On Task Period Assignment in Multiprocessor Real-Time Control Systems

Abhishek Roy, Hakan Aydin
Department of Computer Science
George Mason University
Fairfax, VA 22030
aroy6@gmu.edu, aydin@cs.gmu.edu

Dakai Zhu
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
dakai.zhu@utsa.edu

ABSTRACT

In real-time control systems, a well-known problem is the period assignment to individual tasks, in order to minimize the overall control cost while guaranteeing the task deadlines. In general, the control cost decreases in convex fashion with decreasing periods (increasing invocation rates). Many real-time control systems are increasingly implemented on multiprocessor platforms due to the increased performance requirements. In this paper, we consider the optimal period assignment problem on a homogeneous multiprocessor platform. The problem is intractable in nature. We analyze the performance of the approaches that first partition the tasks, before assigning periods to optimize overall cost on each CPU locally. Then we propose a technique which assigns the periods optimally by reducing the problem to a single-processor problem setting in the first step, and then applying the partitioning algorithms in the second step. Our experimental evaluation shows that the two variants of our proposed technique offer significant advantage, and exhibit a performance close to the theoretical bound achievable by any algorithm.

Keywords

Real-time Control; Period Assignment; Multiprocessors

1. INTRODUCTION

Digital controllers are an integral part of many cyber-physical and embedded systems, including those in industrial control and automation, avionics, transportation, and medical monitoring. A typical real-time control task is activated periodically, and it has to perform the sense-compute-actuate cycle with timing guarantees. However, at design time, there is often some flexibility for assigning the activation period (invocation rate) of a control task. Based on the dynamics of the state variable that the control task is associated with, the designer can derive bounds that must be satisfied by the sampling period. This flexibility can be used towards a better utilization of the system resources [22].

RTNS '16, Oct 19–21, 2016, Brest, France
© 2016 ACM. ISBN 978-1-4503-4787-7/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2997465.2997469

The controller's performance (sometimes referred to as *performance index*) is maximum when a sampling period is set to an ideal value (i.e., the performance of the digital controller approaches that of an ideal analog controller) [17]. Decreasing the period below a limit does not further improve the performance; however the performance degrades as the period value is increased, due to the corresponding decrease in the frequencies of sampling and control signal. In fact increasing the period further may cause stability problems. Hence, from the digital control point of view, there are constraints on the minimum and maximum periods that should be considered at design time.

In general, the computational capacity of the target system may not be sufficient to schedule all the tasks at their preferred (minimum) activation periods, or in other words, at their maximum invocation rates. It may often be necessary to increase the sampling periods of a number of tasks, thereby decreasing their utilizations. Many researchers have worked on characterizing the performance degradation (or, control cost) as a function of the periods for a single processor system [9, 22, 23]. Seto et al. described a general class of performance index, and showed that for a large class of real-time control applications, the performance index can be characterized by convex functions [22]. In other words, in these systems the control cost increases sharply as the sampling period moves further from the lower bound. Bini and Di Natale [5] also considered convex functions, and analyzed the problem for fixed-priority scheduling on single processor systems.

Most of the existing research on optimal period assignments in real-time control systems consider single-processor systems. However, modern control systems are increasingly implemented on multiprocessor platforms due to the increased performance requirements. Hence, the main objective of this paper is to investigate the period assignment problem for homogeneous multiprocessor systems.

In general, multiprocessor real-time scheduling algorithms can be divided in two categories: *partitioned scheduling* and *global scheduling* [8]. In partitioned scheduling, tasks are statically distributed across processors, and tasks are not allowed to migrate during execution. In contrast, global scheduling allows task migration. In general, global scheduling allows more flexibility and optimal system utilization (for periodic tasks) – but partitioned scheduling is known to reduce the run-time overhead by avoiding cache affinity and task migration related overheads. In this paper, we focus on partitioned scheduling: we consider the problem of assigning periods to individual tasks and allocating them to

M processors in order to minimize the overall control cost.

A solution approach for this problem is to first partition the tasks (using one of the heuristics based on initially assigning each task its maximum allowable period), and then on each processor, to make the period assignment to minimize the cost locally. We call this class of algorithms *local period assignment algorithms*. As we demonstrate, a limitation of this approach is that the algorithm does not have any global view of the task set and it cannot use that information towards minimizing control cost. In this paper, we identify another approach which allows assigning the periods *before* the partitioning step, to find the periods that minimize the overall cost *globally*. This is achieved by solving the problem on a single-processor system which is K times faster ($K \leq M$), and then attempting partitioning with the periods obtained in that way. We call this scheme *Reduction-to-Single-Processor (RTSP)* technique. We propose two variants of this approach, one which gets the period assignments by setting $K = M$, and another one which employs a binary search to find the most appropriate value for K.

Based on the Earliest-Deadline-First (EDF) scheduling on each processor [16], we perform an experimental evaluation of the local algorithms and the new *RTSP* approach. Our results indicate that *RTSP* can yield significant gains in minimizing the control cost compared to local algorithms, especially when the system is heavily loaded. We show that for most of the parameter spectrum, the control cost incurred by *RTSP* stays close to the theoretical lower bound. Our technique can be applied to any system with real-time control tasks where the cost is represented by a continuous and differentiable convex function. To the best of our knowledge, this is the first research effort that considers the period assignment problem on a multiprocessor real-time control system.

The rest of the paper is organized as follows: Section 2 describes the system model and assumptions. Section 3 formulates the problem formally and illustrates how it can be solved optimally on a single processor system for general convex functions. In Section 4, we present the algorithms we develop for the multiprocessor platforms. Section 5 presents the experimental results. Section 6 overviews the research area related to our work and Section 7 has our concluding remarks.

2. MODELS AND ASSUMPTIONS

We consider a set of periodic real-time control tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$. The worst-case execution time of task τ_i is denoted by C_i. The periods of the real-time tasks can be selected within a given range. Specifically, the *nominal* (minimum) period of τ_i is given by P_i^{min}; however its period P_i can be increased (at the cost of increased control cost), up to a limit P_i^{max}. P_i^{min} represents the minimum task period for which the controller's performance is best – this is the point where the performance of the discrete-time optimal controller is identical with the performance of the continuous-time optimal controller [22]. As the task period increases, the difference between the discrete-time controller and continuous-time controller increases, and if it is larger than a certain upper bound (P_i^{max}), the stability can no longer be guaranteed. The ratio of the maximum period to the minimum period for a given control task τ_i is called the *elasticity factor* and is denoted by EF_i: $EF_i \triangleq \frac{P_i^{max}}{P_i^{min}}$. We consider implicit-deadline task sets where the relative deadline of each task is equal to its assigned period P_i.

The invocation frequency f_i of task τ_i is the inverse of its period, that is, $f_i = \frac{1}{P_i}$. The minimum and maximum task invocation frequencies are defined, respectively, as $f_i^{min} = 1/P_i^{max}$ and $f_i^{max} = 1/P_i^{min}$. The *utilization* of a task is defined as the ratio of its worst-case execution time to its assigned period. As the utilization is a function of the assigned period, we can define the minimum and maximum utilization of a task τ_i as $U_i^{min} = C_i/P_i^{max} = C_i * f_i^{min}$ and $U_i^{max} = C_i/P_i^{min} = C_i * f_i^{max}$, respectively.

The set of real-time tasks are scheduled upon a *homogeneous* multiprocessor system that consists of M unit-speed processors, denoted by the set $\Pi = \{\Pi_1, \ldots, \Pi_M\}$. We assume that the tasks are partitioned over the set of available processors, that is, migration of tasks from one processor to another is not allowed. The tasks allocated to a given processor are scheduled by the preemptive *Earliest-Deadline-First (EDF)* policy. It is known that the preemptive EDF is optimal for scheduling on uniprocessor systems, in that it can generate a feasible schedule as long as the total utilization of the tasks assigned to that CPU does not exceed 100% [16].

The control cost performance of a task decreases with its invocation frequency. In literature, it is generally assumed that the control performance is a *convex* function of the invocation frequency [22]. The cost function we will use can be any generic convex function which we denote by $J(f)$, where f is the invocation frequency of the task. This function is specific to each task, and the task-specific constants can be used to formulate the function. Therefore, for a task τ_i, the cost function becomes $J_i(f_i)$. We require that $J_i(f_i)$ is convex, continuous, differentiable, and monotonically decreasing, as in [22]. The most common control cost function used in the real-time systems literature is the exponential decay function of the form $J_i(f_i) = \alpha_i \cdot e^{-\beta_i \cdot f_i}$ [22]. In the literature, other cost functions that incorporate additional factors (such as jitter) have been proposed (e.g., [4,15,20]), but in this paper we are considering only the impact of the period assignment on the controller's performance.

Since the nominal (maximum) invocation frequency is the most preferable setting for every task period, we assume that the cost for a task is 0 if it is running on its maximum frequency. Hence, $J_i(f_i^{max}) = 0 \ \forall i$. In other words, for the common exponential decay functions, we are using the form $J_i(f_i) = \alpha_i e^{-\beta_i f_i} - \alpha_i e^{-\beta_i f_i^{max}}$.

For a task set, the total (overall) cost will be the sum of the costs associated with each task.

$$J_{tot} = \sum_{\tau_i \epsilon \Gamma} J_i(f_i)$$

We note that the derivation of worst-case execution time (WCET) figures on a multiprocessor system may be non-trivial, due to the access to the shared resources such as the memory and I/O subsystems. However, this timing analysis aspect is outside the scope of this paper – the objective of this paper is to find the period assignments to a set of real-time control tasks whose timing analysis on the target multiprocessor has been already performed. Similarly considering more general task models (such as those that allow the specification of the precedence constraints) is left as future work.

3. MINIMIZING CONTROL COST

3.1 Problem Definition

The problem addressed in this paper, denoted by MIN-COST-PARTITION, can be stated as follows:

Given M homogeneous processors and a set of periodic real-time tasks, how can we assign the task periods and partition the tasks among the processors such that:

1. *the task set assigned to each processor can be scheduled in a feasible manner using the assigned frequencies, and,*
2. *the overall control cost across all processors is minimized*

This problem can be easily seen to be NP-Hard: Consider a special case where the allowable minimum and maximum periods are the same, i.e., $P_i^{min} = P_i^{max} \quad \forall i$. In this case, the period of each is task is a fixed value, and the problem reduces to the problem of partitioning tasks with due dates on a multiprocessor platform, which is known to be NP-Hard in the strong sense [13]. Hence, MIN-COST-PARTITION is also NP-Hard in the strong sense.

Before discussing efficient heuristic approaches to address this problem, we briefly turn our attention to the single-processor case.

3.2 Optimization on a single processor with arbitrary convex control cost functions

The problem of minimizing the overall cost on a single processor has been considered in the seminal paper by Seto et al. [22]. We re-visit the same problem, and show how a more general solution can be obtained for arbitrary convex cost functions as long as the function is continuous and differentiable (the seminal work in [22] considered only exponential decay functions). We also incorporate lower bounds on the allowed task periods to indicate the task's preferred (nominal/minimum) periods. Finally, we show that this problem can be solved in polynomial-time (in the number of tasks), for arbitrary convex functions.

As stated previously, EDF is used as the scheduling algorithm for each processor. The schedulability condition for EDF is simple: the total utilization of the task set should not exceed 1.0. We can now formally define the problem on a single processor (denoted by UNI-MIN-COST) as follows:

$$\begin{aligned} \underset{f_1,..,f_n}{\text{minimize}} \quad & \sum_{\tau_i \in \Gamma} J_i(f_i) \\ \text{subject to:} \quad & \sum_{\tau_i \in \Gamma} C_i * f_i \leq 1.0 \\ & f_i^{min} \leq f_i \leq f_i^{max} \end{aligned} \tag{1}$$

where $J_i(f_i)$ is a continuous, differentiable, and monotonically decreasing convex function.

We now define $R_i(f_i) = -J_i(f_i)$. Note that $R_i(f_i)$ is a concave, continuous, and monotonically increasing function. Then, we define a new variable y_i such that $f_i = y_i + f_i^{min}$ and a new function $Q_i(y_i)$ such that

$$Q_i(y_i) = R_i(f_i) = R_i(y_i + f_i^{min})$$

Observe that $Q_i()$ is effectively a concave and non-decreasing *reward* function indicating the controller's performance. The transformed problem is defined as:

$$\begin{aligned} \underset{y_1,..,y_n}{\text{maximize}} \quad & \sum_{\tau_i \in \Gamma} Q_i(y_i) \\ \text{subject to:} \quad & \sum_{\tau_i \in \Gamma} C_i * y_i \leq B \\ & 0 \leq y_i \leq f_i^{max} - f_i^{min} \\ \text{where} \quad & B = 1.0 - \sum_{\tau_i \in \Gamma} C_i * f_i^{min} \end{aligned} \tag{2}$$

Observe that $\sum_{\tau_i \in T} C_i * y_i$ has to be bounded by the constant B. If B is large enough, then clearly assigning $y_i = f_i^{max} - f_i^{min} \; \forall i$ would also maximize the objective function due to the non-decreasing nature of the reward function. Otherwise, this quantity B should be used in its entirety since the total reward never decreases by doing so. In this case, we obtain a constrained concave (non-linear) optimization problem with upper and lower bounds.

$$\begin{aligned} \underset{y_1,..,y_n}{\text{maximize}} \quad & \sum_{\tau_i \in \Gamma} Q_i(y_i) \\ \text{subject to:} \quad & \sum_{\tau_i \in \Gamma} C_i * y_i = B \\ & 0 \leq y_i \leq f_i^{max} - f_i^{min} \\ \text{where} \quad & B = 1.0 - \sum_{\tau_i \in \Gamma} C_i * f_i^{min} \end{aligned} \tag{3}$$

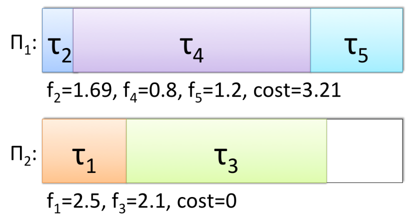
This problem turns out to be an instance of the general reward maximization problem considered in [1]. In fact, Aydin et al. [1] developed an iterative solution that considers the unconstrained optimization problem without the lower and upper bounds, and then iteratively adjusts the solution in time $O(n^2 \log n)$ to solve the constrained optimization problem. We refer the reader to [1] for full details. Hence, UNI-MIN-COST can be also solved in time $O(n^2 \log n)$ for arbitrary convex functions.

3.3 Optimization on a multiprocessor platform

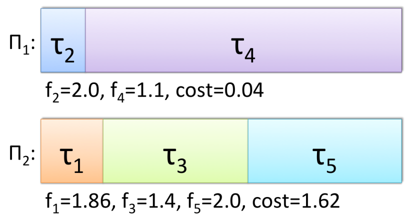
As shown in Section 3.1, overall control cost minimization on a multiprocessor setting with partitioned approach is, in general, intractable[1]. The problem has two important components, assigning periods to minimize the cost, and generating a feasible partitioning. While there are well-known effective partitioning algorithms (such as First-Fit and Worst-Fit), as the following example illustrates, using a different approach may significantly reduce the overall cost.

Example 1. Table 1 shows a task set with 5 tasks ($\tau_1 - \tau_5$) that are to be scheduled on a system with two identical processors Π_1 and Π_2. The given task set can be partitioned among the processors and then the UNI-MIN-COST algorithm can be used to compute the periods to minimize the cost on each processor. However, different partitionings will result in different values of *overall cost*, as can be seen in Figure 1. If we set all the task utilizations to their minimum possible

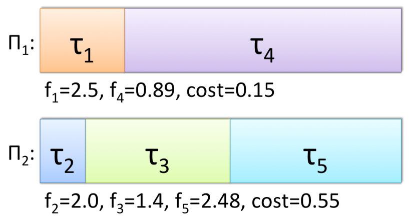
[1]It is worth mentioning that the optimal periods could be obtained for *global scheduling* by replacing the overall utilization bound 1.0 by M in Equation (1), and then using an optimal global scheduler such as *PFair* [3]. However, the focus of this paper is the partitioned approaches which are known to have lower run-time overhead.



(a) Partitioning 1 using FFD/BFD, total cost = 3.21



(b) Partitioning 2 using WFD, total cost = 1.66


Π1: τ1 τ4
f1=2.5, f4=0.89, cost=0.15
Π2: τ2 τ3 τ5
f2=2.0, f3=1.4, f5=2.48, cost=0.55

(c) Partitioning 3, total cost = 0.70

Figure 1: Partitioning Options for Example 1

Table 1: Task set for Example 1

Name	β_i	α_i	$f_i^{min}[Hz]$	$f_i^{max}[Hz]$	$C_i(ms)$
τ_1	0.3	4.42	1.7	2.5	105
τ_2	0.4	9.68	1.3	2.0	45
τ_3	0.6	3.56	1.4	2.1	260
τ_4	0.7	1.42	0.8	1.2	825
τ_5	0.8	9.86	1.2	2.5	220

values (by using the maximum periods), and then apply a well-known partitioning heuristic, e.g., First-Fit-Decreasing (FFD), we obtain the partitioning in Figure 1a. After optimizing the frequencies on each individual processors with `UNI-MIN-COST`, we get a total overall cost of 3.21. Note that, if we had used Best-Fit-Decreasing (BFD), instead of FFD, we would obtain the same partitioning, incurring the same total cost. When Worst-Fit-Decreasing (WFD) is used as the partitioning heuristic, we found the partitioning in Figure 1b, yielding a total cost of 1.66, providing 48% improvement over FFD. However, there is a completely different partitioning, shown in Figure 1c, giving an overall cost of 0.70, and 58% improvement over WFD in terms of minimizing cost. In fact, this partitioning is produced by the RTSP* algorithm that we will develop in the next Section.

4. PROPOSED ALGORITHMS

4.1 Local period assignment algorithms

For a given processor and its assigned task set, `UNI-MIN-COST` provides the optimal invocation frequencies for all tasks within their respective upper and lower bounds. Hence, an intuitive way to address the control cost minimization on a multiprocessor platform is to perform task partitioning first using some well-known partitioning heuristic, and then call `UNI-MIN-COST` in order to assign periods for minimizing the cost *locally* on each processor.

Some well-known partitioning heuristics are *First-Fit*, *Worst-Fit*, and *Best-Fit*. In order to make the partitioning problem easier, respective minimum frequencies (maximum periods) are *tentatively* assigned to each task before partitioning is performed. It is well-known that in general, the performance of the partitioning heuristics improves if the tasks are ordered according to decreasing utilization [18]. This yields heuristics such as *First-Fit-Decreasing (FFD)*, *Worst-Fit-Decreasing (WFD)*, *Best-Fit-Decreasing (BFD)*. After the partitioning phase is over, the final period values to minimize the cost for each processor are obtained separately. This gives us a family of algorithms (that we call *local* algorithms); the algorithms differ only in the special partitioning heuristic used in the initial step. Depending on the specific partitioning algorithm used, we obtain different local algorithms called, for example, *WFD-Local, FFD-Local, BFD-Local*. Hence, a local period assignment algorithm has the following two steps:

1. Partition the given task set among the processors, by tentatively assigning $P_i = P_i^{max}$ to all tasks, and using a well-known heuristic such as FFD or WFD,

2. Invoke `UNI-MIN-COST` to assign optimal task period values in order to minimize cost on each processor, separately.

Note that if we do not have a feasible schedule after Step 1 (with maximum periods), we can declare that the task set is not schedulable using the local approach and the given heuristic. Otherwise, we call `UNI-MIN-COST` on each processor individually to adjust the periods of all tasks to optimal values locally. As it can be easily seen, the complexity of the local algorithms is determined primarily by Step 2, which is $O(M \cdot n^2 \cdot \log n)$.

4.2 Reduction to Single Processor (RTSP) Technique

Despite their intuitive nature, the local period assignment algorithms suffer from an obvious deficiency: The partitioning step is performed in a way which is completely incognizant of the specific cost functions. This is important because at the end of the partitioning, for example, tasks with very steep cost functions may be allocated to the same processor, limiting the potential of cost minimization on that specific processor through the invocation of `UNI-MIN-COST` algorithm.

An alternative approach that we consider is to *reverse* the order of steps undertaken by the local algorithms: If we can make the (near-)optimal period assignments in the first step by considering the *entire* task set, that is very likely to reduce the overall cost. This is, of course, based on the assumption that a feasible partitioning will be found with those suggested period assignments.

Specifically, in this approach, we first consider a *single-processor* Π_0 which is M times faster than each of the individual (unit-speed) processors in the original problem. Observe that Π_0 offers an aggregate computational capacity which is the same as the total computational capacity of all (unit-speed) processors in Π. The entire task set Γ is assigned to the faster processor Π_0, and `UNI-MIN-COST` is applied to get the optimal periods for all tasks. After that, one of the partitioning heuristics, (e.g., FFD, WFD, or BFD) can be used to partition the task set upon the original processor set Π. Note that, instead of using maximum periods for each task, RTSP uses the optimal periods while making partitioning decisions.

It is possible that, due to the inherent difficulty of partitioning problem, we will not be able to allocate some tasks with the assigned optimal periods. During the partitioning phase, the tasks that cannot be assigned to any processor are put in a list of *unassigned-tasks*. Next, this list is ordered according to decreasing value of utilization ($C_i * f_i$). From this list, tasks are picked one after the other and put to the processor that has the least value of *normalized local cost*, which is given by $(\sum_{\tau_i \in \Gamma_j} J_i(f_i) / \sum_{\tau_i \in \Gamma_j} J_i(f_i^{min}))$, where Γ_j is the set of tasks already assigned to the j^{th} processor. Finally, `UNI-MIN-COST` is called on each processor to adjust the frequency for each task to preserve the feasibility with minimum cost increase. Algorithm 1 presents the pseudocode for RTSP.

Complexity: It can be seen that the complexity of the RTSP algorithm is dominated by the for loop in lines 20-24. In that loop, the algorithm `UNI-MIN-COST` is invoked separately for each of the M processors. Given that the number of tasks on each processor is bounded by n and, the complexity of `UNI-MIN-COST` is $O(n^2 \cdot \log n)$, the overall complexity of the algorithm is $O(M \cdot n^2 \cdot \log n)$, which is the same as that of the local algorithms.

RTSP:* The RTSP scheme has a shortcoming which is immediately noticeable: the total utilization of the task set after the optimal period assignment on processor Π_0 is typically very close to M, the number of available unit-speed processors in the original problem. In other words, we attempt to partition a task set with total load equal to the available multiprocessor computing capacity. This is the *hardest* any partitioning problem can become. As a result, the list of *unassigned-tasks* is almost never empty; and those tasks are allocated to some processors using the *minimum normalized-cost* heuristic, before invoking `UNI-MIN-COST` on those processors and potentially modifying the originally assigned optimal period values to preserve the feasibility.

To overcome this shortcoming, we propose an improved version of RTSP, called RTSP*. Similar to RTSP, RTSP* also assumes a hypothetical processor which is x times faster than a unit-speed processor. But now, x is chosen as the largest value $\leq M$ for which the task set can be feasibly partitioned among the M processors, with the suggested periods. After the partitioning has been completed, `UNI-MIN-COST` is called individually on each processor to further reduce the cost, in case there is some idle capacity on certain processors. The value of x is determined using binary search. The search space ranges from $L = \sum_{\tau_i \in \Gamma} U_i^{min}$ to the total capacity of the system, M. The value of the lower bound L is determined by considering that even if all the tasks are assigned their maximum periods (minimum utilizations), they cannot be possibly scheduled on a platform which offers a computational capacity less than the total utilization $\sum_{\tau_i \in \Gamma} U_i^{min}$. The binary search algorithm continues to shrink the search space until the difference between the upper and lower bound comes within a pre-specified error-value, ε. Algorithm 2 presents the pseudocode. In fact,

Algorithm 1 RTSP

```
1: Input: Task set ($\Gamma$), Processor Set ($\Pi$)
2: Output: Task assignments and frequencies
3: $speed\_up \leftarrow M$
4: for all $\tau_i \in \Gamma$ do
5:     $C_i \leftarrow C_i / speed\_up$
6:     $f_i \leftarrow f_i^{min}$
7: end for
8: if $\sum \frac{C_i}{f_i} > 1$ then
9:     return $failure$
10: end if
11: Call UNI-MIN-COST to get optimal frequencies $\{f_i^*\}$ for all tasks in $\Gamma$
12: for all $\tau_i \in \Gamma$ do
13:     $C_i \leftarrow C_i * speed\_up$
14: end for
15: Apply partitioning scheme, (e.g., FFD) to partition $\Gamma$ upon $\Pi$ with the $\{f_i^*\}$ frequencies, put infeasible tasks to unassigned_list
16: Order unassigned_list by decreasing value of $C_i * f_i$
17: for all $\tau_i \in$ unassigned_list do
18:     Assign $\tau_i$ to the processor with the least value of $\sum_{\tau_i \epsilon \Gamma_j} J_i(f_i) / \sum_{\tau_i \epsilon \Gamma_j} J_i(f_i^{min})$, where $\Gamma_j$ is the set of tasks already assigned to the $j^{th}$ processor
19: end for
20: for all processors $\Pi_i \in \Pi$ do
21:     Call UNI-MIN-COST to get optimal frequencies $\{f_i^*\}$ for all tasks on processor $\Pi_i$
22:     if UNI-MIN-COST fails to return frequencies then
23:         return $failure$
24:     end if
25: end for
26: return task-to-processor assignments $\{\Gamma_i^*\}$ and frequency assignments $\{f_i^*\}$
```

in Example 1, the best partitioning shown in Figure 1c was obtained using RTSP*.

Complexity: In each of the binary search iterations from line 6 to line 36, first `UNI-MIN-COST` is called for the entire task set, which takes time $O(n^2 \log n)$ for a given *speed_up* value. To calculate the number of iterations that the binary search takes before it converges to a value within ε of the actual value (or, fails), we divide the speedup value range into equal chunks, each of size ε, which is a configurable search parameter. Let the total number of chunks be k, then $k = (M - \sum_{\tau_i \epsilon \Gamma} U_i^{min})/\varepsilon$, so the binary search would complete in at most $O(\log k)$ iterations. By factoring also the complexity of FFD in line 18, the overall complexity of RTSP* is found as $O(M \cdot n^2 \log n \cdot \log k)$ which is slightly higher than that of RTSP.

5. EVALUATIONS AND DISCUSSIONS

We developed a discrete-event simulator to evaluate the proposed schemes. By varying various system parameters, the proposed schemes are evaluated through extensive simulations with synthetic tasks. In the simulator, the following schemes are implemented.

WFD-local: Based on their minimum activation rates, tasks are first partitioned to CPUs according to the well-known heuristic WFD (Worst-Fit Decreasing). Then, the `UNI-MIN-`

Algorithm 2 RTSP*

```
1:  Input: Task set ($\Gamma$), Processor Set ($\Pi$)
2:  Output: Task assignments and frequencies
3:  $upper \leftarrow M$
4:  $lower \leftarrow \sum_{\tau_i \epsilon \Gamma} U_i^{min}$
5:  $speed\_up \leftarrow lower$
6:  loop
7:      for all $\tau_i \in \Gamma$ do
8:          $C_i \leftarrow C_i / speed\_up$
9:          $f_i \leftarrow f_i^{min}$
10:     end for
11:     if $\sum \frac{C_i}{f_i} > 1$ then
12:         return $failure$
13:     end if
14:     Call UNI-MIN-COST to get optimal frequencies $\{f_i^*\}$ for
        all tasks in $\Gamma$
15:     for all $\tau_i \in \Gamma$ do
16:         $C_i \leftarrow C_i * speed\_up$
17:     end for
18:     Apply partitioning scheme, (e.g., FFD) to partition $\Gamma$
        upon $\Pi$ with the $\{f_i^*\}$ frequencies
19:     if a feasible partitioning is obtained then
20:         $lower \leftarrow speed\_up$
21:         if $upper - lower \leq \varepsilon$ then
22:             for all processors $\Pi_i \in \Pi$ do
23:                 Call UNI-MIN-COST to get optimal frequencies
                    $\{f_i^*\}$ for all tasks on processor $\Pi_i$
24:             end for
25:             return task-to-processor assignments
26:         else
27:             $speed\_up \leftarrow (upper + lower)/2$
28:         end if
29:     else
30:         if $speed\_up = lower$ then
31:             return $failure$
32:         end if
33:         $upper \leftarrow speed\_up$
34:         $speed\_up \leftarrow (upper + lower)/2$
35:     end if
36: end loop
```

COST problem is solved to minimize the cost on each CPU. We note that we also implemented other local algorithms such as FFD and BFD; but they were consistently outperformed by WFD which tends to distribute tasks more evenly across processors and thereby providing better opportunities for minimizing the total cost. Hence, in the evaluation section, we consider WFD as the representative of the local algorithms.

RTSP: The proposed *Reduction-to-Single-Processor (RTSP)* scheme, where a hypothetical processor with the speed of M is adopted to first optimize the activation rates for tasks. Then, based on the resulting optimal activation rates, tasks are partitioned upon M unit-speed CPUs according to the well-known heuristics (such as FFD, BFD and WFD). In what follows, we show only the results for FFD when used in conjunction with RTSP, because BFD and WFD perform similar or worse. Again, once tasks are mapped to CPUs, the activation rates for tasks are further adjusted by solving the UNI-MIN-COST problem on each CPU (See Algorithm 1).

RTSP*: This is the enhanced RTSP scheme. The speed of the hypothetical processor that ensures the feasible partitioning of all tasks is determined through binary search. See Algorithm 2 for details. Again, we use FFD to partition tasks. For binary search error threshold, we adopted $\varepsilon = 0.01$, which leads to acceptable running time and reasonable accuracy for the results.

Bound: By assuming that all tasks run on a single hypothetical processor with the speed of M, we obtain the optimal activation rates of tasks by solving the UNI-MIN-COST problem, which lead to the lowest possible total cost for all tasks. Basically, *Bound* can be considered as a yardstick algorithm to illustrate how well other schemes perform; because it does not consider the challenges of partitioning while keeping total computational capacity the same as the original M-processor system. No feasible partitioning on an M-processor system can yield an overall cost lower than the one provided by *Bound*, in other words, it represents the *lower bound* on the total cost that *any* algorithm can have.

Simulation Settings. In the simulations, we vary the following system parameters: the number of tasks in each task set n, the number of CPUs M, and tasks' elasticity factors. As the computational capacity of a multiprocessor system increases with the number of processors M, we represent the system load as relative to the maximum computational capacity on a given multiprocessor platform. This quantity representing the system load is called *normalized utilization* ($U_{tot}^{\mathcal{N}}$) , and is defined as the ratio of the total utilization to the number of processors, that is, $U_{tot}^{\mathcal{N}} = \frac{\sum U_i^{max}}{M}$.

For a given set of n, M, and $U_{tot}^{\mathcal{N}}$, the synthetic tasks are generated as follows. First, the total system utilization is found as $U_{tot} = U_{tot}^{\mathcal{N}} * M$. Then, the *RandFixedSum* algorithm in [11] is adopted to generate the maximum utilization U_i^{max} for each task τ_i such that: (1) the utilization is randomly chosen between 0 and 1 following a uniform distribution; and (2) the summation of the utilization values for all tasks in the set is exactly equal to the total utilization U_{tot}. Next, the minimum period P_i^{min} for each task τ_i is randomly selected within the range of $[10, 100]$ following a log-uniform distribution. A uniform elasticity factor, EF, is chosen for all tasks in a task set, where $P_i^{max} = EF \cdot P_i^{min}$. The inverse of minimum and maximum periods give maximum and minimum task frequencies (f_i^{max} and f_i^{min}), respectively. Finally, *worst-case execution time (WCET)* for each task can be calculated based on its minimum period and maximum utilization as $C_i = U_i^{max} * P_i^{min}$ (See Section 2).

In our evaluations, we use the exponential decay functions [22] that have the form of $J_i(f_i) = \alpha_i e^{-\beta_i f_i} - \alpha_i e^{-\beta_i f_i^{max}}$. The constant term $\alpha_i e^{-\beta_i f_i^{max}}$ guarantees that $J_i(f_i^{max}) = 0$. We distinguish four different cost functions.

- **Type-0:** For all tasks in a task set, α_i is set to 1, and β_i is set to 0.1 statically (*Uniform* cost functions).
- **Type-1:** For any task in a task set, the *weight* α is chosen randomly from a uniform distribution between 1 to 10. β is set to a constant value of 0.1 for all tasks (*Uniform weight - different decay parameter* cost functions).
- **Type-2:** For all tasks in a task set, the *weight* α is statically set to 1, while β is randomly chosen within the range of $(0.0, 0.25]$ following a uniform distribution (*Different weight - uniform decay parameter* cost functions).

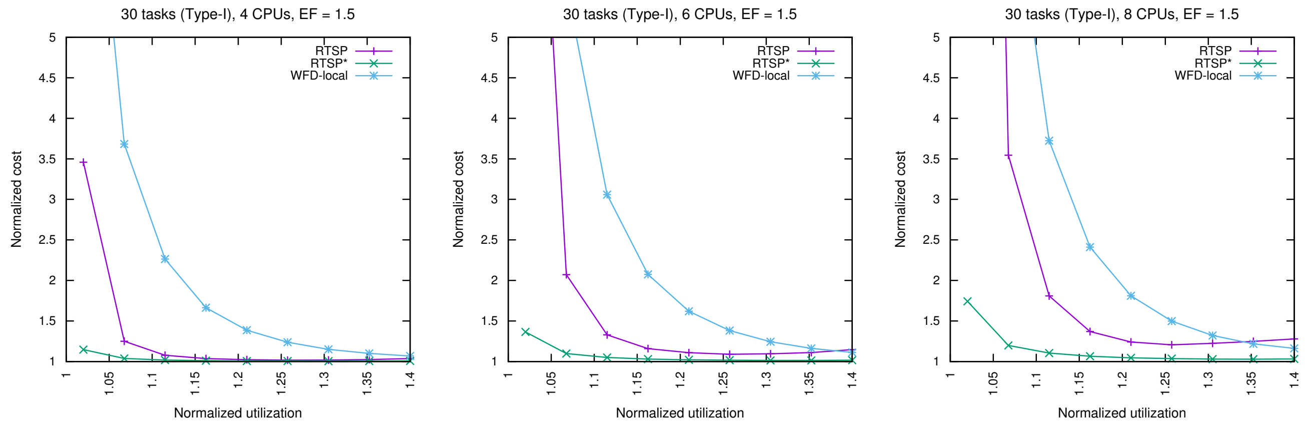


Figure 2: Normalized cost for different schemes with varying system utilization; $EF = 1.5$

- **Type-3:** For any task in a task set, the *weight* α is randomly chosen between 1 to 10, and β between 0.0 to 0.25, both following a uniform distribution (*heterogeneous* cost functions).

We first report results obtained with Type-1 cost functions in the evaluations. Later we comment on results with other types of cost functions. The performance of each scheme is reported as the *normalized cost*, which is defined as the ratio of the total cost incurred by the scheme to the total cost of *Bound*. Recall that *Bound* represents the scheme that assigns the periods by fully utilizing the total computational capacity of all M processors by ignoring the partitioning aspect; hence, it represents the upper bound on the performance of any partitioning-based algorithm. By definition, the normalized cost can never be smaller than 1.0. For each data point in the result plots, 25,000 synthetic task sets are generated and the average result of these task sets is reported.

Impact of System Utilization. Figure 2 shows the achieved normalized cost as a function of the utilization under different schemes for systems with different number of CPUs. We can see that the performance levels of both RTSP and RTSP* are better than that of WFD-local. The reason is that, compared to RTSP and RTSP* where tasks are mapped to processors based on their global optimal activation rates, WFD-local maps tasks to CPUs based on their minimum activation rates, which limits the opportunities for tasks to exploit the entire system computing capacity for optimal activation rates. RTSP, while doing better than WFD in general, is outperformed by RTSP*. This comes from the fact that RTSP, in order to schedule the infeasible tasks, allocates them on certain processors based on a heuristic rule, and the activation rates of all tasks on those processors need to be reduced, increasing the overall cost.

As another observation we can see that as the the system load (i.e., the *normalized utilization*) increases, the *normalized cost* for all schemes gradually approaches to the level of *Bound*, even though different schemes converge at different rates. In contrast, with modest load (e.g., less than 1.1), the normalized cost of the schemes (with respect to *Bound*) increases sharply.

This is because, when the normalized utilization exceeds 1.0 only by a small margin, *Bound* is able to guarantee the feasibility by increasing the periods only by very small amounts, yielding a total cost very close to zero. While the *absolute* cost of other schemes is also quite low in that region, the *normalized cost* increases given that *Bound*'s cost is close to zero. As the utilization increases the cost incurred by *Bound* also increases quickly and the normalized cost of the schemes improves.

With more (e.g., 8) processors in the system, there will be fewer number of tasks per CPU, and the performance difference of the schemes becomes much more pronounced. However, the performance of RTSP* is, even at medium loads, is very close to *Bound*; even on 8-processor systems, the difference is less than 20% as soon as the load exceeds 1.07, and becomes less than 5% at the increased load values.

Impact of Elasticity Factor. Next we consider the impact of different elasticity factors, which indicate the flexibility for assigning tasks' activation rates (i.e., periods), on the normalized cost of the system. Figures 3 and 4 show the results for two different system loads $U_{tot}^{\mathcal{N}} = 1.2$ and $U_{tot}^{\mathcal{N}} = 1.4$, respectively.

As the elasticity factor increases, we have more flexibility in terms of assigning larger periods, and the *absolute total cost* for all schemes tends to decrease. We can see that RTSP and RTSP* are, in most cases, able to maintain their normalized cost performance with increased elasticity factor; implying that the cost improvement due to the increased maximum period values is reflected in those schemes *in the same proportion* as in *Bound*. In fact, the *absolute cost* of WFD-Local monotonically decreases with increasing elasticity factor as well; however, the rate of decrase is *smaller* compared to that of *Bound* – that is why we are observing an increase in *normalized cost.* in the plots.

We note that for small elasticity factor (e.g., $EF < 1.5$ for the case of $U_{tot}^{\mathcal{N}} = 1.4$), RTSP can perform slightly worse than WFD-local. The reason is that, for such inflexible task sets, RTSP cannot take much advantage of the global optimality and the WFD-local scheme tends to perform well. But, by virtue of using the globally assigned activation rates and the exact speedup ratio for making partitioning decisions, RTSP* can still outperform WFD-local. These plots again confirm that RTSP* stays within 5% of *Bound* for most of the spectrum. RTSP, performing close to *Bound* most of the time, drifts away for small EF values.

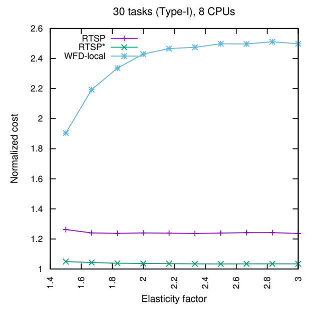


Figure 3: Impact of the elasticity factor; $U_{tot}^{\mathcal{N}} = 1.2$

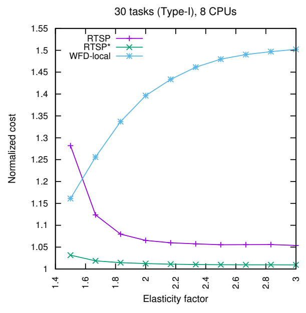


Figure 4: Impact of the elasticity factor; $U_{tot}^{\mathcal{N}} = 1.4$

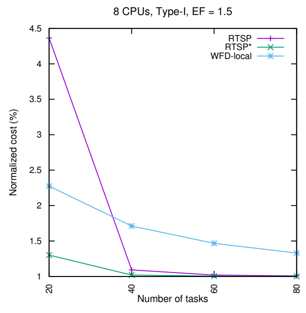


Figure 5: Impact of the number of tasks; $U_{tot}^{\mathcal{N}} = 1.2$

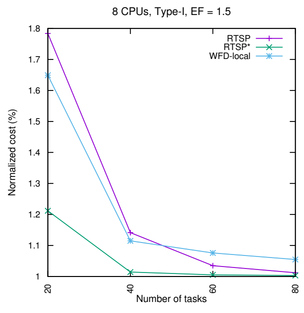


Figure 6: Impact of the number of tasks; $U_{tot}^{\mathcal{N}} = 1.4$

Impact of Number of Tasks. Figures 5 and 6 further show the normalized cost of tasks under different schemes when the number of tasks varies for systems with 8 CPUs. Here, we can see that, for a given system load ($U_{tot}^{\mathcal{N}} = 1.2$ or $U_{tot}^{\mathcal{N}} = 1.4$), the normalized cost for tasks under all schemes decreases as the number of tasks increases. The reason is that, with more tasks, the size (i.e., utilization) of each task gets smaller. With the smaller granularity of task sizes, partitioning tasks to CPUs becomes relatively easier, even with small periods. When there are more than 60 tasks on a system with 8 CPUs, each CPU, on the average, can get more than 7 tasks, which enables RTSP and RTSP* to get very close to *Bound* (within 2%).

On the other hand, when there are fewer number of tasks, both RTSP and RTSP* start to drift away from *Bound*, but they still perform better than WFD-local when the system has more than 40 tasks. However, when the system has only 20 tasks, where each CPU has around 2-3 tasks, RTSP can perform even worse than WFD-local for the case of $U_{tot} = 1.2$ due to the increased granularity of tasks. For the case of slightly higher system load ($U_{tot} = 1.4$), WFD-local can perform as good as RTSP for 30 tasks (Figure 6). Therefore, we can conclude that, the granularity of tasks affects the performance of RTSP the most, followed by WFD-local and RTSP*. However, RTSP* is quite close to *Bound*, except for the case of small number of tasks (20) when the difference is maximum (around 20%).

Impact of Cost Functions. When tasks have different types of cost functions, Figure 7 shows the normalized cost for tasks under different schemes when $EF = 1.5$, with varying system load. For all the types of cost functions considered in the evaluations, RTSP* performs the best and has performance level close to that of *Bound* (except when $U_{tot} \leq 1.1$). Moreover, the different types of exponential cost functions have similar impacts on the performance of the proposed schemes regarding the normalized cost of tasks. In particular, as shown in Figure 2 for Type-1 tasks, RTSP is gradually outperformed by WFD with the increased load, with Type-0, Type-2 and Type-3 cost functions as well.

Comparison with the Optimal Solution. An interesting question is the relative performance of RTSP schemes with respect to the optimal *partitioning-based* algorithm. Obviously, since the problem is in general intractable, the optimal solution cannot be obtained in polynomial time. However, we implemented an exhaustive-search based solution that computes the optimal solution by enumerating all possible task-to-processor assignments, computing the best frequency assignments for each possible partitioning, and then picking up the best solution at the end. Due to the prohibitive computation time, the experiments were performed only for small number of tasks (6-12) running on relatively small number of processors (4). Each data points we show represents the average of 500 runs.

In Figures 8 and 9, we show the impact of the utilization and the impact of the number of tasks, respectively, on RTSP, RTSP* and Optimal scheme, with respect to *Bound*. It can be observed that RTSP* follows very closely the Optimal scheme even for medium normalized utilization and small number of tasks. It is also worth observing that even the performance of Optimal partitioning scheme starts to deviate significantly from that of *Bound* at low load or small number of tasks cases.

6. RELATED WORK

In their seminal work, Seto et al. [22] presented a model where the periods of real-time control tasks could be changed up to a limit, in order to optimize some *control performance index*. They showed that in some industrial control applications, an exponential decay function can be used to model the tasks' performance indices. Seto formulated the problem as a non-linear programming problem and solved it using the *Lagrange multipliers* technique. Bini and Di Natale [5] suggested a generalized framework for maximizing various objective functions in the context of fixed-priority scheduling. They defined the set of all period assignments that can produce feasible schedules as the optimization domain and then used mathematical properties of the objective function to find the optimal assignment. They used Time-Demand Analysis technique to assess the feasibility in a single processor environment. Our work is different in terms of the problem settings: we use dynamic priority scheduler (EDF) and we target multiprocessor platforms.

Buttazzo et al. [7] proposed a scheduling framework (called *elastic scheduling*) in which periodic tasks can change their rates in order to tackle the overload conditions. For each task, their model has a minimum period, a maximum period and an elasticity coefficient. When the system is overloaded, the tasks can increase their periods (compressing); and when the system is underloaded, the task periods can be reduced (decompressing). The authors developed an efficient algorithm where they modeled each task as a spring with minimum and maximum length and a rigidity coefficient. This model proved useful for supporting both multimedia and control applications in which execution rates of some compu-

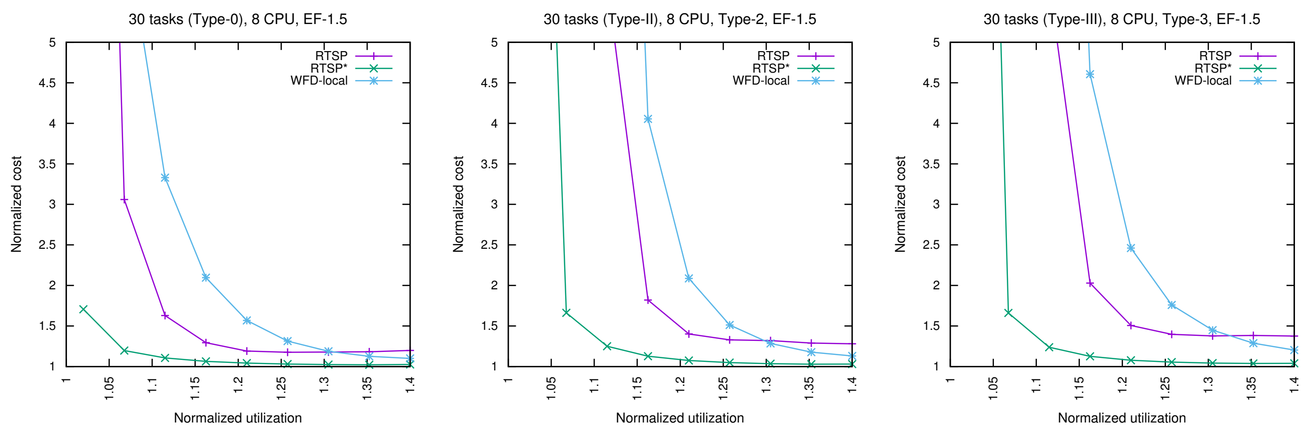


Figure 7: Impact of utilization on different types of task sets (30 tasks) on 8 CPUs

tational activities have to be dynamically tuned as a function of the current system state. However, their work does not consider arbitrary control cost functions, and is developed for single-processor systems. Marinoni et al. [19] incorporated DVS technique to the elastic task model.

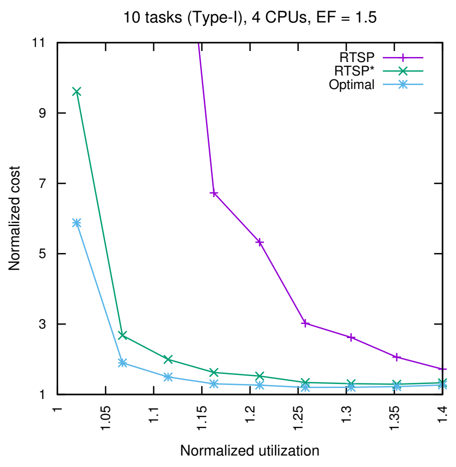


Figure 8: Impact of Utilization - Comparison to Exhaustive-Search Based Optimal Solution

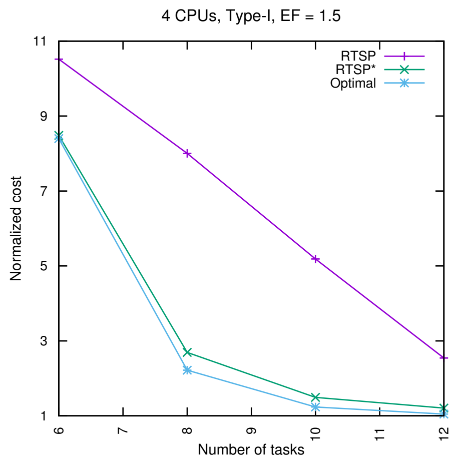


Figure 9: Impact of number of tasks - Comparison to Exhaustive-Search Based Optimal Solution; $U_{tot}^{\mathcal{N}} = 1.2$

Chantem et al. [10] considered a setting where task periods and deadlines are chosen in order to improve the schedulability. Later, the same research group explored a setting similar to ours where minimum and maximum allowable periods are associated with each task, and a performance index is optimized [9]. Their performance index is a quadratic function of utilization of each task. They focused on uni-processor system settings. Wu et al. also used a quadratic cost function as their performance loss index which depends on the period of the tasks [24]. Fontanelli et al. [12] explored similar problem settings associated with real-time control tasks and cost functions, but their results were obtained under stochastic model. Tian et al. [23] leveraged the elasticity of tasks to improve the quality of control in applications using a utilization-based quadratic cost function. Kim et al. [14] proposed a task model for automotive systems called *rhythmic tasks* where period and worst-case execution time of each task can vary continuously based on some external event. They did not use any performance index, rather, their main goal was to improve schedulability. In a recent paper, Buttazzo et al. [6] extended the idea and proposed rate-adaptive tasks where period, worst-case execution time and deadline can vary according to the angular velocity of the crank-shaft. However, their main objective was to improve schedulability, not optimizing any performance index. Rajkumar et al. [21] proposed a model called Q-RAM where each real-time application are assigned resources to maximize overall system utility along multiple and discrete QoS dimensions, for single-processor systems. After resources are allocated, each application can choose its own period, WCET and deadline.

Several heuristics have been proposed and analyzed for partitioning real-time tasks efficiently in a near optimal way [2,8,18]. These heuristics order tasks according to some certain criteria (e.g., decreasing utilization) and typically allocate tasks by using one of the well-known rules such as First-Fit, Best-Fit or Worst-Fit [18]. Lopez et al. [18] derived utilization bounds for partitioned scheduling on homogeneous multiprocessor systems when each processor uses EDF and a

Reasonable Allocation Decreasing (RAD) algorithm is used to distribute the tasks. RAD algorithms includes FFD, BFD and WFD, which happen to share the same worst-case utilization bound. Carpenter et al. [8] classified the scheduling approaches on multiprocessor systems based on the complexity of the priority assignment of the tasks, and the degree of migration allowed. More recently, Baruah [2] used the metric called *speedup factor* to compare the performances of various allocation schemes for partitioned scheduling on multiprocessors. Pointing out the ineffectiveness of the utilization bound for partitioned scheduling, Baruah suggested that the speedup factor provides a deeper insight into the behavior of partitioning techniques and derived the speedup bounds for various RAD algorithms.

7. CONCLUSIONS

Period assignment to real-time control tasks can make important difference in terms of control performance. While several studies considered the problem for uniprocessor settings and scheduling models, in this work, we considered multiprocessor platforms that are increasingly used in view of the performance requirements. We assumed partitioned EDF scheduling and arbitrary convex control cost functions.

Finding optimal solution to this problem on a multiprocessor platform is, in general, computationally intractable. We identified *local* period assignment algorithms as the ones that first partition the tasks and then optimize the periods on each processor locally. To address the limitations of the local approaches, we proposed the Reduction-to-Single-Processor (RTSP) technique which first assigns tentative optimal periods by considering all the tasks at once on a hypothetical faster single processor, and then attempts partitioning with those period values. We developed two variants of the algorithm that differ on the way they determine the speedup factor for the hypothetical faster single processor. Our experimental results indicate that the RTSP technique generally outperforms the local algorithms, and follows the theoretical upper bound on the control performance closely.

Acknowledgments

This work was supported by the US National Science Foundation Awards CNS-1422709 and CNS-1421855.

8. REFERENCES

[1] Aydin, H., Melhem, R., Mosse, D., and Mejía-Alvarez, P. Optimal reward-based scheduling for periodic real-time tasks. *Computers, IEEE Transactions on 50*, 2 (2001), 111–130.

[2] Baruah, S. Partitioned edf scheduling: a closer look. *Real-Time Systems 49*, 6 (2013), 715–729.

[3] Baruah, S. K., Cohen, N. K., Plaxton, C. G., and Varvel, D. A. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica 15*, 6 (1996), 600–625.

[4] Bini, E., and Cervin, A. Delay-aware period assignment in control systems. In *Real-Time Systems Symposium* (2008), IEEE.

[5] Bini, E., and Natale, M. D. Optimal task rate selection in fixed priority systems. In *Real-Time Systems Symposium. 26th International* (2005), IEEE.

[6] Buttazzo, G. C., Bini, E., and Buttle, D. Rate-adaptive tasks: Model, analysis, and design issues. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)* (2014), IEEE.

[7] Buttazzo, G. C., Lipari, G., Caccamo, M., and Abeni, L. Elastic scheduling for flexible workload management. *Computers, IEEE Transactions on 51*, 3 (2002), 289–302.

[8] Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., and Baruah, S. A categorization of real-time multiprocessor scheduling problems and algorithms. *Handbook on Scheduling Algorithms, Methods, and Models and Performance Analysis* (2004).

[9] Chantem, T., Hu, X. S., and Lemmon, M. D. Generalized elastic scheduling for real-time tasks. *Computers, IEEE Transactions on 58*, 4 (2009), 480–495.

[10] Chantem, T., Wang, X., Lemmon, M. D., and Hu, X. S. Period and deadline selection for schedulability in real-time systems. In *Euromicro Conference on Real-Time Systems* (2008), IEEE.

[11] Emberson, P., Stafford, R., and Davis, R. I. Techniques for the synthesis of multiprocessor tasksets. In *Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)* (2010).

[12] Fontantelli, D., Palopoli, L., and Greco, L. Optimal cpu allocation to a set of control tasks with soft real–time execution constraints. In *Proceedings of the 16th International Conference on Hybrid systems: Computation and Control* (2013), ACM.

[13] Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[14] Kim, J., Lakshmanan, K., and Rajkumar, R. R. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In *Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems* (2012), IEEE.

[15] Lincoln, B., and Cervin, A. Jitterbug: A tool for analysis of real-time control performance. In *Proceedings of the 41st Conference on Decision and Control* (2002), IEEE.

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

[17] Liu, J. W. S. *Real-Time Systems.* Pearson Education, 2000.

[18] López, J. M., Díaz, J. L., and García, D. F. Utilization bounds for edf scheduling on real-time multiprocessor systems. *Real-Time Systems 28*, 1 (2004), 39–68.

[19] Marinoni, M., and Buttazzo, G. Elastic dvs management in processors with discrete voltage/frequency modes. *Industrial Informatics, IEEE Transactions on 3*, 1 (2007), 51–62.

[20] Melzer, S. M., and Kuo, B. C. Sampling period sensitivity of the optimal sampled data linear regulator. *Automatica 7*, 3 (1971), 367–370.

[21] Rajkumar, R., Lee, C., Lehoczky, J., and Siewiorek, D. A resource allocation model for qos management. In *Real-Time Systems Symposium* (1997), IEEE.

[22] Seto, D., Lehoczky, J. P., Sha, L., and Shin, K. G. On task schedulability in real-time control systems. In *Real-Time Systems Symposium* (1996), IEEE, pp. 13–21.

[23] Tian, Y.-C., and Gui, L. Qoc elastic scheduling for real-time control systems. *Real-Time Systems 47*, 6 (2011), 534–561.

[24] Wu, Y., Buttazzo, G., Bini, E., and Cervin, A. Parameter selection for real-time controllers in resource-constrained systems. *Industrial Informatics, IEEE Transactions on 6*, 4 (2010), 610–620.