Computation Offloading by Using Timing Unreliable Components in Real-Time Systems

Wei Liu [1] [2], Jian-Jia Chen [3], Anas Toma [2], Tei-Wei Kuo [4], Qingxu Deng [1]
[1] Northeastern University, China, [2] Karlsruhe Institute of Technology (KIT), Germany
[3] TU Dortmund University, Germany, [4] National Taiwan University, [4]Academia Sinca, Taiwan

ABSTRACT

There are many timing unreliable computing components in modern computer systems, which are typically forbidden in hard real-time systems due to the timing uncertainty. In this paper, we propose a computation offloading mechanism to utilise these timing unreliable components in a hard real-time system, by providing local compensations. The key of the mechanism is to decide (1) how the unreliable components are utilized and (2) how to set the worst-case *estimated* response time. The local compensation has to start when the unreliable components do not deliver the results in the estimated response time. We propose a scheduling algorithm and its schedulability test to analyze the feasibility of the compensation mechanism. To validate the proposed mechanism, we perform a case study based on image-processing applications in a robot system and simulations. By adopting the timing unreliable components, the system can handle higher-quality images and with better performance.

1. INTRODUCTION

Nowadays, embedded systems are commonly used in medical devices, robots, transportation vehicles, etc. In these critical systems, the system correctness depends not only on the function reliability, but also on the timing correctness. That is, the response time of an application must be within a specific relative deadline. Especially in hard real-time systems, missing deadlines may lead to catastrophic consequences.

However, modern embedded systems are increasingly integrated with complicated applications. The limited resources, such as the battery capacity, the memory sizes, and the processor speed, cannot satisfy the demand for such complex applications. Due to the stringent resource availability and computation power in embedded systems, the tasks typically cannot process high volumes of data. To resolve these issues, offloading heavy computation to some powerful components has been shown as an attractive solution, including optimizations for system performance [12] and energy saving [7].

If the components that are used to serve the offloaded tasks as well as the communication channel are timing predictable, these components can be adopted and used well in hard real-time embedded systems. However, there are many computing components in modern computer systems, which are difficult for the system designers to analyze the tight and useful worst-case execution time or worst-case response time. For example, the modern commercial Graphics Processing Units (GPUs) and Commercial off-the-shelf (COTS) computing components are often not very easy to characterize the worst-case timing behavior since the implementation details are not revealed. Without the details, the system designers usually have no possibility to analyze the *useful and tight* worst-case execution time or response time. Therefore, such components are usually considered as *timing unreliable* components and forbidden in hard real-time systems.

Motivation Example: However, adopting GPUs or COTS components has many benefits, as they are typically faster for some applications and can process higher volumes of data. For example, a mobile robot commonly uses the Scale-Invariant Feature Transform (SIFT) algorithm for object recognition in a dynamic environment. If we consider a Nvidia GeForce GT 630M GPU as the component for executing the offloaded tasks. By an image size of 300×200, the average execution time on GT 630M is about $7ms$ without any interference by other tasks. Its average execution time on Intel Core i3-2310M CPU is about $278ms$. Therefore, if the relative deadline is $100ms$, we can either reduce the image size by executing SIFT on the CPU or we can use the GPU, as it has a *high potential* to return the results in the desired relative deadline. It is clear that GPU is a high performance device, but running simultaneous tasks on the GPU may result in much worse response time. Therefore, it is also possible that the results are not returned before the desired deadline.

Our Contributions: In this paper, we propose a computation offloading mechanism to utilise timing unreliable components in a hard real-time system, by providing *local compensations* to handle the exceptional cases if the results are not returned in time. Therefore, to design a sound computation offloading mechanism by using timing unreliable components, we have to deal with two issues: (1) how the timing unreliable components are utilized and (2) how to set the worst-case *estimated* response time, in which the local compensation has to start as soon as the unreliable components do not deliver the results in the estimated response time. To evaluate the benefit for offloading a certain task to a timing unreliable component, the benefit function for offloading is characterized as a discrete function with respect to the estimated response time. Towards these two issues, based on the benefit functions for offloading, we have the following concrete technical contributions in this paper:

- One important technical issue in the mechanism is to

Published in Proceedings of the 51st Annual Design Automation Conference (DAC '14). ACM, New York, USA, Article 39, 6 pages. DOI=10.1145/2593069.2593109 http://doi.acm.org/10.1145/2593069.2593109

schedule the set of real-time tasks after the targeted estimated response times are specified. Towards this, we propose a scheduling algorithm based on the earliest-deadline-first (EDF) scheduling policy, and its schedulability test to analyze the feasibility of the compensation mechanism. The scheduling algorithm assigns different deadlines to the subtasks, one for preparing the offloading and one for handling the local compensation, proportionally to their computation times.

- Moreover, to maximize the system performance, we reduce the studied optimization problem to the multiple-choice knapsack problem based on the proposed schedulability test, and adopt existing solutions of the multiple-choice knapsack problem.
- To validate the proposed mechanism, we perform a case study based on image-processing applications in a robot system and estimate the benefit functions with respect to the image qualities under an unreliable networking environment and GPU servers. Moreover, to understand the impacts of inaccurate response time estimations in the timing unreliable components, we also provide simulations to show the effectiveness of the mechanism.

2. RELATED WORKS

The restrictions of limited resources on embedded systems can be alleviated by computation offloading mechanism. For example, in the literature, computation offloading has been utilized for improving system performance, saving the energy consumption and improving quality of service on embedded systems [3,7,12]. However, these results do not consider to satisfy the timing constraints for real-time applications.

Computation offloading for real-time systems has been recently studied [8,10,11]. Nimmagadda et al. in [8] develop a system for real-time moving object recognition and tracking by computation offloading. In their system, a task is offloaded to the server once the execution time on the server together with the required data transfer time (called *offloading response time* here) is shorter than the local execution time on the client. Toma and Chen [11] propose a pseudo-polynomial-time algorithm to decide when and what to offload without violating timing constraints even when the local execution time may be shorter than the offloading response time. As computation offloading also requires the server to be timing predictable, Toma and Chen [10] further adopt resource reservations in the server site for ensuring the offloading latency.

The above results [8,10,11] can work only when the servers (components) to serve the offloaded tasks and the communication channel are timing reliable. When a task is greedily offloaded but the results do not return in the estimated response time, their approaches cannot be applied for ensuring hard real-time properties.

3. TIMING RELIABLE OFFLOADING

This section presents the software architecture for reliable computation offloading to be executed on timing unreliable components. We consider the flexibility to either execute a task locally or to execute a task on a *server* by computation offloading. Here, a task is an active entity of a program in an embedded system, and a server is an abstraction of any components that can be used for executing the offloaded tasks. We emphasize that the server may be timing unreliable. To characterize the execution behavior of a task τ_i on an embedded system, we consider the following properties:

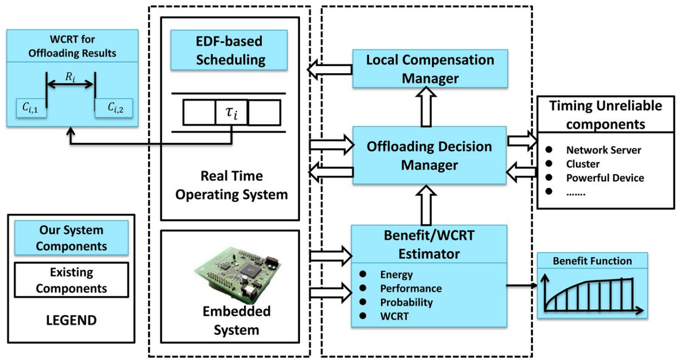

Figure 1: Software Architecture

- C_i, as the *local execution time*, is the worst-case execution time to execute task τ_i locally on the embedded system.
- $C_{i,1}$, as the *setup time* for offloading task τ_i, is the worst-case execution time on the embedded system to prepare the local preprocessing for offloading task τ_i. It involves some local operations such as data compression, initialization, data transmission, etc.
- $C_{i,2}$, as the *local compensation*, is the worst-case execution time on the embedded system to handle the local compensations when the results are not delivered in the estimated time to ensure the timing and *baseline* quality satisfactions. For example, to make sure that the quality is at least as good as the one with local execution, we can simply use the version for the local execution time, in which $C_{i,2}$ is C_i under such a case.
- $C_{i,3}$, as the *post-processing time* for offloading task τ_i, is the worst-case execution time on the embedded system to process the results from the server to ensure the execution correctness. We assume that $C_{i,3} \leq C_{i,2}$.

Moreover, as we focus on timing unreliable components, to prepare for the worst cases, it is clear that we do not need to consider $C_{i,3}$ under the assumption $C_{i,3} \leq C_{i,2}$. However, if the timing unreliable components can still have some pessimistic upper bound of the worst-case response time, $C_{i,3}$ may be used if the expected worst-case response time R_i is set to be longer than the upper bound. The extension to handle this case is pretty straightforward and trivial. Due to the space limitation, and for the simplicity of presentation, we will not consider this case.

3.1 Software Architecture

Based on the timing characteristics of real-time applications defined above, here, we present the software architecture for the proposed mechanism, as illustrated in Figure 1. The important system components in our mechanism include *Benefit and Response Time Estimator*, *Offloading Decision Manager*, and *Local Compensation Manager*. The Benefit and Response Time Estimator evaluates and estimates the corresponding benefit if task τ_i is offloaded to be executed on the unreliable component under an estimated response time. The benefit depends on the properties required by the embedded system for exploiting the timing unreliable component, which will be detailed in Section 3.2. The Offloading Decision Manager, to be detailed in Section 3.3, receives the *discretized* benefit function (from the Benefit and Response Time Estimator) and decides which tasks to be offloaded and the corresponding expected worst-case response times to start the local compensations. One important goal in the Offloading Decision Manager is to ensure the timing correctness even if the results do not return in time. That is, the time to start the local compensation should be further passed to the *Local Compensation Manager* to handle exceptional cases, which can be implemented by setting up timer-interrupts.

3.2 Benefit and Response Time Estimator

The benefit for offloading task τ_i to a timing unreliable component is important for the proposed mechanism. Without any timing information for the potential response time from the unreliable components or the benefits for offloading task τ_i, we are not able to make proper offloading decisions. Even though the timing unreliable component may not provide *worst-case* guarantees, typically, the average cases or the percentile cases can be provided to the embedded system. Moreover, if we focus on the performance improvement, we would focus on the improvement of the resulting quality by exploiting the timing unreliable components.

Even though the estimation for the response time from the unreliable component may not be fully correct, with a *proper* local compensation mechanism, we can still guarantee the timing correctness. However, it is also noticeable that the accuracy of the response time estimation is also very important for making offloading decisions. If the response time estimation is too pessimistic, the offloading option will not be taken. On the other hand, if the response time estimation is too optimistic, the offloading option may be taken, and the local compensation is frequently adopted.

For the rest of this paper, we denote $G_i(r_i)$ as the benefit function of task τ_i if the estimated response time is set to r_i. According to the definition, $G_i(r_i)$ is a non-decreasing function with respect to r_i. The potential benefit values can be (1) the probability to get computation results within response time r_i, (2) the performance index improvement of the results within response time r_i, etc.

Specifically, there exist several frameworks and models, for example probabilistic execution time and queuing theory when $G_i(r_i)$ is the probabilistic distribution, to characterize $G_i(r_i)$ when the system is well-defined. Another possibility is to use statistical analysis to estimate $G_i(r_i)$. In our paper and our case study based on image processing applications, we will consider that $G_i(r_i)$ is obtained based on statistical analysis and measurement, which will be explained in Section 6.1.2.

For the rest of this paper, we will assume that $G_i(r_i)$ is a non-decreasing function, and the value changes at only a fixed number of points. That is, $G_i(r_i)$ is discretized. Specifically, $G_i(0)$ stores the benefit for local execution. For brevity, we suppose that there are Q_i points in the discretized benefit function, including the point at time 0. For each of the Q_i discrete points in the benefit function for task τ_i, suppose that $r_{i,j}$ is the j-th point (from the smallest). By definition, $r_{i,1}$ is 0 and $r_{i,j} > 0$ when $j > 1$.

3.3 Offloading Decision Manager

After the benefit function $G_i(r_i)$ is established for each task τ_i, Offloading Decision Manager decides whether task τ_i should be executed locally or offloaded to maximize the total system benefit under timing satisfactions. Offloading Decision Manager builds a wrapper library with communication methods to send data from the embedded system to the timing unreliable components. To execute an offloaded task τ_i without sacrificing the timing correctness, the following parameters should be properly set in Offloading Decision Manager:

- The relative deadline of task τ_i is set to $D_{i,1}$ to specify the timing requirement to finish the setup execution (i.e., $C_{i,1}$).
- The estimated worst-case response time from the server R_i to specify the *expected* worst-case response time to receive the results for starting the post-processing (i.e., $C_{i,3}$). If the offloaded task τ_i returns within the response time R_i, we can start the post-processing; otherwise, the local compensation (i.e., $C_{i,2}$) will be started.

The settings of $D_{i,1}$ and R_i are very important for the proposed mechanism. On one hand, $D_{i,1}$ and R_i should be maximized to achieve high benefit for offloading. On the other hand, R_i cannot be set too high, as the local compensation may not be done in time.

4. PROBLEM DEFINITION

As shown in Section 3.3, the proposed mechanism requires proper settings of several parameters for offloaded tasks to utilize the timing unreliable components. Here, we consider the most traditional real-time recurring task model, *sporadic real-time task model*. Each task τ_i represents an infinite sequence of jobs with the same properties, in which τ_i is characterized by its minimum inter-arrival time (also called period) T_i and relative deadline D_i. That is, if a task τ_i releases an instance (called *job*) at time t, this job has an absolute deadline $t + D_i$ and the next job released by task τ_i cannot be earlier than $t + T_i$. Moreover, according to Section 3, each task τ_i is also characterized with its benefit function $G_i(r_i)$ and its execution time properties, i.e., $C_i, C_{i,1}, C_{i,2}$, and $C_{i,3}$. For the simplicity of presentation, we consider implicit-deadline tasks, in which D_i is equal to T_i for every task τ_i. The approach can be easily extended for constrained-deadline tasks, in which $D_i \leq T_i$.

We are given a set $\mathbf{T}$ of independent and preemptable real-time tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$, defined above. The objective of the *Offloading Decision Manager* (ODM) problem is to select a subset $\mathbf{T}'$of the above n tasks and derive a schedule to maximize $\sum_{\tau_i \in \mathbf{T}'} G_i(R_i)$ in which all the jobs released by all the tasks in $\mathbf{T}$ can still meet their deadline constraints. The ODM problem is a NP-hard problem, which can be reduced from the knapsack problem.

5. OUR ALGORITHM

To solve the ODM problem, we decompose the problem into two subproblems. In the first subproblem, a scheduling algorithm and its schedulability test should be provided to analyze the feasibility of the compensation mechanism when the estimated response times for offloaded tasks are given. For this subproblem, we propose a scheduling algorithm based on the earliest-deadline-first (EDF) scheduling policy and its schedulability test to analyze the feasibility of the compensation mechanism, which will be presented in Section 5.1. The second subproblem is to select tasks for offloading with proper settings of the estimated response times for maximizing the benefit under feasible schedulability. For this subproblem, we reduce to the multiple-choice knapsack problem based on the proposed schedulability test, and adopt existing solutions of the multiple-choice knapsack problem, which will be presented in Section 5.2.

5.1 Scheduling under Given R_i

Now, suppose that we are given a task partition, in which $\mathbf{T}^o$ is the subset of the given task set $\mathbf{T}$ for being offloaded and $\mathbf{T}^\ell$ is the subset of $\mathbf{T}$ for being locally executed on the embedded system. Moreover, we assume that the estimated response time R_i is given for each task τ_i in $\mathbf{T}^o$. Clearly, $\mathbf{T}^o \cup \mathbf{T}^\ell$ is $\mathbf{T}$ and $\mathbf{T}^o \cap \mathbf{T}^\ell = \emptyset$.

The scheduling problem by considering $\mathbf{T}^o$ to meet the timing constraints can be considered as a task system in which a job may *self-suspend* itself during its execution *once*. That is, after executing $C_{i,1}$ on the embedded system, the task τ_i suspends itself by at most R_i amount of time (via offloading) and resumes for post processing or local compensation. The model is the same as the self-suspending task model, as described in [9]. From [9], it is known that fixed-priority or earliest-deadline-first (EDF) are not efficient to schedule self-suspending tasks. Therefore, most of the existing results for scheduling sporadic real-time tasks cannot be applied directly to handle $\mathbf{T}^o$ effectively. For example, the naive EDF scheduling considers the two execution phases $C_{i,1}$ and $C_{i,2}$ as one job with the same absolute deadline, but this performs poorly.

To satisfy the feasibility on the real-time embedded system, in this section, we propose an EDF-based scheduling algorithm by setting different absolute deadlines for the two executions of $C_{i,1}$ and $C_{i,2}$, as follows:

For a job of task τ_i in $\mathbf{T}^o$ (that is offloaded), arriving at time t, we split this job into two sub-jobs:

- The first sub-job is *triggered immediately at time t*, its relative deadline is $D_{i,1} = \frac{C_{i1}*(D_i-R_i)}{C_{i1}+C_{i2}}$, i.e., absolute deadline is $t + D_{i,1}$, and has worst-case execution time $C_{i,1}$.
- The second sub-job is *triggered immediately when the result returns from the server or R_i expires* and the job of generated by this subtask has an *absolute deadline* $t + D_i$ with worst-case execution time $C_{i,2}$.

Note that the sub-jobs generated by $C_{i,1}$ for task τ_i in task set $\mathbf{T}^o$ is also periodic with period T_i, but the sub-jobs generated by $C_{i,2}$ may not be periodic.

Moreover, for a job of task τ_i in $\mathbf{T}^\ell$ (that is executed locally), arriving at time t, the relative deadline is set to D_i, i.e., absolute deadline is $t + D_i$. After the absolute deadlines are assigned, the scheduling policy will strictly follow the original earliest-deadline-first scheduling by giving the job in the ready queue with the earliest absolute deadline the highest priority.

In order to analyze the feasibility of our algorithm, we define the *demand bound function $dbf(\tau_i, t)$* for each task τ_i, which is the maximum execution time of the *sub-jobs* generated by task τ_i that must be finished within any interval length equal to t. Suppose that the window of interest is an interval $(A, A+t]$. The demand that must to be finished in this interval includes the jobs (of task τ_i) that arrive no earlier than A and have absolute deadline no later than $A+t$. The definition is similar to the original demand bound function definition from Baruah et al. [2].

Theorem 1. *For task $\tau_i \in \mathbf{T}^o$, the demand bound function $dbf(\tau_i, t)$ can be upper bounded by*

$$dbf(\tau_i, t) \leq \frac{C_{i1} + C_{i2}}{D_i - R_i} * t. \tag{1}$$

Proof. The proof is omitted. □

Theorem 2. *For task $\tau_i \in \mathbf{T}^\ell$, the demand bound function $dbf(\tau_i, t)$ can be upper bounded by*

$$dbf(\tau_i, t) \leq \frac{C_i}{T_i} * t \tag{2}$$

Proof. This comes directly from the definition of the demand bound function for sporadic real-time tasks [2]. □

Theorem 3. *For a given task partition, $\mathbf{T}^o$ and $\mathbf{T}^\ell$ and the estimated response time R_i for each task τ_i in $\mathbf{T}^o$, the EDF-based scheduling algorithm in Section 5.1 can feasibly schedule $\mathbf{T}^o$ if*

$$\sum_{\tau_i \in \mathbf{T}^o} \frac{C_{i,1} + C_{i,2}}{D_i - R_i} + \sum_{\tau_i \in \mathbf{T}^\ell} \frac{C_i}{T_i} \leq 1. \tag{3}$$

Proof. Due to the space limitation, we only sketch the proof by using the *contrapositive* argument. That is, assuming the task set is not schedulable, which will lead to the violation of Equation (3). There must be a sub-job which will misses its absolute deadline. Suppose that the first time that a sub-job misses the deadline is time t. Let t_0 be the time before t when the system is idle. Therefore, the necessary condition to have deadline misses is that the demand received from t_0 to t with absolute deadline less than or equal to t is larger than $t - t_0$ [4]. Therefore,

$$\begin{aligned} t - t_0 &< \sum_{\tau_i \in \mathbf{T}} dbf(\tau_i, t - t_0) \\ &\leq_1 \left(\sum_{\tau_i \in \mathbf{T}^o} \frac{C_{i,1} + C_{i,2}}{D_i - R_i} + \sum_{\tau_i \in \mathbf{T}^\ell} \frac{C_i}{T_i} \right) * (t - t_0), \end{aligned}$$

where $\leq_1$ comes from Theorems 1 and 2. Therefore, the deadline miss enforces

$$1 < \sum_{\tau_i \in \mathbf{T}^o} \frac{C_{i,1} + C_{i,2}}{D_i - R_i} + \sum_{\tau_i \in \mathbf{T}^\ell} \frac{C_i}{T_i}, \tag{4}$$

which proves this theorem due to the contrapositive argument. □

5.2 Selection of Offloaded Tasks

Under the scheduling scheme and its schedulability test in Section 5.1, we still have to determine and set the estimated worst-case response time R_i from the server if task τ_i is offloaded. We have to identify their contribution when using the schedulability test in Theorem 3. There are two cases: (1) when $r_{i,j}$ is 0 (i.e., the first point among the Q_i discrete points), the contribution is denoted by $w_{i,1} = \frac{C_i}{T_i}$, and (2) when $r_{i,j} > 0$, the contribution is denoted by $w_{i,j} = \frac{C_{i,1}+C_{i,2}}{D_i - r_{i,j}}$.

For each of the above choice, we define a decision variable $x_{i,j}$. If $x_{i,j} = 1$, it means the estimated worst-case response time $r_{i,j}$ is selected. On the other hand, if $x_{i,j} = 0$, $r_{i,j}$ is not selected. Then the problem to maximize the total benefits of our system can be formulated as follows:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{Q_i} x_{i,j} * G_i(r_{i,j}) \tag{5a}$$

$$\text{s.t.:} \quad \sum_{i=1}^{n} \sum_{j=1}^{Q_i} x_{i,j} * w_{i,j} \leq 1 \tag{5b}$$

$$\sum_{j=1}^{Q_i} x_{i,j} = 1, \qquad \forall i \tag{5c}$$

$$x_{i,j} \in \{0, 1\}, \qquad \forall i, j \tag{5d}$$

The problem described in Equation (5) is the well-known multiple choice knapsack problem. We adopt the dynamic programming algorithm in [5] with pseudo-polynomial time and the HEU-OE heuristic algorithm from [6] to find the near optimal results. After applying these two algorithms, we can decide for each task to offload or not to offload. In addition, the estimated worst-case response time for each task τ_i can be decided. Moreover, based on Theorem 3, the feasibility of the timing satisfactions of these decisions is also guaranteed with local compensations.

For the brevity of notations, we assume that the setup time $C_{i,1}$ and the local compensation $C_{i,2}$ remain the same regardless of the expected estimated response time R_i. With the presentation in this Section, it has also become clear now that the proposed approach can also be used when the benefit to achieve with $r_{i,j}$ estimated response time only needs $C^j_{i,1}$ for setup time and $C^j_{i,2}$ for local compensation. We will evaluate our case study based on this extension.

6. EXPERIMENTS AND EVALUATIONS

6.1 Case Study

6.1.1 *Experimental Setup*

We consider a system equipped with cameras, in which the cameras can capture the images from the environment. Based on the captured images, some image processing algorithms can help the embedded system to make decisions. This is a typical use case when considering mobile robots for making dynamic decisions during the navigation. We consider 4 sporadic real-time tasks (1) Stereo Vision, (2) Edge Detection, (3) Object recognition, and (4) Motion Detection.

When the cameras capture the images from the environment, to satisfy the timing constraint, the four tasks can only handle these images with smaller sizes. We assume that the camera is more powerful than the local computation capability. That is, the camera can provide higher-resolution images, but the local computation capability can only process smaller sizes by scaling the images to satisfy the timing constraints. When the images are scaled, some important informations on the pixels are lost. Sometimes, with the lost information, these image processing applications can not achieve good performance, which will affect the normal execution of the embedded system. However, offloading the complicated computations to the powerful components can reduce the task's execution time. In our system, with the saved execution time, we try to improve the total image qualities for these applications.

In the case study, the client can be any mobile embedded system. The client is connect with a GPU server by the local wireless network. The GPU server has two Telsa M2050 GPUs, which can provide the remote GPU acceleration. Derived from the framework of rCUDA [1], we implement a software proxy application running on the server side. To offer the remote GPU acceleration, the proxy application can generate multiple parallel threads to collect computations from the client and dispatch these computations on GPUs. The proxy application is mainly implemented based on OpenMP 4.0 and CUDA 5.0.

6.1.2 *Benefit and Response Time Estimation*

In our system, for each task τ_i, in the stage of image scaling, we divide the scaled images into Q_i levels. For the different levels, the lost information and image sizes are also different. The scaling level will directly affect the task execution time, the transfer time and the performance. So for each level j, $C^j_{i,1}$ contains data initialization, image scaling time and data transfer time. In our system, the GPU server in the network environment is a timing unreliable component.

For the given images of level j, the response time for task τ_i from the GPU server depends on many factors. We can estimate the worst-case response time $r_{i,j}$ by using coarse-grained statistic estimation of $r_{i,j}$ under the considerations of the network transfer time, receiving time, processing time on the server host (which handles the GPU boards), and the response time on the GPU.

Moreover, to construct the benefit function $G_i(r_i)$, we need to define the benefits of different levels for each task τ_i. The benefits of the four tasks in our case study are different. Specifically, these benefits are related with the scaling levels and image qualities. In this case study, we use the peak signal-to-noise ratio (PSNR) as a quantitative benefit value, which represents the image quality of each scaling level.

Then we can establish $G_i(r_i)$ for the four tasks as Table 1 with statistic data.

6.1.3 *Experimental Results*

We evaluate the following setting of periodic tasks. In order to satisfy the feasibility of all tasks on the CPU, we set the relative deadline of τ_1 and τ_2 as $1.8s$. The relative deadline of τ_3 and τ_4 are $2s$. Then according to the importance of each task, we define the weight value of each task τ_i as $1, 2, 3, 4$.

We consider the three scenarios as follows: the first scenario is that the GPU server in the network condition is busy to process other applications. Only a small number of offloaded tasks can get computation results. The second scenario is that the GPU server in the network condition is not busy, but it still processes some other applications. a part of offloaded tasks can get computation results successfully. The third scenario is that the GPU server is idle and it only process these offloaded tasks. A large number of offloaded tasks can get computation results. We measure the total image quality values of the three scenarios for $10s$.

There are 24 different combinations for the four tasks with four different weight values. We can use dynamic programming algorithm in Section 5.2 to get the offloading decisions for each task, that is optimal for the integer linear programming in Equation (5). As Figure 2, the total weighted image qualities of the three scenarios are normalized to that of the worst case, when no offloaded task get computation results respectively.

From the Figure 2, we can see that when the number of tasks is small, the dynamic programming can always find the optimal results for the integer linear programming in Equation (5). When the timing unreliable component can not deliver computation results, the local compensation mechanism can effectively guarantee the feasibility of the real-time embedded system. For the average case, the image processing applications in our mechanism can achieve better image qualities improvement, which is beneficial to the system.

6.2 Simulation setup and results

In this section, we evaluate the effectiveness of the proposed mechanism when the benefit function $G_i(r_i)$ for task τ_i is erroneous, which aligns with our assumptions. We simulate the dynamic programming algorithm [5] and the HEU-OE algorithm [6] under different estimation errors. The simulated case is to evaluate a system by offloading tasks to a timing unreliable component for higher-performance output, in which $G_i(r_i)$ is the probability to obtain the successful results of task τ_i within response time r_i. The system objective for $\sum_i G_i(R_i)$ is the expected number of higher-performance

Table 1: The construction of $G_i(r_i)$

Task	Description	$G_i(0)$	$r_{i,2}$	$G_i(r_{i,2})$	$r_{i,3}$	$G_i(r_{i,3})$	$r_{i,4}$	$G_i(r_{i,4})$	$r_{i,5}$	$G_i(r_{i,5})$
τ_1	Stereo Vision	22.4897	195.2814 ms	30.5918	207.4508 ms	33.2853	222.2878 ms	36.6047	236.502 ms	99
τ_2	Edge Detection	28.1574	253.3242 ms	35.0431	312.4523 ms	37.7277	362.4235 ms	41.4977	420.341 ms	99
τ_3	Object recognition	23.9059	148.2351 ms	28.5648	161.4224 ms	31.9884	174.3242 ms	35.3082	188.803 ms	99
τ_4	Motion Detection	21.0324	343.637 ms	28.3015	485.459 ms	32.957	622.091 ms	36.1414	891.36 ms	99

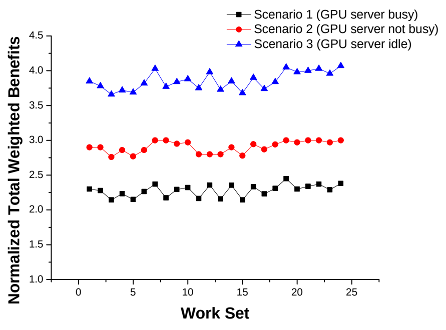

Figure 2: Case study results

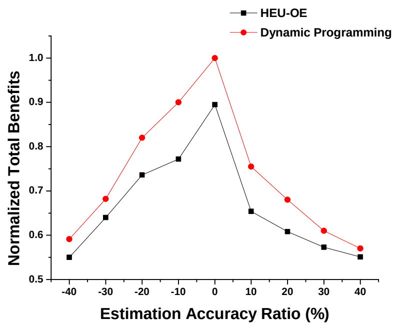

Figure 3: Simulation Results

tasks returned in time in the schedule.

A set of 30 real-time tasks are randomly generated. In the simulation, each task τ_i is generated as follows:

- $C_{i,1}$ and C_i are random values from 0 to $20ms$, $C_{i,2}$ is equal to C_i. D_i, which is equal to T_i, is a random integer value from $600ms$ to $700ms$.
- In benefit function $G_i(r_i)$, the benefit values are probability values to get computation results 10%, 20%, ..., 100%. The associated estimated response time is randomly generated from $100ms$ to $200ms$ with an increasing order.

Since the Benefit and Response Time Estimator does not require perfect information of the $G_i(r_i)$, we simulate the algorithms under different estimation accuracy ratios. Specifically, when the estimate accuracy ratio is x, the Benefit and Response Time Estimator uses $G((1 + x) \cdot r_i)$. The negative estimation accuracy ratio (i.e., $x < 0$) means an under-estimation of the response time. Therefore, the probability function to get the result within response time r_i is over-estimated. The positive estimation accuracy ratio (i.e., $x > 0$) means an over-estimation of the response time. Therefore, the probability function to get the result within response time r_i is under-estimated.

Figure 3 presents the normalized total benefit derived from the dynamic programming algorithm and the HEU-OE algorithm under different estimation accuracy ratio, by normalizing to the perfect estimation (i.e., $x = 0$) with dynamic programming. From the Figure 3, we can see that the accurate benefit function $G_i(r_i)$ is important for the system improvement. Under-estimating or over-estimating the response time from the timing unreliable component will introduce wrong decisions for offloading tasks. If the response time estimation is over-estimated, the offloading option will not be taken. On the other hand, if the response time estimation is under-estimated, the offloading option may be taken, and the local compensation is more frequently adopted than expectations.

7. CONCLUSIONS

In this paper, we propose a computation offloading mechanism to utilise some powerful timing unreliable components in a hard real-time system. We propose an Earliest-Deadline-First based algorithm to schedule offloaded tasks with estimated response time and analyse the feasibility of the compensation mechanism. By establishing a benefit function for each task, we adopt two algorithms from the multiple choice knapsack problem to decide the estimated worst-case response time for each task. Then, from a case study, we can see that our mechanism can effectively improve the system performance. In addition, from the simulations, we can see the effectiveness of the algorithms from multiple choice knapsack problem with different estimation accuracy.

Acknowledgement This work was partially sponsored by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems", Deutscher Akademischer Austauschdienst (DAAD), Taiwan MOST under Grant No. 100-2221-E-002-120-MY3, the National Key Technology Research and Development Program of China Grant No. 2012BAF13B08 and Grant No. 2012BAK24B0104, Research Fund for the Doctoral Program of China Grant No. 20110042110021.

References

[1] The rCUDA website. `http://www.rcuda.net/`.

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

[3] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli. Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices. *Parallel and Distributed Systems, IEEE Transactions on*, 15(9):795–809, 2004.

[4] H. Chetto and M. Silly-Chetto. Scheduling periodic and sporadic tasks in a real-time system. *Inf. Process. Lett.*, 30(4):177–184, 1989.

[5] K. Dudziński and S. Walukiewicz. Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research*, 28(1):3–21, 1987.

[6] S. Khan. Quality adaptation in a multi-session adaptive multimedia system: model and architecture. *Canada: Department of Electronical and Computer Engineering, University of Victoria. Thesis (PhD)*, 1998.

[7] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *CASES*, pages 238–246, 2001.

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

[9] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Real-Time Systems Symposium*, pages 47–56, 2004.

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

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

[12] 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.