In *Proceedings of USENIX OSDI*, pages 93–106, 2012.

[11] F. Hao, M. Kodialam, T. V. Lakshman, and S. Mukherjee. Online allocation of virtual machines in a distributed cloud. In *Proceedings of IEEE INFOCOM*, 2014.

[12] G. Huerta-Canepa and D. Lee. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM MCS Workshop*, 2010.

[13] E. Jung, Y. Wang, I. Prilepov, F. Maker, X. Liu, and V. Akella. User-profile-driven collaborative bandwidth sharing on mobile phones. In *Proceedings of the 1st ACM MCS Workshop*, 2010.

[14] A. Kansal and F. Zhao. Fine-grained energy profiling for power-aware application design. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):26–31, 2008.

[15] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: A computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*, pages 59–79. Springer, 2012.

[16] 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 *Proceedings of INFOCOM*, 2012.

[17] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *IEEE Computer*, 43(4):51–56, 2010.

[18] C. Lanczos. A precision approximation of the gamma function. *Journal of the Society for Industrial & Applied Mathematics*, 1(1):86–96, 1964.

[19] F. Mehmeti and T. Spyropoulos. Is it worth to be patient? analysis and optimization of delayed mobile data offloading. In *Proceedings of IEEE INFOCOM*, 2014.

[20] K. E. Muller. Computing the confluent hypergeometric function, m (a, b, x). *Numerical Mathematics*, 90(1):179–196, 2001.

[21] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan. Odessa: Enabling interactive perception applications on mobile devices. In *Proceedings of MobiSys*, pages 43–56, 2011.

[22] J. S. Rellermeyer, O. Riva, and G. Alonso. Alfredo: An architecture for flexible interaction with electronic devices. In *Proceedings of the 10th International Middleware Conference*, pages 22–41, 2008.

[23] P. Rong and M. Pedram. Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach. In *Proceedings of DAC*, pages 906–911, 2003.

[24] M. Satyanarayanan. Mobile computing: The next decade. *ACM SIGMOBILE Mobile Computing and Communications Review*, 15(2):2–10, 2011.

[25] G. A. F. Seber and A. J. Lee. *Linear Regression Analysis*, volume 936. John Wiley & Sons, 2012.

[26] W.-T. Tsai, X. Sun, and J. Balasooriya. Service-oriented cloud computing architecture. In *Proceedings of the 7th International Conference on Information Technology*, pages 684–689, 2010.

[27] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li. Ready, set, go: Coalesced offloading from mobile devices to the cloud. In *Proceedings of IEEE INFOCOM*, 2014.

[28] L. Xu and M. Jordan. On convergence properties of the em algorithm for gaussian mixtures. *Neural Computation*, 8(1):129–151, 1996.

[29] J. Zhang, F. Ren, and C. Lin. Delay guaranteed live migration of virtual machines. In *Proceedings of IEEE INFOCOM*, 2014.

[30] B. Zhao, Q. Zheng, G. Cao, and A. S. Energy-aware web browsing in 3g based smartphones. In *Proceedings of ICDCS*, 2013.