Server Resource Reservations for Computation Offloading in Real-Time Embedded 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*—Mobile devices have become very popular nowadays. They are used nearly everywhere. They run complex applications where the multimedia data are heavily processed. For example, ubiquitous applications in smart phones and different surveillance tasks on mobile robots. However, most of these applications have real-time constraints, and the resources of the mobile devices are limited. So, it is challenging to finish such complex applications on these resource-constrained devices without violating the real-time constraints. One solution is to adopt the *Computation Offloading* concept by moving some computation-intensive tasks to a powerful server. In this paper, we use the total bandwidth server (TBS) as resource reservations in the server side, and propose two algorithms based on the computation offloading to decide which tasks to be offloaded and how they are scheduled, such that the utilization (i.e., bandwidth) required from the server is minimized. We consider frame-based real-time tasks, in which all the tasks have the same arrival time, relative deadline and period. The first algorithm is a greedy algorithm with low complexity based on a fast heuristic. The second one is a pseudo-polynomial-time algorithm based on dynamic programming. Finally, the algorithms are evaluated with a case study for surveillance system and synthesized benchmarks.**

I. Introduction

Mobile devices are running more and more complex applications that include computation-intensive data processing. For example, surveillance robots perform real-time monitoring for security, rescue and safety purposes [4, 9, 11]. Also, smart phones are increasingly used to run heavy tasks such as 3D navigation [18, 26], ubiquitous and multimedia applications [3, 19], etc. Google Glass [2] is an example of the next-generation ubiquitous systems, which performs voice recognition, navigation, translation, video streaming and recording, etc. Such multimedia mobile applications usually require high computation power. However, these mobile devices are resource-constrained. They have limited computation capabilities and battery life. They may not be able to complete the execution of the computation-intensive tasks in time. *Computation offloading* concept has been adopted in the literature to improve the performance of the mobile devices with the help of external resources, i.e., remote processing unit. Figure 1 shows an example of the offloading mechanism. We use the term *client* to refer to the mobile device that offloads the tasks, and the term *server* to refer to the remote processing unit that executes the offloaded tasks.

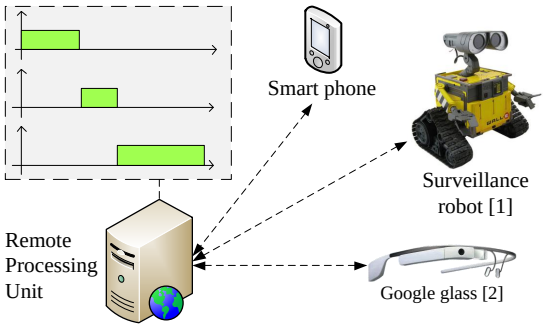


Fig. 1: Example of the offloading Mechanism.

In this paper, we exploit computation offloading to schedule frame-based real-time tasks in a client, where all the tasks have the same arrival time, period and relative deadline. For instance, in surveillance systems once the camera captures an image, several tasks start processing it, in which all of them should be completed before the next capture, which represents the relative deadline of the task execution. Usually, the tasks in the surveillance systems run periodically (with each image capture) and usually include motion detection, object recognition, behavioral analysis, and analysis of the stereo vision [24, 25]. These tasks are computation intensive for resource-constrained devices, and may not finish before the deadline. Therefore, our proposed algorithms use the computation offloading to guarantee the execution of the tasks without violating the real-time constraints. To solve this problem, two decision-making points should be addressed: (1) which tasks to be offloaded (i.e., offloading decision), and (2) when to offload the selected tasks for offloading (i.e., task ordering).

In real-time systems, when the client decides to offload a task to a server, such computation offloading should still make the offloaded task meet its (hard or soft) timing constraint, i.e., the task should be completed in time under the given (hard or soft) timing parameters. Therefore, the server should also provide some timing properties for executing the offloaded tasks. The offloading decision in the most of existing studies, e.g., [7, 16, 27], is based on a simple comparison to check whether it is beneficial in terms of processing time (or processing cost) to offload a task.

In our previous study in [23], we design and analyze the computation offloading mechanism for frame-based real-time

Published in the 11th IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia), Montreal, QC, Canada, October 3-4, 2013. http://dx.doi.org/10.1109/ESTIMedia.2013.6704500

tasks when the server provides *round-trip timing information*. That is, when an offloaded task arrives at the server at time t, the client can receive the computation at time t plus the provided round-trip time. However, such a round-trip timing information requires resource reservations in the server(s) before the offloading decisions are made. If a task is not offloaded such a reservation becomes useless and the capability of the server(s) is wasted. In another study in [22], a total bandwidth server (TBS) [21] is used in the server side to serve a client for computation offloading. Specifically, when the bandwidth (utilization) of the TBS is given, the algorithms in [22] minimize the finishing time (makespan) by using a heuristic algorithm and a dynamic programming approach.

In [22], if the given bandwidth (utilization) of the TBS for a client is too high, the capability in the server side is still wasted. Determining the bandwidths for individual clients remains open. In this paper, we consider computation offloading for real-time embedded systems, in which the objective is to find the minimum required bandwidth reservation for TBS in the server side to meet the timing constraint of the real-time tasks. Therefore, depending on the availability of the remaining bandwidth in the server, the server will grant the clients the required utilization (or even adjust the bandwidths from other clients to make all the clients meet their timing constraints).

The considered problem is a dual problem of the problem studied in [22]. That is, the bandwidth (utilization) of the TBS is a given constraint and the minimization of the finishing time of the tasks is the objective for the problem studied in [22], whereas the finishing time of the tasks is a constraint and the minimum required bandwidth (utilization) of the TBS is the objective in this paper. It has been shown in [22] that the decision version for studied problem in this paper and also in [22] is also $\mathcal{NP}$-complete in the weak sense.

Our Contribution: Our contribution can be summarized as follows:

- In our model, we consider the scheduling on both client and server sides, where the server can serve more than one client.
- We propose two computation offloading algorithms, a greedy algorithm with low time complexity, and a dynamic programming algorithm with pseudo-polynomial time complexity. The algorithms schedule the real-time tasks on the client side, and decide which tasks to be offloaded to the server, such that the real-time constraints are satisfied and the required utilization is minimized.
- We evaluate our algorithms using a surveillance system as a case study and randomly synthesized benchmarks.

II. Related Work

In this section, we provide a summary for the recent studies in the field of computation offloading. Also, we discuss the limitations of the existing approaches.

Nimmagadda et al. [16] use the computation offloading to improve the performance and satisfy the real-time constraints for a mobile robot. The robot performs real-time moving object recognition and tracking. For each task, the offloading decision is taken if the summation of the communication time and the server execution time is less than the local execution time on the robot. The computation offloading is also considered by Wolski et al. [27] and Gurun et al. [7] to improve the performance for computational grid settings. The offloading decision in their work is based on the prediction of the client execution time, the transfer time and the server execution time. They assume that the offloading of a task is beneficial only if the expected cost of the remote execution, including the server execution time and the data transfer time, is less than the cost of the local execution.

The Offloading problem is represented as a graph partitioning problem in [6, 14, 15, 17, 29] and has been solved using different approaches. Each vertex of the graph represents a computational component, such as program class in [6, 17, 29] or a task in a program as in [14, 15]. An edge between two vertices represents the communication cost between them. The main idea is to partition the graph into two parts, client side and server side.

A middleware for mobile Android platforms is proposed in [12] to offload the computation intensive tasks to a remote clouds. The offloading problem is represented as an optimization problem and solved using integer linear programming. To reduce the energy consumption in battery-powered systems, the computation offloading is adopted in [28] based on a timeout mechanism, and also adopted in Hong et al. [8] for a system that performs content-based image retrieval. Ferreira et al. [5] explore the computation offloading to improve the quality of service in the adaptive real-time systems.

The offloading decision in the most of existing studies is based on either simple comparison for each task alone, or representing a program as a graph and then partition it [13]. In both approaches, the task scheduling is not considered neither on the client nor on the server. For example, the client remains idle during offloading until the result returns back from the server. Instead, an independent local task can be executed during the offloading of another task to improve the performance. Also, it is important to consider the server model, and how it schedules the tasks to serve more than one client. Based on the existing approaches, the powerful server is always ready to execute the offloaded tasks from the client immediately, which means that the server is dedicated for one client.

III. System Model

This section presents the system model of the paper. As the problem is a dual problem of [22], the terminologies and system architecture are similar to our previous work in [22]. However, the problem definition here is different from [22]. We say that a schedule for a set of tasks is *feasible* if the total finishing time for all of the tasks is within the deadline D.

A. Client Side

On the client side, a set $\mathcal{T}$ of n independent frame-based real-time tasks arrive periodically at the same time $t = 0$, have the same period D, and require execution within the same relative deadline D. The client schedules the tasks and

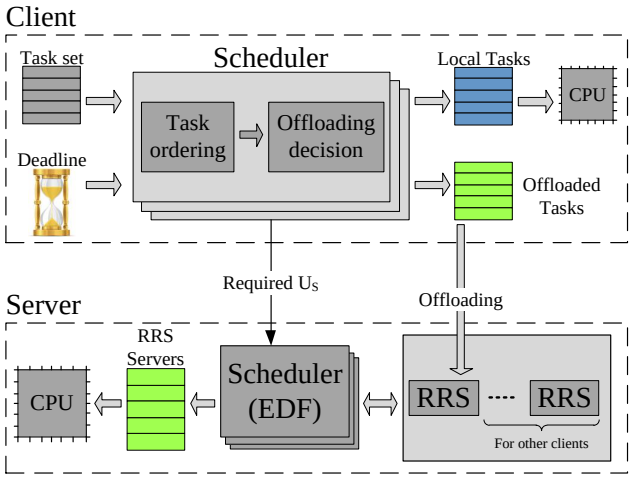


Fig. 2: Offloading System Architecture.

decides which of them to be offloaded to the server in order to satisfy the real-time constraints. Each task $\tau_i \in \mathcal{T}$ (for $i = 1, 2, \ldots, n$) is characterized by the following timing parameters:

- C_i: Local execution time on the client side.
- S_i: Setup time. The execution time required on the client side to be ready for offloading. It includes any preprocessing operations such as encoding and compression. It also includes the sending time of the data of the task from the client to the server.
- R_i: Remote execution time. The execution time on the server side.

A task is said *locally executed* if it is executed with at most C_i amount of time only on the client side. A task is said *offloaded* if it is executed on the server side after setting up on the client side. That is, if a task is offloaded, it has to be executed (at most) S_i amount of time on the client, and then executed with (at most) R_i amount of time on the server. The timing information C_i, S_i, and R_i can be either *soft* based on profiling or *hard* based on static timing analysis with predictable communication fabrics. When the information is soft, we provide soft real-time guarantees, and vice versa.

B. Server Side

The server is able to serve more than one client. *Total Bandwidth Server* (TBS) is considered in our server model [20, 21]. The client finds the feasible schedule that requires the minimum utilization (or bandwidth) U_s from the server. If it is possible, the server assigns a TBS for each requesting client such that the total utilization of all given TBS's are less than or equal to 100%. Figure 2 shows the architecture of the offloading system. The TBS of each client assigns an absolute deadline $d_s(t)$ for each incoming (offloaded) task τ_i from that client, where τ_i arrives at the server side at time t. Let $d_s(t^-)$ be the absolute deadline of the previous offloaded task, and $d_s(0^-)$ is equal to 0. Then, the absolute deadline $d_s(t)$ can be calculated as follows:

$$d_s(t) = \max\{t, d_s(t^-)\} + \frac{R_i}{U_s}$$

The server schedules the offloaded tasks according to the *Earliest Deadline First* (EDF) algorithm based on the assigned TBS deadlines. By having at most 100% total utilization, the server guarantees that all the offloaded tasks meet their assigned deadlines [21].

C. Problem Definition

The problem addressed in this paper can be defined as follows: *Given a set $\mathcal{T}$ of n frame-based real-time tasks and a deadline D. The problem is to find a schedule that meets the deadline D, and the offloading decisions for the tasks in $\mathcal{T}$ with the minimum required TBS utilization U_s from the server.*

IV. Known Results for Minimizing Makespan

In this subsection, we summarize the technique used in our previous work in [22] to find the optimal ordering of the tasks for a given Utilization U_s. We define the vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ to represent the offloading decisions of the tasks in $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. $x_i = 1$ means that the task τ_i is assigned for offloading, and $x_i = 0$ means that it is assigned for local execution. We suppose that the offloading decisions $\vec{x}$ are known for all the tasks (i.e., *what to offload*), and we want to determine here the ordering of the tasks (i.e., *when to offload*). The *makespan* of a task set $\mathcal{T}$ is the total completion time of the tasks in $\mathcal{T}$ on both client and server sides.

For each task τ_i, let $P_{i,c}$ be its execution time on the client side, and $P_{i,s}$ its remote execution time divided by the utilization of the TBS. The two terms can be defined as follows:

$$P_{i,c} = x_i S_i + (1 - x_i) C_i, \tag{1}$$

$$P_{i,s} = x_i \frac{R_i}{U_s}. \tag{2}$$

Based on Johnson's rule [10] we divide the tasks into two sets $\mathcal{T}_1$ and $\mathcal{T}_2$ as follows:

- $\mathcal{T}_1 = \{\tau_i | P_{i,c} \leq P_{i,s}\}$.
- $\mathcal{T}_2 = \{\tau_i | P_{i,c} > P_{i,s}\}$.

First, we schedule the tasks in the set $\mathcal{T}_1$ in increasing order according to $P_{i,c}$. Then, the tasks in the set $\mathcal{T}_2$ in decreasing order according to $P_{i,s}$.

Lemma 1: For a given $\vec{x}$ and utilization U_s, the ordering of the tasks execution based on Johnson's rule is optimal for minimizing the makespan of the execution.

Proof: The proof is in the paper [22]. ■

Corollary 1: All the local tasks belong to the set $\mathcal{T}_2$ and scheduled at the end.

Proof: This comes from the definition that $P_{i,s} = 0$ if τ_i is locally executed. ■

For a given set of tasks $\mathcal{T}$, the proposed dynamic programming algorithm $F(\mathcal{T}, D, U_s)$ in [22] returns $true$ if there exists a schedule, under a given utilization U_s, with a makespan less than or equal to D. Otherwise it returns $false$.

V. Optimal Task Ordering

As we discussed in Section I, the offloading decision and the task ordering are important to solve our offloading problem. In this paper, we use the optimal ordering from our previous work in [22] to minimize the makespan of the execution. In Section IV we discussed the optimal ordering of the tasks for a given U_s. But in our problem, we want to minimize the required utilization from the server. Changing the utilization may also changes the execution order of the tasks. In Subsection V-A we determine all the possible utilization levels that may change the ordering of the tasks. In Subsection V-B we find the interval that contains the minimum required utilization.

A. Utilization Levels

Lemma 2: For any given $0 \leq U_s \leq 1$, and according to the increasing order of the tasks by $\frac{R_i}{S_i}$, if $i < j$ and the task τ_j is in the set $\mathcal{T}_2$ then τ_i is also in the same set.

Proof: By the definition of the set $\mathcal{T}_2$, $\tau_j \in \mathcal{T}_2$ implies that $S_j > \frac{R_j}{U_s} \Rightarrow U_s > \frac{R_j}{S_j}$. But we know that $\frac{R_i}{S_i} < \frac{R_j}{S_j}$ (because of the ordering and $i < j$). Then, $U_s > \frac{R_i}{S_i} \Rightarrow S_i > \frac{R_i}{U_s} \Rightarrow \tau_i \in \mathcal{T}_2$. ∎

Based on the definition of the sets $\mathcal{T}_1$ and $\mathcal{T}_2$, changing the utilization U_s may change the relation between S_i and $\frac{R_i}{U_s}$ and then moves a tasks(s) from one set to another, which will result in a new combination of $\mathcal{T}_1$ and $\mathcal{T}_2$. The following theorem determines the maximum number of possible combinations for $\mathcal{T}_1$ and $\mathcal{T}_2$.

Theorem 1: The combinations of the sets $\mathcal{T}_1$ and $\mathcal{T}_2$ are at most $n+1$ combinations.

Proof: Let task τ_i be the task with the maximum $\frac{R_i}{S_i}$ in the set $\mathcal{T}_2$. According to Lemma 2 and the increasing order of the tasks by $\frac{R_i}{S_i}$, $\mathcal{T}_2 = \{\tau_1, \tau_2, \ldots, \tau_i\}$ and $\mathcal{T}_1 = \{\tau_{i+1}, \tau_{i+2}, \ldots, \tau_n\}$, which are at most $n-1$ combinations for $i = \{1, 2, \ldots, n-1\}$. Clearly, by considering also the two cases when $\mathcal{T}_1$ and $\mathcal{T}_2$ are empty, we have in total $n+1$ combinations. ∎

Theorem 2: The decreasing of the utilization U_s from 1 to 0 moves the tasks from the set $\mathcal{T}_2$ to the set $\mathcal{T}_1$.

Proof: Assume that U_{s_1} and U_{s_2} are two different utilization values, such that $U_{s_1} \geq U_{s_2}$. Also, let $\mathcal{T}_1(U_s)$ and $\mathcal{T}_2(U_s)$ be the two sets $\mathcal{T}_1$ and $\mathcal{T}_2$ for a given utilization U_s. We want to proof that if $U_{s_1} \geq U_{s_2}$, $\mathcal{T}_1(U_{s_1}) \subseteq \mathcal{T}_1(U_{s_2})$ and $\mathcal{T}_2(U_{s_1}) \supseteq \mathcal{T}_2(U_{s_2})$.

Suppose for contradiction that there exists a task τ_i such that $\tau_i \in \mathcal{T}_1(U_{s_1})$ and $\tau_i \notin \mathcal{T}_1(U_{s_2})$. Clearly, if $\tau_i \in \mathcal{T}_1(U_{s_1})$ we have:

$$S_i \leq \frac{R_i}{U_{s_1}} \Rightarrow U_{s_1} \leq \frac{R_i}{S_i} \tag{3}$$

Also, if $\tau_i \notin \mathcal{T}_1(U_{s_2})$ we have:

$$S_i > \frac{R_i}{U_{s_2}} \Rightarrow U_{s_2} > \frac{R_i}{S_i} \tag{4}$$

From Equations 3 and 4, we have $U_{s_1} < U_{s_2}$ which contradicts our assumption that $U_{s_1} \geq U_{s_2}$. The same argument applies for $\mathcal{T}_1(U_{s_1}) \supseteq \mathcal{T}_1(U_{s_2})$. ∎

Algorithm 1 Utilization levels

1: $\mathcal{T}_1 = \{\tau_i | S_i <= R_i\}$;
2: $\mathcal{T}_2 = \{\tau_i | S_i > R_i\}$;
3: $\forall \tau_i \in \mathcal{T}_2$, order them according to $\frac{R_i}{S_i}$;
4: Initialize a vector $V[|\mathcal{T}_2| + 2]$;
5: $k \leftarrow 1$, $V_k \leftarrow 0$, $V_{size(V)} \leftarrow 1$;
6: **while** $\mathcal{T}_2 ! = \emptyset$ **do**
7: pick the task τ_j from $\mathcal{T}_2$ with the maximum $\frac{R_j}{S_j}$;
8: $k \leftarrow k + 1$;
9: $V_k \leftarrow \frac{R_j}{S_j}$;
10: $\mathcal{T}_2 \leftarrow \mathcal{T}_2 \setminus \{\tau_j\}$;
11: $\mathcal{T}_1 \leftarrow \mathcal{T}_1 \cup \{\tau_j\}$;
12: **end while**
13: $return\ \vec{V}$;

Algorithm 1 finds all possible utilization levels that may change the ordering of the tasks; i.e., the ordering of the tasks will never change if the utilization changes within the same interval (between two consecutive utilization levels). Starting from $U_s = 1$, we order the tasks according to Lemma 1 by offloading all of them, to construct the sets $\mathcal{T}_1$ and $\mathcal{T}_2$ (Lines 1 and 2). A vector $\vec{V}$ is initialized to store the utilization levels, including $U_s = 0$ and $U_s = 1$ (Lines 4 and 5). According to Theorem 1, all the combinations of $\mathcal{T}_1$ and $\mathcal{T}_2$ are known. The next combination happens by decreasing the utilization according to Theorem 2, which moves the task τ_i with the maximum $\frac{R_i}{S_i}$ from $\mathcal{T}_2$ to $\mathcal{T}_1$ (Lines 6-12). This task moves once its $S_i = \frac{R_i}{U_s}$, which implies that $U_s = \frac{R_i}{S_i}$ is the next utilization level.

Clearly, the time complexity of the ordering (Line 3) is $O(n \log n)$, and the while loop (Lines 6-12) is $O(n)$. So, the time complexity of the algorithm is $O(n \log n)$.

B. Minimum Utilization Interval

We determine the utilization intervals in subsection V-A, where the execution order does not change by changing the utilization within the same interval. Now, we want to find the utilization interval that contains the minimum utilization required from the server such that the schedule is feasible. We call it here the *minimum utilization interval*.

The function $F(\mathcal{T}, D, U_s)$, from our previous work in [22], is used in Algorithm 2 to check if there exists a feasible schedule. The algorithm returns NULL, if there is no feasible schedule under the highest possible utilization $U_s = 1$ (Lines 2-4). We use the binary search (Lines 5-12) to find the minimum utilization interval $(V_l, V_h]$. This interval is between two utilization levels V_i and V_{i+1}, such that there exists a feasible schedule under the utilization $U_s = V_{i+1}$, and there is no feasible schedule under the utilization $U_s = V_i$. The time complexity of the function $F(\mathcal{T}, D, U_s)$ is $O(nD^2)$, and the binary search is $O(\log n)$. Therefore, the total time complexity of Algorithm 2 is $O(D^2 n \log n)$.

VI. Hardness of the Offloading Problem

Theorem 3: The offloading problem is $\mathcal{NP}$-hard.

Algorithm 2 Minimum utilization interval

$l \leftarrow 1$, $h \leftarrow size(V)$
if $!F(\mathcal{T}, D, V_h)$ **then**
  $retrun$ NULL;
end if
while $h - l > 1$ **do**
  $i \leftarrow \lfloor \frac{h+l}{2} \rfloor$
  if $!F(\mathcal{T}, D, V_i)$ **then**
    $l \leftarrow i$;
  else
    $h \leftarrow i$;
  end if
end while
$return$ $(V_l, V_h]$;

Proof: The current offloading problem is an optimization problem to minimize the required utilization. The decision version of this problem is the same as the COTBS problem in our previous paper [22], which has been proved that it is an $\mathcal{NP}$-complete problem. ■

VII. Our Approach

The solution of our offloading problem is composed of two parts: the task ordering (from Section V) and the offloading decision. In this section, we propose two algorithms to determine the offloading decision.

Algorithm 2 is used to determine the minimum utilization interval $(V_l, V_h]$. Then, the binary search within this interval can be used to find the minimum required utilization in order to be feasible. The time complexity to check if there exists a feasible schedule under a given utilization is $O(nD^2)$, and the binary search complexity is $O(\log \frac{V_h - V_l}{\epsilon})$, where ϵ is the tolerance of the search. So, the total time complexity is $O(nD^2 \log \frac{V_h - V_l}{\epsilon})$. Our proposed algorithms in this section provide solutions that are close to the optimal, and faster than the binary search.

In our system, the client finds the minimum required utilization from the server that guarantees a feasible schedule. The following theorem shows that the feasibility is also guaranteed if the server provides the client with a utilization value that is more than the required.

Theorem 4: A feasible schedule based on the minimum required utilization remains feasible even if a higher utilization is provided from the server.

Proof: If the given utilization from the server does not affect the ordering (i.e., the ordering remains according to Lemma 1), the schedule remains feasible. If the order of the tasks changes after the given utilization, the makespan decreases due to the optimality of the ordering of Lemma 1, and then the schedule remains also feasible. In both cases the response time from the server decreases. ■

A. Greedy algorithm

We propose a greedy algorithm that requires only low complexity, where its time complexity is $O(n \log^2 n)$. The algorithm is faster than the binary search. Based on the offloading decision from this algorithm, and the ordering of the tasks according to Lemma 1, the next utilization level that achieves a feasible solution can be easily determined using the utilization levels from Algorithm 1. For notational brevity, we use the following notations in this subsection:

- $Y = \sum_{\tau_i \in \mathcal{T}} S_i$: the summation of the setup time for all of the tasks.
- $a_i = (C_i - S_i)$ for τ_i: the difference between the local execution time and the setup time.

We know that a schedule is feasible if the following two conditions hold:

1) the offloaded tasks finish their setup time and the local tasks finish their local execution time within the deadline, i.e., $Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i \leq D$, and
2) the last TBS deadline of the offloaded tasks, which is at least $\sum_{\tau_i \in \mathcal{T}} x_i \frac{R_i}{U_s}$, is less than or equal to the deadline.

Where x_i is equal to 1 if task τ_i is decided to be offloaded; otherwise, x_i is equal to 0. Our algorithm minimizes the value of $\sum_{\tau_i \in \mathcal{T}} x_i R_i$ without violating the first feasibility condition mentioned above (we will use the term *condition 1*), which can be represented by the following integer linear programming (ILP):

$$\text{minimize} \quad \sum_{\tau_i \in \mathcal{T}} x_i R_i \tag{5a}$$

$$\text{s.t } Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i \leq D \tag{5b}$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, 2, \ldots, n. \tag{5c}$$

The utilization $U_s = \frac{\sum_{\tau_i \in \mathcal{T}} x_i R_i}{D}$ is a lower bound of the minimum required utilization from the server. Here, we present a heuristic to decide whether a task is locally executed or offloaded by using the heuristic presented in Algorithm 3.

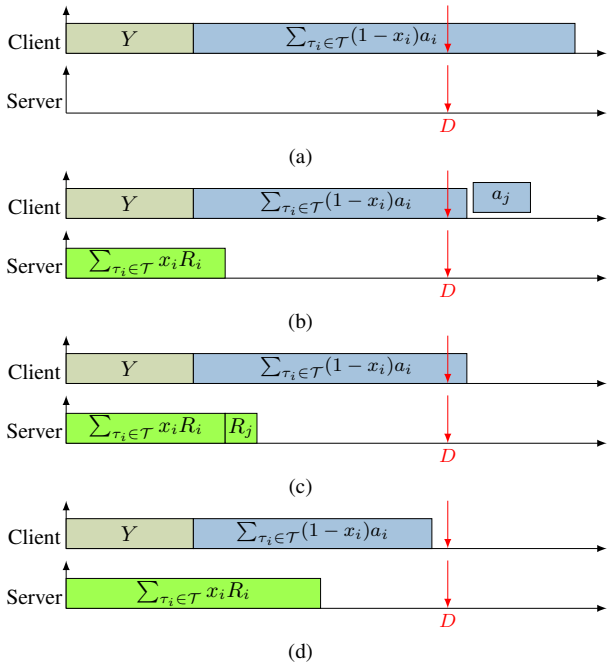


Fig. 3: Illustration of Algorithm 3.

Algorithm 3 Greedy Algorithm

1: $\forall \tau_i \in \mathcal{T}$, $x_i \leftarrow 0$, $Y = \sum_{\tau_i \in \mathcal{T}} S_i$, $a_i = (C_i - S_i)$;
2: $\forall \tau_i \in \mathcal{T} | S_i < C_i$, order them according to $\frac{R_i}{a_i}$ in a list $\mathcal{L}$;
3: **while** $(Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i) > D$ **do**
4: pick the task τ_j from $\mathcal{L}$ with the minimum $\frac{R_j}{a_j}$;
5: $x_j \leftarrow 1$;
6: $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\tau_j\}$;
7: **end while**
8: $min \leftarrow \sum_{\tau_i \in \mathcal{T}} x_i R_i$, $ind \leftarrow j$;
9: $x_j \leftarrow 0$;
10: **while** $\mathcal{L}! = \emptyset$ **do**
11: pick the task τ_j from $\mathcal{L}$ with the minimum $\frac{R_j}{a_j}$;
12: $x_j \leftarrow 1$;
13: **if** $(\sum_{\tau_i \in \mathcal{T}} x_i R_i < min) \wedge (Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i \leq D)$ **then**
14: $min \leftarrow \sum_{\tau_i \in \mathcal{T}} x_i R_i$, $ind \leftarrow j$;
15: **end if**
16: $x_j \leftarrow 0$;
17: $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\tau_j\}$;
18: **end while**
19: $x_{ind} \leftarrow 1$;
20: return $\frac{min}{D}$;

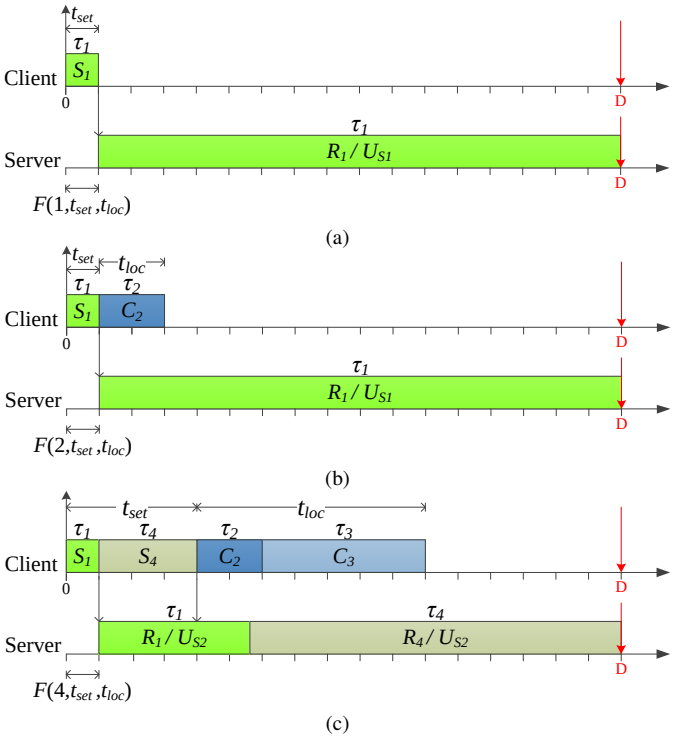


Fig. 4: Illustration of the dynamic programming algorithm.

The greedy algorithm, which is described in Algorithm 3, works as follows:

- At the beginning, all the tasks are assigned for local execution. Only the tasks that may be beneficial for offloading, their $S_i < C_i$, are ordered in the list $\mathcal{L}$ (Lines 1 and 2).
- As long as condition 1 is not satisfied (as represented in Figure 3a), the algorithm keeps picking the task τ_j with the maximum value of $\frac{R_j}{a_j}$ from the list $\mathcal{L}$ and assigns it for offloading, as illustrated in Figures 3b and 3c (Lines 3 and 7).
- The algorithm stores the index of the first task τ_j that makes the condition 1 satisfied as soon as it is assigned for offloading, and also stores the value of $\sum_{\tau_i \in \mathcal{T}} x_i R_i$ as a minimum possible value. Then, it is assigned for local execution without returning it back to the list $\mathcal{L}$ (Lines 8 and 9).
- For the remaining tasks in the list $\mathcal{L}$, the algorithm searches for a task τ_j, such that if it is assigned for offloading we gain the minimum possible $\sum_{\tau_i \in \mathcal{T}} x_i R_i$ value, without violating the condition1. The resulting value $\frac{min}{D}$ returned in Line 20 provides a lower bound of the utilization for this decision of x_is.

The time complexity for deciding whether a task τ_i is offloaded or not for all the tasks is $O(n \log n)$, which is dominated by the sorting of $\frac{R_j}{a_j}$. After the decision is done, we still have to find a suitable utilization to ensure the feasibility for this offloading decision. Here, we incrementally check the utilization levels defined in Section V-A, by considering the utilization levels that are larger than $\frac{min}{D}$. The algorithm stops when a certain utilization returns a feasible schedule. Verifying the feasibility of a schedule under a given utilization level requires $O(n \log n)$ time complexity, whereas the search of the minimum utilization levels defined in Section V-A for the offloading decision to be feasible takes $O(\log n)$ iterations. Therefore, the overall time complexity is $O(n \log^2 n)$.

B. Dynamic Programming Algorithm

Based on dynamic programming, we present a pseudo-polynomial-time scheduling algorithm to minimize the required utilization from the server, without violating the timing constraints. To construct the dynamic programming table, we need to order the tasks first. We consider the ordering of the tasks according to Lemma 1 when all of them are offloaded. We use Lemma 1 just for ordering. The offloading decision is determined using the dynamic programming algorithm.

For a given set of tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, consider the subproblem for the first i tasks, i.e., $\{\tau_1, \tau_2, \ldots, \tau_i\}$. Let $U(i, t_{set}, t_{loc})$ be the minimum required utilization from the server, such that the schedule of the tasks is feasible, under the following constraints:

- The total setup time for the offloaded tasks among the first i tasks is less than or equal to t_{set}.
- The total execution time for the local tasks among the first i tasks is less than or equal to t_{loc}.

Also, let $F(i, t_{set}, t_{loc})$ be the setup time of the first offloaded task under the same constraints above. Figure 4c shows an example for a schedule of four tasks to illustrate the dynamic programming parameters, where the tasks $\{\tau_1, \tau_4\}$ are offloaded, and the tasks $\{\tau_2, \tau_3\}$ are executed locally.

We construct two tables U and F, with three dimensions and size $n \times D \times D$. All the elements in $U(0, t_{set}, t_{loc})$ and $F(0, t_{set}, t_{loc})$ are initialized to zero. For $1 \leq i \leq n$, we fill

the table U according to the following recursion:

$$U(i, t_{set}, t_{loc}) = \begin{cases} \text{Equation 7} & t_{set} + t_{loc} \leq D. \\ \infty & \text{otherwise} \end{cases} \quad (6)$$

Where $0 \leq t_{set} \leq D$ and $0 \leq t_{loc} \leq D$.

For any given t_{set} and t_{loc}, such that $t_{set} + t_{loc} \leq D$, the algorithm determines the offloading decision for the task τ_i that achieves the minimum required utilization for the subproblem $\{\tau_1, \tau_2, \ldots, \tau_i\}$. If $t_{set} + t_{loc} > D$, there is no feasible solution and ∞ is stored in the table.

According to Equation 7, the task τ_i can be offloaded if it is *beneficial* (i.e., $S_i < C_i$), and there is enough setup time t_{set} (i.e., $t_{set} \geq S_i$). Also, it can be executed locally if there is enough local execution time t_{loc} (i.e., $t_{loc} \geq C_i$). If the task τ_i is assigned for local execution, the utilization remains the same as of the previous subproblem $U(i-1, t_{set}, t_{loc} - Ci)$. Also, the the value of $F(i, t_{set}, t_{loc})$ remains the same as $F(i-1, t_{set}, t_{loc} - C_i)$, because the setup time of the first offloaded tasks does not change by adding local tasks. Figure 4b shows an example where assigning the task τ_2 for local execution does not change the utilization U_{s_1} of the previous subproblem shown in Figure 4a (where t_{loc} is equal to 0).

If the task τ_i is assigned for offloading, the value of $F(i, t_{set}, t_{loc})$ also remains the same as $F(i-1, t_{set} - S_i, t_{loc})$. It only changes if τ_i is the first offloaded task, where $F(i-1, t_{set} - S_i, t_{loc}) = 0$. Then, we consider $F(i-1, t_{set} - S_i, t_{loc}) = S_i$, which leads the utilization to be equal to $\frac{R_i}{D - S_i}$. Therefore, the two parameters that change the system utilization by offloading τ_i (where it is not the first offloaded task) are: its remote execution time R_i and the time at which the server start executing it. So, the new value of the system utilization $U(i, t_{set}, t_{loc})$, in the case of offloading, is the maximum of the following:

- $U(i-1, t_{set} - S_i, t_{loc}) + \frac{R_i}{D - F(i-1, t_{set} - S_i, t_{loc})}$: where the utilization of the task τ_i in the period $D - F(i-1, t_{set} - S_i, t_{loc})$ is added to the system utilization of the previous subproblem.
- $\frac{R_i}{D - t_{set}}$: In this case, the t_{set} is too long such that the utilization above violates the system feasibility. Then, the utilization of task τ_i in the period $D - t_{set}$ restricts the system utilization to maintain the feasibility.

The algorithm stores the offloading decisions combined with all elements of the table U.

Finally, to minimize the required utilization, we find the minimum of $U(n, t_{set}, t_{loc})$ for all possible values of t_{set} and t_{loc}, such that their summation is less than or equal to D. Then, we backtrack the table starting from the location that achieves the minimum value above. If the task τ_i is assigned for offloading, we backtrack to $U(i-1, t_{set} - S_i, t_{loc})$. If it is assigned for local execution, we backtrack to $U(i-1, t_{set}, t_{loc} - C_i)$. The time complexity of the algorithm is $O(nD^2)$.

VIII. Experimental Evaluation and Simulation

In this section, we evaluate our algorithms using a case study of surveillance system, and synthesis workload simulation. The terms *Greedy* and *DP* refer to the greedy and the dynamic programming algorithms respectively. Also, the term *Search* is used to refer to the binary search for the minimum required utilization. The search is performed within the minimum utilization interval from Subsection V-B. The Search algorithm is used here as a baseline to evaluate the efficiency of the other algorithms.

TABLE I: Timing parameters of case study tasks (ms)

τ_i	Description	C_i	S_i	R_i
τ_1	Motion Detection	30	7	21
τ_2	Object Recognition	220	2	102
τ_3	Stereo Vision	88	16	41
τ_4	Motion Recording	18	7	14

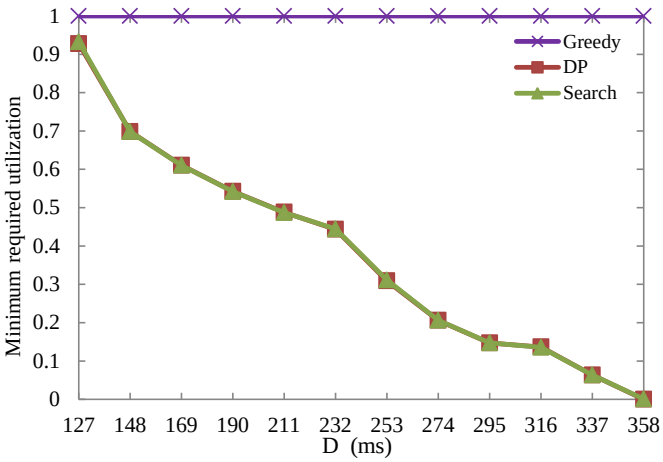


Fig. 5: Case study results.

A. Case Study of a Surveillance System

The surveillance system consists of a client and a server. The client performs four frame-based real-time tasks on the input video. The server processes the offloaded tasks from the client, and returns the results back. The tasks are independent of execution and can be described as follows:

- Motion Detection: to detect any moving objects.
- Object Recognition: to recognize and track the object of the interest.
- Stereo Vision: to construct a depth map of the view.
- Motion Recording: to record the detected video for further examination.

All the tasks can be executed either on the client or on the server. Table I shows the timing parameters of the tasks in milliseconds (ms). The algorithms are implemented on the client to find the schedule that minimizes the required utilization from the server without violating the real-time constraints.

Figure 5 shows the minimum required utilization using each algorithm for different deadlines. We observe that there exists a feasible solution for deadline $D \geq 127$, where the required utilization is close to 1 at $D = 127$. As the deadline increases, the required utilization decreases, because the possibility of executing more local tasks increases. For $D \geq 356$, the required utilization is equal to 0 because there is enough time to execute all of the tasks locally.

$$\min \left\{ \begin{array}{ll} \max\{U(i-1, t_{set} - S_i, t_{loc}) + \frac{R_i}{D-F(i-1,t_{set}-S_i,t_{loc})}, \frac{R_i}{D-t_{set}}\} & \mathrm{S}_i < C_i \wedge t_{set} \geq S_i \\ U(i-1, t_{set}, t_{loc} - Ci) & t_{loc} \geq C_i \end{array} \right\} \quad (7)$$

B. Simulation Setup and Results

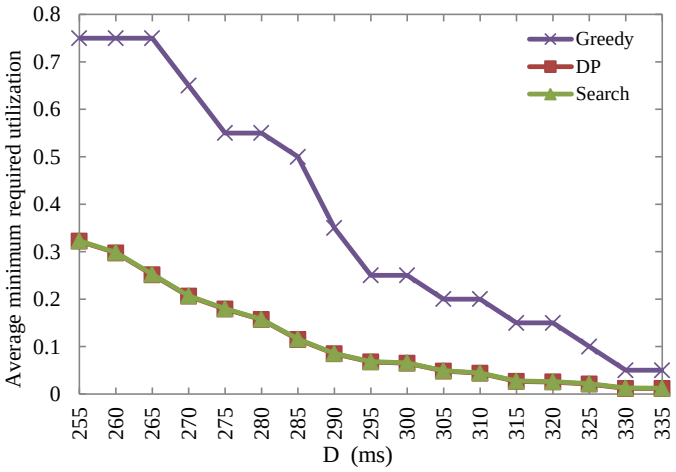
A synthetic workload is also used to evaluate our algorithms. Timing parameters of the tasks are generated as follows:

- C_i: Randomly generated integer values from 1 to 50 ms with uniform distribution.
- S_i: Randomly generated integer values from 1 to C_i ms with uniform distribution.
- R_i: $R_i = \frac{C_i}{\alpha}$, where α is the speed-up factor of the server.

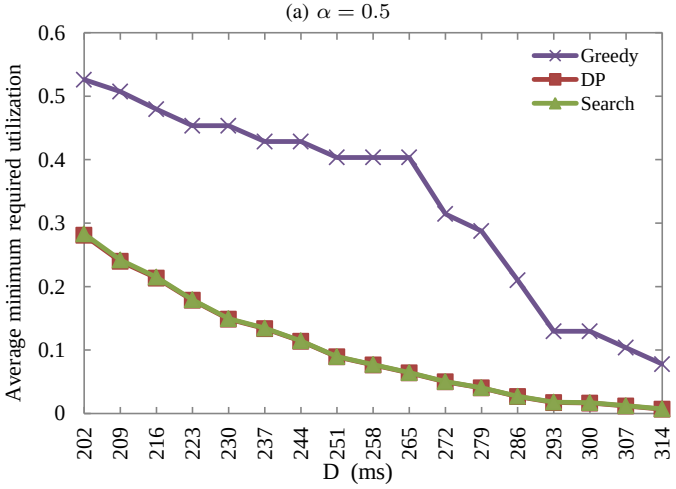
For each value of $\alpha = \{0.5, 2, 4\}$, we perform 20 rounds. In each round, a set of 10 frame-based real-time tasks is randomly generated and evaluated for different deadline values.

Figure 6 shows the average minimum required utilization for the generated tasks above. If the deadline increases, the client uses the additional time for local execution, which reduces the number of offloaded tasks and then the required utilization. Also, the speed-up factor of the server affects the required utilization. For a specific deadline (among Figures 6a, 6b and 6c), the utilization decreases if the speed-up factor of the server (α) increases, because the response time of the offloaded tasks becomes shorter. The figures also show that the dynamic programming and the search algorithms have nearly the same results.

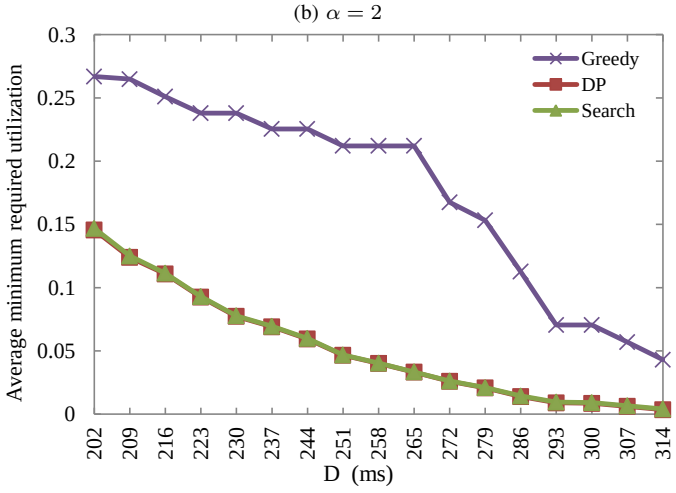
In Figure 6a, the server is two times slower than the client. Nevertheless, our system offloads tasks to the server. Because the local tasks are executed during the offloading of the other tasks, instead of remaining idle until the results return back from the server.



(a) $\alpha = 0.5$



(b) $\alpha = 2$



(c) $\alpha = 4$

Fig. 6: Minimum required utilization for all algorithms.

IX. Conclusion

In this paper, we explore the computation offloading to satisfy the real-time constraints in mobile devices. We adopt the total bandwidth server (TBS) on the server side for resource reservation. There are two challenges to perform computation offloading: which tasks to offload, and at what time each task should be offloaded. Therefore, the client uses the proposed algorithms to schedule frame-based real-time tasks and decide what to offload to the server, such that the required utilization from the server is minimized. The time complexity of the greedy algorithm is less than the time complexity of the dynamic programming algorithm, but our experimental evaluation and simulation show that the dynamic programming is more efficient in the term of minimizing the required utilization. We plan to extend our work to consider periodic/sporadic real-time tasks and other resource reservation servers.

Acknowledgement This work is supported in parts by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems", by Baden Württemberg MWK Juniorprofessoren-Programms and Deutscher Akademischer Austauschdienst (DAAD).

REFERENCES

[1] Wall-e robot. URL http://pixarplanet.com/blog/the-ultimate-walle-robot.

[2] Google glass project. URL http://www.google.com/glass/start/.

[3] R. Ballagas, J. Borchers, M. Rohs, and J. Sheridan. The smart phone: a ubiquitous input device. *Pervasive Computing, IEEE*, 5(1):70–77, 2006.

[4] J. Fernandez, D. Losada, and R. Sanz. Enhancing building security systems with autonomous robots. In *IEEE International Conference on Technologies for Practical Robot Applications*, pages 19 –24, 2008.

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

[6] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading inference for delivering applications in pervasive computing environments. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 107 – 114, 2003.

[7] S. Gurun, R. Wolski, C. Krintz, and D. Nurmi. On the efficacy of computation offloading decision-making strategies. *Int. J. High Perform. Comput. Appl.*, 22(4): 460–479, 2008.

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

[9] K. S. Hwang, K. J. Park, D. H. Kim, S.-S. Kim, and S. H. Park. Development of a mobile surveillance robot. In *International Conference on Control, Automation and Systems (ICCAS)*, pages 2503 –2508, 2007.

[10] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[11] K. Kim, S. Bae, and K. Huh. Intelligent surveillance and security robot systems. In *IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pages 70 –73, 2010.

[12] D. Kovachev, T. Yu, and R. Klamma. Adaptive computation offloading from mobile devices into the cloud. In *IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 784 –791, 2012.

[13] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava. A survey of computation offloading for mobile systems. *Mob. Netw. Appl.*, 18(1):129–140, Feb. 2013.

[14] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *International conference on Compilers, architecture, and synthesis for embedded systems (CASES)*, pages 238–246, 2001.

[15] 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.

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

[17] S. Ou, K. Yang, and A. Liotta. An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 10 –125, 2006.

[18] L. Pei, R. Chen, J. Liu, Z. Liu, H. Kuusniemi, Y. Chen, and L. Zhu. Sensor assisted 3d personal navigation on a smart phone in gps degraded environments. In *Geoinformatics, 2011 19th International Conference on*, pages 1–6, 2011.

[19] N. Ravi, P. Stern, N. Desai, and L. Iftode. Accessing ubiquitous services using smart phones. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 383–393, 2005.

[20] M. Spuri and G. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Real-Time Systems Symposium*, pages 2 –11, 1994.

[21] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10:179–210, 1996.

[22] A. Toma and J.-J. Chen. Computation offloading for frame-based real-time tasks with resource reservation servers. In *Euromicro Conference on Real-Time Systems (ECRTS)*, page to appear, 2013.

[23] A. Toma and J.-J. Chen. Computation offloading for real-time systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1650–1651. ACM, 2013.

[24] M. Valera and S. Velastin. Intelligent distributed surveillance systems: a review. *Vision, Image and Signal Processing, IEE Proceedings* -, 152(2):192 – 204, 2005.

[25] X. Wang. Intelligent multi-camera video surveillance: A review. *Pattern Recognition Letters*, 34(1):3 – 19, 2013.

[26] Y. Wang, K. Virrantaus, L. Pei, R. Chen, and Y. Chen. 3d personal navigation in smart phone using geocoded images. In *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, pages 584–589, 2012.

[27] 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.

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

[29] K. Yang, S. Ou, and H.-H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *IEEE Communications Magazine*, 46(1):56 –63, 2008.