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
Karlsruhe Institute of Technology, Germany
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.**

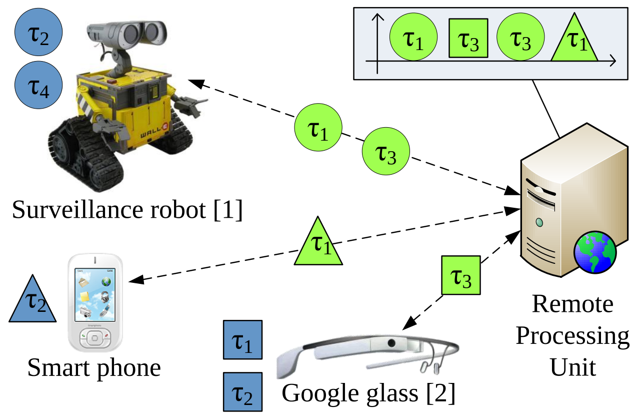


Fig. 1: Example of the offloading Mechanism.

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 executed on the mobile device and the green ones 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

Published in the 2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, Chongqing, China, August 20-22, 2014. http://dx.doi.org/10.1109/RTCSA.2014.6910509

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**.

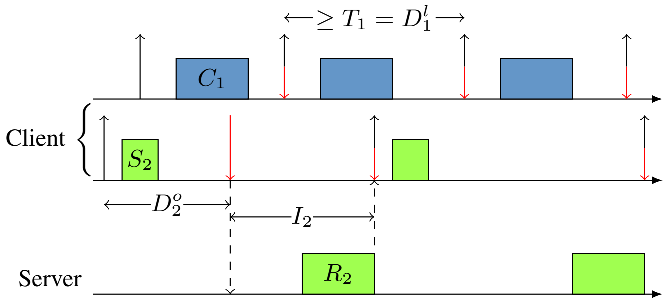


Fig. 2: Timing parameters for two tasks.

- T_i: **Minimum inter-arrival time**. The minimum period of time between the arrival of two consecutive jobs of the same task.
- S_i: **Setup time**. The execution time required for the task 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 for the data of the task from the client to the server.
- I_i: **Round-trip offloading time**. The period of time starting from the end of setting up the task τ_i until getting the result from the server.

We say that a task τ_i is *executed locally* if it is executed with (at most) C_i amount of time on the client. And we say it is *offloaded* if it is executed (at most) S_i amount of time on the client for setting up, and then its result returns back from the server after (at most) I_i amount of time. Each task can be executed locally or offloaded. Suppose that x_i is equal to 1 if the task τ_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 τ_i can be expressed as follows:

$$D_i = x_i D_i^o + (1 - x_i) D_i^l. \quad (1)$$

Where, the setting up of the task τ_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 mange 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 $\frac{1}{U_s}$ times slower than the speed of the server.

C. Problem Definition

Given a set $\mathcal{T}$ 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 τ_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 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 τ_i within the time interval of length t can be defined as follows:

$$\text{dbf}(\tau_i, t) = \max\left\{0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right\} \times C_i. \quad (2)$$

The task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ 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

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

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:

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

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

$$\forall t > 0, \sum_{i=1}^{n} \text{dbf}^*(\tau_i, t) \leq t. \tag{5}$$

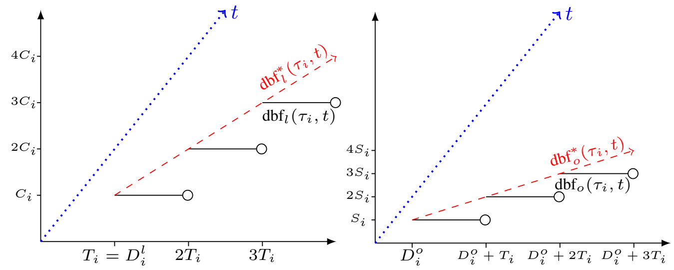
According to the approximation approach, only the first job of each task is considered in the schedulability analysis. The relative deadline D_i of each task τ_i represents its maximum test interval, supposing that the first jobs of all the tasks are released simultaneously at the beginning of the interval time t [4]. Therefore, the following conditions represent sufficient schedulability test:

$$\sum_{j=1}^{n} u_j = \sum_{j=1}^{n} \frac{C_j}{T_j} \leq 1 \tag{6a}$$

$$\forall \tau_i \in \mathcal{T}, \sum_{j=1}^{i} \text{dbf}^*(\tau_j, D_i) \leq D_i, \tag{6b}$$

where u_j is the utilization of task τ_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 ϵ *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 ϵ 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.



(a) $\text{dbf}_l(\tau_i, t)$ and $\text{dbf}_l^*(\tau_i, t)$ functions.

(b) $\text{dbf}_o(\tau_i, t)$ and $\text{dbf}_o^*(\tau_i, t)$ functions.

Fig. 3: Illustration for the demand bound function and its approximation.

V. Our Approach

In our offloading problem, two decision-making points should be considered: the offloading decision $\vec{x}_n$ 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 $\vec{x}_n$ 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 I_i.

A. 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^l. 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 τ_i in our model are defined respectively as follows:

- τ_i is executed locally:

$$u_i^l = \frac{C_i}{T_i},$$
$$\text{dbf}_l(\tau_i, t) = \max\left\{0, \left\lfloor \frac{t}{T_i} \right\rfloor\right\} \times C_i,$$
$$\text{dbf}_l^*(\tau_i, t) = \begin{cases} 0 & \text{if } t < D_i^l \\ t\frac{C_i}{T_i} & \text{otherwise.} \end{cases}$$

- τ_i is offloaded:

$$u_i^o = \frac{S_i}{T_i},$$
$$\text{dbf}_o(\tau_i, t) = \max\left\{0, \left\lfloor \frac{t - D_i^o}{T_i} \right\rfloor + 1\right\} \times S_i,$$
$$\text{dbf}_o^*(\tau_i, t) = \begin{cases} 0 & \text{if } t < D_i^o \\ \left(\frac{t - D_i^o}{T_i} + 1\right)S_i & \text{otherwise.} \end{cases}$$

Figure 3 illustrates the demand bound function (in black) and its approximation (dashed red line) for the task τ_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:

$$\sum_{j=1}^{n} u_j = \sum_{j=1}^{n} x_j u_j^o + \sum_{j=1}^{n} (1-x_j) u_j^l$$
$$= \sum_{j=1}^{n} x_j \frac{S_j}{T_j} + \sum_{j=1}^{n} (1-x_j)\frac{C_j}{T_j} \leq 1 \tag{9a}$$

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

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

$$\sum_{j=1}^{i} x_j \Big(\frac{D_i - D_j^o}{T_j} + 1\Big) S_j + \sum_{j=1}^{i} (1-x_j) D_i \frac{C_j}{T_j}$$
$$\leq \sum_{j=1}^{i} x_j \Big(\frac{D_i}{T_j} + 1\Big) S_j + \sum_{j=1}^{i} (1-x_j) D_i \frac{C_j}{T_j}$$
$$= D_i \Big(\frac{\sum_{j=1}^{i} x_j S_j}{D_i} + \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1-x_j) \frac{C_j}{T_j}\Big). \tag{10}$$

Therefore, based on the condition in (9b) and the over-approximation 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 $\vec{x}_n$, if $\forall \tau_i \in \mathcal{T}$,

$$D_i \Big(\frac{\sum_{j=1}^{i} x_j S_j}{D_i} + \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1-x_j) \frac{C_j}{T_j}\Big) \leq D_i.$$

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:

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

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.

$$\leq \Big(\frac{\sum_{j=1}^{i} x_j S_j}{t} + \sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1-x_j) \frac{C_j}{T_j}\Big) t \leq t,$$
$$\Big(\tfrac{\sum_{j=1}^{i} x_j S_j}{t} + \textstyle\sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1-x_j) \frac{C_j}{T_j}\Big) \leq 1$$

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

■

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

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

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 τ_i, if

$$\frac{\sum_{j=1}^{i} x_j S_j}{D_i} + \Big(\sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1-x_j) \frac{C_j}{T_j}\Big) > 1$$

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

$$\frac{\sum_{j=1}^{i} x_j S_j}{D_i} + \Big(\sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1-x_j) \frac{C_j}{T_j}\Big) > 1,$$

then we have two possibilities:

1) $\frac{\sum_{j=1}^{i} x_j S_j}{D_i} > 0.5$, then $\sum_{j=1}^{i} x_j S_j > 0.5 D_i$. The infeasibility by slowing down can be verified as follows: $\sum_{j=1}^{n} \text{dbf}(\tau_j, D_i) \geq \sum_{j=1}^{i} x_j \cdot \text{dbf}_o(\tau_j, D_i) \geq \sum_{j=1}^{i} x_j S_j > 0.5 D_i$.
2) $\big(\sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1-x_j) \frac{C_j}{T_j}\big) > 0.5$. By slowing down to 0.5 of the original speed, the utilization becomes: $\sum_{j=1}^{i} x_j \frac{2S_j}{T_j} + \sum_{j=1}^{i} (1-x_j) \frac{2C_j}{T_j} = 2\big(\sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i} (1-x_j) \frac{C_j}{T_j}\big) > 1$.

■

B. Dynamic Programming Algorithm

The feasibility condition in Lemma 1 can be verified for a given offloading decision vector $\vec{x}_n$. One solution to determine the offloading decision $\vec{x}_n$ 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 can not be offloaded ($D_i^o < S_i$), are assigned and fixed for local execution with $D_i = D_i^l$. 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 $\{\tau_1, \tau_2, \dots, \tau_i\}$. Recall the feasibility condition in (12), let $\frac{\sum_{j=1}^{i} x_j S_j}{D_i}$ be the *effective density* for the first i tasks at time D_i, and $\left(\sum_{j=1}^{i} x_j \frac{S_j}{T_j} + \sum_{j=1}^{i}(1-x_j)\frac{C_j}{T_j}\right)$ 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 δ. A two-dimensional dynamic programming table $L(i, \delta)$ is constructed for all possible values of i and δ, such that $0 \leq i \leq n$ and $0 \leq \delta \leq 1$. Where, all the possible values of δ are considered as the integer multiples of ρ (i.e., ρ is a user-specified granularity), and $\frac{1}{\rho}$ 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.

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

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 τ_i is assigned for offloading and if it is assigned for local execution respectively.

Lemma 3: For given i and δ, 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 $\vec{x}_{i-1}^o$ be the optimal offloading decision for $\{x_1, x_2, \dots, x_{i-1}\}$ 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 τ_i is considered for offloading). Similarly, let $\vec{x}_{i-1}^l$ be the optimal offloading decision for $\{x_1, x_2, \dots, x_{i-1}\}$ 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 τ_i is considered for local execution).

Suppose for contradiction that $\vec{x}_i^*$ is an offloading decision for $\{x_1, x_2, \dots, x_i\}$ in which $\frac{\sum_{j=1}^{i} x_j^* S_j}{D_i} \leq \delta$ and $\left(\sum_{j=1}^{i} x_j^* \frac{S_j}{T_j} + \sum_{j=1}^{i}(1-x_j^*)\frac{C_j}{T_j}\right) < L(i, \delta)$. There are two cases for the task τ_i in $\vec{x}_i^*$:

- **Case 1:** τ_i is executed locally in $\vec{x}_i^*$ ($x_i^* = 0$). Under the assumption we have $\left(\sum_{j=1}^{i} x_j^* \frac{S_j}{T_j} + \sum_{j=1}^{i}(1-x_j^*)\frac{C_j}{T_j}\right) < L(i, \delta)$. But
$$\begin{aligned}&\Big(\sum_{j=1}^{i} x_j^* \frac{S_j}{T_j} + \sum_{j=1}^{i}(1-x_j^*)\frac{C_j}{T_j}\Big)\\ =&\Big(\sum_{j=1}^{i-1} x_j^* \frac{S_j}{T_j} + \sum_{j=1}^{i-1}(1-x_j^*)\frac{C_j}{T_j}\Big) + \frac{C_i}{T_i}\\ <&L(i,\delta) \leq \Big(\sum_{j=1}^{i-1} x_j^l \frac{S_j}{T_j} + \sum_{j=1}^{i-1}(1-x_j^l)\frac{C_j}{T_j}\Big) + \frac{C_i}{T_i},\end{aligned}$$
Hence, the offloading decision $\vec{x}_{i-1}^*$ is an offloading decision for the first $i-1$ tasks in which $\frac{\sum_{j=1}^{i-1} x_j^* S_j}{D_{i-1}} \leq \frac{\delta D_i}{D_{i-1}}$ and $\left(\sum_{j=1}^{i-1} x_j^* \frac{S_j}{T_j} + \sum_{j=1}^{i-1}(1-x_j^*)\frac{C_j}{T_j}\right) < \left(\sum_{j=1}^{i-1} x_j^l \frac{S_j}{T_j} + \sum_{j=1}^{i-1}(1-x_j^l)\frac{C_j}{T_j}\right)$, which contradicts the optimality of $L(i-1, \delta)$.
- **Case 2:** The same argument in case 1 applies when τ_i is offloaded in $\vec{x}_i^*$ ($x_i^* = 1$).

Hence, based on the induction hypothesis, the lemma is proved. ■

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 $L(n, \delta) + \delta$, 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 τ_i, starting from the solution found, as follows: (1) If τ_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 $O(n \log n + n\frac{1}{\rho})$.

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).

Algorithm 1 Estimating the value of I_i

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

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 $\mathcal{T}_o \in \mathcal{T}$ 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 $\mathcal{T}_o$. 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 $\mathcal{L}$ (Lines 1 and 2). As long as the list $\mathcal{L}$ is not empty, the algorithm keeps doing the following steps:

- Pick the task τ_j with the maximum heuristic value $\frac{C_j - S_j}{R_j}$ from the list $\mathcal{L}$ (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).
- 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 $\tau_i \in \mathcal{T}_o$ a TBS_i with a utilization U_i such that their total utilization is equal to the given utilization from the server U_s, i.e., $\sum_{\tau_i \in \mathcal{T}_o} U_i = U_s$ (Line 8).
- The dynamic programming algorithm is used to find a feasible schedule. If there exists one, Algorithm 1 returns the offloading decision $\vec{x}$ (Lines 9 - 11).

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

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

τ_i	Description	C_i	S_i	R_i	T_i
τ_1	Motion Detection	30	7	21	115
τ_2	Object Recognition	220	2	102	418
τ_3	Stereo Vision	88	16	41	695
τ_4	Motion Recording	18	7	14	63

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 τ_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

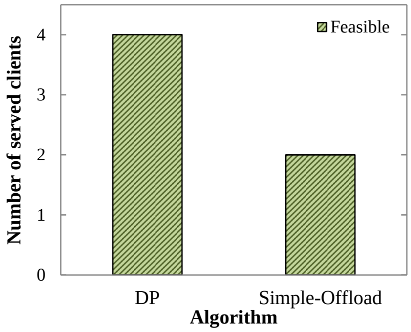


Fig. 4: Case study results.

four clients, or $U_s \leq 0.2$, the algorithm is not able to find a feasible schedule. But, it guarantees that there is no feasible by slowing down the processor to 0.5 of the original speed. The Simple-Offload algorithm can find a feasible schedule for $U_s = \{0.5, 1\}$, i.e., the server provides offloading services for two clients or is dedicated for one.

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 τ_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 $S_i + I_i < C_i$.

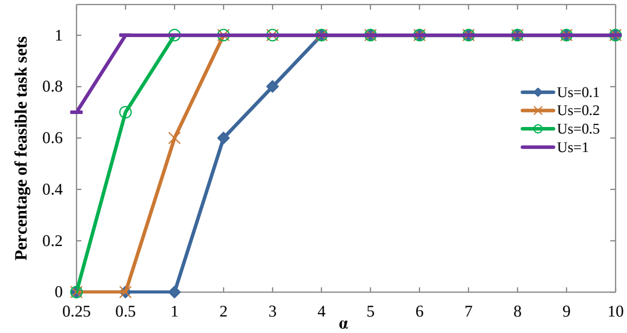
B. Simulation Setup and Results

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

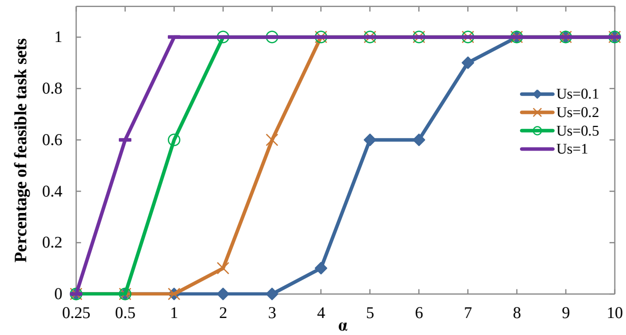
- T_i: Randomly generated integer values between 50 and 150 ms with uniform distribution.
- C_i: Randomly generated floating-point values, such that the total utilization of each task set without offloading is equal to U_{local}.
- S_i: Randomly generated integer values from 1 to C_i ms with uniform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

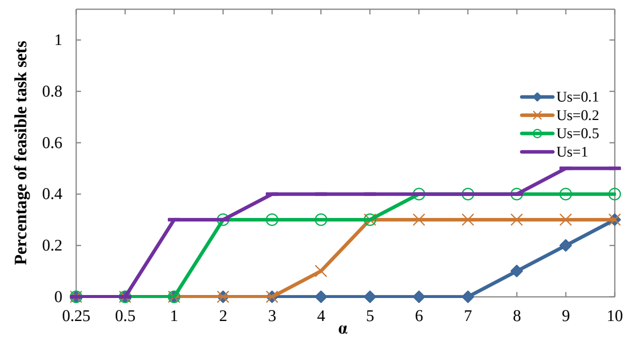
Figures 5 and 6 show the percentage of the feasible task sets obtained by the dynamic programming algorithm and



(a) Percentage of the feasible task sets for $U_{local} = 1.1$.



(b) Percentage of the feasible task sets for $U_{local} = 1.2$.

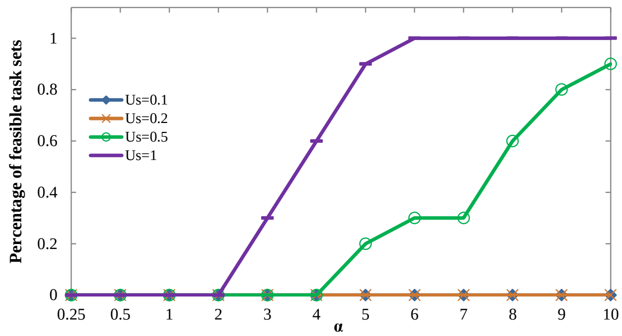


(c) Percentage of the feasible task sets for $U_{local} = 1.3$.

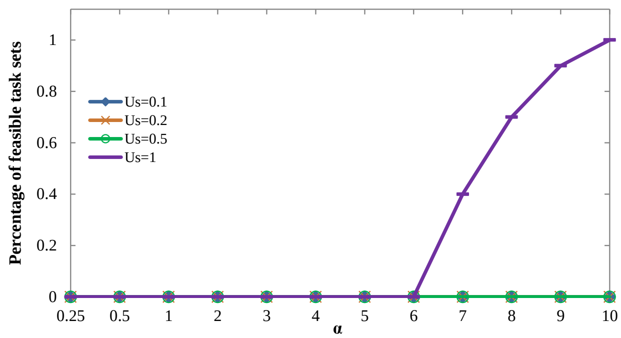
Fig. 5: Feasible task sets obtained by the dynamic programming algorithm.

Simple-Offload algorithm respectively, for all possible values of α and U_{local}. As the value of α increases, the number of obtained feasible schedules increases for all given utilizations from the server. Because with a faster server (higher α 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.

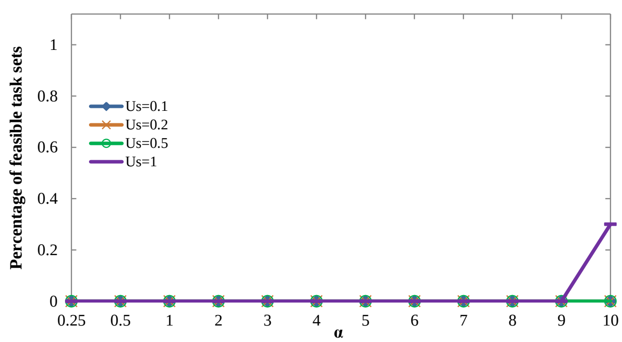
Figure 5 shows that the dynamic programming algorithm finds feasible schedules for $\alpha = \{0.25, 0.5, 1\}$, which means that the algorithm offloads tasks to servers that have the same speed of the client or even slower. But, the Simple-Offload algorithm finds feasible schedules just when the server 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



(a) Percentage of the feasible task sets for $U_{local} = 1.1$.



(b) Percentage of the feasible task sets for $U_{local} = 1.2$.



(c) Percentage of the feasible task sets for $U_{local} = 1.3$.

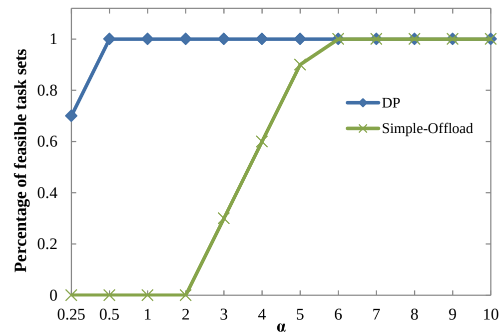
Fig. 6: Feasible task sets obtained by the Simple-Offload algorithm.

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 α.

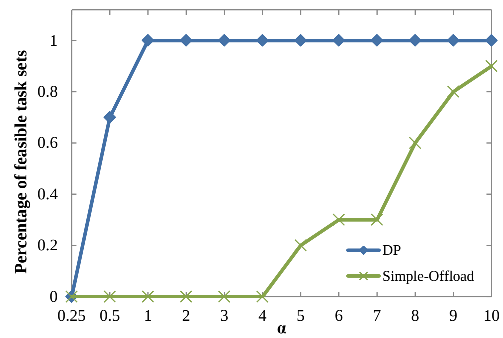
In general, the simulation shows that the computation offloading technique helps to reduce the local processor demand by offloading part of the tasks to the server, and then find feasible schedules.

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



(a) Percentage of the feasible task sets for $U_s = 1$.



(b) Percentage of the feasible task sets for $U_s = 0.5$.

Fig. 7: Feasible task sets obtained by both algorithms for $U_s = \{1, 0.5\}$ and $U_{local} = 1.1$

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

[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] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pages 187–195, 2004.

[4] K. Albers and F. Slomka. Efficient feasibility analysis for real-time systems with edf scheduling. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 492–497 Vol. 1, 2005.

[5] H. Albitar, A. Ananiev, and I. Kalaykov. New concept of in-water surface cleaning robot. In *Mechatronics and Au-*

tomation (ICMA), 2013 IEEE International Conference on, pages 1582–1587, 2013.

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

[7] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190, 1990.

[8] G. C. Buttazzo. *Hard Real-time Computing Systems*. Springer US, 2011.

[9] S. Chakraborty, S. Kunzli, and L. Thiele. Approximate schedulability analysis. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 159–168, 2002.

[10] J.-J. Chen and S. Chakraborty. Resource augmentation bounds for approximate demand bound functions. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 272–281, 2011.

[11] 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.

[12] 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.

[13] 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.

[14] S.-Y. Juang and J.-G. Juang. Real-time indoor surveillance based on smartphone and mobile robot. In *IEEE International Conference on Industrial Informatics (INDIN)*, pages 475 –480, 2012.

[15] S. Kantawong. Exterior climbing mirror cleaning robot based on hybrid fuzzy-pid and pneumatic control system. In *Knowledge and Smart Technology (KST), 2013 5th International Conference on*, pages 101–106, 2013.

[16] A. Khairy, H. Ammar, and R. Bahgat. Smartphone energizer: Extending smartphone's battery life with smart offloading. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, pages 329–336, 2013.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.

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

[24] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Cambridge, MA, USA, 1983.

[25] 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.

[26] 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.

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

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

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

[30] A. Toma and J.-J. Chen. Server resource reservations for computation offloading in real-time embedded systems. In *the 11th IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia 2013)*, pages 31–39, Oct 2013.

[31] 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.

[32] C.-C. Tseng, C.-L. Lin, B.-Y. Shih, and C.-Y. Chen. Sip-enabled surveillance patrol robot. *Robotics and Computer-Integrated Manufacturing*, 29(2):394 – 399, 2013.

[33] 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.

[34] 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.

[35] 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.

[36] 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.