Computation Offloading for Sporadic Real-Time Tasks

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

Jian-Jia Chen  
Department of Informatics  
TU Dortmund University, Germany  
jian-jia.chen@cs.uni-dortmund.de

Wei Liu  
Northeastern University, China  
liuwei-neu@hotmail.com

Abstract—The applications of the mobile devices are increasingly being improved. They include computation-intensive tasks, such as video and audio processing. However, the mobile devices have limited resources, which may make it difficult to finish these tasks in time. Computation offloading can be used to boost the capabilities of these resource-constrained devices, where the computation-intensive tasks are moved to a powerful remote processing unit. This paper considers the computation offloading problem for sporadic real-time tasks. The total bandwidth server (TBS) is adopted on the remote processing unit (the server side) for resource reservation. On the client side, a dynamic programming algorithm is proposed to determine the offloading decision of the tasks such that their schedule is feasible (i.e., all the tasks meet their deadlines). The algorithm is evaluated using a case study of surveillance system and synthesized benchmarks.

I. INTRODUCTION

With the recent advances in mobile and wireless technologies, the mobile computing devices have become very important in our daily life. They include smart phones, mobile robots and wearable computers. For example, the smart phones are used nowadays for different purposes, such as Internet access and multimedia applications, in addition to the traditional phone calls [6, 33]. Also, the mobile robots are increasingly used for surveillance [14, 32], security [17] and cleaning [5, 15]. Furthermore, the wearable computers such as Google glass [2] and the smart watches are the next-generation ubiquitous technologies. These mobile devices run multiple tasks simultaneously, which include voice and image recognition, navigation, video processing, etc. For instance, the mobile robots that used for exploration collect the data from different sensors and process it periodically, in order to analyze the surrounding environment. Also, they process the captured images periodically to perform object recognition and motion planning. In addition, the robot may perform further tasks such as self control of different components and analysis of stereo vision. These tasks are executed periodically and must finish within specific time, i.e., the deadline.

However, the mobile devices have limited resources such as computation capabilities, memory capacity and battery life. Therefore, they may not be able to finish the execution of all the tasks in time, specially the computation-intensive tasks. In this case, the results may become useless or even harmful to the system if the deadlines are missed. One solution is to use the computation offloading, where the mobile device (i.e., the resource-constrained device) offloads the computation-intensive tasks to a powerful remote processing unit. The remote processing unit executes the offloaded tasks and returns the results back to the mobile device. Figure 1 illustrates the computation offloading mechanism, where the blue tasks are offloaded to the remote processing unit. We denote the mobile device as the client and the remote processing unit as the server.

Several studies have adopted the computation offloading technique. The offloading decision in [13, 25, 34] is based on the comparison between the time consumed during the local execution and the time consumed during the offloading for each task. The approaches in [12, 20, 21, 26, 36] use the graph partitioning method to solve the computation offloading problem. The approaches above focus on the offloading decision without considering task scheduling or the server model. Also, most of them either do not consider the timing satisfaction requirement for real-time properties, or use pessimistic offloading mechanism for deciding whether a task can be offloaded or not [19]. In our previous study in [29], the total bandwidth server (TBS) [27, 28] is used on the server side to provide resource reservation for the offloaded tasks. The server assigns a TBS with a specific utilization (i.e., bandwidth) for each client. On the client side, two algorithms are proposed to minimize the finishing time of the tasks (i.e., makespan) based on the given utilization. But for some clients, the tasks may be scheduled using less utilization than the given from the server, without violating the real-time constraints. Therefore, in our work in [30] the client finds a feasible schedule for the tasks such that the required utilization from the server is minimized, in order to avoid wasting the resources of the server. The tasks
in our two approaches above are frame-based real-time tasks, where all of the tasks have the same arrival time, relative deadline and period.

In this paper, we adopt the idea of computation offloading to schedule the tasks in the resource-constraints mobile devices such that all of them meet their real-time constraints (i.e., their deadlines). We consider the sporadic real-time tasks, which also include the periodic tasks, where each task consists of an infinite sequence of identical instances (called jobs) separated by at least $T_i$ period of time (i.e., the minimum inter-arrival time). Each task should be executed within its relative deadline $D_i$. To perform computation offloading in real-time systems, the server should provide response time guarantee for the offloaded tasks to be executed and returned back to the client before their deadlines. Therefore, we use the total bandwidth server on the server side. Two decision-making points should be addressed in this paper: (1) task scheduling and (2) offloading decision.

**Our Contribution:** Our contribution can be summarized as follows:

- In our model, the server can serve more than one client and provides response time guarantee to the offloaded tasks.
- We present schedulability test analysis for sporadic real-time tasks that can be executed locally or offloaded.
- We propose a computation offloading algorithm based on dynamic programming. The algorithm schedules the sporadic real-time tasks and decides which of them to be offloaded, based on the given utilization from the server, such that the real-time constraints are satisfied.
- We evaluate our algorithm using a case study of surveillance system and synthesized benchmarks.

**II. Literature Review**

Different techniques have been proposed to perform computation offloading in order to improve performance [12, 13, 16, 18, 26, 34, 36], save energy [16, 18, 20, 21, 35] and satisfy real-time requirements [11, 25].

The computation offloading is adopted in [13, 34] to improve the performance for computational grid settings. The system predicts the local execution time, the remote execution time and the transmission time of the tasks. Then, the offloading problem is represented as a statistical decision problem. The task is considered beneficial for offloading if the expected cost of the remote execution is less than the expected cost of the local execution.

Without loss of generality, the main idea in [12, 20, 21, 26, 36] is to represent the computation offloading problem as a graph partitioning problem, where the first partition represents the client side and the other one represents the server side. Each vertex in the graph is combined with a cost and represents a task in a program as in [20, 21], or a computational component as in [12, 26, 36]. The edges between the vertices are combined with the communication costs. The costs of the vertices and the edges could be either energy costs, performance costs or a combination between them. Different algorithms are proposed to partition the graph into two parts in order to minimize the communication cost or the total cost on one side.

Khairy et al. [16] propose a “Smartphone Energizer” technique for context-aware computation offloading in order to extend the battery life of the smart phone. The proposed technique predicts the energy consumption and the execution time costs of a computation on both client and server sides, and combines them into one cost. The computation is considered to be beneficial for offloading if the expected combined cost on the server is less than the one on the client. The prediction is performed based on supervised learning with the contextual information such as network, service, device and user characteristics.

The offloading decision in [18] is represented as an optimization problem based on different parameters such as CPU load, available memory, remaining battery, and the bandwidth. The Integer linear Programming (ILP) is used to solve the problem on the mobile device. Then, the computation-intensive tasks are offloaded to a remote cloud.

Nimmagadda et al. [25] propose an offloading framework for mobile robots to perform the tasks of object recognition and tracking without violating the real-time constraints. A task is assigned for offloading, if the summation of its expected remote execution time on the server and its data transfer time is less than its local execution time on the robot. Ferreira et al. [11] explore the computation offloading to improve the quality of service in the adaptive real-time systems. The proposed mechanism offloads the services from the smart phone to several surrogate nodes.

In most of the current studies, the offloading decision is either based on: (1) the comparison between the cost of the local execution and the cost of the remote execution for each task alone, or (2) the partitioning of the graph that represents the tasks combined with their local and remote execution costs [19]. The cost may include the execution time, the energy consumption, the memory usage, etc. The first approach may not be optimal if we consider all of the system tasks together. Both approaches do not consider the scheduling of the tasks. Changing the order of the task execution, if it is possible, may improve the performance of the system. Also, they do not consider the server model, or how it handles and executes the offloaded tasks from more than one client. According to the existing approaches, the 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 and Problem Definition**

In this section, we present our client-server system model, and the problem definition.

**A. Client and Task Model**

Given a set $\mathcal{T}$ of $n$ independent sporadic real-time tasks. A task $\tau_i \in \mathcal{T}$ (for $i = 1, 2, \ldots, n$) represents an execution unit, and consists of an infinite sequence of identical instances, called jobs [8, 24]. Each task is characterized by the following timing parameters:

- $C_i$: Local execution time on the client side.
- $R_i$: Remote execution time. The execution time on the server side.
- $D_i$: Relative deadline.
the setting up of the task \(\tau\) within the server. In the case of local execution, the task should be executed offloading relative deadline can be expressed as \(D\) if the task \(\tau\) is executed locally or offloaded. Suppose that \(x_i\) is equal to 1 if the task \(\tau_i\) is chosen for offloading; otherwise, \(x_i\) is equal to 0. We use the vector \(\vec{x}_n = (x_1, x_2, \ldots, x_n)\) to denote an offloading decision vector of the tasks.

The relative deadline \(D_i\) of a task \(\tau_i\) can be expressed as follows:

\[
D_i = x_i D_i^o + (1 - x_i) D_i^l.
\]

Where, the setting up of the task \(\tau_i\) should be finished within the offloading relative deadline \(D_i^o\) in the case of offloading. In the case of local execution, the task should be executed within the local relative deadline \(D_i^l\). As the result of the offloaded task returns after at most \(I_i\) amount of time, the offloading relative deadline can be expressed as \(D_i^o = D_i^l - I_i\). On the client side, we consider the sporadic real-time task model with implicit local relative deadlines, where the local relative deadline is equal to the minimum inter-arrival time, i.e., \(D_i^l = T_i\). This model also includes the periodic tasks, in which the jobs of the same task are activated at a constant rate. We assume that the returned result from the server needs very short post processing time, which is negligible. Figure 2 shows the timing parameters for an example of two tasks, where the first task (in blue) is executed locally and the second one (in green) is offloaded to the server.

### B. Server Model

In our system model, the server is able to serve more than one client. In this case, the server should provide a certain resource reservation for each client in order to guarantee the response time of the offloaded tasks. The Total Bandwidth Server (TBS) [27, 28] is used as a resource reservation server to manage the sharing of the server processor, and then preserve the real-time property of the system.

The server assigns a TBS for each requesting client with a specific utilization (or bandwidth) \(U_s\), if it is possible. The total given utilization for all clients should be less than or equal to 100\%, in order to preserve the system feasibility. For the client, the speed of the given TBS seems \(\frac{1}{x}\) times slower than the speed of the server.

### C. Problem Definition

Given a set \(T\) of \(n\) sporadic real-time tasks. A schedule of the tasks is said to be feasible if the timing constraints of all the tasks are satisfied. As the resources of the client are limited, it may not be able to schedule the tasks without violating the timing constraints. Therefore, the client may offload some of the tasks to the server for faster execution. A task \(\tau_i\) can be offloaded if it is beneficial for offloading (i.e., \(S_i < C_i\)), and its result returns back from the server before the deadline. The problem is to find a schedule and an offloading decision such that all the tasks meet their real-time constraints.

### D. Hardness of the Problem

The problem in this paper is similar to the problem in our previous work in [31], but with a general task model. A special case of the sporadic real-time model, called frame-based real-time task model, is considered in [31]. In the frame-based real-time task model, all the tasks have the same arrival time, relative deadline and period. It has been shown in [31] that this offloading problem is \(\mathcal{NP}\)-complete even for the special case of the general task model.

**Theorem 1:** The computation offloading problem for sporadic real-time tasks is \(\mathcal{NP}\)-hard problem.

**Proof:** The current offloading problem is a general case of the offloading problem in [31], which has been proved that it is an \(\mathcal{NP}\)-complete problem.

### IV. Task Scheduling and Feasibility Test

In this section, we present how the tasks are scheduled and how the feasibility of a schedule can be verified. For the task scheduling, this paper considers the Earliest Deadline First (EDF) algorithm, which is a preemptive scheduling algorithm with a dynamic priority policy. It is an optimal algorithm for dynamic-priority scheduling on preemptive uniprocessors [22]. According to EDF, the job with the earliest absolute deadline, among of the ready jobs, is assigned with the highest priority [23].

The processor demand analysis is used to verify the feasibility of a schedule under EDF [7]. The demand bound function \(\text{dbf}(\tau_i, t)\) of a task \(\tau_i\) within the time interval of length \(t\) can be defined as follows:

\[
\text{dbf}(\tau_i, t) = \max \left\{ 0, \left\lceil \frac{t - D_i}{I_i} \right\rceil + 1 \right\} \times C_i.
\]

The task set \(T = \{\tau_1, \tau_2, \ldots, \tau_n\}\) can be feasibly scheduled by EDF if and only if the total demand of all the tasks
within any interval of time $t$ is no greater than the available processing time \[7\]; that is, if and only if

$$\forall t > 0, \sum_{i=1}^{n} dbf(\tau_i, t) \leq t.$$  \hspace{1cm} (3)

Several approximations on the demand bound function have been proposed by Chakraborty et al. \[9\], and Albers and Slomka \[3, 4\] to reduce the time complexity of the feasibility analysis. The main idea of the approximation algorithms is to limit the number of test points by considering only a constant number of points for each task, and then use the linear approximation for the rest of the test interval. According to the approximation in \[3\], the approximate demand bound function can be defined as follows:

$$dbf^*(\tau_i, t) = \begin{cases} 0 & \text{if } t < D_i \\ \frac{(t-D_i)}{D_i} + 1 & \text{otherwise.} \end{cases}$$ \hspace{1cm} (4)

It represents an upper bound of the Equation (2). In this case, there exists a feasible schedule if:

$$\forall t > 0, \sum_{i=1}^{n} dbf^*(\tau_i, t) \leq t.$$

$$
\begin{align*}
\sum_{j=1}^{n} u_j &= \sum_{j=1}^{n} \frac{C_j}{T_j} 
\leq 1 & \text{if } \tau_i \text{ is executed locally,} \\
\sum_{j=1}^{n} \sum_{i} dbf^*(\tau_j, D_i) &= D_i & \text{if } \tau_i \text{ is offloaded.} 
\end{align*}
$$ \hspace{1cm} (6a)  

\hspace{1cm} (6b)

where $u_j$ is the utilization of task $\tau_j$.

The analysis in \[3, 4\] shows that the above schedulability test in (6) has a 2 resource augmentation factor. Moreover, the recent result in \[10\] gives a tighter analysis to show that the test by (6) gives a 1.6322 resource augmentation factor. If a given algorithm has an $\epsilon$ resource augmentation factor to solve a scheduling problem, it guarantees that the solution (i.e., the derived schedule) is feasible on a processor by speeding it up to $\epsilon$ times as fast as the original speed, provided that there exists a feasible schedule for the original speed. If the condition in (6b) does not hold, we can not say there is no feasible solution. According the the resource augmentation technique, if the algorithm fails to find a solution, there is no feasible schedule by slowing down the processor to $\frac{1}{\epsilon}$ of the original speed.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig3.png}
\caption{Illustration for the demand bound function and its approximation.}
\end{figure}

\section{Our Approach}

In our offloading problem, two decision-making points should be considered: the offloading decision $x_o$ and the task scheduling. EDF is used to schedule the tasks as shown in Section IV. Subsection V-A defines the feasibility test for our system model. The offloading decision of the tasks $x_o$ is determined using the dynamic programming algorithm in Subsection V-B. In Subsection V-C, an iterative algorithm is proposed to estimate the value of $t_i$.

\subsection{Feasibility Test and Analysis}

The conditions in (6) can not be used to test the schedulability of the tasks in our problem. Because the task in our model either is executed locally or offloaded. In the case of offloading, the task is executed $S_i$ amount of time on the client instead of $C_i$ amount of time in the case of local execution. Also, the relative deadline becomes $D_i^o$ instead of $D_i$. Therefore, we want to define our own schedulability (or feasibility) test, based on the conditions in (6), which is applicable for our problem. The utilization, the demand bound function and the approximate demand bound function for a task $\tau_i$ in our model are defined respectively as follows:

- $\tau_i$ is executed locally:

\begin{align*}
u_i^l &= \frac{C_i}{T_i}, \\
dbf_l(\tau_i, t) &= \max \left\{ 0, \frac{t}{T_i} \right\} \times C_i,
\end{align*}

- $\tau_i$ is offloaded:

\begin{align*}
u_i^o &= \frac{S_i}{T_i}, \\
dbf_o(\tau_i, t) &= \max \left\{ 0, \frac{t-D_i^o}{T_i} + 1 \right\} \times S_i,
\end{align*}

- $\tau_i$ is offloaded:

\begin{align*}
u_i^o &= \frac{S_i}{T_i}, \\
dbf_o^*(\tau_i, t) &= \begin{cases} 0 & \text{if } t < D_i^o \\ \frac{(t-D_i^o)}{D_i^o} + 1 & \text{otherwise.} \end{cases}
\end{align*}

Figure 3 illustrates the demand bound function (in black) and its approximation (dashed red line) for the task $\tau_i$ in the case of local execution (Figure 3a), and in the case of
offloading (Figure 3b). It also shows the available processing time \( t \).

According to our task model, the conditions in (6) can be expressed as follows:

\[
\sum_{j=1}^{n} u_j = \sum_{j=1}^{n} x_j u^f_j + \sum_{j=1}^{n} (1 - x_j) u^l_j \\
= \sum_{j=1}^{n} x_j \frac{S_j}{T_j} \sum_{j=1}^{n} (1 - x_j) \frac{C_j}{T_j} \leq 1 \tag{9a}
\]

\[
\forall \tau_i \in \mathcal{T}, \sum_{j=1}^{i} \text{dbf}^* (\tau_j, D_i) \\
= \sum_{j=1}^{i} x_j \cdot \text{dbf}^* (\tau_j, D_i) + \sum_{j=1}^{i} (1 - x_j) \cdot \text{dbf}^*_l (\tau_j, D_i) \\
= \sum_{j=1}^{i} x_j \left( \frac{D_i - D_j^o}{T_j} + 1 \right) S_j + \sum_{j=1}^{i} (1 - x_j) D_i \frac{C_j}{T_j} \leq D_i. \tag{9b}
\]

Recall that \( x_i \) is equal to 1 if the task \( \tau_i \) is chosen for offloading; otherwise, \( x_i \) is equal to 0. Also, the relative deadline of the task \( \tau_i \) is represented as: \( D_i = x_i D_j^o + (1 - x_i) D_j^l \). A very safe upper bound to approximate the condition in (9b) is to discard the value of \( D_j^j \) (or replace \( \frac{D_i - D_j^o}{T_j} \) with \( \frac{T_j}{T_j} \)) As a result we have:

\[
\sum_{j=1}^{i} x_j \left( \frac{D_i - D_j^o}{T_j} + 1 \right) S_j + \sum_{j=1}^{i} (1 - x_j) D_i \frac{C_j}{T_j} \\
\leq \sum_{j=1}^{i} x_j \left( \frac{D_i}{T_j} + 1 \right) S_j + \sum_{j=1}^{i} (1 - x_j) D_i \frac{C_j}{T_j} \\
= D_i \left( \frac{\sum_{j=1}^{i} x_j S_j}{D_i} \right) + \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \tag{10}
\]

Therefore, based on the condition in (9b) and the overapproximation in (10), the following condition in Lemma 1 can be used to test the feasibility of a schedule in our problem under EDF.

**Lemma 1:** For a given offloading decision vector \( \hat{x}_n \), if

\[
\forall \tau_i \in \mathcal{T}, \frac{\sum_{j=1}^{i} x_j S_j}{D_i} + \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \leq 1, \tag{11}
\]

then the tasks can be feasibly scheduled by EDF, where \( D_i \leq D_{i+1} \).

**Proof:** Suppose that the if part of the lemma is true, then we have:

\[
\forall \tau_i \in \mathcal{T}, \frac{\sum_{j=1}^{i} x_j S_j}{D_i} + \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \leq 1, \tag{11}
\]

Now, we want to show that the feasibility condition in (3) is also true. For \( D_i \leq t < D_{i+1} \), we have:

\[
\sum_{j=1}^{n} \text{dbf}(\tau_j, t) \\
= \sum_{j=1}^{n} x_j \cdot \text{dbf}_o (\tau_j, t) + \sum_{j=1}^{n} (1 - x_j) \cdot \text{dbf}_l (\tau_j, t) \\
\leq \sum_{j=1}^{n} x_j \cdot \text{dbf}_o (\tau_j, t) + \sum_{j=1}^{n} (1 - x_j) \cdot \text{dbf}_l^* (\tau_j, t) \\
= \sum_{j=1}^{n} x_j \cdot \text{dbf}_o (\tau_j, t) + \sum_{j=1}^{n} (1 - x_j) \cdot \text{dbf}_l^* (\tau_j, t),
\]

because \( \text{dbf}_o (\tau_j, t) \) and \( \text{dbf}_l^* (\tau_j, t) \) are equal to 0 for all \( t < D_j^o \) and \( t < D_j^l \) respectively.

\[
\left( \sum_{j=1}^{i} x_j S_j + \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \right) \leq t,
\]

\[
\left( \sum_{j=1}^{i} x_j S_j + \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \right) \leq 1
\]

for \( D_i \leq t < D_{i+1} \), see Equation (11).

The feasibility condition in Lemma 1 can be expressed as follows:

\[
\forall \tau_i \in \mathcal{T}, \frac{\sum_{j=1}^{i} x_j S_j}{D_i} + \left( \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \right) \leq 1, \tag{12}
\]

which also includes the utilization condition in (9a). The following lemma shows that the approximation in (12) has a resource augmentation factor equals to 2.

**Lemma 2:** For the task \( \tau_i \), if

\[
\sum_{j=1}^{i} x_j S_j \frac{D_i}{D_i} + \left( \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \right) > 1
\]

then there is no feasible schedule for the tasks \( \{ \tau_1, \tau_2, \ldots, \tau_i \} \) if the system is slowed down to 0.5 of the original speed.

**Proof:** If

\[
\sum_{j=1}^{i} x_j S_j \frac{D_i}{D_i} + \left( \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \right) > 1
\]

then we have two possibilities:

1) \( \sum_{j=1}^{i} x_j S_j \frac{D_i}{D_i} > 0.5 \), then \( \sum_{j=1}^{i} x_j S_j > 0.5 D_i \). The infeasibility by slowing down can be verified as follows: \( \sum_{j=1}^{i} x_j \cdot \text{dbf}_o (\tau_j, D_i) \geq \sum_{j=1}^{i} x_j \cdot \text{dbf}_o (\tau_j, D_i) \geq \sum_{j=1}^{i} x_j S_j > 0.5 D_i \).

2) \( \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} > 0.5 \). By slowing down to 0.5 of the original speed, the utilization becomes: \( \sum_{j=1}^{i} x_j \frac{2 S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} = 2 \left( \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \right) > 1 \).

\]
B. Dynamic Programming Algorithm

The feasibility condition in Lemma 1 can be verified for a given offloading decision vector $\bar{x}_n$. One solution to determine the offloading decision $\bar{x}_n$ is to find all the combinations for the offloading decisions of the tasks, and then test the feasibility. However, this method needs exponential execution time. Therefore, we propose a dynamic programming algorithm to determine the offloading decisions of the tasks, and to test the feasibility of the schedule at the same time. The tasks are ordered according to the relative deadline $D_i$, i.e., $D_i \leq D_j$ if $i < j$, to build the dynamic programming table. The tasks that are not beneficial for offloading ($S_i \geq C_i$) or cannot be offloaded ($D_i^o < S_i$), are assigned and fixed for local execution with $D_i = D_i^o$. For the other tasks, i.e., that can be offloaded or executed locally, we consider at the beginning that $D_i = D_i^o$, which is used for ordering of the tasks and testing. If the dynamic programming algorithm finds a feasible schedule, the obtained offloading decisions are used to determine the relative deadlines of the tasks according to (1). And then, EDF is used to schedule the tasks according to these deadlines.

Consider the sub-problem for the first $i$ tasks $\{\tau_1, \tau_2, \ldots, \tau_i\}$. Recall the feasibility condition in (12), let $\sum_{j=1}^{i} x_j S_j / T_j$ be the effective density for the first $i$ tasks at time $D_i$, and $(\sum_{j=1}^{i} x_j S_j / T_j + \sum_{j=1}^{i} (1 - x_j) C_j / T_j)$ be the effective utilization for the first $i$ tasks. Suppose that $L(i, \delta)$ is the minimum effective utilization for the first $i$ tasks, such that their effective density at time $D_i$ is less than or equal to $\delta$. A two-dimensional dynamic programming table $L(i, \delta)$ is constructed for all possible values of $i$ and $\delta$, such that $0 \leq i \leq n$ and $0 \leq \delta \leq 1$. Where, all the possible values of $\delta$ are considered as the integer multiples of $\rho$ (i.e., $\rho$ is a user-specified granularity), and $\frac{\rho}{\rho}$ is considered as an integer number. We start by initializing all the elements of $L(0, \delta)$ to zeros. Then, the following recursion is used to fill the table for $i$ from 1 to $n$.

$$L(i, \delta) = \min \left\{ \begin{array}{ll} L(i - 1, \frac{\delta D_i - S_i}{D_i - 1}) + \frac{S_i}{T_i} & \text{if } S_i < C_i \land \delta \geq \frac{S_i}{C_i} \\ \infty & \text{otherwise} \end{array} \right. \tag{13}$$

where $\frac{\delta D_i - S_i}{D_i - 1}$ and $\frac{\delta D_i}{D_i - 1}$ are the effective density for the first $i - 1$ tasks at time $D_{i-1}$ if the task $\tau_i$ is assigned for offloading and if it is assigned for local execution respectively.

Lemma 3: For given $i$ and $\delta$, the recursive function defined in 13 computes the optimal solution for $L(i, \delta)$.

Proof: This Lemma can be proved by induction. For the base case ($i = 1$) if the task can be offloaded, the function stores the minimum between the offloading case $\frac{S_i}{T_i}$ and the local case $\frac{C_i}{T_i}$ which is optimal.

Inductive step: Assume that $L(i - 1, \delta)$ is optimal for the subproblem of the first $i - 1$ tasks with $i \geq 2$ and any given $0 \leq \delta \leq 1$. Let $x_{i-1}^*$ be the optimal offloading decision for $\{x_1, x_2, \ldots, x_{i-1}\}$ when the effective density of the first $i - 1$ tasks is no more than $\frac{\delta D_i - S_i}{D_i - 1}$ (i.e., when task $\tau_i$ is considered for offloading). Similarly, let $x_{i-1}^l$ be the optimal offloading decision for $\{x_1, x_2, \ldots, x_{i-1}\}$ when the effective density of the first $i - 1$ tasks is no more than $\frac{\delta D_i}{D_i - 1}$ (i.e., when task $\tau_i$ is considered for local execution).

Suppose for contradiction that $x_i^* \neq x_i^l$. Then, the following recursion is used to fill the table $L$ (in 13) computes the optimal solution for $L(i, \delta)$. But $L(i, \delta)$ would be equal to $L(i - 1, \delta)$.

Finally, we check if there exists a feasible schedule using the following theorem.

Theorem 2: There exists a feasible schedule under EDF for our offloading problem if the minimum of $L(n, \delta)$, for $0 \leq \delta \leq 1$, is less than or equal to 1. Otherwise, there is no feasible schedule by slowing down to 0.5 of the original speed.

Proof: This comes from Lemma 1 and the sub-optimality property of the dynamic programming scheme shown in Lemma 3. The infeasibility by slowing down the original platform comes from Lemma 2.

If we have a feasible schedule, we backtrack the dynamic programming table to obtain the offloading decision for each task $\tau_i$, starting from the solution found, as follows: (1) If $\tau_i$ is assigned for local execution, we backtrack to $L(i - 1, \frac{\delta D_i}{D_i - 1})$. If it is assigned for offloading, we backtrack to $L(i - 1, \frac{\delta D_i - S_i}{D_i - 1})$.

The time complexity of the dynamic programming algorithm is $O(n \log n + n^2 \rho)$.

C. Estimating the Value of $I_i$

Based on the given $I_i$ values, the dynamic programming algorithm in Subsection V-B determines the tasks that should

It comes from the construction of $L(i, \delta)$ in (13).
be offloaded to have a feasible schedule. However, the value of $I_i$ depends on the number of offloaded tasks. Because if the number of offloaded tasks increases, the server needs longer time to finish them, which affects the value of $I_i$. The value of $I_i$ also depends on the given utilization from the server $U_s$. Fortunately, this value ($U_s$) is predefined and fixed. So, it is difficult to calculate the exact value of $I_i$ without knowing the number of the offloaded tasks to the server. Therefore, we present in this subsection an iterative algorithm, which is described in Algorithm 1, to estimate the value of $I_i$.

The main idea of the algorithm is to nominate a set of tasks $T_o$ for offloading, i.e., predict the number of offloaded tasks, based on the heuristic value $\frac{C_i-S_i}{R_i}$. Then, the $I_i$ value is calculated just for the candidate tasks in the set $T_o$. For the other tasks, the $I_i$ value is assigned to infinity. The tasks with $S_i \geq C_i$ are not beneficial for offloading. Thus, Algorithm 1 assigns all of them for local execution by setting their $I_i$ value to infinity. All other tasks, that are beneficial for offloading, are added to the list $L$ (Lines 1 and 2). As long as the list $L$ is not empty, the algorithm keeps doing the following steps:

1. Pick the task $\tau_j$ with the maximum heuristic value $\frac{C_i-S_i}{R_i}$ from the list $L$ (i.e., beneficial for offloading) and nominate it for offloading. Using this heuristic, the algorithm tries to decrease the load of the client as much as possible and to increase the load of the server as less as possible (Lines 5 - 2).
2. Calculate the value of $I_i$ for the candidate tasks. According to the system model, the server assigns a TBS with a specific utilization $U_s$ to the client. To be feasible, the algorithm on the client assigns for each task $\tau_i \in T_o$ a TBS, with a utilization $U_i$ such that their total utilization is equal to the given utilization from the server $U_s$, i.e., $\sum_{\tau_i \in T_o} U_i = U_s$ (Line 8).
3. The dynamic programming algorithm is used to find a feasible schedule. If there exists one, Algorithm 1 returns the offloading decision $x$ (Lines 9 - 11).

If the list $L$ becomes empty before finding a feasible schedule, then our algorithm fails to find a feasible schedule.

---

**Algorithm 1 Estimating the value of $I_i$**

1. $\forall \tau_i \in T, I_i \leftarrow \infty$;
2. $\forall \tau_i \in T | S_i < C_i$, Order them according to $\frac{C_i-S_i}{R_i}$ in the list $L$;
3. $T_o \leftarrow \emptyset$;
4. while $L \neq \emptyset$ do
5. Pick the task $\tau_j$ from $L$ with the maximum $\frac{C_i-S_i}{R_i}$;
6. $T_o \leftarrow T_o \cup \{\tau_j\}$;
7. $L \leftarrow L \setminus \{\tau_j\}$;
8. $\forall \tau_i \in T_o, U_i \leftarrow \frac{U_s}{TBS_s}$, $I_i \leftarrow \frac{R_i}{L}$;
9. Run the dynamic programming algorithm from Subsection V-B;
10. if There exists a feasible schedule then
11. Return $x$;
12. end if
13. end while
14. return “Fails to find a feasible schedule”;

---

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

<table>
<thead>
<tr>
<th>Task</th>
<th>$C_i$</th>
<th>$S_i$</th>
<th>$R_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$ Motion Detection</td>
<td>30</td>
<td>7</td>
<td>21</td>
<td>115</td>
</tr>
<tr>
<td>$T_2$ Object Recognition</td>
<td>220</td>
<td>2</td>
<td>102</td>
<td>418</td>
</tr>
<tr>
<td>$T_3$ Stereo Vision</td>
<td>88</td>
<td>16</td>
<td>41</td>
<td>695</td>
</tr>
<tr>
<td>$T_4$ Motion Recording</td>
<td>18</td>
<td>7</td>
<td>14</td>
<td>63</td>
</tr>
</tbody>
</table>

---

VI. EXPERIMENTAL EVALUATION AND SIMULATION

In this section, we evaluate our algorithm by implementing a case study of a surveillance system, and synthesis workload simulation. We use the abbreviation $DP$ to refer to the dynamic programming algorithm. The term Simple-Offload is used to refer to the algorithm in which the offloading decision is taken based on the comparison between the data transfer time added to the round-trip offloading time, and the local execution time, i.e., a task $\tau_i$ is simply offloaded if $S_i + I_i < C_i$. To show the effectiveness, we report the results by adopting these two algorithms.

The $I_i$ values are calculated using Algorithm 1. The evaluation is performed for different values of the given utilization from the server $U_s$. The server uses the fair-sharing policy, where its utilization is partitioned between the connected clients equally, i.e., the given utilization $U_s$ for each client is equal to 1 divided by the number of served clients. For example, if the given utilization for a client is $U_s = 1$, this means that the server is dedicated to this client. And if $U_s = 0.1$, then the server serves 10 connected clients at the same time.

A. Case Study of a Surveillance System

A surveillance system is implemented as a case study to evaluate our algorithm and to compare it with the Simple-Offload approach. The server is Pentium(R) Dual-Core 2.8 GHz 64-bit CPU with 4 G memory. The client has Centrino Duo 1.73 GHz 32-bit CPU and 512 MB of memory, and is provided with two cameras (left and right). The client performs four independent sporadic real-time tasks on the input video streams. The tasks can be described as follows:

- **Motion Detection**: Detects the moving objects.
- **Object Recognition**: Recognizes and tracks a given object.
- **Stereo Vision**: Calculates the distance between the camera and the object of interest by generating a depth map for left and right images.
- **Motion Recording**: Records the video of the detected motion for records and any further human observations.

Motion detection, object recognition and motion recording process the images captured by the left camera. Table I shows the parameters of the tasks above, where the time is measured in milliseconds. The total utilization without offloading is equal to nearly 120%, i.e., there is no feasible schedule if all of the tasks are executed locally. The offloading algorithms are implemented to find a feasible schedule.

Figure 4 shows that our dynamic programming algorithm can find a feasible schedule for the tasks of the case study when the number of the served clients is up to four, i.e., the given utilization is $U_s = \{0.25, 0.333, 0.5, 1\}$. For more than
The difference between the results of the two algorithms comes from the fact that the dynamic programming algorithm tries to find the offloading decision that minimizes the total demand of all the tasks, while the offloading decision in Simple-Offload algorithm is based on each task alone. Also, the dynamic programming algorithm nominates a task \( \tau_i \) for offloading if \( S_i < C_i \), which reduces the demand of this task in the case of offloading, while in the Simple-Offload algorithm the task is offloaded only if \( S_i + I_i < C_i \).

B. Simulation Setup and Results

A synthetic workload is also used to evaluate our algorithm. Sporadic real-time tasks were generated randomly as follows:

- \( T_i \): Randomly generated integer values between 50 and 150 ms with uniform distribution.
- \( C_i \): Randomly generated floating-point values, such that the total utilization of each task set without offloading is equal to \( U_{local} \).
- \( S_i \): Randomly generated integer values from 1 to \( C_i \) ms with uniform distribution.
- \( R_i \): \( R_i = \frac{C_i}{\alpha} \), where \( \alpha \) is the speed-up factor of the server.

For each value of \( U_{local} = \{1.1, 1.2, 1.3\} \) and of \( U_s = \{0.1, 0.2, 0.5, 1\} \), a total of 10 task sets were generated and evaluated for different values of \( \alpha = \{0.25, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \). Each task set contains 20 sporadic real-time tasks that generated randomly according to the conditions above. All of the generated task sets are not feasible if they are executed locally, because their total utilization \( U_{local} \) is greater than one for all cases. Our dynamic programming algorithm and the Simple-Offload approach were implemented to find feasible schedules for the generated task sets by the help of computation offloading. The two algorithms are evaluated by considering the percentage of the obtained feasible task sets (or schedules), which is equal to the number of obtained feasible schedules divided by the total number of generated task sets for each simulation case above.

Figures 5 and 6 show the percentage of the feasible task sets obtained by the dynamic programming algorithm and Simple-Offload algorithm respectively, for all possible values of \( \alpha \) and \( U_{local} \). As the value of \( \alpha \) increases, the number of obtained feasible schedules increases for all given utilizations from the server. Because with a faster server (higher \( \alpha \) values), the value of \( I_i \) decreases, and then the client may offload more tasks. We also observe that with faster servers we need less given utilization \( U_s \) to find feasible schedules.

Figure 5 shows that the dynamic programming algorithm finds feasible schedules for \( \alpha = \{0.25, 0.5, 1\} \), which means that the algorithm offloads tasks to servers that have the same speed of the client or even slower. But, the Simple-Offload algorithm finds feasible schedules just when the server is faster than the client as shown in Figure 6. Because the offloading decision in this algorithm is based on the relation between \( S_i + I_i \) and \( C_i \), and \( S_i + I_i \) is always greater than \( C_i \) in the case of slower server (or server with the same speed of the client). In the contrary, the dynamic programming algorithm
nominates any task with $S_i < C_i$ for offloading, while its result returns before the deadline, to reduce the demand of the tasks and then find a feasible schedule. See Figure 7 that presents a comparison between the two algorithms for $U_s = \{1, 0.5\}$, $U_{local} = 1.1$ and all values of $\alpha$.

In general, the simulation shows that the computation offloading technique helps to reduce the local processor demand by offloading part of the tasks to the server, and then find feasible schedules.

VII. CONCLUSION

In this paper, the computation offloading mechanism is used to satisfy the real-time constraints in mobile devices, where the sporadic real-time tasks are considered. According to this mechanism, the computation intensive tasks are offloaded from the client (i.e., the mobile device) to the server (i.e, a powerful remote processing unit). On the server side, we adopt the total bandwidth server (TBS) to provide response time guarantee for the offloaded tasks. There are two challenges in our problem: determine which tasks to be offloaded, and schedule all of the tasks on the client without violating their real-time constraints. Therefore, a dynamic programming algorithm is proposed to determine the offloading decision of the tasks, such that their schedule is feasible. The algorithm is evaluated using a case study of surveillance system and synthesized benchmarks. The evaluation shows that our algorithm can find a feasible schedule using computation offloading. For future research, we plan to use the computation offloading to minimize the energy consumption in the mobile devices.

REFERENCES