An Energy-Efficient Middleware for Computation Offloading in Real-Time Embedded Systems*

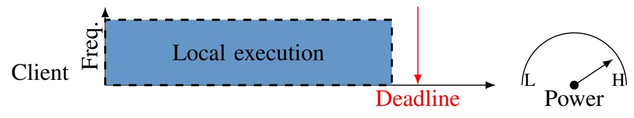
Anas Toma[1], Santiago Pagani[2], Jian-Jia Chen[1], Wolfgang Karl[2] and Jörg Henkel[2]
[1] Department of Informatics, TU Dortmund University, Germany
[2] Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany
E-mail: {anas.toma, jian-jia.chen}@tu-dortmund.de, {santiago.pagani, karl, henkel}@kit.edu

***Abstract*—Embedded systems have limited resources, such as computation capabilities and battery life. The *Dynamic Voltage and Frequency Scaling* (DVFS) technique is used to save energy by running the processor of the embedded system at low voltage and frequency levels. However, this prolongs the execution time, which may cause potential deadline misses for real-time tasks. In this paper, we propose a general-purpose middleware to reduce the energy consumption in embedded systems without violating the real-time constraints. The algorithms in the middleware adopt the computation offloading concept to reduce the workload on the processor of the embedded system by sending the computation-intensive tasks to a powerful server. The algorithms are further combined with the DVFS technique to find the running frequency (or speed) such that the energy consumption is minimized and the real-time constraints are satisfied. The evaluation shows that our approach reduces the average energy consumption down to nearly 60%, compared to executing all the tasks locally at the maximum processor speed.**

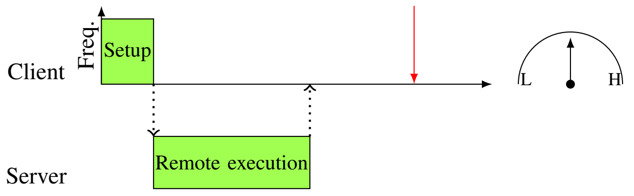
I. Introduction

Embedded systems have become very widespread and used nearly everywhere. Nowadays, over 96% of the computer chips are produced to be used in these systems, while the remaining production of 4% is used in the normal computers [1]. Embedded systems have recently been used to run more and more complex applications that include computation-intensive data processing such as video and audio processing in smart phones, wearable devices, mobile robots, etc. However, the gap between the demand for running complex programs and the amount of the available resources is on the rise [8]. Although embedded systems continue to evolve and improve, their computation and power resources will stay limited. Therefore, an embedded system may become slow and not be able to finish all the tasks in time (or not be able to execute some of them at all), specially the computation-intensive tasks. For instance, the battery of a portable or mobile embedded system may not be powerful enough to finish some important tasks at a certain time, and may be depleted before finding a power source.

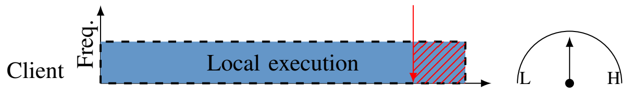
Upgrading or adding more hardware components to prolong the battery life will mostly increase the weight and the cost of the embedded system. Additionally, an embedded system may be designed in such a way that its components cannot be replaced or extended at all. Hence, the need arises to reduce the energy consumption in embedded systems and use their resources efficiently. *Dynamic Voltage and Frequency Scaling* (DVFS) is an efficient technique to reduce the power consumption by running the processor at a low frequency [4]. However, this prolongs the execution time, which may cause potential deadline misses for real-time tasks. Timing requirements are very important for real-time embedded systems, in which the results may become useless or even harmful if the deadlines are not met. To overcome such a problem, we propose a middleware that performs computation offloading to reduce the energy consumption in embedded systems without violating the real-time constraints. The algorithms in the middleware are further combined with the DVFS to minimize the total energy consumption.



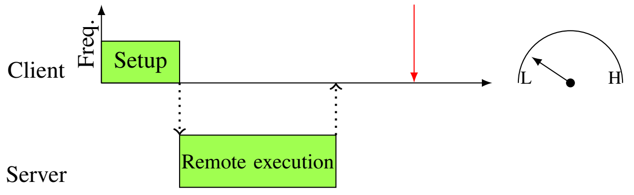
(a) Running at a high frequency.



(b) The task is offloaded.



(c) Running at a lower frequency.



(d) Running at a lower frequency with computation offloading.

Fig. 1: Computation offloading and DVFS example.

Computation offloading reduces the workload on the processor of the embedded system (referred to as *client*) by moving the computation-intensive tasks to a powerful remote processing unit (referred to as *server*). The server executes the offloaded tasks and returns the results back to the client. Energy savings come from the idle or sleep state of the processor during the remote execution. To our knowledge, this is the first study that adopts computation offloading to reduce the energy consumption for real-time systems. The combination with the DVFS can even save more energy, by executing the remaining workload on the processor at a low frequency. Figure 1 illustrates computation offloading and DVFS techniques, where the blue task with a dashed frame

*Published in the 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2016), Daegu, South Korea, August 2016. https://doi.org/10.1109/RTCSA.2016.50

is executed on the client and the green one is offloaded to the server. The symbolic power meter on the right side roughly indicates the power consumption. Figure 1b shows that offloading the task in Figure 1a reduces the power consumption by reducing the processing time on the client side, where the processor consumes idle power during the remote execution. The power consumption is also reduced by running at a lower frequency than in Figure 1a, using DVFS as shown in Figure 1c. As a consequence, the local task in Figure 1c misses its deadline. Figure 1d shows that the combination of both techniques saves more energy without missing the deadline.

Our Contribution: In this paper, a middleware is proposed to reduce the energy consumption in real-time embedded systems based on the computation offloading. Our contribution can be summarized as follows:

- In our model, the server can serve more than one client and provides response time guarantees for the offloaded tasks.
- We propose a set of offloading algorithms for sporadic and frame-based real-time tasks. The algorithms schedule the real-time tasks and decide which tasks should be offloaded to reduce the energy consumption on the client without violating the real-time constraints.
- We further combine the offloading algorithms with DVFS to determine the frequency (or speed) of the client's processor, such that the total energy consumption is minimized and all the tasks meet the deadline.
- The middleware is evaluated using a real world case study of a surveillance system in addition to randomly synthesized benchmarks.

II. Literature Review

This section summarizes the recent studies in the field of computation offloading to save energy. We also discuss the limitations of the existing approaches. Patra et al. [12] present a framework that performs computation offloading to save energy in handheld devices. The tasks or applications are offloaded from the handheld device to a selected resourceful host in a P2P network of smart devices. Applications are chosen for offloading based on some of their parameters such as CPU cycles, memory usage, energy consumption and input/output data. Liu et al. [11] explore the computation offloading technique by using timing unreliable components in real-time systems. They utilize these components in hard real-time systems by estimating the worst-case response time and then provide local compensations if the unreliable components do not deliver the results within the estimated response time.

The main idea in [9, 10] is to represent the computation offloading problem as a graph partitioning problem, where the first partition represents the client side and the second one represents the server side. Each vertex in the graph is combined with the energy cost of the execution, and represents a task. The edges between the vertices are combined with the energy cost of the communication between them. The proposed algorithms divide the graph into two parts in order to minimize the communication cost and the total cost on one side.

Khairy et al. [6] propose a "Smartphone Energizer" technique for context-aware computation offloading in order to extend the battery life of the smart phones. 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 of the execution on the server is less than the one on the client. The offloading decision in [7] is represented as an optimization problem based on different parameters such as CPU load, available memory, remaining battery, and the bandwidth. Integer linear Programming is used to solve the problem on the mobile device. Then, the computation-intensive tasks are offloaded to a remote cloud.

In most of the current studies, the offloading decision is either based on: (1) the comparison between the energy consumption of the local execution and the offloading for each task alone, or (2) the partitioning of the graph that represents the tasks combined with their energy consumption for the local and remote execution [8]. The first approach may not be optimal if we consider all of the system tasks together. None of both approaches considers the server model, or how the server 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. Furthermore, most of the approaches with computation offloading 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.

III. System Model and Problem Definition

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

A. Server Model

In our system model, we suppose that the server can serve more than one client. Therefore, 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) [13] is used as a resource reservation server to manage the sharing of the server processor, and then preserve the real-time property of the system. The server reserves a specific utilization (or bandwidth) for each requesting client, if it is possible. Based on the reserved utilization, the client schedules the tasks on its side and determines their offloading decisions. The offloading decisions on the client do not control or influence how the server schedules the offloaded tasks. After that, the server uses Earliest Deadline First (EDF) algorithm to schedule the offloaded tasks (from one or more than one client) according to their absolute deadlines assigned by the TBS. The total given utilization for all the clients should not exceed 100% in order to preserve the system feasibility. Any other resource reservation server can be used to provide the response time guarantee to the offloaded tasks.

B. Client and Task Models

On the client side, a set of real-time tasks arrive periodically and require execution in time. A task τ_i represents an execution unit, and consists of an infinite sequence of identical instances, called jobs. We have two types of task models: Sporadic and frame-based task models. Based on the utilization given from the server, the client determines the speed of its processor, schedules the tasks and decides which

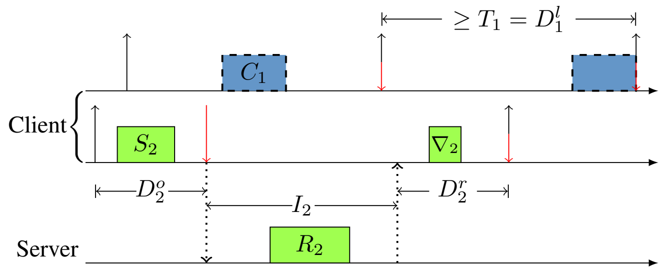


Fig. 2: Timing parameters for two sporadic real-time tasks.

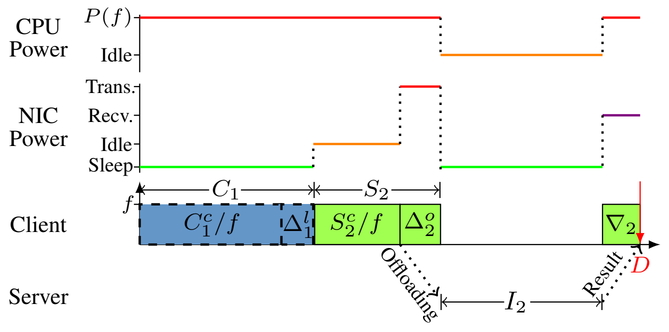


Fig. 3: Timing and power parameters for two frame-based real-time tasks.

of them should be offloaded in order to minimize the energy consumption and meet the real-time constraints. The tasks are independent in execution without precedence constraints.

1) Sporadic Real-time Task Model: Given a set $\mathcal{T}$ of n independent sporadic real-time tasks. Suppose that f is the frequency (or the speed) of the client's processor. Each task $\tau_i \in \mathcal{T}$ (for $i = 1, 2, \ldots, n$) is characterized by the following worst-case timing parameters:

- C_i: **Local execution time** on the client side. It can be represented as $C_i = \frac{C_i^c}{f} + \Delta_i^l$, where:
  - C_i^c is the execution cycles on the client side, i.e. **local execution cycles**.
  - Δ_i^l is the amount of **frequency-independent execution time** required on the client side in the case of local execution.
- S_i: **Setup time**. It can be represented as $S_i = \frac{S_i^c}{f} + \Delta_i^o$, where:
  - S_i^c is the amount of execution cycles required on the client side to be ready for offloading, i.e. **setup cycles**. It includes any preprocessing operations such as encoding and compression.
  - Δ_i^o is the amount of **frequency-independent execution time** required on the client side during setting up in the case of offloading, e.g. transfer time of the task from the client to the server.
- R_i: **Remote execution time**. The execution time on the server side.
- ∇_i: **Reception time**. The amount of time required to receive the result of an offloaded task τ_i from the server.
- D_i: **Relative deadline**, which is equal to either:
  - **local relative deadline** D_i^l: The deadline for executing the task on the client in the case of local execution.
  - **offloading relative deadline** D_i^o: The deadline for setting up the task in the case of offloading. Another deadline, called **reception relative deadline** D_i^r, is assigned for receiving the result from the server.
- T_i: **Minimum inter-arrival time**. The minimum period of time between the arrival of two consecutive jobs of the same task.
- I_i: **Remote response time**. The interval length starting from the time when the task arrives to the server until the time when the client starts receiving the result from the server.

We say that a task τ_i is *executed locally* if it is executed with C_i amount of time on the client. We say it is *offloaded* if it is executed S_i amount of time on the client for setting up, and then its result returns back from the server after I_i amount of time. Also, it requires ∇_i amount of time to receive the result. Figure 2 shows the timing parameters for an example of two sporadic real-time tasks, where the first task (the blue one with a dashed frame) is executed locally and the second one (the other green one) is offloaded to the server.

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. Each task τ_i should be executed within its local relative deadline D_i^l. In the case of offloading, where the task is executed on the server side, the result should be available on the client within this deadline. 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.

In many systems, the size of the result is much smaller than the size of the data sent to the server. Thus, the offloaded task requires a very short reception time and considered negligible compared to the transmission time of the data. For instance, in the systems that include image processing tasks, the size of the result (e.g. coordinates or distance) is very small compared to the size of the image sent to the server.

2) Frame-based Real-time Task Model: Frame-based real-time task model is a special case of the sporadic task model in III-B1. A set $\mathcal{T}$ of n independent frame-based real-time tasks arrive periodically at the same time $t = 0$, have the same period T, and require execution within the same relative deadline D, where $T = D$. The results of all offloaded tasks should be available on the client within the deadline D. Figure 3 shows the timing parameters for an example of two frame-based real-time tasks, where the first task (the blue one with a dashed frame) is executed locally and the second one (the other green) is offloaded to the server.

Such a model is widely adopted in surveillance, exploration, environmental inspection, and medical diagnosis systems. It also exists in wearable devices and multimedia applications. For instance, several image processing tasks should be carried out once an image is captured, and all the tasks should be completed before the next capture (i.e., the relative deadline). The execution sequence of the tasks continues along the video stream in a periodic manner. Another example can be found in the exploration and inspection systems. These systems process and analyze the collected data from different sensors periodically. Although this task model is a special case of the general task model , i.e., the sporadic task model, we show that minimizing the energy consumption using this model is $\mathcal{NP}$-hard.

3) Response Time Guarantee I_i: Suppose that the given utilization from the server is equal to U_s. We assume that the client distributes the utilization U_s equally among all of its tasks. Then, a TBS is assigned for each task with a utilization equals to ψ_i, where $\psi_i = \frac{U_s}{n}$. For the task τ_i, the speed of the given TBS seems $\frac{1}{\psi_i}$ times slower than the original speed of the server. According to this model, the value of I_i can be calculated as $I_i = \frac{R_i}{\psi_i}$. This model provides response time guarantee for all of the offloaded tasks. It can be further improved by choosing a subset of tasks $\mathcal{T}_o \in \mathcal{T}$ to be nominated for offloading according to a specific heuristic, e.g., the amount of energy reduction for each task if it is offloaded. Then the utilization from the server U_s is distributed just among the tasks of the set $\mathcal{T}_o$, and the algorithm chooses tasks to be offloaded just from this nominated set.

C. Power Model

We assume that the system on the client can change the voltage and the frequency of its processor by adopting the DVFS, where the available frequencies are in the range of $[f_{min}, f_{max}]$. We denote the power consumption function of the client's processor executing a task at frequency f as $P(f)$. It can be represented by $P(f) = \alpha f^{\gamma} + \beta$, where α is a constant depends on the effective switching capacitance, γ is a constant related to the hardware, and β represents the static power consumption [16]. We assume that the idle power of the processor is fixed and represented by P^{idle}_{cpu}. The *Network Interface Card* (**NIC**) consumes P^{sleep}, P^{idle}, $P^{trans.}$ and $P^{recv.}$ power during its sleep, idle, transmit and receive states respectively. The energy consumption of executing a task τ_i at frequency f can be represented as follows:

- In the case of local execution, i.e., $x_i = 0$:

$$E_l(f, \tau_i) = P(f) \cdot (\frac{C^c_i}{f} + \Delta^l_i)$$

- In the case of offloading, i.e., $x_i = 1$:

$$E_o(f, \tau_i) = P(f) \cdot (\frac{S^c_i}{f} + \Delta^o_i + \nabla_i) + E^o_i$$

  - E^o_i: Energy consumed by the network card in the case of offloading. It includes the energy consumed by the idle state during the setting up, transmitting the data to the server, and receiving the result from the server i.e., $E^o_i = P^{idle} \cdot \frac{S^c_i}{f} + P^{trans.} \cdot \Delta^o_i + P^{recv.} \cdot \nabla_i$. It may also include any other amount of energy consumed by the network card in the case of offloading, such as the wake-up energy.

Figure 3 shows the power consumption of the client's CPU and the NIC during the execution of two frame-based tasks. We assume that the timing and the power parameters are given based on the system setup. If all these parameters are specified for (the upper bounds of) the worst cases, and the communication fabric between the client and the server is timing predictable, the proposed algorithms provide hard real-time guarantees. Otherwise, if the information is based on estimation, they are more suitable for soft real-time systems and we would like to meet the timing constraint by exploiting the services provided from the server.

D. Problem Definition

In our model, the client schedules the tasks and decides which of them should be offloaded. The tasks will be partitioned into two sets: (1) local tasks and (2) offloaded tasks. A schedule is considered to be *feasible* if the timing constraints of all locally-executed and offloaded tasks are satisfied. In this paper, we address the problem of **Min**imizing the **En**ergy consumption (*MinEng*) of the embedded real-time systems with the help of the computation offloading technique. *Given a set $\mathcal{T}$ of n real-time tasks and a bandwidth U_s provided from the server, the MinEng problem is to find a feasible schedule (including the offloading decisions of the tasks) and determine the frequency of the processor so that the energy consumption of the client is minimized.* The problem can be divided into two subproblems according to the task model as follows:

- Minimize the energy consumption for frame-based tasks.
- Minimize the Energy consumption for sporadic tasks.

As making offloading decisions to satisfy timing constraints for a set of tasks is $\mathcal{NP}$-complete [14], further optimization in energy consumption will not be easier. As a result, the problem addressed in this paper is $\mathcal{NP}$-hard.

IV. Our Approach

In this section, we propose various algorithms to schedule real-time tasks on the client side and decide which of them should be offloaded to the server, because it is difficult to propose just one algorithm to fulfill all system requirements and to handle all task models. According to our system model, the client requests an offloading service from the server to reduce the energy consumption. The server provides the client with a specific utilization called U_s, if it is available. The client uses the given utilization to calculate the remote response time I_i of the tasks as shown in Subsection III-B3. Then, the proposed algorithms in this section can be used on the client to find a feasible schedule for the tasks (including their offloading decision) and determine the processor speed such that the energy consumption is minimized, if it is possible. If any input parameter changes (e.g. number of the tasks, the deadline, the given utilization, etc), the algorithms are executed again to find a new feasible schedule with the minimum energy consumption.

A. Dynamic Programming Algorithm for Frame-based Tasks

The proposed **D**ynamic **P**rogramming algorithm in this subsection, called *DPF* algorithm, finds a feasible schedule that minimizes the energy consumption for **F**rame-based real-time tasks. For a given frequency f, the objective of the

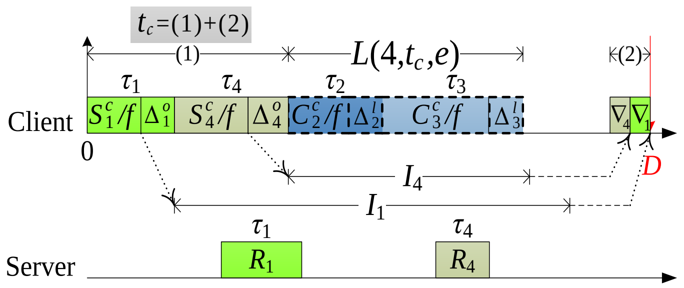


Fig. 4: An example illustrates the sequence, the timing and the dynamic programming parameters of four frame-based tasks.

algorithm can be represented as follows:

$$\text{minimize} \sum_{j=1}^{n} x_j E_o(f,\tau_j) + \sum_{j=1}^{n}(1-x_j)E_l(f,\tau_j)$$

$$= \sum_{j=1}^{n} x_j E_j^o +$$

$$P(f)\cdot\Big[\sum_{j=1}^{n} x_j(\frac{S_j^c}{f}+\Pi_j) + \sum_{j=1}^{n}(1-x_j)(\frac{C_j^c}{f}+\Delta_j^l)\Big], \tag{1a}$$

such that

$$\Big[\sum_{j=1}^{n} x_j(\frac{S_j^c}{f}+\Pi_j) + \sum_{j=1}^{n}(1-x_j)(\frac{C_j^c}{f}+\Delta_j^l)\Big] \le D \tag{1b}$$

$$x_k I_k + \sum_{j=1}^{k} x_j(\frac{S_j^c}{f}+\Pi_j) \le D, \qquad \forall k = 1,2,\dots,n. \tag{1c}$$

Where $\Pi_j = \Delta_j^o + \nabla_j$ for notational brevity. For the tasks with a negligible reception time, $\Pi_j = \Delta_j^o$. The two conditions in (1b) and (1c) test the feasibility of the schedule. The condition in (1b) guarantees that the processing time on the client side for both local and offloaded tasks finishes within the deadline D, where the condition in (1c) guarantees that the results for all of the offloaded tasks return from the server before the deadline D. Figure 4 presents a feasible schedule of four tasks $\{\tau_1,\tau_2,\tau_3,\tau_4\}$ with their timing parameters, where tasks $\{\tau_1,\tau_4\}$ are offloaded and $\{\tau_2,\tau_3\}$ are executed locally. To build the dynamic programming table, the tasks are ordered according to I_i non-increasingly. If there is a feasible schedule based on the offloading decision obtained by the dynamic programming algorithm, all offloaded tasks are executed (or setting up) at the beginning of the schedule, where the offloaded task with the longest I_i time is executed first. Then, local tasks are executed during the remote execution of the offloaded tasks. The results of the offloaded tasks are buffered on the server and received at the end of the schedule. This ordering grants the offloaded tasks with long I_i more time to be executed and return back before the deadline. Furthermore, it reduces the number of transitions between the idle and the transmit/receive states of the network card. The order for an example of four tasks is shown in Figure 4.

Consider the sub-problem of the first i tasks $\{\tau_1,\tau_2,\dots,\tau_i\}$. For the offloaded tasks in $\{\tau_1,\tau_2,\dots,\tau_i\}$, let $\sum_{j=1}^{i} x_j E_j^o$ be their *total NIC energy* and $\sum_{j=1}^{i} x_j(\frac{S_j^c}{f}+\Pi_j)$ be their *total setup and communication time*. Suppose that $L(i,t_c,e)$ is the *minimum total local execution time* for the locally-executed tasks in $\{\tau_1,\tau_2,\dots,\tau_i\}$, such that the total setup and communication time for the offloaded tasks in $\{\tau_1,\tau_2,\dots,\tau_i\}$ is less than or equal to t_c and their total NIC energy is less than or equal to e. Figure 4 shows the dynamic programming parameters for the example of the tasks. A three-dimensional dynamic programming table $L(i,t_c,e)$ is constructed for all the possible values of i, t_c, and e such that $0 \le i \le n$, $0 \le t_c \le D$ and $0 \le e \le \sum_{j=1}^{n} E_j^o$. All possible values of t_c and e are considered as the integer multiples of ρ and σ respectively (i.e. ρ and σ are user-specified granularity), and the values of $\frac{1}{\rho}$ and $\frac{1}{\sigma}$ are considered as integer numbers. We start by initializing all the elements of $L(0,t_c,e)$ to zeros. Then, the following recursion is used to fill the table:

$$L(i,t_c,e) = \min \begin{cases} \begin{cases} L(i-1,t_c-(\frac{S_i^c}{f}+\Pi_i),e-E_i^o) \\ \qquad \text{if } \; t_c \ge (\frac{S_i^c}{f}+\Pi_i) \wedge t_c + I_i \le D \wedge e \ge E_i^o \\ \infty \qquad\qquad\qquad \text{otherwise} \end{cases} \\ L(i-1,t_c,e) + (\frac{C_i^c}{f}+\Delta_i^l) \end{cases}$$

The algorithm chooses the offloading decision for the task τ_i that minimizes the total local execution time $L(i,t_c,e)$ for each sub-problem $\{\tau_1,\tau_2,\dots,\tau_i\}$ and for all the possible values of t_c and e. The inequation $t_c + I_i \le D$ in the recursion checks for each task if it can return before the deadline, in order to satisfy the feasibility condition in (1c). If a task τ_i is chosen for the local execution, its execution time $(\frac{C_i^c}{f}+\Delta_i^l)$ is added to the total local execution time of the previous sub-problem $L(i-1,t_c,e)$. If it is chosen for offloading, its setup and communication time $(\frac{S_i^c}{f}+\Pi_i)$ and its NIC energy E_i^o are considered in the available t_c and e of the previous sub-problem respectively. The algorithm also maintains the offloading decision of each task for backtracking later on.

After filling the table, we can verify whether a feasible schedule exists or not. We search for the minimum of $\{e+P(f)(t_c+L(n,t_c,e))\}$ (the objective function in (1a)) for all the possible values of t_c and e such that $t_c+L(n,t_c,e) \le D$ (the feasibility condition in (1b)). Then, we backtrack the table starting from the location that achieves the minimum value above to find the offloading decision x_i for each task. If the task τ_i is assigned for offloading, we backtrack to $L(i-1,t_c-(\frac{S_i^c}{f}+\Pi_i),e-E_i^o)$. If it is assigned for local execution, we backtrack to $L(i-1,t_c,e)$. The time complexity of the dynamic programming algorithm is $O(n\frac{D}{\rho}\frac{\sum_{j=1}^{n}E_j^o}{\sigma})$, in addition to the time complexity of $O(n\log n)$ for tasks' ordering at the beginning. Because the frequency levels of the processor are limited, we can search them (by running the algorithm for each frequency) to find the frequency that minimizes the energy consumption without violating real-time constraints.

B. Greedy Algorithm for Frame-based Tasks

In this subsection, we propose a greedy algorithm, called *GreedyF* algorithm, with time complexity less than the dynamic programming algorithm in IV-A. At the beginning, the algorithm takes as an input a schedule that is feasible when the

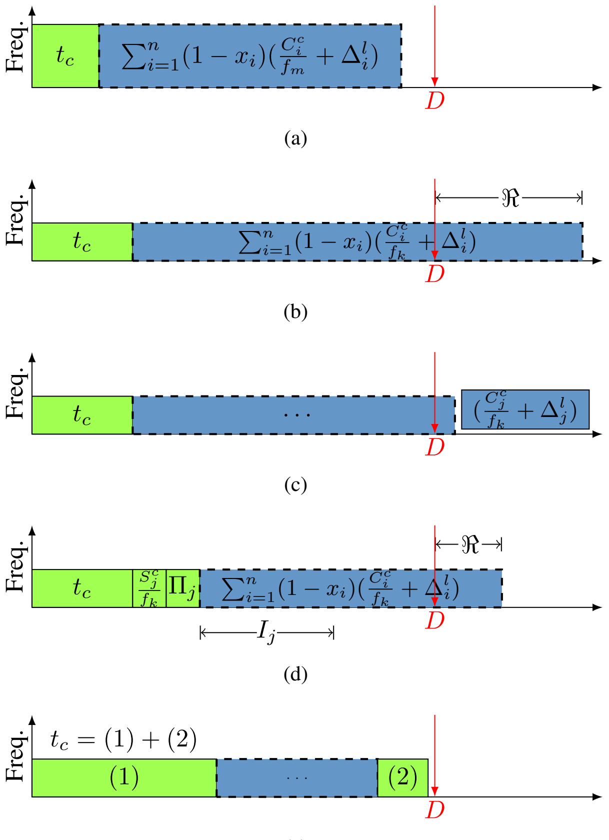
Algorithm 1 GreedyF($k, \vec{x}_n$)

```
1: $\vec{x}_n^* \leftarrow \vec{x}_n, k \leftarrow k-1$;
2: if condition (1c) does not hold then
3:     $k \leftarrow k+1$
4:     return $k, \vec{x}_n$;
5: end if
6: $\forall \tau_i \in \mathcal{T}, a_i \leftarrow (\frac{C_i^c}{f_k} + \Delta_i^l) - (\frac{E_i^o}{P(f_k)} + \frac{S_i^c}{f_k} + \Pi_i)$;
7: $t_c \leftarrow \sum_{i=1}^n x_i(\frac{S_i^c}{f_k} + \Pi_i)$;
8: $\Re \leftarrow (t_c + \sum_{i=1}^n (1-x_i)(\frac{C_i^c}{f_k} + \Delta_i^l)) - D$;
9: if $\Re \leq 0$ then
10:     return GreedyF($k, \vec{x}_n$);
11: end if
12: $\forall \tau_i \in \mathcal{T} | x_i = 0$ and $a_i > 0$, Order them according to $a_i$
    into the list $\mathcal{L}$;
13: while $\mathcal{L} \neq \emptyset$ and $\Re > 0$ do
14:     Pick the task $\tau_j$ from $\mathcal{L}$ with the maximum $a_j$;
15:     if $t_c + \frac{S_j^c}{f_k} + \Pi_j + I_j \leq D$ then
16:         $x_j^* \leftarrow 1$;
17:         $t_c \leftarrow t_c + \frac{S_j^c}{f_k} + \Pi_j$;
18:         $\Re \leftarrow \Re - \Big((\frac{C_i^c}{f_k} + \Delta_i^l) - (\frac{S_i^c}{f_k} + \Pi_i)\Big)$;
19:     end if
20:     $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\tau_j\}$;
21: end while
22: if $\Re \leq 0$ then
23:     retrun GreedyF($k, \vec{x}_n^*$);
24: else
25:     $k \leftarrow k+1$
26:     return $k, \vec{x}_n$;
27: end if
```

processor runs at the maximum frequency f_{max} (e.g. all the tasks are executed locally and meet the deadline). Then, it tries to reduce the energy consumption without violating the real-time constraints. The greedy algorithm does not guarantee to find a feasible schedule with the minimum energy consumption as in the dynamic programming algorithm, but it is much faster. If the greedy algorithm does not find a feasible schedule that reduces the energy consumption, it returns the feasible schedule that was taken as an input. The objective function in 1a can be written as follows:

$$\begin{aligned}\text{minimize } & P(f)\Big(\sum_{i=1}^{n}(\frac{C_i^c}{f} + \Delta_i^l) - \\ & \sum_{i=1}^{n} x_i\Big[(\frac{C_i^c}{f} + \Delta_i^l) - (\frac{E_i^o}{P(f)} + \frac{S_i^c}{f} + \Pi_i)\Big]\Big) \\ =& P(f)\Big(\sum_{i=1}^{n}(\frac{C_i^c}{f} + \Delta_i^l) - \sum_{i=1}^{n} x_i a_i\Big), \qquad (2) \\ \text{where } & a_i = (\frac{C_i^c}{f} + \Delta_i^l) - (\frac{E_i^o}{P(f)} + \frac{S_i^c}{f} + \Pi_i).\end{aligned}$$

If a task τ_i with $a_i > 0$ is offloaded, it reduces the energy consumption with the amount of $P(f) \cdot a_i$, and also reduces the total execution time on the client side. Therefore, the value of a_i is used as a heuristic in our greedy algorithm. We assume that the frequency range of $[f_{min}, f_{max}]$ is discretized into m frequency levels $\{f_1, f_2, \ldots, f_m\}$, where f_1 and f_m are the



Freq.
t_c
$\sum_{i=1}^n(1-x_i)(\frac{C_i^c}{f_m}+\Delta_i^l)$
D
(a)
$\Re$
$\sum_{i=1}^n(1-x_i)(\frac{C_i^c}{f_k}+\Delta_i^l)$
(b)
$\cdots$
$(\frac{C_j^c}{f_k}+\Delta_j^l)$
(c)
$\frac{S_j^c}{f_k}$
Π_j
I_j
(d)
$t_c = (1) + (2)$
(1)
(2)
(e)

Fig. 5: Illustration of Algorithm 1.

minimum and the maximum possible frequencies respectively. The algorithm starts from a feasible schedule at the maximum frequency f_m as shown in Figure 5a. For each iteration, the frequency is lowered one level, and the algorithm tries to find a feasible schedule that reduces the energy consumption. The greedy algorithm GreedyF($k, \vec{x}_n$), which is described in Algorithm 1 and illustrated in Figure 5, takes two inputs: a feasible schedule with the offloading decision $\vec{x}_n$, and the frequency index k. The value of k is equal to m at the first iteration of the algorithm. The algorithm works as follows:

- We start by reducing the frequency one level (Line 1). If the result of any offloaded task does not return back to the server within the deadline (according to condition (1c)), the algorithm stops and returns the previous feasible schedule with the previous frequency index (Lines 2 - 5).
- All variables are initialized, where $\vec{x}_n^*$ is used to maintain the last feasible schedule, t_c is the total setup and communication time for the offloaded tasks, and $\Re$ represents the difference between the total local finishing time and the deadline D (Lines 1, 6 - 8).
- If the schedule is still feasible with the lower frequency, the algorithm calls itself again with the new lower frequency level (Lines 9 - 11). Otherwise, the total local finishing time exceeds the deadline and then we have $\Re > 0$, as illustrated in Figure 5b.
- Only the local tasks, that reduce the energy consumption and the total local finishing time in the case of offloading, are ordered according to $a_i > 0$ into the list $\mathcal{L}$ (Line 12).
- As long as the schedule is not feasible ($\Re > 0$), the algorithm keeps picking a task τ_j from the list $\mathcal{L}$ with

the maximum value of a_j. If it is feasible for offloading, the algorithm assigns it for offloading and updates the values of t_c and $\Re$, as shown in Figures 5c and 5d (Lines 13 - 21).

- Finally, if the algorithm finds a feasible schedule as in Figure 5e, it calls itself again with the index of the lower frequency and the new offloading decision $\vec{x}^*_n$. Otherwise, it stops and returns the last feasible schedule with the previous frequency index (Lines 22 - 27).

The total setup and communication time t_c consists of two parts: (1) total setup and transmission time for the offloaded tasks which is equal to $\sum_{i=1}^{n} x_i(\frac{S_i^c}{f_k} + \Delta_i^o)$ and this part is executed at the beginning of the schedule, (2) total receiving time of the results which is equal to $\sum_{i=1}^{n} x_i \nabla_i$ and this part is executed at the end of the schedule. The local tasks are executed between these two parts as shown in Figure 5e. The time complexity of Algorithm 1 is $O(m\ n \log n)$, which includes searching the m frequency levels.

C. Greedy Algorithm for Sporadic Tasks

In this subsection, we propose a Greedy algorithm, called *GreedyS* algorithm, to reduce the Energy consumption for Sporadic real-time tasks. To solve this problem, two decision-making points should be considered: task scheduling and offloading decision. If we decide to offload a task, it means that the execution of a job of the task is divided into two parts, separated by an offloading interval. Therefore, we map the problem of scheduling sporadic real-time tasks with computation offloading to the problem of *scheduling sporadic real-time tasks with self-suspensions*. In the problem of scheduling self-suspending tasks, the task can suspend its execution for a specific amount of time when it performs read/write operations with input/output devices or when it waits for an event. The real-time literature has focused on two self-suspending task models: the *dynamic* self-suspension model and the *segmented* suspension model. The dynamic self-suspension task model allows self-suspension to take place as long as it does not suspend longer than the specified worst case self-suspension time. The segmented self-suspending task model defines the execution behaviour of a job of a task by predefined computation segments and self-suspension intervals.

The recent report by Chen et al. [3] provides a comprehensive survey and explains why several analyses in the state-of-the-art of self-suspending tasks are in fact unsafe. Our studied computation offloading problem falls into the category of segmented suspension model with only one self-suspension interval. The best result in this category for scheduling is by Chen and Liu [2].[1] The problem is to find a feasible schedule for such a task model. For the case with at most one self-suspension, the task τ_i consists of two execution units ($C_{i,1}$ and $C_{i,2}$) separated by a suspension interval Λ. According to the system model presented in [2], if a task τ_i does not suspend, its execution time is equal to $C_{i,1}$, and both $C_{i,2}$ and Λ are equal to 0. Now we will link the local and the offloaded tasks in our system model to the tasks that do not suspend and the tasks that suspend in the sporadic task model with self-suspensions respectively. For given offloading decisions x_i and a frequency level f in our problem, we can define the following terms:

- $C_{i,1} = x_i(\frac{S_i^c}{f} + \Delta_i^o) + (1 - x_i)\frac{C_i^c}{f} + \Delta_i^l$,
- $C_{i,2} = x_i \nabla_i$,
- $\Lambda = x_i I_i$,
- $D_{i,1} = D_i^o$ and
- $D_{i,1} = D_i^r$.

The work in [2] presents a *Fixed-Relative-Deadline* (FRD) approach to schedule the real-time tasks that may perform one self-suspension on a uniprocessor so that the tasks meet the deadlines. A proposed approach, called *Equal-Deadline Assignment* (EDA), assigns equal relative deadlines ($D_{i,1}$ and $D_{i,2}$) for the execution units $C_{i,1}$ and $C_{i,2}$ of a task with self-suspension, where $D_{i,1} = D_{i,2} = \frac{T_i - \Lambda_i}{2}$. Then, the EDF algorithm is used to schedule the execution units according to the newly assigned relative deadlines. It has been shown in [2] that the EDA approach provides a feasible schedule for a set of sporadic real time tasks $\mathcal{T}$ with at most one self-suspension if:

$$\sum_{j=1}^{n} U_j = \sum_{j=1}^{n} \frac{C_{j,1} + C_{j,2}}{T_j} \leq 1, \text{ and} \tag{3a}$$

$$\forall \tau_i \in \mathcal{T}, \sum_{j=1}^{i} \text{dbf}_j^{\text{EDA}}(D_{i,1}) \leq D_{i,1}, \tag{3b}$$

where the tasks are ordered according to $D_{i,1}$ (or $D_{i,2}$) such that $D_{i,1} \leq D_{j,1}$ for $i < j$,

$$\text{dbf}_j^{\text{EDA}}(t) = \begin{cases} 0 & 0 \leq t < D_{j,1} \\ C'_j + (t - D_{j,1})U_j & D_{j,1} \leq t \end{cases}, \text{ and}$$
$$C'_j = \max\{C_{j,1}, C_{j,2}, C_{j,1} + C_{j,2} - U_j D_{j,1}\}.$$

The linear-time feasibility analysis described above is used in our proposed algorithm below to test the schedulability of the tasks in our system model.

The second decision-making point is to determine the offloading decision of the tasks. Our proposed algorithm, described in Algorithm 2, determines the offloading decision of the tasks in addition to the speed of the clients processor so that the energy consumption is minimized. The algorithm starts from any feasible schedule (e.g. all the tasks are executed locally at the maximum processor speed and meet their deadlines) and tries to minimize the energy consumption using the DVFS and the computation offloading without violating the feasibility of the schedule. In each round, the frequency of the processor is decreased one level to reduce the energy consumption, where we consider the discretized m frequency levels $\{f_1, f_2, \ldots, f_m\}$ and the heuristic a_i from Subsection IV-B. If the schedule becomes infeasible due to the frequency decrement, the algorithm assigns tasks for offloading according to the heuristic a_i to reduce the total demand of the tasks and then find a feasible schedule. The inputs of the proposed algorithm are the offloading decision vector $\vec{x}_n$ and the index of the frequency level that are necessary to schedule the tasks feasibly under the EDF algorithm. The GreedyS algorithm works as follows:

[1]The recent result by Huang and Chen [5] shows that the same method can still yield good solutions for multiple self-suspension segments, but this is out of the scope of this paper.

Algorithm 2 GreedyS($k, \vec{x}_n$)

```
1:  $\vec{x}^*_n \leftarrow \vec{x}_n, k \leftarrow k-1$;
2:  if condition (3) holds then
3:      return GreedyS($k, \vec{x}_n$);
4:  end if
5:  $\forall \tau_i \in \mathcal{T}, a_i \leftarrow (\frac{C^c_i}{f_k} + \Delta^l_i) - (\frac{E^o_i}{P(f_k)} + \frac{S^c_i}{f_k} + \Pi_i)$;
6:  $\forall \tau_i \in \mathcal{T} | x_i = 0 \wedge a_i > 0 \wedge D^o_i \geq \frac{S^c_i}{f_k} + \Delta^o_i \wedge D^r_i \geq \nabla_i$,
    order them according to $a_i$ into the list $\mathcal{L}$;
7:  while $\mathcal{L} \neq \emptyset$ do
8:      Pick the task $\tau_j$ from $\mathcal{L}$ with the maximum $a_j$;
9:      $x^*_j \leftarrow 1$;
10:     $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\tau_j\}$;
11:     Relocate $\tau_j$ in the schedule;
12:     if condition (3) holds then
13:         return GreedyS($k, \vec{x}^*_n$);
14:     end if
15: end while
16: $k \leftarrow k+1$
17: return $k, \vec{x}_n$;
```

- At the beginning, the algorithm maintains the offloading decision vector of the previous feasible schedule and reduces the frequency one level. If the schedule is still feasible by running the processor at the new lower frequency, the algorithm calls itself again to target a lower frequency level (Lines 1 - 4). If the feasibility analysis does not guarantee a feasible schedule, the algorithm continues to find a feasible schedule by reducing the workload on the client using the computation offloading.
- The heuristic of a_i is calculated for each task (recall that $\Pi_i = \Delta^o_i + \nabla_i$). The tasks that are beneficial ($a_i > 0$) and feasible for offloading ($D^o_i \geq \frac{S^c_i}{f_k} + \Delta^o_i \wedge D^r_i \geq \nabla_i$) are ordered into a list called $\mathcal{L}$ (Lines 5 - 6). The heuristic evaluates the tasks according to the reduction in the energy and the workload gained if they are offloaded.
- To find a feasible schedule, the algorithm offloads the tasks according to the heuristic where the task with the highest energy and time reduction is offloaded first (Lines 7 - 15). If the algorithm finds a feasible schedule, it tries again to target a lower frequency level. Otherwise, the algorithm returns the last feasible schedule (Lines 16 - 17).

After finding a feasible schedule that reduces the energy consumption, the tasks are scheduled under the EDF algorithm. The GreedyS algorithm selects the frequency level that achieves the minimum energy consumption among all levels. If the algorithm finds a feasible schedule at f_{min} and there are still local tasks with $a_i > 0$, it can continue assigning these tasks for offloading to have lower energy consumption. The time complexity of the algorithm for a specific frequency level is $O(n^3)$, and the total time complexity for evaluating the m levels is $O(m \cdot n^3)$.

Another feasibility analysis from [15] can be used to test the feasibility of sporadic real-time tasks, that can be offloaded with a negligible reception time (i.e., $\nabla_i \approx 0$), under EDF. Suppose that the offloading decisions of the tasks are known and they are ordered according to D_i non-decreasingly (i.e., $D_k \leq D_l$ if $k < l$), where $D_i = x_i D^o_i + (1 - x_i) D^l_i$ and $D^o_i = D^l_i - I_i$. According to the feasibility analysis in [15], a set of sporadic tasks with a negligible reception time can be feasibly scheduled by the EDF algorithm on a processor with a speed of f if:

TABLE I: Tasks parameters of the case study.

		$\times 10^4$ cycles		ms	
Task	Description	C^c_i	S^c_i	Δ^o_i	R_i
τ_1	Motion Detection	5190	0	21	21
τ_2	Object Recognition	38060	173	1	102
τ_3	Stereo Vision	15224	2076	22	41
τ_4	Motion Recording	3114	0	21	14

$$\sum_{j=1}^{n} u_j = \sum_{j=1}^{n} x_j \frac{S_j}{T_j} + \sum_{j=1}^{n} (1 - x_j) \frac{C_j}{T_j} \leq 1 \tag{4a}$$

$$\forall \tau_i \in \mathcal{T}, \sum_{j=1}^{i} x_j \left(\frac{D_i - D^o_j}{T_j} + 1 \right) \frac{S_j}{D_i} + \sum_{j=1}^{i} (1 - x_j) \frac{C_j}{T_j} \leq 1, \tag{4b}$$

where $S_j = \frac{S^c_j}{f} + \Delta^o_j$ and $C_j = \frac{C^c_j}{f} + \Delta^l_j$. Algorithm 2 can be easily and slightly modified to consider sporadic tasks with a negligible reception time as follows:

- The feasibility condition in (3) (Lines 2 and 12) is replaced with the condition in (4).
- $\Pi_i = \Delta^o_i$ in Line 5.
- The condition $D^r_i \geq \nabla_i$ in Line 6 is omitted.

V. Experimental Evaluations and Simulations

The proposed middleware was evaluated by implementing a surveillance system as a case study and synthesis workload simulations. A feasible schedule was used as a baseline to evaluate the efficiency of the DPF and GreedyF algorithms, such that all the tasks are executed locally at the maximum speed. For the GreedyS algorithm, the baseline schedule was derived for each simulation case using the proposed algorithm in [15] at the maximum processor speed. The algorithms are compared to the approach that offloads a task if its energy consumption in the case of the offloading is less than its energy consumption in the case of the local execution. We use the abbreviation *LOD* to denote such an approach that considers a Local Offloading Decision for each task. Tight deadlines were assigned so that the schedule of the tasks becomes infeasible if the processor runs at any lower frequency than the maximum one f_{max}. Therefore, the DVFS cannot be used alone here to reduce the energy consumption, because the tasks will miss the deadlines. The *normalized energy saving (or reduction)* of an algorithm is equal to $1-$ (the energy consumption of the derived schedule using this algorithm divided by the energy consumption of the baseline schedule for the same task set).

A. Case Study of a Surveillance System

We implemented a case study of a surveillance system that consists of a client and a server. The client is equipped with two cameras, left and right, to capture images from the environment. The client performs four independent image processing tasks on the captured images which are described in Table I with their parameters. The reception time for the results of the tasks does not exceed 0.2 ms, and the average

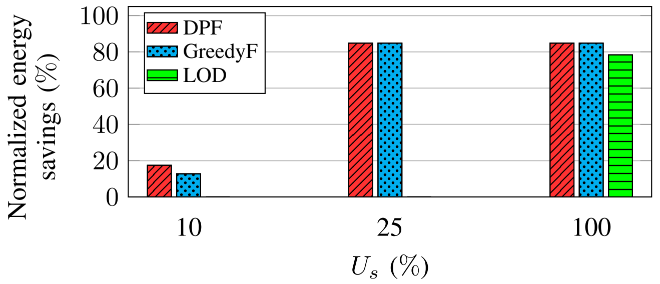


Fig. 6: Case study results for the frame-based tasks.

TABLE II: Case study results for the sporadic tasks.

	$U_s = 100\%$	$U_s = 15\%$
Normalized energy savings (%)	69.91	15.68
Running frequency (MHz)	33	266

value equals 0.042 ms. The parameters represent the worst-case values based on profiling. Tasks execution continues along the input video stream in a periodic manner. The server has a Pentium(R) Dual-Core 2.8 GHz 64-bit CPU. On the client side, we adopted the IBM PPC405LP 32-bit processor model and the Intel PRO/Wireless 3945ABG (802.11a/b/g) card model. The frequency levels of the processor are 33, 100, 266 and 333 MHz. The corresponding power consumption levels are 19, 72, 600, and 750 mW respectively. The idle power of the processor equals 12 mW. The network card consumes 30, 150, 1400 and 1800 mW during its sleep, idle, reception and transmission states respectively.

The results of the case study for frame-based tasks using different utilization values are shown in Figure 6. We observe that the results of the DPF and GreedyF algorithms are the same for the utilization of 25% and 100%. However, the DPF algorithm saves more energy for the utilization of 10% due to the optimal decision for each frequency level. Furthermore, the LOD algorithm was not able to reduce the energy consumption for the utilization of 25% and 10%, i.e., the server serves more than one client. For $U_s = 100\%$, the LOD algorithm offloads all the tasks. However, the DPF and GreedyF algorithms offload just tasks τ_2 and τ_3 which improves the energy saving. The relative deadlines of the sporadic tasks were generated randomly so that the total utilization of the tasks equals 100% when they are executed locally. Table II shows the results of the derived schedules by the GreedyS algorithm for $U_s = 100\%$ and $U_s = 15\%$. In the case of the full utilization, the remote response time of the tasks is relatively shorter than in the case of $U_s = 15\%$. In this case, the algorithm offloads more tasks to reduce the workload of the client's processor and then run it at the lowest frequency in order to save more energy as shown in the table.

B. Simulation Setup and Results

The generation of simulation parameters is inspired by the case study but with wider ranges. The parameters of the tasks were generated randomly with a uniform distribution as follows:

- C_i^c: between 10^6 and 10^9 cycles.
- S_i^c: between 10^6 and C_i^c cycles.
- Δ_i^o: floating-point values between 1 ms and 20 ms.
- ∇_i: floating-point values between 0.04 ms and 0.2 ms.
- R_i: $\frac{C_i^c/f_{max}}{\alpha}$, where α is the speed-up factor of the server.

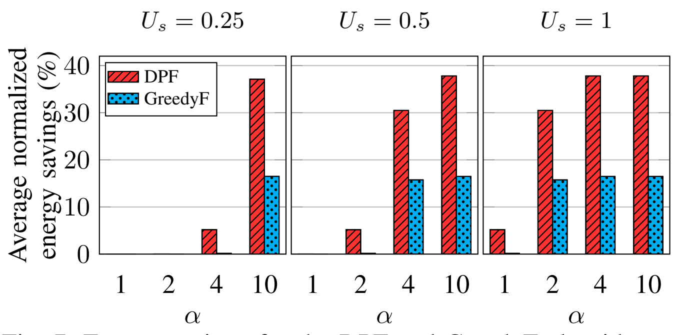


Fig. 7: Energy savings for the DPF and GreedyF algorithms.

A total of 100 rounds were performed for each $U_s = \{0.25, 0.5, 1\}$. The server uses a fair-sharing policy, where its utilization is partitioned between the connected clients equally. A low utilization value, $U_s = 0.25$, means that the server serves four clients at the same time or is busy with other applications and can reserve relatively a short amount of time for the offloaded tasks. Full utilization, $U_s = 1$, means that the server is idle and dedicated completely for one client. In each round, a set of 25 frame-based real-time tasks is randomly generated for all the values of $\alpha = \{1, 2, 4, 10\}$. The power model in the case study was adopted in the simulation. A tight deadline $D = \sum_{i=1}^{n}(C_i^c/f_{max} + \Delta_i^l)$ was assigned for each task set such that the schedule becomes infeasible if the processor runs at any lower frequency than f_{max}. For the sporadic tasks, the same parameters above were used and the relative deadlines of the tasks were generated as random floating-point values so that the total utilization of any task set without offloading (i.e. U_{local}) is equal to 120%. Therefore, DVFS cannot be used alone for both task models to reduce the energy consumption, because the tasks will miss the deadlines.

Figure 7 shows the average normalized energy savings for the schedules derived by the DPF and GreedyF algorithms. In general, as the value of α increases, the energy saving also increases for both algorithms. Because with a faster server (higher α values), the remote response time of the tasks will be relatively shorter and the algorithms may offload more tasks. Furthermore, as the given utilization from the server increases (less served clients), the energy saving also increases. If the client obtains more utilization from the server, the remote response time of the offloaded tasks decreases. Hence, there will be more offloadable tasks (i.e., their results return before the deadline) and the client will be able to offload more of them in order to reduce the energy consumption. Moreover, we observe that there will be no more reduction in the energy consumption if the speed-up factor of the server multiplied by the given utilization is greater than or equal to 2.5. In this case, the client will not be able to offload more tasks due to the communication overhead. The LOD algorithm was not able to derive feasible schedules with reduced energy consumption in this simulation.

Figure 8 presents the average execution time for the DPF and GreedyF algorithms in milliseconds, where $U_s = 1$ and $\alpha = 4$. As the number of tasks n increases, the average execution time also increases, but it increases exponentially for the DPF algorithm. To test the percentage of the feasible schedules obtained by both algorithms, the deadline is reduced to be $D = \gamma * \sum_{i=1}^{n}(C_i^c/f_{max} + \Delta_i^l)$, where

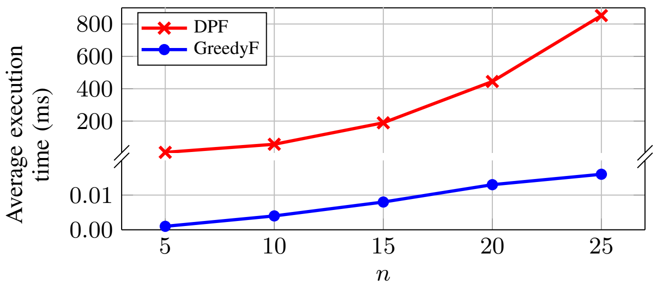


Fig. 8: The execution time of the algorithms.

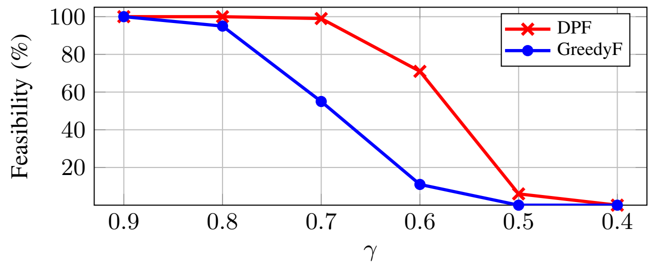


Fig. 9: Percentage of the feasible derived schedules.

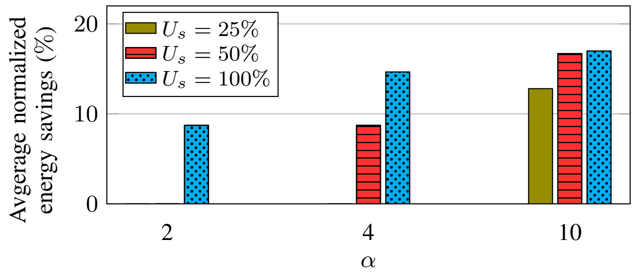


Fig. 10: Simulation results for the GreedyS algorithm.

$\gamma = \{0.9, 0.8, 0.7, 0.6, 0.5, 0.4\}$. Figure 9 shows that the DPF algorithm finds more feasible schedules than the GreedyF algorithm. According to the evaluation, the GreedyF algorithm is much faster than the DPF algorithm. However, the DPF algorithm saves more energy and finds more feasible schedules. The evaluation results of the GreedyS algorithm are presented in Figure 10 for different values of α and U_s. We observe that the energy saving increases as the speed-up factor of the server increases, because the GreedyS algorithm will be able to offload more tasks. Furthermore, as the given utilization from the server increases, the energy savings also increases for the same reason in Figure 7.

VI. Conclusion

In this paper, we propose a middleware to reduce the energy consumption in real-time embedded systems. The algorithms in the middleware performs computation offloading for sporadic and frame-based real-time tasks in order to reduce the workload of the client's processor (i.e., the processor of the embedded system) and then save energy. The tasks are offloaded to a powerful server. On the server side, we use the total bandwidth server to provide response time guarantee for the offloaded tasks. The proposed algorithms determine the speed of the client's processor, schedule real-time tasks and determine their offloading decision in order to find a feasible schedule that minimizes the energy consumption. The algorithms were evaluated using a case study of a surveillance system and synthesized benchmarks. The simulation results show that the derived schedules can save in average up to nearly 40% of the energy consumption, compared to the case that all of the tasks are executed locally at the maximum processor speed. For future researches, we will explore other resource reservation servers.

VII. Acknowledgements

This work is supported by the German Research Foundation (DFG) as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/) and as part of the Transregional Collaborative Research Centre Invasive Computing [SFB/TR 89].

References

[1] HTSM roadmap embedded systems 2015, laying the foundation for intelligent high-tech systems and applications, 2015. URL http://www.esi.nl/research/documents/HTSM-Roadmap_Embedded_Systems-2015-REV1.0.pdf.

[2] J. Chen and C. Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 149–160, 2014.

[3] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, Neil, Audsley, R. Rajkumar, and D. de Niz. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. Technical Report 854, Faculty of Informatik, TU Dortmund, 2016.

[4] C.-H. Hsu and W. chun Feng. A power-aware run-time system for high-performance computing. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 1–1, Nov 2005.

[5] W. Huang and J. Chen. Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling. In *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pages 1078–1083, 2016.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] W. Liu, J.-J. Chen, A. Toma, T.-W. Kuo, and Q. Deng. Computation offloading by using timing unreliable components in real-time systems. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.

[12] B. Patra, S. Roy, and C. Chowdhury. A framework for energy efficient and flexible offloading scheme for handheld devices. In *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, pages 1–6, 2015.

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

[14] A. Toma and J. Chen. Computation offloading for frame-based real-time tasks under given server response time guarantees. *Leibniz Transactions on Embedded Systems*, 1(2):02:1–02:21, 2014.

[15] A. Toma, J. Chen, and W. Liu. Computation offloading for sporadic real-time tasks. In *The 2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, Chongqing, China, August 20-22, 2014*, pages 1–10, 2014.

[16] F. Wu, S. Jin, and Y. Wang. A simple model for the energy-efficient optimal real-time multiprocessor scheduling. In *Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on*, volume 3, pages 18–21, May 2012.