A Computation Offloading Framework for Soft Real-Time Embedded Systems

Yuchuan Liu[1], Cong Liu[1], Xia Zhang[1], Wei Gao[2], Liang He[3], and Yu Gu[4]
[1]The University of Texas at Dallas, USA
[2]The University of Tennessee at Knoxville, USA
[3]Singapore University of Technology and Design, Singapore
[4]IBM Research Austin Laboratory, USA

Abstract

Recent developments in embedded hardware have empowered human experiences through pervasive computing. While embedded systems are becoming more powerful, they still fall short when faced with users' growing desire for running more resource-demanding applications. To bridge this gap, one solution is to leverage powerful resources residing at remote sites by performing computation offloading. Unfortunately, the state-of-the-art offloading frameworks cannot be applied in many embedded systems supporting applications with soft real-time (SRT) constraints or high delay sensitivity, as they typically optimize response times on a "best-effort" basis using heuristics.

This paper establishes a soft real-time offloading framework that optimizes the resource utilization of the embedded system while analytically guaranteeing SRT schedulability. The key idea behind the proposed framework is to view offloading-induced delays as suspensions occurring at the local embedded system side, which allows a task being offloaded to be modelled as a suspending task and thus existing SRT suspension-aware scheduling and analysis techniques to be leveraged. Based on this idea, we propose an offloading algorithm, namely Real-time Offloading Decision-making Algorithm (RODA), to make offloading decisions such that SRT schedulability of the task system can be ensured. The optimality properties of RODA have been proved on both uniprocessors and multiprocessors. We conducted extensive simulations on evaluating schedulability and implemented a case study offloading system on top of real hardware to test runtime response time performance. Results demonstrated that RODA is superior to existing performance-driven offloading algorithms, particularly under heavy workloads.

1 Introduction

Recent developments in embedded hardware (e.g., smartphones, tablet PCs, sensors, etc.) have empowered human experience through pervasive computing. While embedded systems are becoming more powerful, they still fall short when faced with users' growing desire for running more resource-demanding applications in different domains, such as rich media applications, object recognition, and image processing. Many such applications are computationally intensive and have (soft) real-time constraints or high delay sensitivity. To bridge this gap, one solution receiving much attention is to utilize remote resources. The fundamental idea is to leverage powerful resources residing at remote sites by offloading computation-intensive components to the remote site. A good example is the emerging mobile cloud-computing paradigm, where applications running on mobile devices get offloaded to the cloud (e.g., Apple's SiRi application).

The state-of-the-art research on computation offloading, unfortunately, cannot be applied in many embedded systems that support applications with soft real-time (SRT) constraints (e.g., voice recognition and multimedia processing). Most existing research on this topic treats application response times as an optimization objective function and applies different offloading heuristics to optimize it on a "best-effort" basis. However, in order to support applications with real-time constraints, the offloading technique employed in embedded systems has to analytically guarantee timing correctness (i.e., meeting all response time requirements). This lack of support is a serious impediment limiting the applicability of offloading techniques in many real-time embedded systems.

In this paper, we establish a SRT offloading framework that offloads selected components of embedded applications to dedicated remote resources for execution.[1] One key challenge in ensuring SRT correctness in this context is appropriately handling the offloading-induced delay (i.e., communication delay and the remote execution time). We propose to *view such delays as suspension delays occurring at the local embedded system side,*[2] *which is different from all ex-*

[1]We intend to consider remote resources that we can directly access and control, e.g., network-connected computer servers with relatively stable connectivity that are dedicated for executing offloaded computations.

[2]In this context, whenever a task's components are offloaded, it is considered to be suspended in the embedded system.

1068-3070/15 $31.00 © 2015 IEEE
DOI 10.1109/ECRTS.2015.19

isting approaches and allows us to leverage existing SRT suspension-aware schedulability analysis techniques [9, 10]. Consequently, we can model tasks running in an embedded system that are selected to be offloaded as suspending tasks. Hence, by utilizing global earliest-deadline-first (GEDF) based SRT suspension-aware schedulability tests, we have a criteria to check whether a given task system is schedulable under GEDF given a specific set of offloading decisions. Clearly, a key problem in checking schedulability is to decide which tasks' components should be offloaded. If too many components are selected for offloading, then the resulting task suspensions might be too frequent to make the system schedulable. On the other hand, if too few components are selected for offloading, then the resulting task execution times might be too long, possibly causing the system to be unschedulable. To solve this problem, we propose an algorithm called Real-time Offloading Decision-making Algorithm (RODA) to make offloading decisions such that SRT schedulability of the task system can be ensured.

Related Work. Most prior work on computation offloading focuses on non-real-time contexts. Various offloading heuristics and run-time optimization techniques for improving runtime response times and energy performance have been proposed [5, 6, 8, 17, 18]. Unfortunately, these methods cannot guarantee real-time correctness, but instead improve response times on a "best-effort" basis. Furthermore, many works only focus on the single application offloading scenario [7, 16, 18]. Several recent works [11, 13–15] look at the problem of offloading computation in a hard real-time (HRT) system. However, they mainly focused on the resource allocation problem at the remote site and ignored how to manage the offloading-induced delay as well as how these delays affect the scheduling and offloading decisions in the local embedded system. Moreover, in many cases it might not be appropriate to offload HRT workloads due to unpredictable network connectivity.

Contributions. In this paper, we present an offloading framework for SRT embedded systems. The key idea is to view offloading-induced delay as suspension delays. It is thus possible to model any task that is selected for offloading purposes as a suspending task and leverage the existing SRT suspension-aware analysis technique [10] to validate schedulability. To make judicious offloading decisions, we propose an offloading algorithm RODA. We prove that if the embedded system contains a uniprocessor, then RODA can produce a feasible offloading decision set if there exists one that can pass the suspension-aware schedulability test given in [10]. Such optimality properties of RODA can also be proved for the multiprocessor case, but provided an additional condition expressed using task parameters holds. Moreover, RODA considers offloading-induced overheads possibly imposed by tasks in practice when making offloading decisions. Such overheads may have a non-trivial impact on making those de-

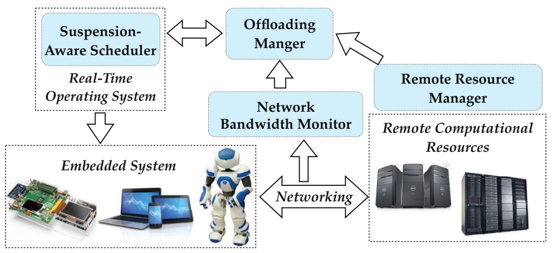

Suspension-Aware Scheduler
Real-Time Operating System
Offloading Manger
Remote Resource Manager
Network Bandwidth Monitor
Remote Computational Resources
Embedded System
Networking

Figure 1: SRT offloading framework.

cisions. To evaluate the proposed offloading framework, we have conducted extensive experiments, including both simulations using randomly generated task sets and a case study implementation using real-world workloads on top of real hardware. Results showed that compared to performance-driven offloading heuristics, RODA proved to be superior and can improve the performance by a large margin in many scenarios.

Organization. The rest of this paper is organized as follows. Sec. 2 defines the formal offloading system model. Sec. 3 describes the RODA algorithm. Sec. 4 presents the uniprocessor and multiprocessor analysis of the RODA algorithm. Sec. 5 shows the experimental results on both schedulability and runtime performance of a case study system. Sec. 6 concludes.

2 The Offloading System Model

In this section, we present a formal system model representing our proposed offloading framework. As illustrated in Fig. 1, our offloading framework consists of several key components: a SRT scheduler that schedules tasks in the local embedded system, an offloading manager that makes offloading decisions, a network bandwidth monitor that monitors the real-time network bandwidth, and a resource manager at the remote site for scheduling offloaded components. The objective of the offloading framework is to decide which task components should be offloaded while satisfying SRT schedulability. We assume a well-studied SRT notion that a SRT task is schedulable if its response time can be provably bounded [4, 9, 10].

Formal model. In our model, an embedded system with $m \geq 1$ processors is comprised of a collection of $n \geq 1$ SRT sporadic tasks, $\tau = \{\tau_1, ..., \tau_n\}$. Each task is assumed to be composed of three components: pre-offloading execution phase, offloadable execution phase, and post-offloading execution phase.[3] A task may also incur additional overheads due to offloading, e.g., if its component is offloaded, the associated data may need to be processed locally for transmission and the remote site needs to receive and process the data ac-

[3]A number of efficient partitioning techniques [1–3, 16] exist that can separate a large set of applications into components for offloading purposes. In partitioning a task, a general guideline is to partition tasks in a way so that computation-intensive components with reasonable data sizes can be identified as offloadable execution phase.

cordingly.

A pre-offloading execution phase is responsible for performing the computation before the task is offloaded. An offloadable execution phase is the one that can either be performed locally or offloaded to the remote site. A post-offloading execution phase is for processing the remaining computation of the corresponding job. A task can thus be represented as

$$\tau_i = \{C_i^1, C_i^2, S_i^2, C_i^3, T_i, O_i\}$$

where C_i^1 is the pre-offloading execution time, C_i^2 is the offloadable execution time if performed locally, S_i^2 is the offloading-induced delay if the offloaded execution phase is indeed offloaded to the remote site. Note that S_i^2 includes the response time of the component executed in the remote site and the data transmission time.[4] C_i^3 is the post-offloading execution time. T_i denotes the period of the sporadic task τ_i. O_i represents the offloading decision of the task. If task τ_i's offloadable component is offloaded, $O_i = 1$, otherwise $O_i = 0$. The utilization of any task τ_i is defined to be $U_i = \frac{C_i^1+C_i^2+C_i^3}{T_i}$, which represents the fraction of the processor capacity required by τ_i if executed locally. The total utilization of the task system τ is defined to be $U_{sum} = \sum_{i=1}^{n} U_i$.

A key step in this modelling process is to model the delays due to the offloading process as suspensions. Thus, if a task τ_i's offloadable component is selected for offloading, then we consider τ_i to suspend for S_i^2 time units at the local embedded system. That is, each job released by τ_i first executes for C_i^1 time units locally, then suspends for S_i^2 time units, and finally execute for C_i^3 time units locally. Fig. 2(a) shows a task τ_i with the following parameters $\{1, 4, 3, 2, 10, 0\}$, where its offloadable component is executed locally and consumes four time units. Fig. 2(b) shows the scenario where this task is offloaded (where $O_i = 1$). After the pre-offloading execution, related data is sent to the remote site and then the offloadable component is executed remotely. When remote computation is done, the results are sent back to the local embedded system and τ_i will resume executing its post-offloading execution phase. Note that in this case since $S_i^2 = 3$, τ_i suspends for three time units during the offloading process.

Overhead consideration. When offloading task components, additional overhead may be generated, which may have impact on making offloading decisions. Thus, it is necessary to take overheads into consideration. We use the case where task components have security requirements as an example scenario to illustrate such overheads. In practice, some

[4]In this paper, we assume that S_i^2 for each task is a known value and it is bounded. This is reasonable given sufficient resources at the remote site and reliable transmission links. Due to space constraints, we leave the research issues associated with obtaining a tight upper bound of the S_i^2 term and dealing with potential networking delay or packet loss as immediate future work.

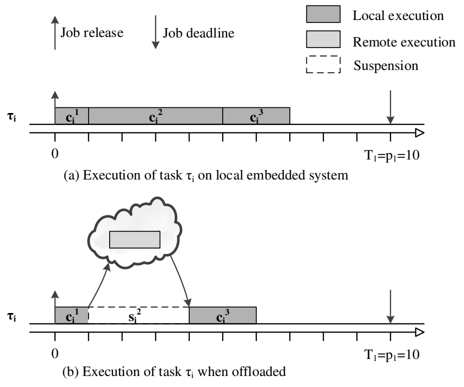


Figure 2: Example offloading task.

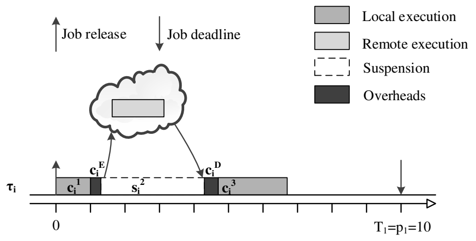


Figure 3: Example task with overhead consideration.

tasks may contain confidential information. It is thus necessary to implement encryption algorithms and secure transmission between the local embedded system and remote site. The encryption/decryption process is described as follow: (*i*) Encrypt data associated with the offloadable component in the local embedded system (client), (*ii*) Send the encrypted data to the remote site (server), (*iii*) Decrypt the data on the server, (*iv*) Execute the offloaded component on the server and encrypt the produced results on the server, (*v*) Send the encrypted results to the client, and decrypt the results on the client. The time used to encrypt the data locally and decrypt the data remotely can be viewed as additional overheads due to offloading.

There are also other types of overhead that could result in additional execution time during offloading. For example, before offloading image recognition applications, it is often more efficient to compress images to reduce the transmission latency, which results in additional overheads in the local embedded system (due to image compression). Let C_i^E and C_i^D denote the computational overhead incurred for task τ_i in the local embedded system before and after the offloading process, respectively, if τ_i's offloadable execution phase is offloaded. If a task τ_i is selected to be offloaded, then for readability, we factor the corresponding overheads occurred at the remote site into the S_i^2 parameter. From the above discussions, we could observe that if a task is selected to be offloaded, then additional overheads may result in longer execution times in both the local and the remote sites, as illustrated in Fig. 3. This may negate the benefits of applying the offloading strategy. Thus, we consider the overhead factor into our offloading algorithm design.

Since our key idea is to view offloading induced delay as suspensions, we now present the self-suspending task model and show how to model the offloading task system as a suspending task system.

The self-suspending task model. Under the self-suspending task model given by [10], each task releases jobs sporadically. Each job consists of interleaved computation and suspension phases. Each job of task τ_i executes at most e_i time unit across all its execution phases and suspends 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 our offloading system can be modeled as a self-suspending task. Specifically, each task has three phases in fixed order, where the second offloadable phase can be viewed as a suspension phase if it is selected to be offloaded (otherwise a computation phase)[5]. Note that if a task τ_i is not selected to be offloaded, then it can be modeled as a self-suspending task with a zero suspension length (i.e., $S_i^2 = 0$). We thus model the offloading task system as a self-suspending task system. This modeling step allows us to leverage existing suspension-aware schedulability analysis techniques to analyze schedulability of the original offloading task system, as discusses next.

Local scheduler. We consider tasks in the local embedded system to be scheduled by EDF (on a uniprocessor) and GEDF (on a multiprocessor). Thus, with a given set of offloading decisions, we obtain a task system containing both ordinary sporadic tasks and suspending tasks (i.e., those that will be offloaded). We choose EDF (or GEDF) as the local system scheduler because it is known to be good schedulers for scheduling SRT suspending task systems [10], as it yields an efficient schedulability test described next.

SRT suspension-aware schedulability test under GEDF. [10] gave a schedulability test with only $O(m)$ utilization loss, which dominates the other existing schedulability tests for SRT suspending task systems. Since our offloading task system can be modeled as a SRT self-suspending task system, we could apply the O(m) SRT suspension-aware schedulability test to our offloading task system[6].

Definition 1. Let V_τ^i denote the i^{th} maximum suspension ratio of $\frac{S_i^2 \times O_i}{T_i}$ in τ (under a given set of O_i). Let $W(\tau)^m$ denote the sum of m maximum suspension ratios among tasks in τ, i.e., $W(\tau)^m = \sum_{i=1}^{m} V_\tau^i$.

[5]The self-suspending task model is more general in the sense that it imposes no restriction on how phases interleave; whereas under the offloading task model, each task has three phases with a fixed interleaving pattern.

[6]The latency bound could be derived by Theorem 1 in [10]. We omit the derivation due to space constrains.

Theorem 1. *[10] The SRT task system τ with a given set of offloading decisions is schedulable under GEDF provided*

$$\sum_{j=1}^{n} \frac{C_j^1 + C_j^3 + C_j^2 \times (1 - O_j) + (C_j^E + C_j^D) \times O_j}{T_j} + W(\tau)^m \leq m. \tag{1}$$

Theorem 1 (specifically Eq. (1)) gives us a condition to check whether a given task system τ is SRT schedulable. As we can observe from Eq. (1), the only variables in this inequality are the set of offloading decisions $\{O_1, ..., O_n\}$. Thus, we next present an efficient offloading algorithm, RODA, to identify such an offloading decision set, under which τ may satisfy Eq. (1).

3 A Suspension-Aware Computation Offloading Algorithm

We now present algorithm RODA, which decides which tasks should be offloaded so that SRT schedulability can be ensured. RODA consists of two main phases. First, we put the set of tasks into two categories. The first category includes tasks with $C_i^E + C_i^D > C_i^2$. These tasks obviously should not be selected for offloading because the additional overhead incurred at the local side is already longer than the execution time if executed locally (not to mention the additional suspension time due to the offloading process). The second category includes tasks with $C_i^E + C_i^D \leq C_i^2$, which are the potential offloading candidates. In the second phase, we'll decide which tasks should be offloaded according to the schedulability condition Eq. (1).

Algorithm 1 shows the pseudocode of RODA. It first separates tasks into two sets τ_{local} and τ_{new}, as discussed earlier. For tasks put into τ_{local}, their offloading decisions are marked as local (lines 2-4). For the tasks put into τ_{new}, they are sorted by largest suspension ratio (i.e., $\frac{S_i^2}{T_i}$) first (line 9). Tasks in τ_{new} are then re-indexed as $\{\tau_1, ..., \tau_{n'}\}$. Then RODA determines the offloading decisions for tasks in τ_{new} (lines 10-20). For each task τ_i in τ_{new} in the new index order defined for τ_{new}, we check whether the inequality shown on line 11 holds. This inequality is a rearrangement of the Eq. (1) by assuming that tasks in $\{\tau_i, \tau_{i+1}, ..., \tau_{n'}\}$ in τ_{new} will be offloaded, while any other task in τ will be executed locally. If this inequality holds for any task τ_i as tested in turn (line 10), then this offloading decision set is a feasible one. Thus, we mark each task's offloading decision variable O_i accordingly (lines 12-18). That is, any task τ_k in $\{\tau_i, ..., \tau_{n'}\}$ in τ_{new} have $O_k = 1$. If such a $\tau_i \in \tau_{new}$ cannot be found, then RODA will return an offloading decision set where all tasks are executed locally (i.e., $O_i = 0$ for any task $\tau_i \in \tau$).

RODA clearly has a runtime complexity of $O(n^2)$.

Example. We now give an example illustrating the RODA algorithm. Suppose we have a task set running on a uniprocessor with parameters shown in Table 1. Through the first

Algorithm 1 RODA

```
1: for each task $\tau_i \in \tau$, starting from $i = 1$ do
2:     if $C_i^E + C_i^D > C_i^2$ then
3:         Mark $O_i = 0$;
4:         Put $\tau_i$ into a new task set $\tau_{local}$;
5:     else
6:         Put $\tau_i$ into another new task set $\tau_{new}$;
7:     end if
8: end for
9: Sort tasks in $\tau_{new}$ by largest ratio of $(\frac{S_i^2}{T_i})$ first, re-index
   them from $\tau_1$ to $\tau_{n'}$;
10: for each task $\tau_i$ in $\tau_{new}$, starting from $i = 1$ do
11:     if $\sum_{j=i}^{i+m-1} \frac{S_j^2}{T_j} \leq m - \sum_{\tau_j \in \tau_{new} \cup \tau_{local}} \frac{C_j^1 + C_j^3}{T_j} - \sum_{\tau_j \in \tau_{local}} \frac{C_j^2}{T_j} - \sum_{j=1}^{i-1} \frac{C_j^2}{T_j} - \sum_{j=i}^{n'} \frac{C_j^E + C_j^D}{T_j}$ then
12:         for each task $\tau_j$ to $\tau_{n'}$ in $\tau_{new}$, starting form $j = i$
            do
13:             Mark $O_j = 1$;
14:         end for
15:         Mark $schedulable = true$;
16:         Break;
17:     else
18:         Mark $O_i = 0$;
19:     end if
20: end for
```

for loop in RODA (lines 1-8), τ_1 is put into τ_{local} with $O_1 = 0$, while all other tasks are put into τ_{new}. After line 9 in RODA, tasks in τ_{new} are ordered and re-indexed accordingly. For tasks in τ_{new} (we now use the new task indexing defined for tasks in τ_{new}), we have $\tau_1 = \tau_3$ (i.e., τ_1 in τ_{new} represents τ_3 in the original set τ), $\tau_2 = \tau_4$, $\tau_3 = \tau_6$, $\tau_4 = \tau_2$, $\tau_5 = \tau_5$. In the second for loop (lines 10-20), RODA identifies that τ_1 does not satisfy the condition on line 11, but τ_2 satisfies the condition. This is because we have $\sum_{j=i}^{i+m-1} \frac{S_j^2}{T_j} = 0.125$ and $m - \sum_{\tau_j \in \tau_{new} \cup \tau_{local}} \frac{C_j^1 + C_j^3}{T_j} - \sum_{\tau_j \in \tau_{local}} \frac{C_j^2}{T_j} - \sum_{j=1}^{i-1} \frac{C_j^2}{T_j} - \sum_{j=i}^{n'} \frac{C_j^E + C_j^D}{T_j} = 0.167$ for this example. Thus, according to lines 12-14 of RODA, τ_1 is decided to execute locally and other tasks in τ_{new} are decided to be offloaded. According to the original task indexing, the offloading decision set returned by RODA for this example task system: $O_1 = 0$, $O_2 = 1$, $O_3 = 0$, $O_4 = 1$, $O_5 = 1$, $O_6 = 1$, which is feasible.

4 Analysis of RODA on Uniprocessors and Multiprocessors

In this section, we provide the analysis of RODA for both the uniprocessor and multiprocessor cases. We prove the relative optimality[7] of RODA. That is, if there exists a feasible

[7]The term"relative optimality" is reflected by the fact that Theorem 1 only gives a sufficient, but not exact schedulability test.

Table 1: Example task system.

	C_i^1	C_i^2	C_i^3	S_i^2	C_i^E	C_i^D	T_i
τ_1	1	1	0	7	2	1	12
τ_2	0	1	1	1	0	0	12
τ_3	1	1	0	6	0	0	12
τ_4	1	3	0	1	0	0	8
τ_5	1	2	0	1	1	0	12
τ_6	0	1	1	1	0	0	8

offloading decision set that makes Eq. (1) hold (thus the task system is schedulable), then RODA is guaranteed to provide a feasible offloading decision set.

4.1 Uniprocessor Analysis

We now prove the "relative optimality" of RODA if the embedded system contains a uniprocessor.

Theorem 2. *If there exists a feasible offloading decision set that makes the task system τ schedulable according to Theorem 1 and $m = 1$, then RODA is guaranteed to find one feasible offloading decision set.*

Proof. Given a task set τ, assume that there exists a feasible offloading decision set β for τ such that Theorem 1 holds.

The proof contains several steps. First, we separate τ into two subsets τ_{local} and τ_{new} and change the decision of every task τ_i in τ_{local} to $O_i = 0$, according to lines 1-8 of RODA. We then prove that the resulting offloading decision set β' is also a feasible one. Finally, we prove that RODA will be able to find a feasible offloading decision set as well.

According to lines 1-8 of RODA, tasks put into τ_{local} are those with $C_i^E + C_i^D > C_i^2$, while those put into τ_{new} are tasks with $C_i^E + C_i^D \leq C_i^2$. For any task $\tau_i \in \tau_{local}$, if $O_i = 1$, we change it to $O_i = 0$. The resulting offloading decision set β' clearly remains feasible because for any such task $\tau_i \in \tau_{local}$, $C_i^E + C_i^D > C_i^2$ holds, thus changing the value of O_i from 1 to 0 can only decrease the left-hand side of Eq. (1).

Now we have a set τ_{local} where all tasks are executed locally and another set τ_{new} where tasks may have offloading decisions that remain to be the same as defined in the original decision set β. If τ_{new} is empty, then RODA returns an offloading decision set of $O_i = 0$ for every task $\tau_i \in \tau$, which is clearly feasible because β' is feasible; thus we assume otherwise.

We now concentrate on analyzing tasks in τ_{new}. We first order tasks in τ_{new} by largest ratio of $\frac{S_i^2}{T_i}$ first, and re-index them as $\tau_{new} = \{\tau_1, ..., \tau_{n'}\}$, according to line 9 of RODA. Then we find the task τ_j with the smallest index in τ_{new} with $O_j = 1$. If we cannot find such a task in τ_{new}, which implies that all tasks in τ_{new} are also executed locally, then according to lines 10-20 of RODA, RODA will output a decision set where all $O_i = 0$, which is clearly feasible (again because this decision set would be identical to β').

We thus consider the case where such a $\tau_j \in \tau_{new}$ exists. (Note that when referring tasks in τ_{new}, we use the new task indexing as defined in line 9 of RODA.) If any task $\tau_k \in \tau_{new}$, where $k > j$, has an offloading decision of $O_k = 0$, we change it to $O_k = 1$. We now prove that the resulting offloading decision set β'' for τ remains to be a feasible set. For clarity, let's further separate τ_{new} into three subsets: (*i*) $\overline{\tau_l}$ represents the set of tasks $\tau_1, ..., \tau_{j-1}$ in τ_{new} (i.e., tasks with smaller indexes than our identified j), (*ii*) $\overline{\tau_c}$ represents the task set containing those tasks τ_i in τ_{new} whose offloading decisions are changed from $O_i = 0$ to $O_i = 1$, and (*iii*) $\overline{\tau_o}$ represents the rest of tasks in τ_{new}. We have $\overline{\tau_l} \cup \overline{\tau_c} \cup \overline{\tau_o} = \tau_{new}$. Notice that among these three task subsets, only tasks in $\overline{\tau_c}$ have their offloading decisions changed from the original values given in β. Also, tasks in $\overline{\tau_c} \cup \overline{\tau_o}$ will be offloaded (i.e., $O_i = 1$), while tasks in $\tau_{local} \cup \overline{\tau_l}$ will be executed locally (i.e., $O_i = 0$).

We now prove that β'' is a feasible offloading decision set. By plugging the O_i values in β'' and $m = 1$ into the left-hand side of Eq. (1), we have

$$\begin{aligned}
&\sum_{i=1}^{n} \frac{C_i^1 + C_i^3 + C_i^2 \times (1 - O_i) + (C_i^E + C_i^D) \times O_i}{T_i} + W(\tau)^1 \\
=&\sum_{\tau_i \in \tau} \frac{C_i^1 + C_i^3}{T_i} + \sum_{\tau_i \in \overline{\tau_l} \cup \tau_{local}} \frac{C_i^2}{T_i} + \sum_{\tau_i \in \overline{\tau_c} \cup \overline{\tau_o}} \frac{C_i^E + C_i^D}{T_i} + \frac{S_j^2}{T_j} \\
\leq&\sum_{\tau_i \in \tau} \frac{C_i^1 + C_i^3}{T_i} + \sum_{\tau_i \in \overline{\tau_l} \cup \overline{\tau_c} \cup \tau_{local}} \frac{C_i^2}{T_i} + \sum_{\tau_i \in \overline{\tau_o}} \frac{C_i^E + C_i^D}{T_i} + \frac{S_j^2}{T_j} \\
\leq& 1,
\end{aligned}$$

which satisfies Eq. (1). In the above equation, the first $=$ comes from the definitions of the task subsets $\overline{\tau_l}$, $\overline{\tau_c}$, and $\overline{\tau_o}$, and the fact that τ_j has the largest ratio of $\frac{S_j^2}{T_j}$ among all tasks with $O_i = 1$; the first $\leq$ comes from the fact that for any task $\tau_i \in \overline{\tau_c}$, $C_i^E + C_i^D \leq C_i^2$ holds; the second $\leq$ comes from the fact that the last inequality is an instance of Eq. (1) after plugging β' into Eq. (1) and we know that β' is a feasible offloading decision set.[8]

We have now proven that β'' is a feasible offloading decision set. According to the way β'' is defined, we know that RODA is able to at least find τ_j when checking each task in turn in τ_{new} (line 11 of RODA). This is because such a τ_j would make the condition on line 11 of RODA hold. Thus, RODA, in the worst case, can find τ_j. Or, RODA may identify another task $\tau_k \in \tau_{new}$ where $k < j$, in which case RODA is still guaranteed to produce a feasible offloading decision set. The theorem thus follows. □

4.2 Multiprocessor Analysis

Now we analyze the multiprocessor case. Unlike the uniprocessor case, we can prove the relative optimality of RODA only if an additional condition on task parameters is satisfied (as discussed later). Intuitively this is because when $m > 1$, it is more difficult to design an efficient algorithm that can minimize the left-hand side of Eq. (1). This is partly due to the definition of $W(\tau)^m$ (defined in Def. 1) appearing in Eq. (1), which makes this problem a combinatorial optimization problem. The following theorem shows that RODA is able to achieve relative optimality if $C_i^E + C_i^D + S_i^2 \leq C_i^2$ holds for every task $\tau_i \in \tau_{new}$ (note again that τ_{new} is obtained according to lines 1-8 of RODA).

Theorem 3. *For any given task system τ, if $C_i^E + C_i^D + S_i^2 \leq C_i^2$ holds for every task $\tau_i \in \tau_{new}$ and $m > 1$, then RODA is guaranteed to find one feasible offloading decision set if there exists such a feasible solution that passes Eq. (1).*

Proof. For a given task set τ, assume that there exists a feasible offloading decision set β for τ. The first part of the proof is similar to the uniprocessor case, where we seek to find a specific task $\tau_j \in \tau_{new}$ with the smallest index in τ_{new} with $O_j = 1$, and thus be able to define the same task subsets $\overline{\tau_l}$, $\overline{\tau_c}$, and $\overline{\tau_o}$ as defined in the uniprocessor case. For completeness, this part of the proof is given in a technical report [12].

After the first part of the proof, we obtained a modified offloading decision set β''. In the rest of this proof, we prove that β'' is a feasible decision set.

For any task $\tau_i \in \tau_{new}$, since $C_i^E + C_i^D + S_i^2 \leq C_i^2$, we have:

$$\begin{aligned}
&C_i^E + C_i^D + S_i^2 \leq C_i^2 \\
&\{add\ up\ every\ task\ in\ set\ \overline{\tau_c}\} \\
\Rightarrow &\sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^E + C_i^D}{T_i} + \sum_{\tau_i \in \overline{\tau_c}} \frac{S_i^2}{T_i} \leq \sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^2}{T_i} \\
&\{W(\overline{\tau_c})^m \leq \sum_{\tau_i \in \overline{\tau_c}} \frac{S_i^2}{T_i}\ by\ Def.\ 1\} \\
\Rightarrow &\sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^E + C_i^D}{T_i} + W(\overline{\tau_c})^m \leq \sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^2}{T_i} \\
&\{add\ W(\overline{\tau_o})^m\ to\ both\ side\ of\ inequation\} \\
\Rightarrow &\sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^E + C_i^D}{T_i} + W(\overline{\tau_c})^m + W(\overline{\tau_o})^m \leq \sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^2}{T_i} + W(\overline{\tau_o})^m \\
&\{W(\overline{\tau_c} \cup \overline{\tau_o})^m \leq W(\overline{\tau_c})^m + W(\overline{\tau_o})^m\} \\
\Rightarrow &\sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^E + C_i^D}{T_i} + W(\overline{\tau_c} \cup \overline{\tau_o})^m \leq \sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^2}{T_i} + W(\overline{\tau_o})^m \\
&\{W(\overline{\tau_c} \cup \overline{\tau_o})^m = \sum_{i=j\ \&\ \tau_i \in \tau_{new}}^{j+m-1} \frac{S_i^2}{T_i},\ by\ the\ definition\ of\ \tau_j\} \\
\Rightarrow &\sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^E + C_i^D}{T_i} + \sum_{i=j\ \&\ \tau_i \in \tau_{new}}^{j+m-1} \frac{S_i^2}{T_i} \leq \sum_{\tau_i \in \overline{\tau_c}} \frac{C_i^2}{T_i} + W(\overline{\tau_o})^m.
\end{aligned} \tag{2}$$

Note that the term $\sum_{i=j\ \&\ \tau_i \in \tau_{new}}^{j+m-1} \frac{S_i^2}{T_i}$ denotes the summation of the $\frac{S_i^2}{T_i}$ ratios for tasks $\{\tau_j, \tau_{j+1}, ..., \tau_{j+m-1}\}$ in τ_{new}

[8]The only difference between β'' and β' is that in β'', tasks in $\overline{\tau_c}$ have their original offloading decisions of $O_i = 0$ given in β' changed to $O_i = 1$, as described earlier.

after tasks in τ_{new} are re-indexed according to line 9 of RODA. Also note that in the above equation, for the last step, if $\overline{\tau_c} \cup \overline{\tau_o}$ contains less than m tasks, then we simply use 0 to replace the needed values as appeared in the term $\sum_{i=j \ \& \ \tau_i \in \tau_{new}}^{j+m-1} \frac{S_i^2}{T_i}$, which does not violate the inequality.

We now input the modified offloading decision set β'' into Eq. (1). Recall that according to β'', tasks in $\overline{\tau_o} \cup \overline{\tau_c}$ have their offloading decisions $O_i = 1$ and all other tasks in $\overline{\tau_l} \cup \tau_{local}$ have their offloading decisions $O_i = 0$. We thus have

$$\sum_{\tau_i \in \tau} \frac{C_i^1 + C_i^3}{T_i} + \sum_{\tau_i \in \overline{\tau_l} \cup \tau_{local}} \frac{C_i^2}{T_i} + \sum_{\tau_i \in \overline{\tau_c} \cup \overline{\tau_o}} \frac{C_i^E + C_i^D}{T_i}$$
$$+ \sum_{i=j \ \& \ \tau_i \in \tau_{new}}^{j+m-1} \frac{S_i^2}{T_i}$$
$$\leq \sum_{\tau_i \in \tau} \frac{C_i^1 + C_i^3}{T_i} + \sum_{\tau_i \in \overline{\tau_l} \cup \tau_{local} \cup \overline{\tau_c}} \frac{C_i^2}{T_i} + \sum_{\tau_i \in \overline{\tau_o}} \frac{C_i^E + C_i^D}{T_i}$$
$$+ W(\overline{\tau_o})^m \leq m.$$

In the above equation, the first $\leq$ holds due to Eq. (2); the second $\leq$ holds due to the fact that this equality represents Eq. (1) with β' as the input offloading decision set, which is feasible as discussed earlier.

We have proven that β'' is a feasible decision set. Similar to the uniprocessor case proof, according to the way β'' is defined, we know that RODA is able to at least find τ_j when checking each task in turn in τ_{new} (line 11 of RODA). This is because such a τ_j would make the condition on line 11 of RODA to hold. Thus, RODA, in the worst case, can find τ_j. Or, RODA may identify another task $\tau_k \in \tau_{new}$, where $k < j$, that satisfies Eq. (1), in which case RODA is still guaranteed to produce a feasible offloading decision set. □

5 Experiments

To evaluate RODA, we conducted both large-scale schedulability simulations using randomly generated tasks, and experiments using a case-study offloading system implemented on top of real hardware.

5.1 Schedulability

The first set of experiments is to evaluate the schedulability loss under RODA and two other scheduling and offloading algorithms (described below), using randomly-generated task sets.

5.1.1 Experiment Setup

In the experiments, task sets were generated regarding following constraints.

- Task period T_i distributed uniformly over $[0.1s, 60s]$.
- Task utilization U_i uniformly distributed in three ranges as $[0.005, 0.1)$ (light), $[0.1, 0.3)$ (medium), $[0.3, 0.6)$ (heavy)
- The offloadable phase's execution time C_i^2 were uniformly distributed over $[T_i \times U_i \times 0.01, T_i \times U_i)$.
- The pre-offloading phase's execution time C_i^1 and the post-offloading phase's execution time C_i^3 are same and equal to $(T_i \times U_i - C_i^2)/2$.
- The offloading-induced delay, S_i^2, is uniformly distributed over $[0.1 \times C_i^2, 1.5 \times C_i^2)$.
- The offloading-induced overhead C_i^E and C_i^D are same and uniformly distributed in three ranges as $[0, 0.05 \times C_i^2)$ (low overhead), $[0.05 \times C_i^2, 0.2 \times C_i^2)$ (medium overhead), $[0.2 \times C_i^2, 0.6 \times C_i^2)$ (high overhead).

We varied the task system utilization U_{sum} in $\{0.1, 0.2, ..., m\}$. For each combination of task utilization distribution, overhead distribution, and U_{sum}, 10,000 task sets were generated for systems with one and four processors (i.e., $m = 1$ and $m = 4$). Each task set was generated by randomly creating tasks until the total task utilization exceeds the corresponding utilization cap. And then the utilization of the last task is reduced to make the total system utilization equal to the corresponding utilization cap.

We compared RODA with two other scheduling and offloading techniques. For supporting suspending task systems, a common approach, namely the suspension-oblivious analysis, is to treat all tasks' suspensions as computation. Doing so transforms any suspending task system into an ordinary sporadic task system and allows us to leverage existing techniques that are designed to handle sporadic task systems. We compared our applied suspension-aware analysis (Theorem 1) against the common suspension-oblivious analysis. The following theorem gives a GEDF-based SRT schedulability test under suspension-oblivious analysis [4].

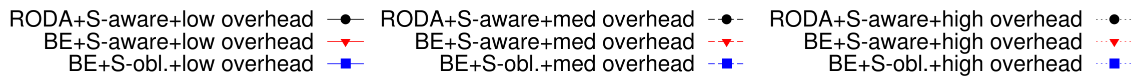
Theorem 4. *[4] For any given task system τ with a specific offloading decision set scheduled under GEDF, its SRT schedulability can be guaranteed if*

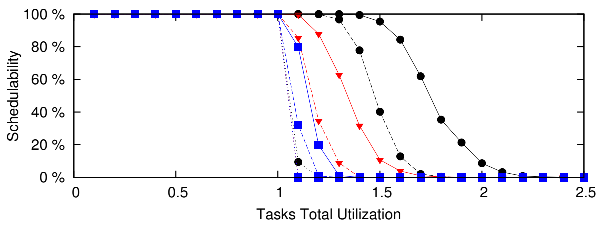
$$\sum_{i=1}^{n} \frac{C_i^1 + C_i^3 + C_i^2 \cdot (1 - O_i) + (C_i^E + C_i^D + S_i^2) \cdot O_i}{T_i} \leq m \tag{3}$$

Regarding the offloading algorithm, we compared RODA with a performance-driven offloading algorithm, denoted "BE", which improves the average-case performance on a best-effort basis. Under BE, any task τ_i will be offloaded if $C_i^2 > C_i^E + C_i^D + S_i^2$. The intuition is straightforward: offloading a task if it takes less time to execute the task remotely.

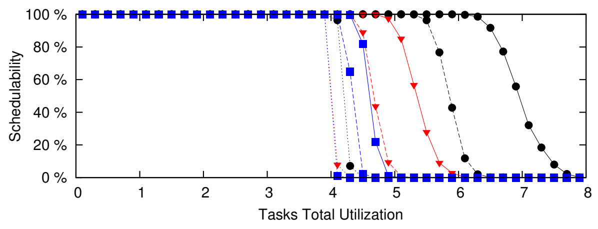
5.1.2 Results

The schedulability results are shown in Fig. 4. In all graphs, the x-axis indicates the total task system utilization and the y-axis indicates the ratio of schedulable task set with respect to the 10,000 task sets generated for each data point.

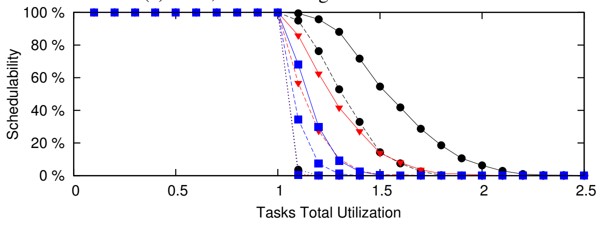




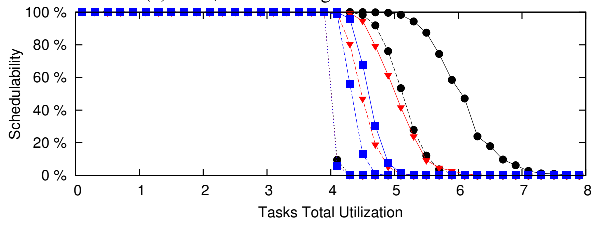
(a) m=1, tasks with light task utilization



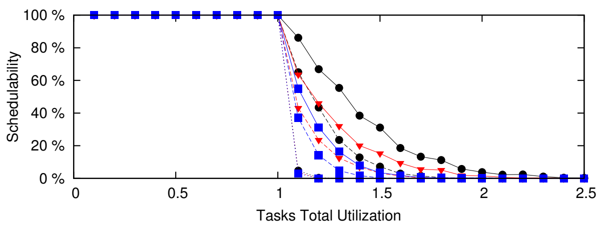
(b) m=4, tasks with light task utilization



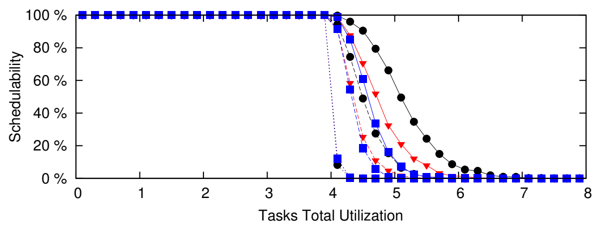
(c) m=1, tasks with medium task utilization



(d) m=4, tasks with medium task utilization



(e) m=1, tasks with heavy task utilization



(f) m=4, tasks with heavy task utilization

Figure 4: Schedulability of tasks with different workload on uni- and multiprocessors

In the first (second) column of graphs, we assume $m = 1$ ($m = 4$). In the first (second, third) row, per-task utilization is low (medium, high). For each graph, we have nine curves, representing low, medium, and high overhead settings, combined with three different approaches: RODA with suspension-aware analysis (denoted "RODA+S-aware"), BE with suspension-aware analysis (denoted "BE+S-aware"), and BE with suspension-oblivious analysis (denoted "BE+S-obl."). The schedulability tests under suspension-aware and suspension-oblivious analysis with a given offloading decision set are given in Theorem 1 and Theorem 4, respectively.

As seen in Fig. 4, RODA yields the best performance in all cases. For example, in Fig. 4(b), under low overhead setting on four processors, RODA is able to guarantee 100% schedulability for task sets with total utilization up to 6.2, while BE+S-aware and BE+S-obl. can do so for task sets with total utilization up to only 4.6 and 4.2, respectively. Such performance improvements are due to two factors. First, the suspension-aware analysis using Theorem 1 is superior to the suspension-oblivious analysis using Theorem 4, thus allowing more task sets to be deemed schedulable. Second, the BE offloading algorithm only considers individual task parameters when making offloading decisions. Such "greedy" offloading decisions may be undesirable to enable the overall task system to pass the schedulability test. An observation made in all the graphs is that when the total task system utilization is no greater than m, the schedulability under all approaches can always reach 100%. This is because in this case, executing all tasks locally (i.e., $O_i = 0$ for any τ_i) would already make the task system to be schedulable according to either Theorem 1 or Theorem 4 (under this case such task systems were automatically deemed schedulable). Another observation is that when task utilizations are lighter, all approaches yield better schedulability. This is because when task utilizations are lighter, the generated task suspension lengths may also become shorter, as discussed in our experiment setup. As seen in both Eqs. (1) and (3), shorter task suspension lengths may give task sets higher possibilities to

	Client	Server
Processor	Intel Celeron Processor 847 1.10GHz Dual Core	Intel Core i7-4700MQ 2.40GHz Quad Core
Operating System	Linux Ubuntu Lucid	Linux Ubuntu Lucid
Wireless Router	N150 Wireless Router WNR 1000V3	
Max. WiFi Speed	Up to 150 Mbps	

Figure 5: Hardware configuration.

Application senario	Computationally intensive	Computationally not intensive
Period	20 seconds	20 seconds
Total execution time (local)	2 seconds	2 seconds
Offloading phase execution time (local)	1.8 seconds	0.7 seconds
Offloadable phase execution time (remote)	0.7 seconds	1.1 seconds

Figure 6: Application parameters.

pass the corresponding SRT schedulability test.

5.2 A Real Case Study

To evaluate the runtime performance of RODA compared to BE, we have implemented a case study offloading system. The metrics evaluated in the experiments are the average and the maximum task response times.

5.2.1 System Setup

Hardware. We used one server and one client with different computation capability. The configuration details are shown in Fig. 5.

Network communication. We used Java sockets for communications between the local system and the server.

Application. We implemented a periodic matrix manipulation application, described as follow. We created a training set consisting of the same applications but with different input data sizes (i.e., different string sizes) and did extensive profiling to obtain the corresponding task parameters.

- Build matrices based on input strings (C_i^1) $\rightarrow$ Perform matrix manipulation operations and Transform matrices to strings (C_i^2) $\rightarrow$ Print String (C_i^3)

Offloading decisions are made by RODA using the profiled parameters before runtime.

We performed three sets of experiments, where the offloadable execution phase of each matrix application is computationally intensive, computationally not intensive, or mixed. According to our profiling, the parameters of these two types of applications are shown in Fig. 6.

5.2.2 Results

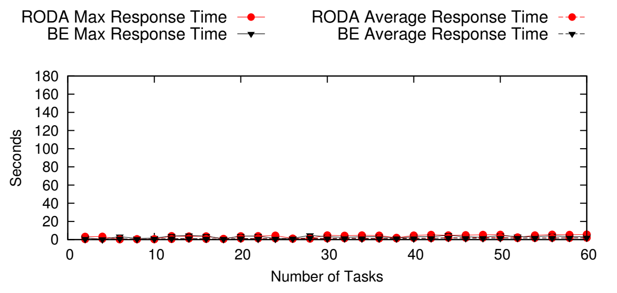
The case study results are shown in Fig. 7. For all the included graphs, the y-axis represents the response time in seconds and the x-axis represents the number of concurrent matrix manipulation applications. For each application, we run them for five releases and record the response time information. Each graph contains four curves, representing the RODA and BE approaches with two metrics. For the mixed application scenario, as shown in Fig. 7(c), the ratio of the number of computationally not intensive tasks to the number of computationally intensive tasks is five to one.

As seen in Fig. 7, RODA yields better performance than BE for both average and maximum task response times in most cases. For example, as seen in Fig. 7(b), with 40 applications running concurrently, both the average and the maximum task response times under RODA is under 20 seconds, while the maximum task response time under BE is over 100 seconds. The results show that with the more intelligent offloading decisions given by RODA, runtime response times can be significantly reduced, particularly with heavy workloads. One observation made in these experiments is that the performance improvement achieved by RODA becomes less significant when the applications in the system are more computationally intensive. This is intuitive because it is clearly more beneficial to offload a computationally intensive application since executing such tasks locally would cause the local system to more likely become overloaded. As shown in Fig. 7(a), when the system only contains computationally intensive applications, RODA and BE yield almost identical performance.

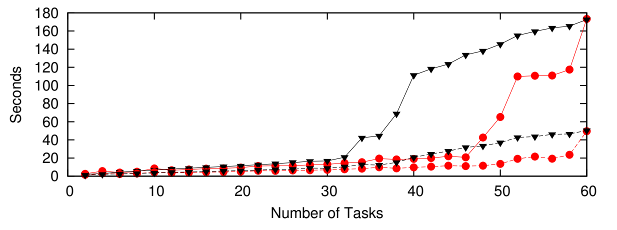
6 Conclusion and Future Work

This paper establishes an offloading framework for embedded systems supporting SRT applications. The underlying key idea is to model offloading-induced delays as suspensions occurring at the local embedded system. It is thus possible to formally model an application being offloaded as a suspending task. Existing SRT suspension-aware schedulability analysis techniques can then be leveraged to ensure SRT schedulability of any given task system. An efficient offloading algorithm, RODA, has been proposed, with provable optimality properties for embedded systems containing either uniprocessors or multiprocessors. Both extensive simulations and a real case study system demonstrated that RODA yields better performance in terms of schedulability and runtime response times than a commonly used performance-driven offloading algorithm.

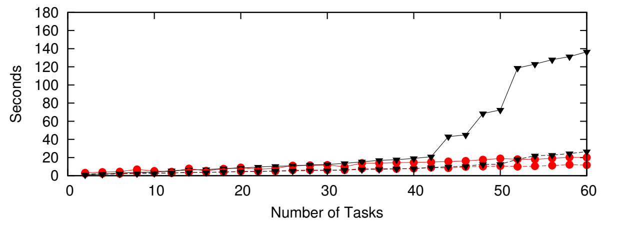
There are several important research issues we plan to address in the near future in order for this framework to be of value in practice. We have made several assumptions in this paper in order to mathematically model the system and describe the problem. First, we plan to investigate methods that can handle networking exceptions, that is, what if the result for an offloaded task component is delayed or even lost during the transmission? Second, how to derive a safe yet tight response time bound for any task component that is offloaded to the remote site? Thirdly, assuming worst-case



(a) Response time performance when tasks are computationally intensive



(b) Response time performance when tasks are not computationally intensive



(c) Response time performance with mixed tasks

Figure 7: Case study results.

execution times and periodic arrival patterns may not be practical for many real-world workloads with SRT requirements. We plan to develop efficient scheduling and offloading techniques with provably analytical properties that can remove these assumptions.

References

[1] G. Chen, B. Kang, M. Kandemir, N. Vijaykrishnan, M. Irwin, and R. Chandramouli. Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):795–809, 2004.

[2] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[3] Y. Dai, M. Xie, and X. Wang. A heuristic algorithm for reliability modeling and analysis of grid systems. *IEEE Transactions on Systems, Man and Cybernetics*, 37(2):189–200, 2007.

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

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

[6] W. Gao, Y. Liu, H. Lu, T. Wang, and C. Liu. On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *Proceedings of the 22nd IEEE International Conference on Network Protocols*, to appear, 2014.

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

[8] Z. Li, C. Wang, and R. Xu. 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*, pages 238–246, 2001.

[9] C. Liu and J. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Proceedings of the 30th Real-Time Systems Symposium*, pages 425–436, 2009.

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

[11] W. Liu, J. Chen, A. Toma, T. Kuo, and Q. Deng. Computation offloading by using timing unreliable components in real-time systems. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6, 2014.

[12] Y. Liu, C. Liu, X. Zhang, W. Gao, L. He, and Y. Gu. A computation offloading framework for soft real-time embedded systems. Technical report, The University of Texas at Dallas, `http://www.utdallas.edu/~cong/TR15a.pdf`, 2015.

[13] Y. Nimmagadda, K. Kumar, Y. Lu, and G. Lee. Real-time moving object recognition and tracking using computation offloading. In *Proceedings of the IEEE Intertional Conference on Intelligent Robots and Systems*, pages 2449–2455, 2010.

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

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

[16] Cheng Wang and Zhiyuan Li. A computation offloading scheme on handheld devices. *Journal of Parallel and Distributed Computing*, 64(6):740 – 746, 2004.

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

[18] Y. Ye, L. Xiao, I. Yen, and F. Bastani. Leveraging service clouds for power and qos management for mobile devices. In *Proceedings of the 4th IEEE International Conference on Cloud Computing*, pages 235–242, 2011.