An Energy-efficient Offloading Framework with Predictable Temporal Correctness*

Zheng Dong
Department of Computer Science
University of Texas at Dallas
zheng@utdallas.edu

Yuchuan Liu, Zhou Husheng
Department of Computer Science
University of Texas at Dallas
Liu.yuchuan@utdallas.edu

Xusheng Xiao
Department of Electrical Engineering & Computer Science
Case Western Reserve University
xusheng.xiao@case.edu

Yu Gu
Watson Health Cloud
IBM Watson Health
yugu@us.ibm.com

Lingming Zhang
Department of Computer Science
University of Texas at Dallas
lingming.zhang@utdallas.edu

Cong Liu
Department of Computer Science
University of Texas at Dallas
cong@utdallas.edu

ABSTRACT

As battery-powered embedded devices have limited computational capacity, computation offloading becomes a promising solution that selectively migrates computations to powerful remote severs. The driving problem that motivates this work is to leverage remote resources to facilitate the development of mobile augmented reality (AR) systems. Due to the (soft) timing predictability requirements of many AR-based computations (e.g., object recognition tasks require bounded response times), it is challenging to develop an offloading framework that jointly optimizes the two (somewhat conflicting) goals of achieving timing predictability and energy efficiency.

This paper presents a comprehensive offloading and resource management framework for embedded systems, which aims to ensure predictable response time performance while minimizing energy consumption. We develop two offloading algorithms within the framework, which decide the task components that shall be offloaded so that both goals can be achieved simultaneously. We have fully implemented our framework on an Android smartphone platform. An in-depth evaluation using representative Android applications and benchmarks demonstrates that our proposed offloading framework dominates existing approaches in term of timing predictability (e.g., ours can support workloads with 100% more required CPU utilization), while effectively reducing energy consumption.

*Work supported by NSF grant CNS 1527727.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SEC '17, San Jose / Silicon Valley, CA, USA

© 2017 ACM. 978-1-4503-5087-7/17/10. . . $15.00

DOI: 10.1145/3132211.3134448

CCS CONCEPTS

•Computer systems organization →Embedded software;

KEYWORDS

Edge Computing, Offloading, Embedded devices, Energy-efficient

ACM Reference format:
Zheng Dong, Yuchuan Liu, Zhou Husheng, Xusheng Xiao, Yu Gu, Lingming Zhang, and Cong Liu. 2017. An Energy-efficient Offloading Framework with Predictable Temporal Correctness. In *Proceedings of SEC '17, San Jose / Silicon Valley, CA, USA, October 12–14, 2017,* 12 pages.
DOI: 10.1145/3132211.3134448

1 INTRODUCTION

1.1 Motivation

Recent developments in embedded systems hardware (e.g., smartphones) have empowered human experience through pervasive computing [1, 7, 11, 20]. While embedded systems hardware is becoming increasingly powerful [5, 6], it still falls short when faced with users' growing desire for running more resource-demanding applications, such as rich media applications, 3D modeling, and image processing. To bridge this gap, one solution receiving much recent attention is computation offloading. The fundamental idea is to leverage powerful remote resources by offloading complex computations to the remote site.

The driving problem that motivates this work is to leverage remote resources to facilitate the development of new simultaneous localization and mapping (SLAM) algorithms in a mobile augmented reality (AR) system. SLAM algorithms are used to correct the system's error accumulation or drift in a real-time fashion. To ensure functional correctness of the operations, it is critical to ensure bounded response times of SLAM-incurred computations along with other workloads in the system. Handling this situation in real-time requires a great amount of time, computation power, and energy. Using a battery-powered embedded device alone to process these

computations is infeasible, due to its limited computational power and stringent energy constraints.

To accelerate computation and avoid battery drain of the embedded processor, our approach is to establish an energy-efficient offloading framework for selectively offloading computations to remote resources (e.g., dedicated servers[1]) while ensuring timing predictability (i.e., ensuring applications to have provably bounded response times[2]). Note that although we focus on using SLAM-based mobile AR system as a motivation, our designed framework is applicable to all relevant system settings where achieving both timing predictability and energy efficiency is necessary.

Although the offloading idea has received much recent attention [12, 13, 23, 33, 34], the existing solutions cannot be applied herein because they separately consider the two performance aspects (i.e., either energy or timing predictability). Although the existing energy-aware offloading methods [3, 19, 21] yield reduced runtime energy consumption, they cannot guarantee timing predictability. Also, most of such solutions focus on single-application scenarios, thus failing to handle the more common multi-application scenarios. On the other hand, a couple of recent works seek to establish real-time offloading frameworks for embedded systems. Unfortunately, such frameworks do not consider energy consumption when making offloading decisions, thus may achieve timing predictability at the cost of significantly sacrificing energy consumption, which is clearly unacceptable for many mobile embedded platforms. Furthermore, existing real-time offloading research mainly focuses on algorithmic design and analysis issues, while largely ignoring implementation and evaluation matters (e.g., only simulations are used for assessment). However, such efforts are needed to assess the efficacy of any proposed real-time offloading methods in practice.

In summary, to enable the offloading idea to be implemented in many embedded systems such as mobile AR systems, an offloading framework that jointly considers timing predictability and energy issues is needed.

1.2 Contribution

This paper presents an offloading framework for embedded systems, which aims to ensure timing predictability (i.e., provably bounded response times) while minimizing energy consumption. Our design of the framework is motivated by the following insights.

First, guaranteeing predictable response time performance requires real-time resource allocation methods. There exists a rich set of soft real-time resource allocation methods [4, 24] that can guarantee bounded response time performance for CPU-intensive workloads. In order to leverage such methods, a notable challenge is to handle the offloading-induced delay in the timing analysis, i.e., the communication delay and remote execution delay. For example, a common SLAM-based task is to recurrently capture and analyze consecutive image frames, where the image analysis component of this task is often computationally intensive. Fig. 1 illustrates a typical execution flow of accelerating this SLAM-based task via offloading, where images are captured by the camera sensor at a certain rate (in this example, one frame per 100ms for a 10 frame-per-second (FPS)), then compressed (efficiently done locally leveraging specialized embedded graphics engines) and analyzed (more efficiently done on remote powerful resources). An important observation herein is that the SLAM-based task is suspended by the operating system when its components are offloaded to the remote site.

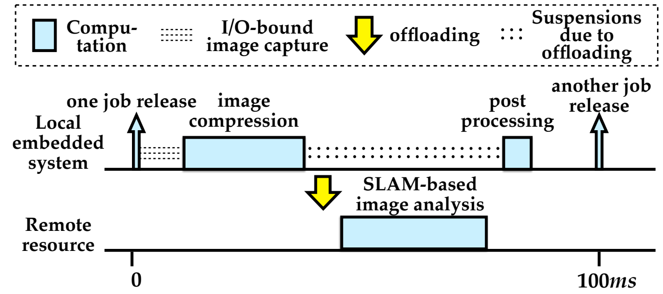


Figure 1: Execution flow of offloading a SLAM-based task

To deal with offloading-induced delay, our key idea is thus to naturally view such offloading-induced delays as suspensions. Therefore, we can model tasks running in an embedded system with components being offloaded as CPU tasks with suspensions, and then leverage existing suspension-based timing analysis techniques [24].

Moreover, another critical issue in designing this framework is: which task components should be offloaded so that the (somewhat conflicting) goals of guaranteeing bounded response time performance and minimizing energy consumption can be achieved simultaneously? Offloading decisions that seek to minimize energy consumption may violate timing predictability; on the other hand, in order to guarantee timing predictability, offloading decisions may significantly sacrifice energy performance, e.g., offloading task components that would consume significantly more energy due to communicating to the remote server. Thus, our offloading algorithm design shall consider both performance metrics as "first-class citizens" and resolve the inherent conflict.

Following these insights, this paper makes the following contributions:

- With the key idea of treating offloading-induced delay as suspensions occurring at the local embedded system side, we present a framework that allows the offloading system to be modeled as a classical real-time suspending task system model, thus further allowing us to leverage two categories of existing timing analysis techniques that can validate whether

[1] We intend to consider remote resources that we can directly access and control, e.g., network-connected computer servers with stable wireless connectivity that are dedicated for executing offloaded computations.
[2] We consider in this paper soft real-time constraints of ensuring bounded application response times, instead of hard real-time constraints where hard deadlines must be met, due to the existence of networking latency.

bounded response times can be ensured for all tasks in the system.

- Based on these two categories of suspension-based timing analysis techniques, we correspondingly propose two new energy-efficient offloading methods that guarantee bounded response time performance: a suspension-oblivious approach and a suspension-aware approach. Under the suspension-oblivious approach, we present a 0-1 ILP that yields a set of offloading decisions that minimize energy consumption while guaranteeing timing predictability; while under the suspension-aware approach, we design an energy-efficient offloading algorithm with low runtime complexity, which guarantees timing predictability while incorporating several optimization steps that effectively reduce energy consumption.
- We have fully implemented our offloading and resource allocation framework on the Android platform. Implementation details are given in Sec. 5. Based on this implementation, an in-depth evaluation is conducted using an Android SLAM-based image recognition application, an AES encryption application that represents common data-intensive workloads, and a set of benchmarks that represent interfering background workloads. Experimental results demonstrate the efficacy of our proposed methods. Specifically, our suspension-aware offloading method dominates existing approaches in term of timing predictability (e.g., can support workloads with 100% more required CPU utilization), while being able to effectively reduce energy consumption.

2 BACKGROUND

In this section, we first describe the overall architecture and a formal representation of the offloading system, and then describe the adopted energy measurement methodologies.

As illustrated in Fig. 2, there are several key components in an offloading system, i.e., a real-time scheduler at the embedded system side that allocates local resources, energy and network monitors that monitor necessary system parameters at runtime, an offloading decision maker that makes offloading decisions based on the monitored parameters, and an offloading engine that communicates with the remote server.

A formal offloading system model. Under our offloading system model, an embedded system contains $m \geq 1$ identical processors and needs to process a set of $n \geq 1$ tasks, $\tau = \{\tau_1, ..., \tau_n\}$. Each task is assumed to be composed of multiple phases: local-only phases and offloadable phases. A local-only phase contains computations that can only be executed locally, such as computations using the resources available only on the phone like GUI (Graphic User Interface), displaying or the acceleration sensor; while an offloadable phase can be either performed locally or offloaded. Our phase classification is inspired by recent work [36] in automatic identifying offloadable components. The identified offloadable components correspond to offloadable phases, while un-offloadable components corresponds to local-only phases. To obtain the phases of a task, we may leverage the existing approaches that identify offloadable components of the program via combining static analysis and cost estimation of network and CPU usage [3, 36]. Note that if a phase is selected to be offloaded, it may incur additional overheads due to offloading, e.g. a phase's associated data may need to be encrypted before transmission due to security concerns.

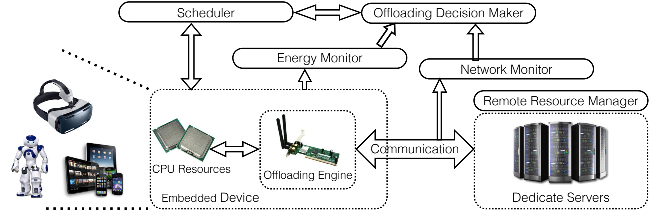


Figure 2: The offloading framework.

Table 1: System Notation

Parameter	Meaning
C_i^L	total execution time of all local-only phases
C_i^O	total execution time of all offloadable phases when they are executed locally
S_i^O	offloading-induced delay that contains the communication time (i.e. time to transmit offloadable components to/from the remote server) and the execution time on the remote server
C_i^{OH}	total offloading-induced overheads if τ_i's offloadable phases are offloaded
O_i	offloading decision: $O_i = 1$ implies τ_i is selected to be offloaded; otherwise $O_i = 0$

Thus, a task can be mathematically modeled as

$$\tau_i = \{C_i^L, C_i^O, S_i^O, C_i^{OH}, O_i, T_i\}.$$

Table 1 gives the explanation of each parameter. Note that we assume S_i^O is a measurable parameter since we choose to use dedicated remote servers with reliable WiFi links. The current technology ensures reliable WiFi links for most scenarios, e.g., Google is launching the project Fi to provide mobile WiFi service directly to users as an alternative to the traditional cellular networks [14].

The last parameter T_i of a task illustrates the recurring nature of the workload, often known as the task period. Many embedded systems commonly perform recurring operations (e.g., periodic sensing). The SLAM-based computation shown in Fig. 1 is also recurrent, where an image is captured and analyzed for every 100ms. Since we focus on analyzing recurrent workloads, we choose to study the sporadic task model [24], which is a widely studied general model of recurrent workloads. Under this model, each task τ_i releases a succession of *jobs* $J_{i,1}$, $J_{i,2}$, ... and is defined by specifying a *period* T_i. Successive jobs of τ_i are released at least T_i time units apart.

Each job $\tau_{i,j}$ released by any task has a *response time* that defines the duration from its release time denoted $r_{i,j}$ to its finish time. A task τ_i's response time equals the maximum job response time among all of its released jobs. The timing predictability considered in this paper is to guarantee tightly bounded response times for all tasks in the system.

Another parameter of a sporadic task is important, which is the task utilization. The utilization of any task τ_i is defined to be $U_i = \frac{C_i^L + C_i^O \cdot O_i}{T_i}$, which represents the fraction of the processor capacity required by τ_i with a specific offloading decision O_i.

The total utilization of the task system τ is defined to be $U_{sum} = \sum_{i=1}^{n} U_i$ when given a set of offloading decisions for all tasks in τ.

Energy model and measurement methodologies. Recent advances in energy measurement techniques for smartphones make it feasible to integrate an energy model into the offloading system model. PowerBooter [35] leverages built-in battery voltage sensors and knowledge of battery discharge behaviour to monitor power consumption, providing an energy model at the hardware component level. Using the model generated by the PowerBooter, PowerTutor provides online energy estimation for various components (e.g. CPU, WIFI, and LCD). Besides energy estimation for components, Li et al. [17, 22] develop more accurate energy measurement techniques that calculate source line level energy consumption by combining hardware-based power measurements with program analysis and statistical modeling.

In this work, we adapt the energy measurement techniques of components as it does not rely on external measurement devices and its measurement granularity meets our needs in defining the energy model. We next present the definition of our energy model.

As a task may use one or more components in a phase, its energy consumption rate is different during different phases. Thus, we introduce a parameter to model each component's energy consumption rate and represent the energy consumption rate of a phase using components' energy consumption rates. In this work, our model focuses on the energy consumption of the WIFI and CPU components as these are the core components that are used by the CPU-intensive workloads considered in this paper. This focus is also due to the fact that our offloading framework aims to reduce energy consumption of the computations occurred in local embedded systems and such computations are either executed by CPUs or offloaded through WIFI. Thus, modeling the CPU and WIFI components can already present sufficient measurement inputs for our model. (Note that our model can be extended for considering other energy-consuming components.) In the energy model, we introduce P_O and P_L to represent the energy consumption rate when the WIFI component is running and the CPU component is computing. We use P_I to represent the energy consumption rate when both the WIFI component and the CPU component are idle.

To integrate these power parameters into the offloading system model and yield more precise energy estimation, we further split the offloadable phase of a task into three consecutive sub-phases: *uploading*, *offload computing*, and *downloading* when the offloading decision is to offload. In the uploading and downloading sub-phases, the WIFI component is running and thus P_O is used to represent its energy consumption; while in the offloading computing sub-phase, both the WIFI component and the CPU component are idle and P_I is used to represent its energy consumption. Accordingly, we use S_i^T and S_i^S to represent the time for data transmission and remote computation respectively (i.e. S_i^T is the sum of the time of all the uploading and downloading sub-phases, and S_i^S is the sum of the time for all the offload computing phases.). Note that $S_i^O = S_i^T + S_i^S$.

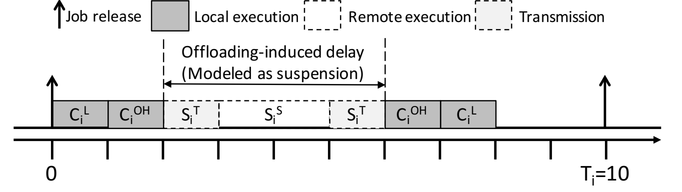


Figure 3: An example illustrating the offloading task model

By incorporating the energy-related parameters, the representation of a task is as follows.

$$\tau_i = \{C_i^L, C_i^O, S_i^T, S_i^S, C_i^{OH}, O_i, T_i\}$$

Fig. 3 shows the phases for an example task with $O_i = 1$, one time unit for uploading, two time units for remote execution, and another time unit for downloading.

According to this model, with a specific offloading decision O_i, the energy consumed by the task τ_i can be calculated using:

$$C_i^L \cdot P_L + (1 - O_i) \cdot C_i^O \cdot P_L + O_i \cdot (S_i^T \cdot P_O + S_i^S \cdot P_I + C_i^{OH} \cdot P_L). \quad (1)$$

In the above equation, $C_i^L \cdot P_L$ represents the energy consumed to execute local-only phases, $(1 - O_i) \cdot C_i^O \cdot P_L$ represents the energy consumed to execute the offloadable phase if being executed locally (it equals 0 if $O_i = 1$), and $O_i \cdot (S_i^T \cdot P_O + S_i^S \cdot P_I + C_i^{OH} \cdot P_L)$ represents the energy consumed to execute the offloadable phase if being offloaded (it equals 0 if $O_i = 0$).

3 TREATING OFFLOADING-INDUCED DELAY AS SUSPENSIONS

In this section, we introduce our modeling approach of treating an offloading task system as a suspending task system, where the key idea is to view a task's offloading-induced delay as suspensions that task experiences locally. We then explain the two existing categories of real-time analysis approaches that are originally designed to validate whether a given suspending task system can achieve bounded response times under a specific scheduling policy.

The suspending task model. To illustrate our modeling approach, we first present the suspending task model. Under the classical suspending task model [8, 26], each task releases jobs recurrently in a sporadic manner. Each job consists of interleaved computation and suspension phases. Each job of task τ_i executes at most C_i time unit across all its execution phases and suspends for at most S_i time unit across all its suspension phases. There are no restrictions on how these phases interleave within a job.

Clearly, each task in an offloading system can be modeled as a suspending task where offloading-induced delay is modeled as suspension. Note that if a task τ_i is not selected to be offloaded, then it is a suspending task with zero suspension length. Thus, for any task τ_i in the offloading system, after modeling it as a suspending task, it will have an execution time of $C_i^L + C_i^O \cdot (1 - O_i) + C_i^{OH} \cdot O_i$ time units and a suspension length of $(S_i^T + S_i^S) \cdot O_i$ time units. As illustrated in Fig. 3, by treating task τ_i's offloading-induced delay as suspensions (assuming $O_i = 1$), the offloading task can be considered as a suspending task with an execution time of four time units, and a suspension length of four time units.

This modeling step allows us to leverage existing real-time analysis techniques to analyze temporal correctness of the original offloading task system, as to be discussed next.

Existing work in the real-time systems community has established two categories of analysis techniques for validating whether a given suspending task system can achieve provably bounded response times under the well-known scheduling policy earliest-deadline-first (EDF) [10, 26]: suspension-oblivious analysis [4] and suspension-aware analysis [24]. In our offloading system, we will also use EDF as the local scheduler, where jobs with the earliest deadlines have the highest priorities and will be scheduled first. In our offloading system model, we may not have job deadlines typically pre-defined by application designers. Thus, for any job $\tau_{i,j}$, we use $r_{i,j} + T_i$ as its deadline value (i.e., a job's release time plus the corresponding task's period).

We now explain these two analysis techniques in detail, which are important because they allow us to verify whether an offloading task system can be guaranteed bounded response times when an offloading decision set is given. In Sec. 4, we will present two offloading algorithms that are designed based on these two analysis techniques. Note that these techniques are designed for analyzing the suspending task system we introduced earlier.

The suspension-oblivious timing analysis technique. Under the suspension-oblivious approach, the suspensions of a task are simply integrated into the per-task worst-case execution time requirements. Note that a task does not occupy CPU during its suspension intervals. However, under this approach, by converting suspensions into computation, the CPU resource is considered to be occupied during a task's suspension intervals. This eases the analysis difficulty, but the resulted timing analysis is thus more pessimistic. Hence, for our offloading task systems, the offloading-induced delay of an offloading task is treated as suspensions first, and then integrated into the execution time parameter under this approach. The following theorem (proved in [4]) gives a closed-form formula that validates whether bounded response times can be guaranteed for a suspending task system (thus an offloading task system) with a given set of offloading decisions (i.e., the O_i values).

Theorem 3.1. *[4] An offloading task system with a given set of offloading decisions can have bounded response times scheduled under EDF if the following condition holds:*

$$\sum_{i=1}^{n} \frac{C_i^L + C_i^O \cdot (1 - O_i) + (S_i^T + S_i^S + C_i^{OH}) \cdot O_i}{T_i} \leq m. \quad (2)$$

As we could observe from Eq. 2, the only variables in this inequality are the set of offloading decisions $\{O_1, \dots, O_n\}$.

The suspension-aware timing analysis technique. The suspension-aware approach [24] is less pessimistic than the suspension-oblivious approach, mainly because this approach explicitly consider suspensions in the analysis. That is, the suspension-aware approach assumes that CPU resources can be used by other tasks during a task's suspension periods. This increases the difficulty in proving the analysis but yields a better solution. The following theorem (proved in [24]) gives a closed-form formula that validates whether bounded response times can be guaranteed for a suspending task system (thus an offloading task system) with a given set of offloading decisions (i.e., the O_i values).

Definition 3.2. Let V_τ^i denotes the ith maximum suspension ratio of $\frac{(S_i^T + S_i^S) \times O_i}{T_i}$ in τ (with a given set of O_i). Let $W(\tau)^m$ denotes the sum of m maximum suspension ratios among tasks in τ. i.e. $W(\tau)^m = \sum_{i=1}^{m} V_\tau^i$.

Theorem 3.3. *[24] An offloading task system with a given set of offloading decisions can have bounded response times scheduled under EDF if the following condition holds:*

$$\sum_{i=1}^{n} \frac{C_i^L + C_i^O \cdot (1 - O_i) + C_i^{OH} \cdot O_i}{T_i} + W(\tau)^m \leq m \quad (3)$$

Similarly, as we could observe from Eq. 3, the only variables in this inequality are the set of offloading decisions $\{O_1, \dots, O_n\}$.

In the following section, we present two new offloading decision making algorithms that identify offloading decisions satisfying Eq. 2 and Eq. 3 (i.e., ensuring timing predictability), respectively, while minimizing energy consumption.

4 OFFLOADING ALGORITHMS

In this section, we present two energy-aware offloading decision making algorithms, which are designed based on the suspension-oblivious and suspension-aware timing analysis techniques (i.e., Theorem 3.1 and Theorem 3.3), respectively.

4.1 Suspension-Oblivious Energy-aware Offloading

Problem statement. We have an offloading task system with n tasks that are scheduled under the EDF scheduler in an embedded system with m processors. The objective is to design an offloading algorithm that satisfies Theorem 3.1 while minimizing energy consumption.

An 0-1 ILP solution. A good offloading algorithm would identify an offloading decision set among all feasible sets (i.e., those satisfying Eq. 2) that yields the minimum energy consumption. As we can observe from Eq. 2, intuitively, if the sum of all tasks' ratio of execution time (after treating offloading-induced suspensions as computation) over the period is at most m, the task system will have bounded response times. Therefore, various combinations of offloading decisions exist for satisfying this equation, which may yield significantly different energy consumption. Since Eq. 2 is linear with $\{O_1, ..., O_n\}$ as the only 0 or 1 variables, we are able to formulate this problem as an 0-1 integer linear programming (ILP) and identify an optimal solution under the suspension-oblivious timing analysis approach, as shown below.

$$\min \sum_{i=1}^{n} \frac{\left(P_O \cdot S_i^T + P_L \cdot C_i^{OH} + P_I \cdot S_i^S - P_L \cdot C_i^O\right) \cdot O_i}{T_i}$$

$$\text{s.t.} \sum_{i=1}^{n} \frac{C_i^L + C_i^O \cdot (1 - O_i) + \left(C_i^{OH} + S_i^S + S_i^T\right) \cdot O_i}{T_i} \leq m$$

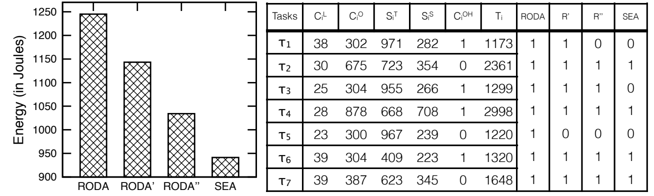
The above ILP formulation has n variables O_i, which represent the offloading decisions for n tasks. All other parameters are constants as introduced in the offloading system model. The objective is to minimize the summation of all n tasks' energy consumption per time unit, where the numerator in this term represents the energy consumption incurred by a task given a specific offloading decision (given by Eq. 1 in Sec. 2). Note that we choose to minimize the ratio of energy consumption divided by the corresponding task period, instead of the energy amount itself due to the recurrent nature of our considered workloads. The first constraint guarantees the condition of guaranteeing bounded response times under the suspension-oblivious approach (i.e., Eq. (2). Thus, a solution given by solving this ILP formulation guarantees bounded response times under the suspension-oblivious analysis technique while yielding minimum energy consumption.

4.2 Suspension-aware Offloading with Energy Optimization

Problem statement. Similarly, we have an offloading task system with n tasks that are scheduled under the EDF scheduler in an embedded system with m processors. The objective is to design an offloading algorithm that satisfies Theorem 3.3 while minimizing energy consumption.

Motivation. Under the suspension-aware approach, it may not be possible to formulate an ILP problem because Eq. 3 in Theorem 3.3 is too complex for such a formulation, which contains a term representing the m largest suspension ratios out of a set of n items. Thus, we instead develop an energy-aware offloading algorithm that seeks to reduce energy consumption while satisfying Eq. 3 in Theorem 3.3.

To develop such an algorithm, we leverage an algorithm named "RODA" presented in an existing work addressing the soft real-time offloading problem [27], which is proved to yield an offloading decision set that guarantees bounded



Tasks	C_i^L	C_i^O	S_i^T	S_i^S	C_i^{OH}	T_i	RODA	R'	R''	SEA
τ_1	38	302	971	282	1	1173	1	1	0	0
τ_2	30	675	723	354	0	2361	1	1	1	1
τ_3	25	304	955	266	1	1299	1	1	1	0
τ_4	28	878	668	708	1	2998	1	1	1	1
τ_5	23	300	967	239	0	1220	1	0	0	0
τ_6	39	304	409	223	1	1320	1	1	1	1
τ_7	39	387	623	345	0	1648	1	1	1	1

Figure 4: Energy consumption under different offloading decisions.

response times under the suspension-aware analysis technique (i.e., satisfying Theorem 3.3). Unfortunately, RODA does not consider energy consumption at all in the design. By reviewing RODA's design, our observation is that it may yield an offloading decision set that significantly sacrifices energy.

To illustrate this point, we performed a measurement-based case study, which run a set of benchmarks on an Android smartphone using the offloading decision set given by RODA and two "twisted" decision sets by changing certain individual task offloading decisions based on RODA's solution (the experimental setup for this case study is the same as our evaluation setup, as discussed in Sec. 6 in detail). The parameters of this case study task system and offloading decision set given by RODA are also shown in the table inside Fig. 4. Accordingly, RODA's overall energy consumption per time unit is 1245.18J.

Fig. 4 shows another two decision sets denoted by RODA' and RODA", which change certain individual task offloading decisions as shown in the table within Fig. 4. From the energy figure, we observe that these two twisted offloading decision sets yield lower energy consumption compared to the original RODA. By fitting these two decision sets into Eq. 3, the equation still holds, which implies that both these two twisted decision sets can guarantee bounded response times. This case study shows that as RODA does not consider energy performance into design, it may yield rather pessimistic energy performance. Motivated by this observation, we intend to smartly change individual task offloading decisions to reduce energy consumption while not violating the timing correctness condition.

The SEA algorithm. We now present a new offloading algorithm, namely SEA (a Suspension- and Energy-Aware offloading algorithm), which guarantees bounded response times while effectively reducing energy consumption. The idea is to smartly reduce energy consumption by changing individual task offloading decisions yielded by RODA. To ensure low runtime complexity (as checking the combinations of these offloading decisions is clearly exponential), our intuition is to first check decisions that have the smallest impact on Eq. 3. By observing Eq. 3, we know that a task with $O_i = 1$ and a large $\frac{S_i^T + S_i^S}{T_i}$ ratio tends to have the smallest impact on Eq. 3 if its offloading decision gets changed. Based on this observation, we order tasks and check each task in turn to determine whether changing a task offloading

decision would result in a reduced energy consumption while still satisfying Eq. 3. Doing so ensures that SEA will never violate guarantees timing predictability.

Algorithm 1 SEA

1: $\tau = \text{RODA}(\tau)$;
2: **if** τ is empty **then**
3: Mark $predictability = false$; **return**;
4: **end if**
5: Split τ into two subsets τ_{local} and $\tau_{offload}$. If $O_i = 0$, then put τ_i into τ_{local}; Otherwise, put it into $\tau_{offload}$;
6: Sort tasks in $\tau_{offload}$ by largest ratio of $\frac{S_i^S+S_i^T}{T_i}$ first;
7: **for** each task τ_i in $\tau_{offload}$, starting from $i = 1$ **do**
8: **if** changing O_i from 1 to 0 satisfies Eq. 3 and $P_O \times S_i^T + P_L \times C_i^{OH} + P_I \times S_i^S - P_L \times C_i^O \geq 0$ holds **then**
9: $O_i = 0$;
10: **end if**
11: **end for**

The pseudocodes of SEA is shown in Algorithm 1. It first obtains an offloading decision set by running the RODA algorithm (lines 1-3). Then it partitions task set based on their offloading decisions to $\tau_{offload}$ and τ_{local} (line 4). Then SEA focuses on checking tasks in $\tau_{offload}$. This is because by analyzing the RODA algorithm, we find out an interesting observation that changing the offloading decision of any task in τ_{local} would quite negatively impact the ability to satisfy Eq. 3. SEA then sorts tasks by largest ratio of $\frac{S_i^S+S_i^T}{T_i}$ first (line 5). This is because by observing Eq. 3, we find out that changing offloading decisions of tasks with larger ratios of $\frac{S_i^S+S_i^T}{T_i}$ yield relatively less negative impact in the ability of satisfying Eq. 3. Then SEA seeks to change each individual task offloading decision in order (lines 6-8). If changing a task's offloading decision reduces energy while still satisfying Eq. 3 (line 7), SEA changes this task's offloading decision. The SEA algorithm clearly has a runtime complexity of $O(nlogn)$.

Fig. 4 also shows the energy consumption for running the same case study task system under SEA. As seen in the figure, SEA is able to produce an offloading decision set that satisfies Theorem 3.3 while yielding a significantly reduced energy consumption of 941.46J (compared to the 1245.18J if using RODA).

5 IMPLEMENTATION

For proof of concept, we implement an offloading system in android application layer to demonstrate the effectiveness of our framework. In this section, we first introduce the overview of the system and then present the detailed design of each component of the system.

5.1 System Overview

The offloading system has a similar architecture as we presented in Fig. 2 and the workflow is shown in Fig. 5. In our system, we assume that there is a stable dedicated server that can effectively handle offloading requests and our design focuses on the offloading client built in the Android phone. Our offloading client is designed to simulate the task scheduling and CPU execution of the operating system. Also, we implement the network communication functionality, which is used in the offloadable phase to support communications with the server. The first version of our offloading client has approximately 7,000 lines of code. In this implementation, we focus on the client side and assume that the remote site already has the source code needed to perform the offloaded computations.

5.2 Offloading Client

Our offloading client consists of 4 major components: *scheduler*, *CPU simulator*, *offloading decision maker*, and *offloading engine*.

The offloading client initializes the scheduler first and accepts incoming tasks as input. The scheduler is the core part of the offloading client, which consists of two major steps: making offloading decisions and scheduling tasks. When the scheduler receives a set of tasks, it invokes the offloading decision maker to make offloading decisions for each task. Based on the offloading decisions, the scheduler uses EDF scheduling policy to assign tasks to the CPU simulator for execution. Once a task finishes, the scheduler is notified and continues to reschedule next set of tasks.

The CPU simulator's major mission is to simulate how an CPU executes a task. As no two tasks can be executed simultaneously at a CPU, our system spawns a worker thread of the Android system to execute each task and no two phases of the task are scheduled to execute in two threads at the same time. Such mechanism simulates the real world scenario of executing a task in only a CPU for a given moment.

To allow the scheduler to reschedule the tasks, the CPU simulator is also responsible for notifying the scheduler about its execution phases. As we described in Section 2, a task consists of multiple offloadable execution phases and local-only execution phases. Between these phases, we instrument the task to introduce rescheduling points, which are the execution points that allow the scheduler to suspend the running task and reschedule other tasks to execute if necessary. We also introduce rescheduling points before the uploading sub-phase in an offloadable phase, since the following uploading sub-phase doesn't use CPU resources and can be suspended. The goal of introducing these rescheduling points is to ensure that our system can preempt the running task, which is a requirement for the EDF scheduling policy, and maximize the real-time schedulability.

The offloading decision maker receives a set of tasks and provide offloading decisions for each of the task based on the offloading decision algorithms, such as the suspension-aware and suspension-oblivious offloading decision algorithms proposed in this paper.

The offloading engine is responsible for the communication between the offloading client and the server. When a task needs to be offloaded, the offloading engine encapsulates the

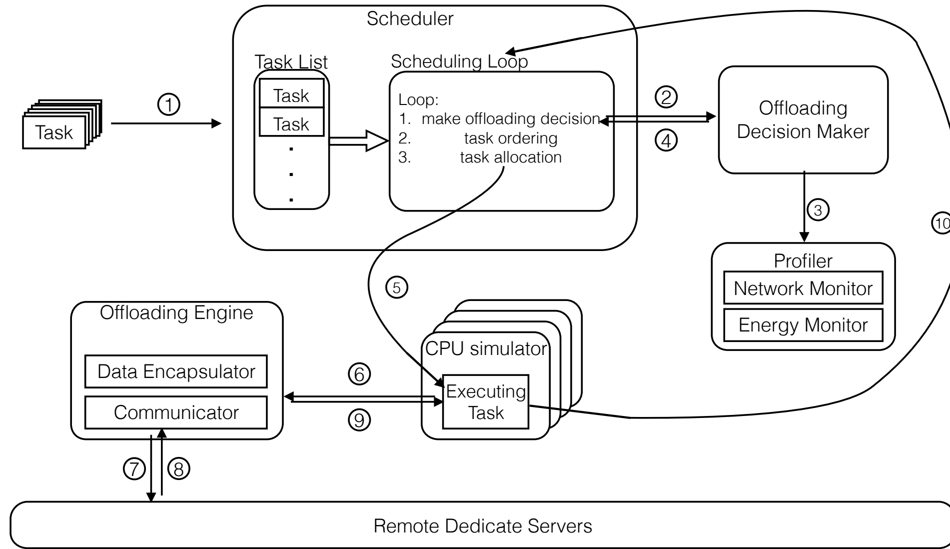


Figure 5: Workflow of our offloading framework. ① scheduler receives a set of tasks; ② offloading decision maker is invoked for making offloading decisions; ③ offloading decision maker requests network and energy measurement values; ④ scheduler receives offloading decisions; ⑤ scheduler schedules tasks to run in CPUs; ⑥ data of some tasks is sent to offloading engine; ⑦ data is encapsulated and transferred to remote server; ⑧ computed data is returned to offloading engine; ⑨ offloaded tasks resume execution; ⑩ after a task is done, CPU notifies scheduler and scheduler continues to process more tasks (back to ②).

offloading data with the necessary metadata to indicate which offloading service to use. It then sends the encapsulated data of the task to the server and get an identifier for the retrieval of the computed data. This identifier is later used to retrieve the results when the server finishes the computation. These network communications are implemented using network sockets.

Within affordable efforts, our implementation in the Android application layer aims to accurately simulate the behaviours of the operation system with additional features required for offloading. In our evaluations, we carefully choose a proper number of background applications to minimize the background interferences for yielding more accurate results.

6 EXPERIMENTAL EVALUATION

6.1 Experimental Setup

To evaluate the performance of the proposed two offloading algorithms, we performed an in-depth evaluation. Next we introduce the hardware platform, power profiling of various device components, evaluation scenarios, and the compared offloading approaches.

6.1.1 Evaluation Platform. We used an Android device and a workstation to conduct the experiments. More specifically, the Android device is a Nexus 5 mobile phone with Snapdragon 800 2.26 GHz Quad-Core and 2 Gigabyte RAM, running Android 4.4.3. The work station is a desktop with Intel Core i7-4770 3.40 GHz Quad-Core and 16 Gigabyte RAM, running Ubuntu 14.04.

Table 2: Application profiling

Applications (in milliseconds)	C_i^L	C_i^O	S_i^O	S_i^S	C_i^{OH}
Image recognition	354	2018	20	1058	1
AES encryption	39	1719	456	468	0
Matrix addition	3	878	546	486	0
Matrix multiplication	9	334	812	754	0
Vector addition	1	642	333	227	0
Vector dot product	1	114	746	94	0

6.1.2 Power Profiling. We used PowerTutor[35] to estimate the energy consumption of the Android device. More specifically, we measured the power of the CPU (1.15 Watt) and WIFI (0.66 Watt) components of the Android device.

6.1.3 Evaluation Scenarios. We now introduce the representative offloadable apps deployed in our system. Generally speaking, there are two types of offloadable apps (i.e. computation-intensive and data-intensive). The computation-intensive apps tend to have longer execution time of offload computing than transmission time. The data-intensive apps, on the contrary, have longer transmission time than the time of offload computing. Off-the-shelf automated app partitioning technique [36] is applied manually to partition each App into local-only and offloadable phases. In total, we use the following apps in our evaluation.

The SLAM-based Image Recognition App. Since a motivational problem is to support SLAM-based computations

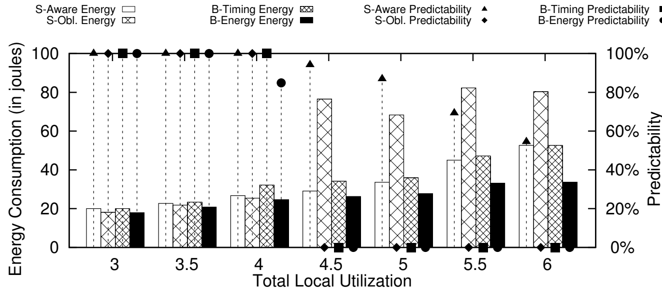


Figure 6: Results on predictability and energy consumption for the SLAM-based image recognition app-based experiments.

in a mobile AR system, we implement a SLAM-based image recognition app using the OpenCV library [28]. We observe that the image recognition algorithm consumes lots of computation resources (i.e. a computation-intensive type). Thus we transform the image recognition app and put the computation-intensive recognition algorithm part in the offloadable phase. Capturing images and displaying recognized object outline are put into local-only phases.

The AES Encryption App. The AES encryption App performs data encryption algorithms. It is of data-intensive type since it processes large amount of data while using small amount of time. Generating large sets of data and saving data to file are put into local-only phases, and the encryption algorithm is put into offloadable phases.

Other Benchmarks. We also implements several common matrix- and vector-based computation benchmarks [27], including matrix addition, matrix multiplication, vector addition, and vector dot product. These benchmark tasks are added into experiments as background interference for the image recognition app or the AES encryption app, which are used to simulate the background apps on embedded devices.

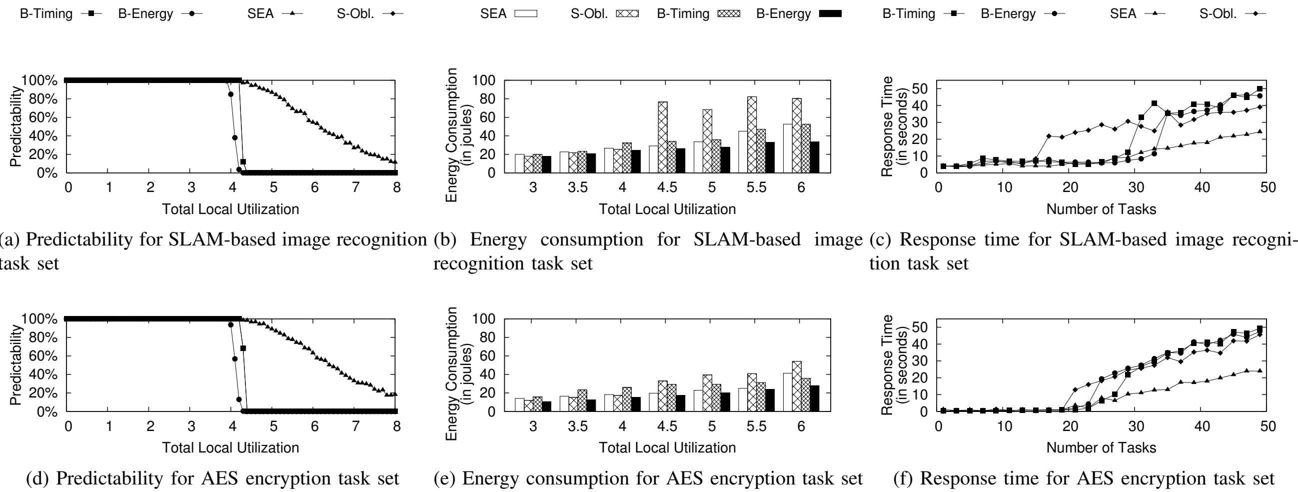
We integrated image recognition, AES encryption, and the other benchmarks into our offloading system. To make runtime offloading decisions, we profiled the execution of each application used in the experiment. Part of our profiled data is shown in Table 2. To simulate real-world scenarios, we set up two categories of experiment task sets. For first category, the AES encryption task is the main task while all the other apps are treated as background interferences. Similarly, the second category has the SLAM-based image recognition task as the main task and all the other apps as background interferences. For each experiment, we generate 1,000 task sets according to this setup. Each task set is generated according to a total local utilization threshold (i.e., total task utilization if all tasks are executed locally), which varies from 0 to 8.

6.1.4 Studied Approaches. In order to evaluate the performance of our two proposed approaches (denoted by "SEA" and "S-Obl", respectively), we also implemented the widely-used best-timing [20] (denoted by "B-Timing") and best-energy [21] (denoted by "B-Energy") approaches for comparison. Under the best-timing approach, the offloading decision is made on a per-task basis. For any task, if its remote execution time is shorter than local execution time, then it is chosen for offloading; otherwise, the task will execute locally. Similarly, under the best-energy approach, for any task, if the local power consumed when a task is offloaded is smaller than that when the task is locally executed, it is chosen for offloading; otherwise, the task will execute locally. Since the B-Energy and B-Timing methods do not consider timing predictability, we apply Theorem 3.1 to assess their performance in terms of timing predictability in all experiments.

We would like to point out that simply combining the B-Energy or B-Timing approach with a timing analysis technique cannot resolve the problem studied in this paper. This is because such heuristic approaches do not consider timing predictability in the design, thus failing to make offloading decisions that are inherently in favor of guaranteeing timing predictability.

6.2 Experimental Results

Key question: whether the (somewhat conflicting) goals of guaranteeing predictable response times and minimizing energy consumption are achieved? In Fig. 6, we combine the predictability (i.e., the percentage of the generated task set that can ensure bounded response times) and energy consumption for the image recognition task set in one graph. The most intuitive observation is that our proposed offloading algorithms, especially the SEA algorithm, are able to yield relatively minimized energy consumption while maximizing the predictability of task sets. For example, when the total task utilization is larger than 4.5, the energy consumption under SEA is slightly larger than the one under B-Energy (and smaller than B-Timing), while the predictability under SEA is significantly better (e.g., SEA achieves a predictability of 96% and 87% when total task utilization equals 4.5 and 5, respectively; while the B-Energy and B-Timing methods fail to support any such workload, i.e., 0% predictability in both cases). Thus, SEA is able to achieve significantly better predictability at the cost of consuming slightly more energy. Note that the energy consumed under S-Obl dramatically increases when task utilization exceeds 4.5. This is due to the pessimism of the suspension-oblivious analysis, where the ILP formulation fails to yield offloading decision sets that could satisfy Eq. 2. In this case, we simply choose to execute all tasks locally in the experiments, which results in a surge of energy consumption. However, when the total local utilization is smaller than 4.5, it actually yields even lower energy consumption than SEA while ensuring 100% predictability. This is because when the ILP formulation under S-Obl is able to give solutions, it yields a decision set among all feasible set which minimizes energy consumption.



(a) Predictability for SLAM-based image recognition task set
(b) Energy consumption for SLAM-based image recognition task set
(c) Response time for SLAM-based image recognition task set
(d) Predictability for AES encryption task set
(e) Energy consumption for AES encryption task set
(f) Response time for AES encryption task set

Figure 7: Results. In the first (respectively, second) row of graphs, SLAM-based image recognition-based (respectively, AES encryption-based) experiments are performed. In the first (respectively, second and third) column of graphs, the predictability (respectively, energy and runtime response time) performance is evaluated.

We next describe the detailed result analysis from three individual aspects: predictability, energy consumption, and runtime response time.

6.2.1 Predictability. The obtained predictability results are shown in Fig. 7 (the organization of which is explained in the figures caption). We have the following observations. (i) SEA yields better predictability than other approaches due to a less pessimistic timing analysis technique. For example, Fig. 7(a) shows that when the total local utilization exceeds 3.9, the predictability under all other three approaches drop dramatically, and becomes 0 after the total utilization exceeds 4.5. In contrast, SEA is able to maintain good predictability performance even when the total utilization reaches 8. Thus, SEA is able to support workloads with 100% more required CPU utilization compared to the other approaches. (ii) B-Timing and S-Obl approaches yield the same predictability performance. The reason is because under B-Timing, a task is offloaded only if the offloading-induced delay is smaller than the local execution time for the offloadable phases. Thus, B-Timing actually yields the smallest possible value of the left-hand side of Eq. 2 among all possible offloading decisions. Thus, if B-Timing cannot find a feasible solution, then S-Obl also cannot find one because there does not exist such a solution. Or, if B-Timing yields a feasible offloading decision set that satisfies Eq. 2, then S-Obl can also find a feasible solution due to the ILP formulation. Vice versa, the argument clearly holds as well.

6.2.2 Energy consumption. Different from experimental setup from assessing predictability, we vary the total local utilization from 3 to 6 run each generated task set for 1 minute. Each task set is executed and we record their breakdown execution time and obtain the energy consumption based on energy profiling. The energy consumption results are shown in Fig. 7(b) and Fig. 7(e). We have the following observations. The B-Energy approach has the lowest energy consumption among all four approaches. This is expected since the B-Energy approach makes offloading decisions according to per-task energy consumption. When the total local utilization is at most 4, S-Obl, SEA and B-Energy yield almost the same energy consumption while B-Timing performs worse. When the total local utilization exceeds 4.5, SEA yields a slightly higher energy consumption than B-Energy. The reason is because SEA has to consider predictability at the front when making offloading decisions. However, B-Energy does not need to consider predictability at all, thus being able to make more greedy decisions to minimize energy consumption.

6.2.3 Runtime response time. Besides predictability and energy consumption, we also measure the runtime response time performance in each experiment under four methods when the number of tasks generated in each task set is varied from 1 to 50. The response time of a job is defined to be from its release time till its completion. The response time of a task is the maximum job response time among all its jobs. Results are shown in Fig. 7(c) and Fig. 7(f). Each dot in the figure represents the average response time for all tasks. As seen in the figure, SEA yields the best response time performance in almost all scenarios, particularly when the number tasks is large. Moreover, B-Timing, B-Energy, and S-obl have an obvious response time increase when the number of tasks reaches a certain value. This is because that S-Obl may fail to give decisions when number of task gets higher and our experiments assume all local executions in this case; while B-Timing and B-Energy make decisions on

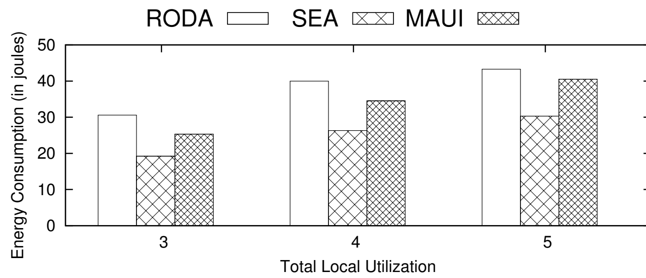


Figure 8: Energy Comparison among SEA, RODA, and MAUI.

a per-task basis, which cannot optimize the global response time performance of the system. On contrary, since SEA inherently considers timing predictability into the design, it guarantees to yield analytically bounded response times for all tasks in the system even when the number of tasks is large (due to its superior predictability performance as discussed earlier). Because of this fundamental advantage, SEA is shown to also yield a superior performance in terms of the observed runtime response times.

6.2.4 Energy Comparison among SEA, RODA, and MAUI. Since a focus of the SEA offloading algorithm is on extending RODA to effectively reduce energy consumption, we have conducted experiments comparing SEA against RODA, and MAUI [3] which is a well-known energy-oriented offloading algorithm that targets at the single-application scenario, in terms of energy consumption. As seen in Fig. 8, SEA yields the best energy performance among three algorithms under all utilization scenarios. SEA is able to reduce energy consumption by more than 30% and 25% compared to RODA and MAUI, respectively. Although SEA yields the same timing predictability as RODA, it effectively extends RODA by smartly changing offloading decisions to be more energy efficient without sacrificing timing predictability. Also, SEA is able to outperform MAUI because MAUI seeks to optimize energy consumption while meeting a specific response time bound which is pre-defined by MAUI (or the user). The solution space for making energy-efficient offloading decisions under MAUI is thus constrained.

7 RELATED WORK

Offloading in general-purpose computing systems. Computation offloading has been used widely to improve performance of software applications. Early works show promising results on desktop applications [29, 30], and recent works have extended the successes to mobile platforms for better energy efficiency and runtime response time performance [3]. Some of these works require developers to manually specify which class to be offloaded, either by source-level annotations or predefined programming patterns. More specifically, MAUI [3] and CloneCloud [2] aim at optimizing both of the energy and response time performance of mobile applications, and offload computations at the granularity of method level. COMET [16] optimizes the energy and response time performance by spawning multiple threads to offload computations to multiple servers. Instead of offloading part of the computations to the server, Tango [15] runs applications on both mobile devices and the server at the same time and sends the output to users when the output is ready from either the mobile devices or the server. Unlike these approaches that seek to optimize response time performance on a "best-effort" basis, our approach provides an offloading framework with guaranteed timing predictability. Another line of recent work [37] studies how to automatically identify offloadable methods using static analysis. These research efforts focus on identifying the offloadable components of the software applications and preparing them for offloading, while our framework proposes techniques to make offloading decisions with timing predictability and energy minimization, complementing these research efforts.

Real-time offloading. Several recent works [9, 18, 25, 27, 31, 32, 38] investigate the problem of real-time offloading in real-time embedded systems, e.g., RODA. However, these works do not consider energy consumption and may achieve timing predictability at the cost of significant energy lost. Moreover, the existing works on developing real-time offloading system frameworks are mostly theory-oriented, i.e., they focus on proving whether the proposed offloading algorithms can guarantee real-time correctness but ignoring matters in terms of real implementation and evaluation using real-world applications.

8 CONCLUSION AND FUTURE WORK

This paper establishes an energy-aware offloading framework with predictable temporal correctness. Two offloading algorithms using two different timing analysis techniques has been proposed, which could yield offloading decision sets that guarantee timing predictability while optimizing energy performance. An in-depth evaluation on top of an Android-based implementation demonstrates that our framework yields the best performance in terms of timing predictability while effectively reducing energy consumption.

There are several important research issues we plan to address in the near future. We have made several assumptions in this paper that allow us to elegantly model the offloading system behaviors. Releasing these assumption requires us to answer the following reserach questions: 1) How to deal with networking exceptions (i.e., the variable network speed and probabilistic transmission failures)? 2) How to derive a safe yet tight remote response time bounds for task components that are offloaded to the remote site? 3) How to analyze certain real-world workloads with SRT requirements that exihibit stochastic execution demands and arrival patterns instead of worst-case parameters? Our near future work aims at developing efficient SRT scheduling and offloading techniques with provably analytical properties that address such practical issues.

REFERENCES

[1] Darren Black, Nils Jakob Clemmensen, and Mikael B Skov. 2012. Pervasive Computing in the Supermarket: Designing a Context-Aware Shopping Trolley. In *Social and Organizational Impacts of Emerging Mobile Devices: Evaluating Use*. IGI Global, 172–185.

[2] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. 2011. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*. ACM, 301–314.

[3] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. 2010. MAUI: Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. 49–62. https://doi.org/10.1145/1814433.1814441

[4] U. Devi and J. Anderson. pages 330-341, 2005. Tardiness bounds under global EDF scheduling on a multiprocessor. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*.

[5] Zheng Dong, Yu Gu, Jiming Chen, Shaojie Tang, Tian He, and Cong Liu. 2016. Enabling Predictable Wireless Data Collection in Severe Energy Harvesting Environments. In *Real-Time Systems Symposium (RTSS), 2016 IEEE*. IEEE, 157–166.

[6] Zheng Dong, Yu Gu, Lingkun Fu, Jiming Chen, Tian He, and Cong Liu. 2017. ATHOME: Automatic Tunable Wireless Charging for Smart Home. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*. ACM, 133–143.

[7] Zheng Dong, Linghe Kong, Peng Cheng, Liang He, Yu Gu, Lu Fang, Ting Zhu, and Cong Liu. 2014. REPC: Reliable and efficient participatory computing for mobile devices. In *Sensing, Communication, and Networking (SECON), 2014 Eleventh Annual IEEE International Conference on*. IEEE, 257–265.

[8] Zheng Dong and Cong Liu. 2016. Closing the Loop for the Selective Conversion Approach: A Utilization-Based Test for Hard Real-Time Suspending Task Systems. In *Real-Time Systems Symposium (RTSS), 2016 IEEE*. IEEE, 339–350.

[9] Zheng Dong, Cong Liu, Lingkun Fu, Peng Cheng, Liang He, Yu Gu, Wei Gao, Chau Yuen, and Tian He. 2016. Energy synchronized task assignment in rechargeable sensor networks. In *Sensing, Communication, and Networking (SECON), 2016 13th Annual IEEE International Conference on*. IEEE, 1–9.

[10] Zheng Dong, Cong Liu, Alan Gatherer, Lee McFearin, Peter Yan, and James H Anderson. 2017. Optimal Dataflow Scheduling on a Heterogeneous Multiprocessor With Reduced Response Time Bounds. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[11] Zheng Dong, Banghui Lu, Liang He, Peng Cheng, Yu Gu, and Lu Fang. 2013. Exploring smartphone-based participatory computing to improve pervasive surveillance. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 69.

[12] L. Ferreira, G. Silva, and L. Pinho. 2011. Service offloading in adaptive real-time systems. In *Proceedings of the Conference on Emerging Technologies and Factory Automation*. 1–6.

[13] W. Gao, Y. Liu, H. Lu, T. Wang, and C. Liu. to appear, 2014. On Exploiting Dynamic Execution Patterns for Workload Offloading in Mobile Cloud Applications. In *Proceedings of the 22nd IEEE International Conference on Network Protocols*.

[14] google fi [n. d.]. Google Project Fi. https://fi.google.com. ([n. d.]).

[15] Mark S Gordon, David Ke Hong, Peter M Chen, Jason Flinn, Scott Mahlke, and Zhuoqing Morley Mao. 2015. Accelerating mobile applications through flip-flop replication. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 137–150.

[16] Mark S Gordon, Davoud Anoushe Jamshidi, Scott A Mahlke, Zhuoqing Morley Mao, and Xu Chen. 2012. COMET: Code Offload by Migrating Execution Transparently.. In *OSDI*, Vol. 12. 93–106.

[17] S. Hao, D. Li, W.G.J. Halfond, and R. Govindan. 2013. Estimating mobile application energy consumption using program analysis. In *Proceedings of the 35th International Conference on Software Engineering*. 92–101. https://doi.org/10.1109/ICSE.2013.6606555

[18] Liang He, Lipeng Gu, Linghe Kong, Yu Gu, Cong Liu, and Tian He. 2013. Exploring adaptive reconfiguration to optimize energy efficiency in large-scale battery systems. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*. IEEE, 118–127.

[19] Y. Hong, K. Kumar, and Y. Lu. 2009. Energy efficient content-based image retrieval for mobile systems. In *Proceedings of the IEEE International Symposium on Circuits and Systems*. IEEE, 1673–1676.

[20] K. Kumar, J. Liu, Y. Lu, and B. Bhargava. 2013. A survey of computation offloading for mobile systems. *Mobile Networks and Applications* 18, 1 (2013), 129–140.

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

[22] D. Li, S. Hao, W.G.J. Halfond, and R. Govindan. 2013. Calculating Source Line Level Energy Information for Android Applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. 78–89. https://doi.org/10.1145/2483760.2483780

[23] Z. Li, C. Wang, and R. Xu. 2001. Computation Offloading to Save Energy on Handheld Devices: A Partition Scheme. In *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. 238–246.

[24] C. Liu and J. Anderson. 2012. An O(m) analysis technique for supporting real-time self-suspending task systems. In *Proceedings of the 33th IEEE Real-Time Systems Symposium*. 373–382.

[25] Cong Liu, Jian Jia Chen, Liang He, and Yu Gu. 2014. Analysis techniques for supporting harmonic real-time tasks with suspensions. In *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*. IEEE, 201–210.

[26] J. Liu. 2000. *Real-Time Systems*. Prentice Hall.

[27] Y. Liu, C. Liu, X. Zhang, W. Gao, L. He, and Y. Gu. 2015. A Computation Offloading Framework for Soft Real-Time Embedded Systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems*.

[28] opencv [n. d.]. OpenCV. http://opencv.org. ([n. d.]).

[29] M. Philippsen and M. Zenger. 1997. JavaParty - Transparent Remote Objects in Java. *Concurrency - Practice and Experience* 9, 11 (1997), 1225–1242.

[30] E. Tilevich and Y. Smaragdakis. 2009. J-Orchestra: Enhancing Java programs with distribution capabilities. *ACM Transaction on Software Engineering and Methodology* 19, 1 (2009).

[31] A. Toma and J. Chen. 2013. Computation offloading for frame-based real-time tasks with resource reservation servers. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems*. 103–112.

[32] A. Toma and J. Chen. 2013. Computation offloading for real-time systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. 1650–1651.

[33] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi. 2008. Using bandwidth data to make computation offloading decisions. In *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing*. 1–8.

[34] Y. Ye, L. Xiao, I. Yen, and F. Bastani. 2011. Leveraging Service Clouds for Power and QoS Management for Mobile Devices. In *Proceedings of the 4th IEEE International Conference on Cloud Computing*. 235–242.

[35] L. Zhang, B. Tiwana, R.P. Dick, Z. Qian, Z.M. Mao, Z. Wang, and L. Yang. 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proccedings of the Hardware/Software Codesign and System Synthesis, 2010 IEEE/ACM/IFIP International Conference on*. 105–114.

[36] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang. 2012. Refactoring Android Java Code for On-demand Computation Offloading. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*. 233–248. https://doi.org/10.1145/2384616.2384634

[37] Ying Zhang, Gang Huang, Xuanzhe Liu, Wei Zhang, Hong Mei, and Shunxiang Yang. 2012. Refactoring android java code for on-demand computation offloading. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 233–248.

[38] Husheng Zhou and Cong Liu. 2014. Task mapping in heterogeneous embedded systems for fast completion time. In *Proceedings of the 14th International Conference on Embedded Software*. ACM, 22.