Computation Offloading for Frame-Based Real-Time Tasks with Resource Reservation Servers

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

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

***Abstract*—Computation offloading concept has been recently adopted to improve the performance of embedded systems by moving some computation-intensive tasks (partially or wholly) to a powerful remote server. In this paper, we consider a computation offloading problem for frame-based real-time tasks, in which all the tasks have the same arrival time and the same relative deadline/period, by adopting the total bandwidth server (TBS) as resource reservations in the server side (remote execution unit). We prove that the problem is $\mathcal{NP}$-complete and propose two algorithms in this paper. The first algorithm is a greedy algorithm with low complexity and provides a quick heuristic approach to decide which tasks to be offloaded and how the tasks are scheduled. The maximum finishing time of the solution derived from the greedy algorithm is at most twice of the finishing time (makespan, maximal on the client and on the server) of any schedule. The second algorithm is a dynamic programming approach, which builds a three-dimensional table and requires pseudo-polynomial time complexity, to make an optimal decision for computation offloading. The algorithms are evaluated with a case study of a surveillance system and synthesized benchmarks.**

I. Introduction

In the recent years, we have seen a significant increase in the use of video surveillance systems for real-time monitoring. They are widely used for security, rescue and safety purposes [22, 23]. Specifically, the mobile robots are the preferred platforms for the surveillance in the difficult and dangerous situations. The mobility of the robots provides a wider range for the vision so that the blind areas can be observed and monitored as shown in [3, 9, 12]. Moreover, such robots have been used for the missions that are hazardous for human beings. For instance, a mobile surveillance robot, called SURBOT, has been used since 1987 for real-time monitoring at the Browns Ferry Nuclear Plant in United States in order to reduce the radiation exposure [24]. Recently, there has been a growing interest in adopting surveillance robots for home security. These robots are used for indoor reconnaissance and patrolling, and also for monitoring of the surroundings [2, 11, 21].

To perform real-time surveillance, the video stream, consisting of a sequence of real-time images captured by a camera (or the cameras) on the robot, should be analyzed under the specified timing constraints. It usually involves several image processing tasks, such as motion detection and recording, object recognition, behavioral analysis, and analysis of the stereo vision [22, 23]. The period of time between two consecutive images is called a frame and represents the relative deadline of the tasks. All of the required surveillance tasks should be executed and finished within this frame to preserve the real-time property of the system.

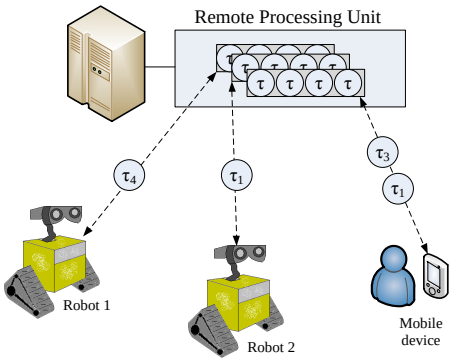


Fig. 1: Offloading Mechanism.

However, the resource limitations in the computational capabilities of the mobile robots may limit the possibility to complete all the tasks in time. One of the solutions for such a resource-constrained problem is to perform *computation offloading*, which offloads a part of the tasks (i.e., computation-intensive tasks) to some remote processing units, i.e., servers. Figure 1 illustrates the computation offloading mechanism. Hereafter, we denote the embedded system that offloads its tasks as *the client*, and the remote processing unit that executes the offloaded tasks from the client as *the server*.

Our Contribution: In this paper, we consider a computation offloading problem for frame-based real-time tasks by adopting the total bandwidth server (TBS) in the server side for resource reservations, in which all the tasks have the same arrival time and the same relative deadline/period. The computation offloading mechanism exploits the idle time on the client when a task is offloaded and does not implicitly assume a dedicated server for the client. Our contribution can be summarized as follows:

- We prove that the offloading problem is $\mathcal{NP}$-complete.
- The proposed approximation algorithm is a greedy algorithm with low complexity. The maximum finishing time of the solution derived from the greedy algorithm is at most twice of the finishing time of any schedule.
- The proposed dynamic programming algorithm builds a three-dimensional table in pseudo-polynomial time complexity to make an optimal decision for computation offloading.
- We present a case study and randomly synthesized bench-

Published in the 25th Euromicro Conference on Real-Time Systems, ECRTS 2013, Paris, France, July 9-12, 2013.
http://dx.doi.org/10.1109/ECRTS.2013.21

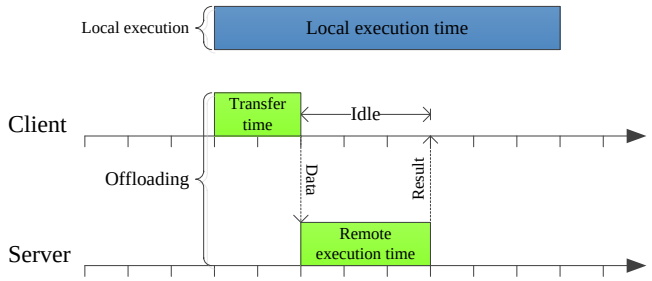


Fig. 2: Simple offloading decision adopted in [16].

marks for evaluating our proposed algorithms.

- The proposed algorithms can also be extended to minimize the period (or maximize the sampling rate) of the tasks by minimizing the finishing time (makespan, maximal on the client and on the server) of the resulting schedule.

The remainder of the paper is organized as follows: Section II provides the related work and discusses their drawbacks and limitations. Section III presents the system model. The optimal ordering of the tasks is shown in Section IV after an offloading decision is made. The $\mathcal{NP}$-completeness of the studied problem is shown in Section V. Section VI presents our approaches. Experimental evaluations and simulations are presented in Section VII, and Section VIII concludes the paper.

II. Related Work for Computation Offloading

This section summarizes the related work in computation offloading and provides the explanations for the limits of their approaches for real-time systems.

A framework for computational offloading is proposed in [6, 25] for computational grid settings to improve the performance. The offloading decision is represented as a statistical decision problem. The scheduler determines when to move parts of computation to more capable devices based on the prediction of the local execution time, remote execution time and the transmission time. Nimmagadda et al. [16] propose a framework for mobile robots in order to perform recognition and tracking for moving objects without violating the real-time constraints. The offloading decision in the two previous approaches is basically based on the comparison between the data transfer time added to the remote execution time, and the local execution time, as shown in Figure 2.

The computation offloading mechanism is used in [17, 27] for mobile handsets. The mobile application is represented as a directed graph, where each vertex represents a Java class with memory and CPU costs. The edges of the graph represent the communications between the classes of the application. Graph partitioning algorithms are developed in [5, 14, 15] to decide how to partition the application to be executed on the servers and the client.

Furthermore, Ferreira et al. [4] explore the computation offloading to improve the quality of service in the adaptive real-time systems. A timeout mechanism is used for the offloading decision in [26] to reduce the energy consumption on battery-powered systems. Hong et al. [7] propose an offloading strategy to save energy for mobile systems. The strategy is used for performing content-based image retrieval.

Kovachev et al. [13] develop the Mobile Augmentation Cloud Services (MACS) middleware for mobile Android platforms. The middleware offloads the computation intensive tasks to a remote clouds. Offloading decision is represented as an optimization problem based on some input parameters, such as CPU load, available memory, remaining battery, and the bandwidth. Integer linear Programming (ILP) is used to solve the problem on the mobile device.

However, these approaches suffer from three drawbacks: (1) The client in all of the existing solutions remains idle during the offloading, and awaits the result from the server side. Also, they do not consider the scheduling on the client. Changing the execution order of the tasks may improve the performance. For example, a task that is executed locally on the client without offloading can be executed during the remote execution of an offloaded task instead of idling. (2) They do not consider the server system model or explain how the server can execute the offloaded tasks. They implicitly assume that the powerful server is dedicated for one specific client and can execute any offloaded task immediately. In practice, a powerful server can serve more than one client, because the client might not require a complete server for offloading. Therefore, the capability of the server is not fully exploited. (3) Furthermore, most of the computation offloading researches do not consider the real-time systems in their approaches.

In our recently study in [20], we explore an offloading scheme for frame-based real-time tasks by assuming that the server(s) provides a (worst-case) round-trip time guarantee for an offloaded task as an input. That is, if an offloaded task arrives at the server at time t, its processing in the corresponding server will be done by at most t plus the round-trip time. It does not require dedicated servers for the client. However, the round-trip time guarantee requires a certain reservation in the server. If the schedule in the client does not offload a task, the server can release the reservation for the task. This may result in the reduction of the round-trip times of the other tasks. But the scheme in [20] cannot handle such dynamic behavior. In this paper, we adopt the total bandwidth server (TBS) in the server side to resolve the drawbacks mentioned above.

III. System Model

This section presents the system model and the problem definition. We use the terms *locally executed* and *offloaded* for a given task to refer to the processing of that task on the client processor and on the server, respectively.

A. Task Model

We explore the scheduling of a set $\mathcal{T}$ of n independent frame-based real-time tasks that are independent in execution. All the tasks arrive at time $t = 0$, have the same period D, and require execution within a common relative deadline D. Each task $\tau_i \in \mathcal{T}$ (for $i = 1, 2, \ldots, n$) can be executed locally or offloaded, and is characterized by the following parameters:

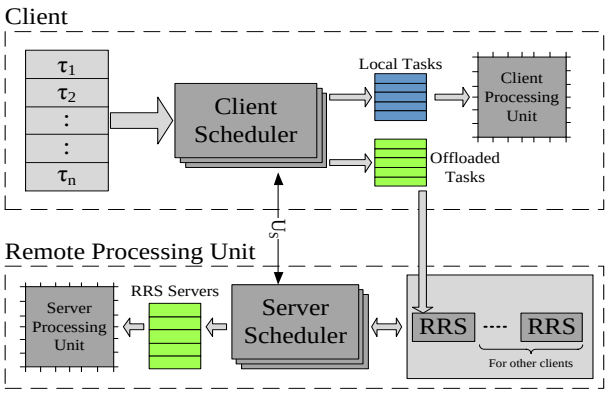


Fig. 3: Offloading System Architecture.

- C_i: Local execution time on the client side.
- S_i: Setup time, which is the execution time required on the client side for task τ_i to be ready for execution on the server side. It includes the sending time of the data from the client to the server and any preprocessing operations (such as encoding and compression). For example, sending time can be estimated as $\frac{Z}{\beta}$, where Z is the size of the offloaded data and β is the estimated network bandwidth between the client and the server.
- R_i: Remote execution time, which is the execution time on the server side. For example, if the server side uses the same program as in the client for local execution, it can be estimated as $\frac{C_i}{\alpha}$, where α is the speed-up factor of the server. Under such a setting, if $\alpha > 1$, we can say that the server is α times faster than the client; otherwise, if $1 > \alpha > 0$, the server is said $\frac{1}{\alpha}$ times slower than the client.

We assume that the parameters C_i, S_i, and R_i are given based on the system setup. When all these parameters are specified for (the upper bounds of) the worst cases, we would like to provide the hard real-time guarantees. Otherwise, the information is based on estimations, and we would like to meet the timing constraint by exploiting the services provided from the server.

We assume that the post processing time of any offloaded task is negligible, because the results returned from the server need very short post processing time. For instance, the returned results in our case study of a surveillance system are the coordinates of the moving object or the distance between it and the cameras.

B. Server (Remote Processing Unit)

When adopting a server to handle the offloaded tasks for real-time systems, the server has to provide a guarantee for the response time of the offloaded tasks to preserve the real-time property of the system. The server should provide a certain *resource reservation* for a client. In the *Resource Reservation Server* (RRS) model[1], one can receive bandwidth guarantees, such as the total bandwidth server (TBS), or receive budget guarantees for a given time interval, such as a constant bandwidth server (CBS), a deferrable server (DS), and a sporadic server (SS) [1].

In this paper, we adopt the *Total Bandwidth Server* (TBS) [18, 19] as RRS in the server side. The server allocates a TBS for each requesting client with a utilization (or bandwidth) value U_s, if it is possible. The TBS assigns an absolute deadline $d_s(t)$ for an incoming (offloaded) task τ_i that arrives at the server at time t as follows:

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

where $d_s(t^-)$ is the absolute deadline of the previous offloaded task served by the TBS, and $d_s(0^-)$ is defined as 0.

The server side will execute the offloaded tasks by using the *Earliest Deadline First* (EDF) scheduling algorithm under the assigned TBS deadlines. To meet the absolute deadlines assigned to the offloaded tasks (from one or more than one client) in the server, the total utilization of all the TBS's in the server should be less than or equal to 100% [19]. Figure 3 shows the offloading system architecture.

For the rest of this paper, we will assume that U_s provided from the server side to the client does not violate the utilization constraint (100%). Therefore, the assigned absolute deadline for an offloaded task will always be met, and we will take this absolute deadline as the worst-case finishing time of the offloaded task.

C. Problem Definition

Based on the TBS with U_s provided from the server to the requesting client, the client decides which tasks should be offloaded and how. Therefore, the tasks in $\mathcal{T}$ will be divided into two sets: (1) the set of local tasks and (2) the set of offloaded tasks, as shown in Figure 3.

Based on the above model, the scheduling problem that is focused on this paper can be defined as follows:

Given a set $\mathcal{T}$ of n frame-based real-time tasks and a TBS with bandwidth U_s provided by the server, the Computation Offloading with TBS (COTBS) problem is to schedule the tasks and decide which to be offloaded and which to be locally executed without violating their timing constraints.

A schedule is considered to be *feasible* if the finishing times of all locally-executed and offloaded tasks are within the deadline D. A scheduling algorithm is said to be *optimal* algorithm if it is able to find a feasible schedule, if and only if one exists.

The *finishing time* on the client of a task τ_i is the time that τ_i finishes its setup (if τ_i is offloaded) or its local execution if τ_i is not offloaded. Suppose that the finishing time of τ_i is t_i. Moreover, the *finishing time* on the server of a task τ_i is defined as the TBS deadline for task τ_i (i.e., $d_s(t_i)$) if τ_i is offloaded or the maximal between t_i and *last* TBS deadline set before τ_i (i.e., $\max\{t_i, d_s(t_i^-)\}$) if τ_i is not offloaded.

The *makespan* is defined as the maximum of the finishing time on the client and the finishing time on the server. Suppose that x_i is equal to 1 if task τ_i is decided to be offloaded; otherwise, x_i is 0. We use a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ to denote an *offloading decision* for the tasks.

[1]This is a logical server, inherited from the literature

IV. Optimal Task Ordering

In this section, we will explore the task ordering on the client to decide the execution sequence under the assumption that the decision of offloading or local execution for all the tasks is already known, i.e., $\vec{x}$ is given. Based $\vec{x}$, we can define the following two terms:

- $P_{i,c} = x_i S_i + (1 - x_i) C_i$, and
- $P_{i,s} = x_i \frac{R_i}{U_s}$,

where $P_{i,c}$ is the execution time on the client for task τ_i and $P_{i,s}$ is the remote execution time divided by the utilization of the TBS.

For completeness, we will link the ordering of execution for the COTBS problem to the well-known *two-stage flow shop* problem [10]. In the two-stage flow shop problem, there are n jobs, arriving at time 0. Each job J_i is characterized by the the execution time $P_{i,c}$ on the first machine (or stage), and the execution time $P_{i,s}$ on the second machine. Moreover, each job must be processed on the first machine first, and then on the second machine. The two-stage flow shop problem is to schedule the jobs in order to minimize the makespan (maximum completion time for the entire operation on both machines) [10]. Based on the $P_{i,c}$ and $P_{i,s}$, we call the above concrete instance spanned by $\vec{x}$ as the *corresponding two-stage flow shop* problem.

We use the terms *first machine*, *second machine* and *jobs* to refer to the two-stage flow shop problem. And the terms *client*, *server* and *tasks* to refer to our problem. When considering the scheduling of the corresponding two-stage flow shop scheduling problem, the ordering on the second machine is the same as the first machine.

The following lemma shows the properties of the finishing times of the tasks on the client and on the server of any sequence of ordering for the COTBS problem with known $\vec{x}$.

Lemma 1: For a given order of tasks $(\tau_1, \tau_2, \ldots, \tau_n)$ with a known offloading decision $\vec{x}$, the finishing time of task τ_k on the client (server, respectively) is the same as the finishing time of the corresponding job J_k on the first (second, respectively) machine for the corresponding two-stage flow shop problem.

Proof: Without loss of generality, we can consider the execution order follows the index from $1, 2, \ldots, n-1$ to n. This lemma can be proved by induction. In the corresponding two-stage flow shop problem, the second machine starts the execution of the first job J_1 at time $P_{1,c}$ (after finishing the execution on the first machine), and finishes at time $P_{1,c} + P_{1,s}$. In our problem, The TBS starts executing the first task after setup time $P_{1,c}$ and assigns a deadline of $d_s(P_{1,c}) = \max\{P_{1,c}, 0\} + \frac{R_1}{U_s} = P_{1,c} + P_{1,s}$.

Suppose that J_i finishes at time z_i on the second machine in the corresponding two-stage flow shop problem under this ordering. We assume that the finishing time of τ_i on the server is set to z_i. Now, we consider J_{i+1}. Clearly, J_{i+1} finishes on the first machine at time $\sum_{j=1}^{i+1} P_{j,c}$. Moreover, τ_{i+1} also finishes on the client at time $\sum_{j=1}^{i+1} P_{j,c}$. Let's consider two cases for the finishing times on the server and the second machine:

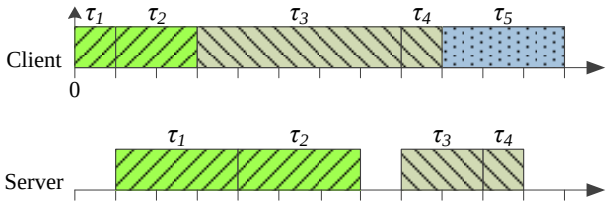


Fig. 4: An example of optimal ordering for five tasks.

- τ_{i+1} is not offloaded: by definition $P_{i+1,s}$ is 0, there is no need to execute on the second machine or on the server. The finishing time of J_{i+1} on the second machine is defined as $\max\{\sum_{j=1}^{i+1} P_{j,c}, z_i\}$. The finishing time of τ_{i+1} on the server is also equal to $\max\{\sum_{j=1}^{i+1} P_{j,c}, z_i\}$.
- τ_{i+1} is offloaded: J_{i+1} finishes on the second machine at time $\max\{\sum_{j=1}^{i+1} P_{j,c}, z_i\} + P_{i+1,s}$. Moreover, according to the definition of TBS server, the finishing time of τ_{i+1} on the server is $\max\{\sum_{j=1}^{i+1} P_{j,c}, z_i\} + P_{i+1,s}$.

According to the induction hypothesis, the lemma is proved. ■

It has been shown in [10] that Johnson's rule is optimal for minimizing the makespan for the two-stage flow shop problem. The tasks are divided into two sets $\mathcal{T}_1$ and $\mathcal{T}_2$ as follows:

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

Tasks in set $\mathcal{T}_1$ are scheduled first by the increasing order of $P_{i,c}$. Then, the tasks in set $\mathcal{T}_2$ are scheduled by the deceasing order of $P_{i,s}$. Figure 4 presents an example of five tasks with this ordering. Tasks τ_1 and τ_2 are offloaded and belong to the set $\mathcal{T}_1$. Tasks τ_3, τ_4 and τ_5 belong to the set $\mathcal{T}_2$, where the first two of them are offloaded and the last one is executed locally.

Therefore, we have the following lemma for the optimal ordering when $\vec{x}$ is given.

Lemma 2: For a given $\vec{x}$, the ordering of the tasks execution based on Johnson's rule is optimal to meet the deadline.

Proof: This comes from Lemma 1 and the optimality of the Johnson's rule ordering. ■

Corollary 1: For a given $\vec{x}$, all the locally-executed tasks belong to the set $\mathcal{T}_2$ and scheduled at the end.

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

V. Hardness of COTBS Problem

When the decisions for offloading is known, i.e., $\vec{x}$ is known, Section IV shows that following Johnson's rule is optimal. However, the difficulty of the COTBS problem comes from the unknown $\vec{x}$. The following theorem shows that COTBS problem is $\mathcal{NP}$-complete.

Theorem 1: The COTBS problem is $\mathcal{NP}$-complete.

Proof: The COTBS problem is in $\mathcal{NP}$, since we can use the task ordering in Lemma 2 to verify whether a solution is feasible or not in polynomial time. The $\mathcal{NP}$-completeness can be proved by a reduction from the SUBSET SUM problem: Given a set of $n-1$ positive integers $\mathcal{V} = \{v_1, v_2, \ldots, v_{n-1}\}$

and an integer A, the objective is to find a subset $\mathcal{V}' \subseteq \mathcal{V}$, such that $\sum_{v_i \in \mathcal{V}'} v_i = A$.

Suppose that δ and U_s are positive real numbers, in which $0 < \delta < 1$ and $0 < U_s \leq 1$. In the reduction, we first define $D = \delta + \sum_{v_j \in \mathcal{V}} 2v_j - A$. Moreover, we define $\rho = \frac{D-\delta-A}{A} = \frac{2((\sum_{v_j \in \mathcal{V}} v_j)-A)}{A}$. Furthermore, the reduction creates n tasks. Task τ_n is with the following parameters:

- $S_n = \delta$ with $0 < \delta < 1$,
- $C_n = D$, and
- $R_n = A \cdot U_s$.

Then, for each v_i in $\mathcal{V}$, the reduction creates τ_i (for $i = 1, 2, \ldots, n-1$) with:

- $S_i = v_i$,
- $C_i = 2v_i$, and
- $R_i = v_i \cdot \rho \cdot U_s$.

The above reduction can be done in polynomial time. Clearly, task τ_n should be offloaded because of the very long local execution time; otherwise, the solution is not feasible. According to Lemma 2, it is also clear that task τ_n should be the first task to be executed in the client.

Suppose that $\mathcal{T}'$ is the set of tasks to be offloaded, in which, by definition, τ_n is in $\mathcal{T}'$. We would like to prove that $\mathcal{T}'$ is a feasible solution for deciding the computation offloading of the reduced problem if and only if $\sum_{\tau_i \in \mathcal{T}'} S_i = A + \delta$. To finish $\mathcal{T}'$ before D in the client, we know that $\sum_{\tau_i \in \mathcal{T}'} S_i + \sum_{\tau_i \in \mathcal{T} \setminus \mathcal{T}'} C_i \leq D$, which implies that

$$\sum_{\tau_i \in \mathcal{T}'} S_i \geq A + \delta. \tag{1}$$

Now, let's focus on the offloaded tasks in the server side. According to Lemma 2, the offloaded tasks should be always executed before the locally executed tasks. As $S_n = \delta$ and $\frac{R_n}{U_s} = A$, we know that the TBS deadline will be set to $A + \delta$ for task τ_n in the server. Therefore, any offloaded tasks that arrive the server before $A+\delta$ will always have to set up the new TBS deadline according to the existing TBS deadline before it arrives. As a result, if $\sum_{\tau_i \in \mathcal{T}'} S_i \leq A + \delta$, we know that the TBS deadline for the last offloaded task will be exactly $\delta + \frac{\sum_{\tau_i \in \mathcal{T}'} R_i}{U_s} = \delta + A + \frac{\sum_{\tau_i \in \mathcal{T}' \setminus \{\tau_n\}} R_i}{U_s} \leq D$. On the other hand, if $\sum_{\tau_i \in \mathcal{T}'} S_i > A + \delta$, we know that the TBS deadline for the last offloaded task will be at least $\delta + \frac{\sum_{\tau_i \in \mathcal{T}'} R_i}{U_s} = \delta + A + \frac{\sum_{\tau_i \in \mathcal{T}' \setminus \{\tau_n\}} R_i}{U_s} > D$. As a result, to meet the deadline constraint in the server side, we need

$$\sum_{\tau_i \in \mathcal{T}'} S_i \leq A + \delta. \tag{2}$$

Therefore, by (1) and (2), we know that there exists a feasible task set $\mathcal{T}'$ for offloading in the reduced input instance of the COTBS problem if and only if there exists $\mathcal{V}' \subseteq \mathcal{V}$ with $\sum_{v_i \in \mathcal{V}'} v_i = A$. Since the COTBS problem is in $\mathcal{NP}$, by the above reduction, we know that the COTBS problem is $\mathcal{NP}$-complete. ■

VI. Our Approach

This section presents our proposed algorithms for the COTBS problem. As determining the feasible solution is $\mathcal{NP}$-complete, we seek for approximations to minimize the makespan of the schedule in Section VI-A. If the makespan is less than or equal to the deadline D, then this schedule is feasible. Section VI-B presents a dynamic programming approach for the COTBS problem with pseudo-polynomial time and space complexity.

A. Approximation Algorithm

In this subsection, we propose a greedy algorithm to minimize the makespan, and show that the algorithm provides a solution with an *approximation factor*, which is defined as the derived makespan divided by the minimum makespan, equals to 2 for any input instance. That is, the makespan of the solution derived from our algorithm is at most twice of the optimal makespan. For notational brevity, we use the following notations in this subsection:

- $Y = \sum_{\tau_i \in \mathcal{T}} S_i$: represents the summation of the setup times of all tasks,
- $a_i = (C_i - S_i)$ for τ_i: the difference between the local execution time of and the setup time,
- $b_i = \frac{R_i}{U_s}$ for τ_i: the minimum response time of task τ_i on the server.

Recall that x_i is equal to 1 if task τ_i is decided to be offloaded; otherwise, x_i is 0. Therefore, we know that the last TBS deadline of the offloaded tasks must be at least $\sum_{\tau_i \in \mathcal{T}} x_i \frac{R_i}{U_s} = \sum_{\tau_i \in \mathcal{T}} x_i b_i$. Moreover, the tasks will finish their setup and local executions on the client side at time $\sum_{\tau_i \in \mathcal{T}} (1 - x_i)(C_i - S_i) + S_i = Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i$. Therefore, the optimal solution M of the following integer linear programming (ILP) provides a lower bound of the optimal makespan:

$$\text{minimize } M \tag{3a}$$

$$\text{s.t } Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i \leq M \tag{3b}$$

$$\sum_{\tau_i \in \mathcal{T}} x_i b_i \leq M \tag{3c}$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, 2, \ldots, n. \tag{3d}$$

Therefore, the relaxation of the integral constraint in (3d) gives a lower bound M^* for the optimal makespan:

$$\text{minimize } M^* \tag{4a}$$

$$\text{s.t } Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i \leq M^* \tag{4b}$$

$$\sum_{\tau_i \in \mathcal{T}} x_i b_i \leq M^* \tag{4c}$$

$$0 \leq x_i \leq 1 \qquad \forall i = 1, 2, \ldots, n. \tag{4d}$$

We can use the linear programming toolkit to find the optimal M^* in (4). Based on the extreme point theory, there exists an optimal solution for (4), in which at most one task τ_j is with $0 < x_j < 1$.

Algorithm 1 Approximation Algorithm

1: $\forall \tau_i \in \mathcal{T}$, $x_i \leftarrow 0$, $Y = \sum_{\tau_i \in \mathcal{T}} S_i$,$a_i = (C_i - S_i)$,$b_i = \frac{R_i}{U_s}$;
2: $\forall \tau_i \in \mathcal{T} | S_i < C_i$, $x_i \leftarrow 1$; and order them according to $\frac{b_i}{a_i}$ in a list $\mathcal{L}$;
3: $r \leftarrow (\sum_{\tau_i \in \mathcal{T}} x_i b_i) - (Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i)$;
4: **while** $r > 0$ **do**
5: pick task τ_j from $\mathcal{L}$ with the max $\frac{b_j}{a_j}$;
6: $r \leftarrow (\sum_{\tau_i \in \mathcal{T} \setminus \{\tau_j\}} x_i b_i) - (Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i)$;
7: **if** $r \geq a_j$ **then**
8: $x_j \leftarrow 0$
9: **else**
10: find x_j s.t. $(\sum_{\tau_i \in \mathcal{T} \setminus \{\tau_j\}} x_i b_i) + x_j b_j = Y + (\sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i) + (1 - x_j) a_j$;
11: break;
12: **end if**
13: $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\tau_j\}$;
14: $r \leftarrow r - a_j$;
15: **end while**
16: $M^* \leftarrow \max \left\{ Y + \sum_{\tau_i \in \mathcal{T}} (1 - x_i) a_i, \sum_{\tau_i \in \mathcal{T}} x_i b_i \right\}$;
17: **if** $\exists \tau_j$ with $0 < x_j < 1$ **then**
18: generate $\vec{x}$ by setting x_j to 0 and generate $\vec{x}^*$ by setting x_j to 1;
19: **if** the makespan of $\vec{x}^*$ is better than the makespan of $\vec{x}$ according to the ordering based on Lemma 2 **then**
20: replace $\vec{x}$ by $\vec{x}^*$;
21: **end if**
22: **end if**
23: order $\{\tau_1, \tau_2, \ldots, \tau_n\}$ according to Lemma 2 based on $\vec{x}$;

Our greedy algorithm, described in Algorithm 1, is based on two main steps: First, it finds the optimal solution M^* based on a greedy approach in which at most one task τ_j will have $0 < x_j < 1$. Second, the offloading decision $\vec{x}$ is decided in which x_j for task τ_j with $0 < x_j < 1$ will be greedily set to test whether setting x_j to 1 or setting x_j to 0 is the better solution.

Fortunately, the linear programming in (4) can be solved by using a simple heuristic. Clearly, only the tasks that may be beneficial for offloading (with $S_i < C_i$) are assigned at the beginning for offloading and ordered into the list $\mathcal{L}$. Let r be a variable that represents the difference between the total execution time (divided by U_s) on the server side and the total execution time on the client side. See Figure 5a. While there is still a positive difference r, the algorithm keeps picking the task τ_j of the maximum value of $\frac{b_j}{a_j}$ and checks if there is enough space for it to be executed locally (Lines 4-7), as shown in Figure 5b. This will maximize the decrease of the value $\sum_{\tau_i \in \mathcal{T}} x_i b_i$ per unit in the decrease of $\sum_{\tau_i \in \mathcal{T}} (1 - x_i) b_i$. If there is enough space for the local execution of the task τ_j, the task is assigned for local execution, removed from the list $\mathcal{L}$ and r is updated (Lines 7,8,13,14), as shown in Figure 5c. Otherwise, the algorithm calculates the value of x_j that achieves the best balance between the client and the server, as illustrated in Figure 5d (Line 10).

The optimal solution M^* for the linear programming in (4) is hence derived. However, because there may exist a task τ_j that may be partially offloaded (i.e., $0 < x_j < 1$), we need to handle such a situation to decide whether x_j is 0 or 1. For such a case, our algorithm tests two possible solutions to know whether it is better to set x_j to 0 or to 1 by testing the ordering according to Lemma 2. The time complexity of the algorithm is $O(n \log n)$.

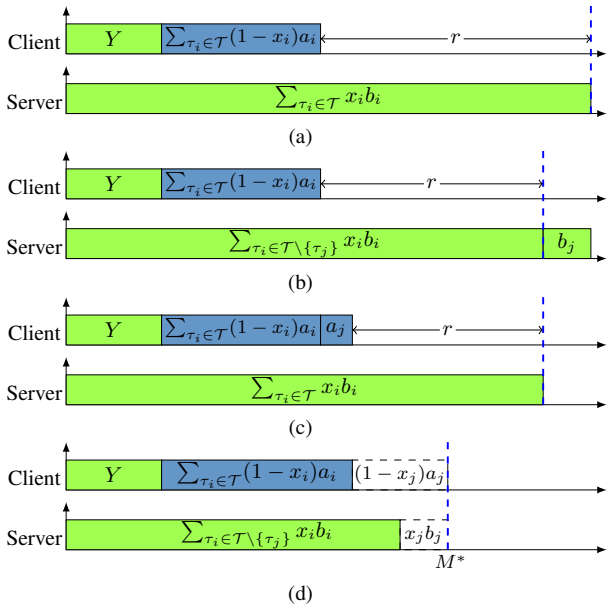


Fig. 5: Illustration of Algorithm 1.

Theorem 2: The M^* derived in Line 16 in Algorithm 1 is an optimal solution for the linear programming in (4), and, hence, a lower bound of the optimal makespan.

Proof: The proof is very similar to the proof for the fractional knapsack problem [8]. The details are omitted due to space limitations. ■

Now, we can prove the approximation factor for optimization of the makespan for Algorithm 1.

Theorem 3: The makespan of the solution derived from Algorithm 1 is at most twice of the makespan of the optimal makespan.

Proof: By Theorem 2, we know that M^* is a lower bound of the optimal makespan. Suppose that there is no task τ_j with $0 < x_j < 1$ in Line 16 in Algorithm 1. Then, we know that the maximum finishing time on the server is at most $2M^*$ and the maximum finishing time on the client is at most M^*. Therefore, the makespan of the resulting solution is at most $2M^*$. We know that the approximation factor is at most 2 for such a case.

For the rest of the proof, we focus on the case that there exists a task τ_j with $0 < x_j < 1$ in Line 16 in Algorithm 1. Note that the value of x_j in the analysis is the fractional value before it is set to integral values after Line 16. Let $A = \sum_{\tau_i \in \mathcal{T} \setminus \{\tau_j\}} x_i b_i$ and $B = Y + (\sum_{\tau_i \in \mathcal{T} \setminus \{\tau_j\}} (1 - x_i) a_i)$. Based on the existence of $0 < x_j < 1$ and Line 16 in Algorithm 1, we know

$$A + x_j b_j = B + (1 - x_j) a_j = M^*.$$

Therefore, we can have $b_j = \frac{(1 - x_j) a_j + B - A}{x_j}$. Now, we can consider the following two cases:

- Case 1: $0 < x_j \leq \frac{\frac{B}{a_j} + 1}{2}$: Let's consider the case that task τ_j is executed locally (i.e., $\vec{x}$ in Line 18 in Algorithm 1).

For such a solution, the finishing time on the client is at most $B+a_j$, while the finishing time on the server is at most $B+A$. Therefore, the makespan of the resulting solution is at most $\max\{B+A, B+a_j\}$. The approximation factor is $\max\{\frac{B+A}{M^*}, \frac{B+a_j}{B+(1-x_j)a_j}\}$. Clearly, as $B \leq M^*$ and $A \leq M^*$, we know that $\frac{B+A}{M^*} \leq 2$. Moreover, $\frac{B+a_j}{B+(1-x_j)a_j}$ is an increasing function with respect to x_j. Therefore, $\frac{B+a_j}{B+(1-x_j)a_j}$ is maximized when x_j is equal to $\frac{\frac{B}{a_j}+1}{2}$. Then the approximation factor for such a case is,

$$\begin{aligned} &\max\{\frac{B+A}{M^*}, \frac{B+a_j}{B+(1-x_j)a_j}\} \\ \leq &\max\{2, \frac{B+a_j}{B+(1-\frac{\frac{B}{a_j}+1}{2})a_j}\} = 2. \end{aligned} \tag{5}$$

- Case 2: $\frac{\frac{B}{a_j}+1}{2} < x_j < 1$: Let's consider the case that task τ_j is offloaded (i.e., $\vec{x}^*$ in Line 18 in Algorithm 1). For such a solution, the finishing time on the client is at most B, while the finishing time on the server is at most $B+A+b_j$. Therefore, the makespan of the resulting solution is at most $B+A+b_j$. The approximation factor for this case is at most $\frac{B+A+b_j}{B+(1-x_j)a_j} = \frac{B+A+\frac{(1-x_j)a_j+B-A}{x_j}}{B+(1-x_j)a_j}$. For notational brevity, let $A=\gamma a_j$ and $B=\beta a_j$, in which $\gamma \geq 0$ and $1 > \beta > 0$. (If $\beta > 1$, this case is never reached since $1 < x_j < 1$ is a contradiction. Moreover, $\beta > 0$ due to the setting that $S_i > 0$ for all tasks $\tau_i \in \mathcal{T}$.) Therefore, the approximation factor can be rewritten as

$$\begin{aligned} & \frac{(\beta+\gamma)x_j + (1-x_j) + (\beta-\gamma)}{x_j(\beta+(1-x_j))} \\ = & \frac{\beta x_j + \beta + 1 - x_j}{x_j(\beta+1-x_j)} + \frac{\gamma x_j - \gamma}{x_j(\beta+1-x_j)} \\ \leq & \frac{\beta x_j + \beta + 1 - x_j}{x_j(\beta+1-x_j)}, \end{aligned} \tag{6}$$

where the inequality comes from the fact that $\frac{\gamma x_j - \gamma}{x_j(\beta+1-x_j)} \leq 0$ when $x_j < 1$ and $\gamma \geq 0$. The second order derivative of $\frac{\beta x_j+\beta+1-x_j}{x_j(\beta+1-x_j)}$ with respect to x is equal to $\frac{2}{x_j^3} + \frac{2\beta}{(\beta+1-x_j)^3}$, which is positive when $\frac{\beta+1}{2} < x_j < 1$ and $\beta > 0$. Therefore, $\frac{\beta x_j+\beta+1-x_j}{x_j(\beta+1-x_j)}$ is maximized when x_j is either $\frac{\beta+1}{2}$ or 1. When x_j is 1, we have $\frac{\beta x_j+\beta+1-x_j}{x_j(\beta+1-x_j)} = 2$. When x_j is $\frac{\beta+1}{2}$, we have $\frac{\beta x_j+\beta+1-x_j}{x_j(\beta+1-x_j)} = 2$. Therefore, the approximation factor for such a case is also 2.

According to all the cases above, as we choose a better solution between $\vec{x}$ and $\vec{x}^*$ in Line 20 in Algorithm 1, we reach our conclusion of the theorem. ■

The above analysis in Theorem 3 is tight by considering the following example with one task, in which $C_1 = 1+\epsilon, S_1 = 1, \frac{R_1}{U_s} = 1$ with $\epsilon > 0$. This task will be offloaded in Algorithm 1, with a resulting schedule with makespan 2, while the optimal solution for minimizing the makespan is to execute this task locally with makespan equals to $1+\epsilon$. When ϵ approaches to 0, the above example gives the tightness of our analysis.

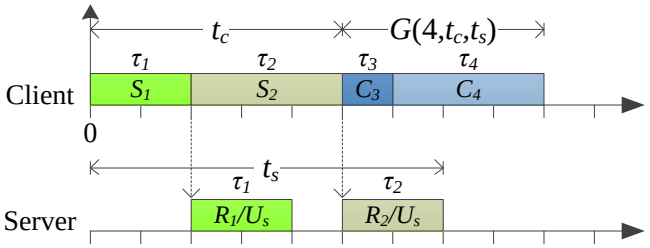


Fig. 6: An example for illustrating the dynamic programming parameters.

B. Dynamic Programming Algorithm

In this subsection, we present a pseudo-polynomial-time scheduling algorithm based on dynamic programming. Our proposed algorithm derives a feasible solution for the COTBS problem if it exists.

At the beginning, in the dynamic programming approach, the tasks $\tau_i \in \mathcal{T}$ for $i = 1, 2, ..., n$ are ordered according to Lemma 2, by supposing that all of them are offloaded. Then, a dynamic programming table is built to maintain the offloading decision for each subproblem, i.e., the first i tasks $\{\tau_1, \tau_2, \ldots, \tau_i\}$. Consider the subproblem for the first i tasks $\{\tau_1, \tau_2, \ldots, \tau_i\}$. Let $G(i, t_c, t_s)$ be the minimum total local execution time for the locally executed tasks under the following two constraints:

- The total setup time for the offloaded tasks among the first i tasks is less than or equal to t_c.
- The finishing time of all offloaded tasks among the first i tasks on server is less than or equal to t_s.

Figure 6 presents an example of four tasks $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ with the dynamic programming parameters, where $\{\tau_1, \tau_2\}$ are offloaded and $\{\tau_3, \tau_4\}$ are executed locally.

A 3-dimensional table $G(i, t_c, t_s)$ is constructed for all values of i, t_c and t_s, where $0 \leq i \leq n$, $0 \leq t_c \leq D$ and $0 \leq t_s \leq D$. We start by initializing all the elements of $G(0, t_c, t_s)$ to zeros. Then, the following recursion is used to fill the table for i from 1 to n:

$$G(i, t_c, t_s) = \min \begin{cases} \begin{cases} G(i-1, t_c - S_i, t_s - \frac{R_i}{U_s}) & \text{if } S_i < C_i \wedge t_c \geq S_i \\ & \wedge t_s \geq t_c + \frac{R_i}{U_s} \\ \infty & \text{otherwise} \end{cases} \\ G(i-1, t_c, t_s) + C_i \end{cases} \tag{7}$$

For each subproblem $\{\tau_1, \tau_2, \ldots, \tau_i\}$ and available times t_c and t_s, the algorithm chooses the offloading decision (offload or local) for τ_i that minimizes the total local execution time. The task τ_i is able for offloading if it is *beneficial* (i.e., $S_i < C_i$) and *feasible* (i.e., $t_c \geq S_i \wedge t_s \geq t_c + \frac{R_i}{U_s}$). Otherwise, offloading task τ_i will violate the timing specification in

finishing time t_s, and, hence, it is assigned for local execution. If the task τ_i is chosen for local execution, its execution time C_i is added to the total local execution time of the previous subproblem ($G(i-1, t_c, t_s) + C_i$). If it is offloaded, its setup time S_i and remote execution time $\frac{R_i}{U_s}$ are considered from the available times t_c and t_s of the previous subproblem respectively. The algorithm also stores the offloading decision for each task for further backtracking.

The following lemma shows the optimality the recursive function for building $G(i, t_c, t_s)$.

Lemma 3: Given, i, t_c, t_s, the recursive function in 7 minimizes $G(i, t_c, t_s)$, when C_j, S_j, and $\frac{R_j}{U_s}$ are all integers for τ_j in $\mathcal{T}$.

Proof: According to Corollary 1, Lemma 2, and our initial ordering by adopting Johnson's rule before starting the dynamic programming, it is clear that (1) if τ_i is offloaded, τ_i should execute its setup time before any task τ_j with $j > i$ on the client no matter whether task τ_j is offloaded or not, and (2) if τ_i is locally executed, τ_i should execute its local execution time after all the offloaded tasks finish their setup time on the client. Therefore, to satisfy the constraint of t_s and t_c, we do not have to consider any task τ_j with $j > i$. Moreover, it is also clear that the best decision to minimize the total local execution time for the locally-executed tasks for the first i tasks depends on the sub-optimality for the first $i-1$ tasks. The sub-optimality property can be easily proved by the induction hypothesis. The details are omitted due to space limitations. ■

After filling the table, we can verify whether a feasible schedule exists for the COTBS problem or not. We just have to check if there exists $t_c \leq D$ and $t_s \leq D$ in the table such that $\max\{(G(n, t_c, t_s) + t_c), t_s\} \leq D$. Then, 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 $G(i-1, t_c, t_s)$. (2) If τ_i is assigned for offloading, we backtrack to $G(i-1, t_c - S_i, t_s - \frac{R_i}{U_s})$.

Clearly, if the objective is further to minimize the makespan, we can find the minimum of $\max\{(G(n, t_c, t_s) + t_c), t_s\}$. The time complexity of the above dynamic programming algorithm is $O(n \log n + nD^2)$ when C_i, S_i, and $\frac{R_i}{U_s}$ are all integers for τ_i in $\mathcal{T}$. We conclude the section with the following theorem.

Theorem 4: The dynamic programming algorithm can find a feasible schedule to minimize the makespan, if and only if feasible schedules exist.

Proof: This comes directly from the dynamic programming scheme and the sub-optimality property shown in Lemma 3. ■

VII. Experimental Evaluation and Simulation

In this section, we evaluate our algorithms by implementing a case study of a surveillance system, and synthesis workload simulation. We use the abbreviation *Approximation* to refer to Algorithm 1, *DP* to refer to the dynamic programming algorithm in Section VI-B, and *Offload-Wait* [16] to refer to the algorithm in which the offloading decision is taken based on the comparison between the data transfer time added to the remote execution time divided by the TBS bandwidth, and the local execution time. To show the effectiveness, we report the resulting makespan by adopting all these three algorithms.

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

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

A. Case Study of a Surveillance System

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

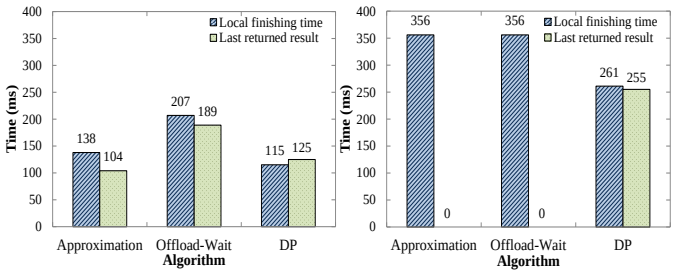
- **Motion Detection**: Detects the moving objects.
- **Object Recognition**: Recognizes and tracks a given object.
- **Stereo Vision**: Calculates the distance between the camera and the object of interest by generating a depth map for left and right images.
- **Motion Recording**: Records the video 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 time is measured in milliseconds. If all the tasks are locally executed, the finishing time on the client is by 356 ms. Offloading algorithms are implemented to reduce the makespan.

Table II shows the offloading decisions of the evaluated algorithms for $U_s = 1$ and $U_s = 0.25$. Based on these decisions, the finishing times on the client and on the server are presented in Figure 7. For $Us = 1$, which represents a dedicated server, all of the algorithms offload task τ_2 and execute τ_4 locally. Offload-Wait also offloads τ_1 and τ_3 because the setup time plus the remote execution time is less than the local execution time for these offloaded tasks. Moreover, Offload-Wait keeps the system idle during offloading. Therefore, the reduction of the makespan (comparing to the pure local executions) by Approximation and DP is more than the reduction by the Offload-Wait, as shown in Figure 7a. For $Us = 0.25$, the server provides offloading services for four clients. In this case, both Offload-Wait and Approximation do not offload any task. But, DP offloads τ_1 and τ_3, and then reduces the makespan of the schedule, as shown in Figure 7b.

B. Simulation Setup and Results

We also evaluate the offloading algorithms using synthetic workload for the tasks. The parameters of the tasks are summarized as follows:



(a) $U_s = 1$ (One client). (b) $U_s = 0.25$ (four clients).

Fig. 7: Case study results.

TABLE II: The Offloading decisions of the algorithms.

τ_i	Approximation		Offload-Wait		DP	
	Us:1	Us:0.25	Us:1	Us:0.25	Us:1	Us:0.25
τ_1	0	0	1	0	1	1
τ_2	1	0	1	0	1	0
τ_3	0	0	1	0	0	1
τ_4	0	0	0	0	0	0

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

Each value of $U_s = \{0.1, 0.111, 0.125, 0.143, 0.167, 0.2, 0.25, 0.333, 0.5, 1\}$ is combined with all values of $\alpha = \{0.25, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ to get a total of 120 configurations for simulations. We perform 100 rounds for each configuration. In each round, a set of 25 frame-based real-time tasks is randomly generated according to the conditions above.

The *normalized finishing time reduction* of an algorithm for a task set is the finishing time (makespan) for the task set execution of the derived schedule divided by the finishing time for the same task set if all the tasks are executed locally. For the rest of this subsection, we will discuss the simulation results for the offloading algorithms using the task sets described above.

Figure 8 shows the average normalized finishing time reduction for all values of U_s and α. In general, as the α value increases, the makespan decreases for all algorithms. Because with a faster server (higher α values), the offloaded tasks are finished earlier. Also, The makespan decreases more rapidly for larger values of U_s. The best reduction is when $U_s = 1$, i.e., when the server is dedicated for one client.

Figure 9 presents a comparison of the normalized finishing time reduction among the three offloading algorithms for $U_s = \{1, 0.25, 0.1\}$ and all values of α. Offload-Wait reduces the local execution time only when the server is faster than the client. So, there is no reduction when $\alpha \leq 1$, $\alpha \leq 4$ and $\alpha \leq 10$ for the corresponding values of $U_s = \{1, 0.25, 0.1\}$ respectively. In contrast, Approximation and DP reduce the makespan even by offloading to a slower server ($\frac{R_i}{U_s}$ is larger than C_i), while the offloading decision is feasible. The figure also shows that the improvement by DP is more than that by Approximation, because the DP algorithm considers the slack time and finds the optimal minimum makespan.

VIII. Conclusion

In this paper, we explore the idea of computation offloading by exploiting the total bandwidth server (TBS) in the server side to guarantee the worst-case behavior when the tasks are offloaded to the server side. We show that the difficulty of the problem comes from the decisions to determine which tasks to be offloaded. We propose an approximation algorithm to make decisions greedily and develop a dynamic-programming approach to make the decisions optimally. The evaluation, using a case study of surveillance system and synthesized benchmarks, shows that our algorithms also improve the performance even with a slower remote processing unit by virtual parallel execution of the tasks in the client and the server.

The dynamic programming approach can also be extended to allow a fully polynomial-time approximation scheme (FPTAS) for minimizing the makespan, to trade the complexity with the error. We do not present the FPTAS due to space limitations. We focus ourselves on the case that $0 \leq U_s \leq 1$ is already specified for the client. Alternatively, a corresponding dual problem is to minimize U_s under the timing constraints. The proposed algorithms in this paper can be iteratively adopted to search a minimal U_s as well. However, the $\mathcal{NP}$-completeness proof in Theorem 1 also implies the difficulty of approximation when the optimal solution has $U_s = 1$.

For future researches, we will explore systems with periodic/sporadic real-time tasks, other resource reservation servers on the server side, and offloading schemes to have more than one server for a client.

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

References

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

[2] C.-Y. Chen, B.-Y. Shih, C.-H. Shih, and W.-C. Chou. The development of autonomous low-cost biped mobile surveillance robot by intelligent bricks. *Journal of Vibration and Control*, 18(5):577–586, 2012.

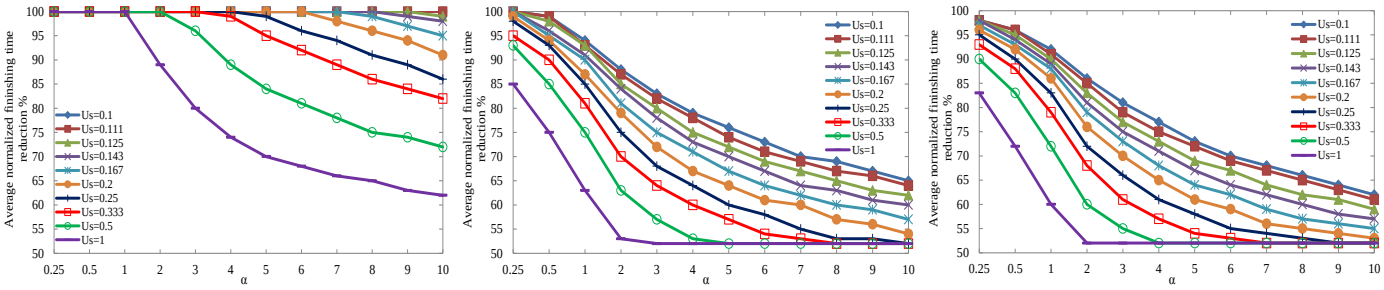
[3] J. Fernandez, D. Losada, and R. Sanz. Enhancing building security systems with autonomous robots. In *IEEE International Conference on Technologies for Practical Robot Applications*, pages 19 –24, 2008.

[4] 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.

[5] 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.

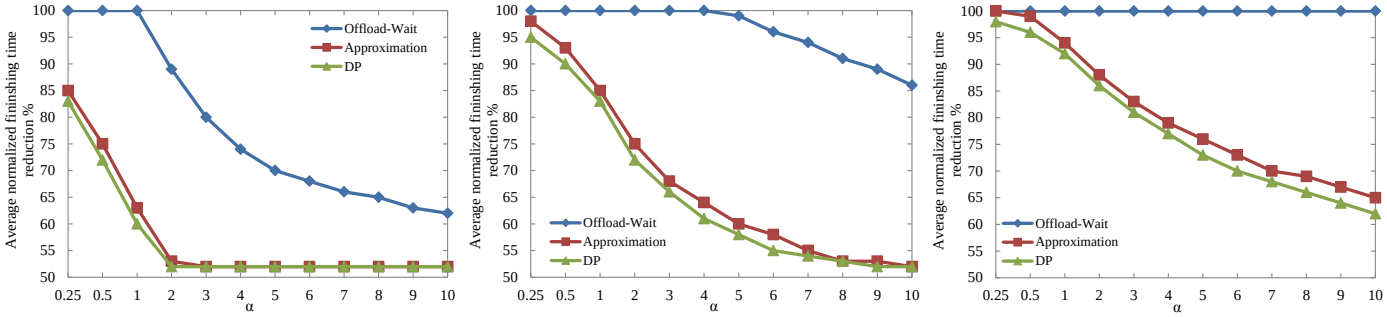
[6] S. Gurun, R. Wolski, C. Krintz, and D. Nurmi. On the efficacy of computation offloading decision-making strategies. *Int. J. High Perform. Comput. Appl.*, 22(4):460–479, 2008.

[7] Y.-J. Hong, K. Kumar, and Y.-H. Lu. Energy efficient content-based image retrieval for mobile systems. In *IEEE International*



(a) Finishing time reduction for Offload-Wait. (b) Finishing time reduction for Approximation. (c) Finishing time reduction for DP.

Fig. 8: Finishing time reduction for all algorithms.



(a) $U_s = 1$ (For one client). (b) $U_s = 0.25$ (For four fair-sharing clients). (c) $U_s = 0.1$ (For ten fair-sharing clients).

Fig. 9: Finishing time reduction for $U_s = 1$, 0.2, and 0.1.

Symposium on Circuits and Systems (ISCAS), pages 1673 – 1676, 2009.

[8] E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms*. W. H. Freeman and Company, NY, USA, 1st edition, 1996.

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

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

[11] 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.

[12] 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.

[13] 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.

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

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

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

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

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

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

[20] A. Toma and J.-J. Chen. Computation offloading for real-time systems. In *ACM 28th Symposium On Applied Computing (SAC)*, page to appear, 2013.

[21] 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.

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

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

[24] J. White, H. Harvey, and K. Farnstrom. Testing of mobile surveillance robot at a nuclear power plant. In *IEEE International Conference on Robotics and Automation. Proceedings*, volume 4, pages 714 – 719, 1987.

[25] 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.

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

[27] 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.