Computation Offloading for Real-time Systems

Anas Toma
Department of Informatics
Karlsruhe Institute of Technology, Germany
anas.toma@student.kit.edu

Jian-Jia Chen
Department of Informatics
Karlsruhe Institute of Technology, Germany
jian-jia.chen@kit.edu

ABSTRACT

Computation offloading has been adopted to improve the performance of embedded systems by offloading the computation of some tasks, especially computation-intensive tasks, to servers or clouds. This paper explores computation offloading for real-time embedded systems to decide which tasks should be offloaded to get the results in time. Such a problem is $\mathcal{NP}$-complete even for frame-based real-time tasks with the same period and relative deadline. We develop a pseudo-polynomial-time algorithm for deriving feasible schedules, if they exist.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*Scheduling*.

Keywords

Computation offloading, task scheduling, real-time systems.

1. INTRODUCTION

Mobile devices are getting increasingly popular nowadays. They have become devices that are often used for multiple functionalities. Specifically, many of their applications are computation intensive, such as video processing,image and voice recognition, etc.

However, even though the performance improvement for mobile devices will continue, their computation capabilities are still quite limited, due to the resource constraints on these devices. It may not be worth to improve the embedded systems just to execute some computation-intensive applications for extreme cases. Therefore, *computation offloading*, as illustrated in Figure 1, can be adopted in the embedded systems with constrained resources by moving a task from a resource-constrained device (here, we call it a *client*) to one or more devices (here, we call them *servers*). The task can be a part of an active program or a complete one. The servers can either provide faster execution in general or accelerate the execution for some specific tasks. Moreover, even when the servers are slower, offloading may also be beneficial for the client as the computation is done remotely so that the energy consumption of the client can be reduced or another task can be executed on the client while awaiting the results from the servers.

We explore the computation offloading mechanism to meet the timing constraint of real-time tasks. Although the idea of computation offloading has been studied previously [1–7], all of the existing approaches decide whether a task is executed locally or is offloaded without scheduling the execution order, even for independent tasks. Therefore, during the remote execution of an offloaded task, the client always remains idle until the result of the task returns from the server. Also, they assume that a server, implicitly, is dedicated for the client to run an offloaded task immediately.

SAC'13 March 18-22, 2013, Coimbra, Portugal.
Copyright 2013 ACM 978-1-4503-1656-9/13/03
http://doi.acm.org/10.1145/2480362.2480670 ...$10.00.

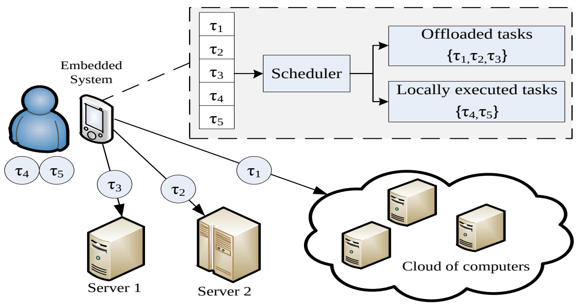

Figure 1: Offloading Mechanism.

Moreover, timing requirements are important for real-time applications, in which the results may become useless or even harmful to the client if the deadlines are not met. Most of the approaches with computation offloading either do not consider the timing satisfaction requirement for real-time properties, e.g., in [2–4, 6, 7], or use pessimistic offloading mechanism for deciding whether a task can be offloaded [5]. Although the offloading mechanism by Nimmagadda et al. [5] can improve the response time and the local execution time on the client, they do not fully exploit the potential of computation offloading for satisfying the timing requirement or achieving better performance by increasing the sampling rates.

Due to the space limitation, we only present the key conceptual observations, and omit all the detailed proofs.

2. SYSTEM MODEL

Suppose that we are given a set $\mathcal{T}$ of n independent frame-based real-time tasks. All the tasks have the same arrival time 0, period D and relative deadline D . Each task $\tau_i \in \mathcal{T}$ is associated with the following timing parameters:

- **Worst-case *local* execution time** C_i.
- **Setup time** S_i**:** It includes any local pre-processing operations before offloading, and also includes the transmission time of the offloaded task to the server.
- **Round-trip offloading time** I_i**:** the interval length starting from the end of setting up S_i for task τ_i until getting the result from the server. The client contacts the server/s before scheduling to get the values of I_i.

Figure 2 shows these timing parameters, where tasks τ_1 and τ_2 are offloaded, and tasks τ_3 and τ_4 are locally executed. The client has to schedule task executions to satisfy the real-time constraints, whereas the servers have their own scheduling policies to handle the tasks that are offloaded from the clients and to ensure I_i value.

2.1 Problem Definition

Given a set $\mathcal{T}$ of n frame-based real-time tasks, the SElective

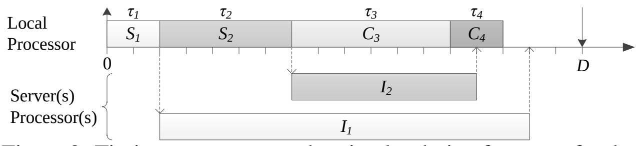

Figure 2: Timing parameters and optimal ordering for a set of tasks.

Real-Time Offloading (SERTO) problem is to schedule the tasks and to decide when and what to offload without violating timing constraints for a client.

A schedule is *feasible* if all the locally-executed and offloaded tasks are finished no later than the deadline D. A scheduling algorithm is said to be *optimal offloading scheduling* algorithm if it is able to find a feasible schedule, if and only if one exists.

Suppose that x_i is equal to 1 if task τ_i is decided to be offloaded; otherwise, x_i is 0. We use a vector $\vec{x}_n = (x_1, x_2, \ldots, x_n)$ to denote an *offloading decision* for the given n tasks.

3. HARDNESS OF THE SERTO PROBLEM

Suppose that the computation offloading decisions have been made. The following lemma decides the optimal ordering.

LEMMA 1. *If the execution order is not specified, all the offloaded tasks should be executed before any locally-executed task.*

When the offloading decision $\vec{x}_n$ for the tasks is known, we define $d_i = x_i(D - I_i) + (1 - x_i)D$ as the *virtual offloaded deadline*. If there is a feasible schedule , then executing the tasks by following the order of d_i non-decreasingly is also a feasible schedule. This ordering is called Earliest Virtual Offloaded Deadline First (**EVODF**). Please refer to Figure 2, as an illustration example for an optimal ordering for a given set of four tasks.

LEMMA 2. *If the execution order is not specified and there is a feasible schedule based on the offloading decisions, the schedule by using **EVODF** is also a feasible schedule.*

LEMMA 3. *Suppose that tasks $\tau_i \in \mathcal{T}$ are ordered non-decreasingly according to $D - I_i$. An offloading decision $\vec{x}_n$ results in a feasible schedule (by using **EVODF**) if and only if*
(a) $\sum_{j=1}^{n} x_j S_j + (1 - x_j) C_j \leq D$, and
(b) $x_k I_k + \sum_{j=1}^{k} x_j S_j \leq D, \forall k = 1, 2, \ldots, n$.

The $\mathcal{NP}$-completeness can be proved by a reduction from the SUBSET SUM problem.

THEOREM 1. *The SERTO problem is $\mathcal{NP}$-complete if the execution order is not given.*

4. OUR APPROACH

Based on dynamic programming, we introduce a pseudo-polynomial-time algorithm called *Dynamic Real-time Scheduling (*DRS) algorithm to find a feasible solution for the SERTO problem. Initially, all tasks are ordered non-decreasingly according to $D - I_i$.

An offloading decision $\vec{x}_i$ for the first i tasks, i.e., $\{\tau_1, \tau_2, \ldots, \tau_i\}$, is said *partially feasible for offloading* if the offloaded tasks can finish the execution in the servers before the given deadline D. Similar to Lemma 3, we know that a vector $\vec{x}$ is partially feasible for offloading for $\{\tau_1, \tau_2, \ldots, \tau_i\}$ if and only if $x_k I_k + \sum_{j=1}^{k} x_j S_j \leq D, \forall k = 1, 2, \ldots, i$.

Our strategy is to build a dynamic programming table by maintaining and storing some scheduling results for the partially feasible offloading decisions for the first i tasks. Specifically, among all the partially feasible offloading decisions for $\{\tau_1, \tau_2, \ldots, \tau_i\}$, let $G(i, t)$ be the minimum total local execution time for the locally-executed tasks under the constraint that the total setup time for the offloaded tasks in $\{\tau_1, \tau_2, \ldots, \tau_i\}$ is less than or equal to t. That is, for a given i and t, the value $G(i, t)$ is the objective function of the following integer linear programming (ILP):

$$\text{minimize} \sum_{j=1}^{i} (1 - x_j) C_j \tag{1a}$$

$$\text{s.t} \sum_{j=1}^{i} x_j S_j \leq t \tag{1b}$$

$$x_k I_k + \sum_{j=1}^{k} x_j S_j \leq D \qquad \forall k = 1, 2, \ldots, i \tag{1c}$$

$$x_j \in \{0, 1\} \qquad \forall j = 1, 2, \ldots, i. \tag{1d}$$

For notational brevity, when the above ILP has no feasible solution, $G(i, t)$ is defined as ∞. Moreover, $G(i, t) = \infty$ when $t < 0$.

Clearly, when i is 1, we know that

$$G(1, t) = \begin{cases} 0 & \text{if } S_1 \leq t \leq D - I_1 \\ C_1 & \text{otherwise.} \end{cases} \tag{2}$$

The construction of $G(i, t)$, for $i \geq 2$, can be achieved by using the following recurrence:

$$G(i, t) = \min \begin{cases} \begin{cases} G(i-1, t - S_i) & \text{if } t \leq D - I_i \\ \infty & \text{otherwise} \end{cases} \\ G(i-1, t) + C_i \end{cases} \tag{3}$$

The recursive function in (3) represents the selection of the minimum solution by comparing two cases:

- Case 1: task τ_i is offloaded when its local setup execution finishes at time t. For such a case, if $t + I_i > D$, offloading τ_i is an infeasible offloading decision; otherwise, we consider the offloading decision for the first $i - 1$ tasks, in which the total local execution time of this solution is $G(i-1, t - S_i)$.
- Case 2: task τ_i is locally executed. Therefore, we consider the offloading decision for the first $i-1$ tasks. As a result, the total local execution time of this solution is $C_i + G(i-1, t)$.

The standard dynamic-programming procedure can be applied by constructing a table with n rows for $i = 1, 2, \ldots, n$ and $D + 1$ columns for $t = 0, 1, 2, \ldots, D$. The time complexity of the algorithm is $O(n \log n + nD)$, which is pseudo-polynomial.

For an input task set $\mathcal{T}$, to verify whether a feasible schedule exists for the SERTO problem or not, we just have to check whether there exists $0 \leq t \leq D$ with $G(n, t) + t \leq D$.

References

[1] L. Ferreira, G. Silva, and L. Pinho. Service offloading in adaptive real-time systems. In *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1 –6, 2011.

[2] Y.-J. Hong, K. Kumar, and Y.-H. Lu. Energy efficient content-based image retrieval for mobile systems. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 1673 –1676, may 2009.

[3] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *CASES*, pages 238–246, 2001.

[4] Z. Li, C. Wang, and R. Xu. Task allocation for distributed multimedia processing on wirelessly networked handheld devices. In *Parallel and Distributed Processing Symposium., Proceedings International*, IPDPS 2002, pages 79 –84, 2002.

[5] Y. Nimmagadda, K. Kumar, Y.-H. Lu, and C. Lee. Real-time moving object recognition and tracking using computation offloading. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 2449 –2455, 2010.

[6] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi. Using bandwidth data to make computation offloading decisions. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–8, 2008.

[7] C. Xian, Y.-H. Lu, and Z. Li. Adaptive computation offloading for energy conservation on battery-powered systems. In *Parallel and Distributed Systems, 2007 International Conference on*, volume 2, pages 1–8, 2007.