Task Handler Based on (m,k)-firm Constraint Model for Managing a Set of Real-Time Controllers

Ning Jia, Ye-Qiong Song, Françoise Simonot-Lion

Task Handler Based on (*m,k*)-firm Constraint Model for Managing a Set of Real-Time Controllers

Jia Ning, Song YeQiong, Simonot-Lion Françoise
LORIA – Nancy Université
Campus Scientifique – BP 239
54506 – Vandoeuvre lès Nancy France
{Ning.Jia,song,Francoise.Simonot}@loria.fr

Abstract

In this paper, we study how to schedule a set of real-time tasks where each task implements a control law. These tasks share a limited computing resource. The set of tasks can switch on line from one given configuration to another one depending on the working modes of the global application. This means that some tasks may appear while other ones may be no longer activated and that the WCET of some tasks may be modified. We propose a scheduling architecture for the handling of such task instances. At each mode switching, the task handler consider the new task parameters; then it determines on line a (m,k)-constraint based scheduling strategy to apply to each task; this strategy aims to selectively discard task instances so that the schedulability of tasks is guaranteed and the overall control performance is maintained at a high level.

1. Introduction

Let us consider an application composed of *n* physical sub-systems. Each sub-system is controlled by one dedicated controller that is implemented as a real-time task responsible for carrying out the control law computation for this sub-system. Therefore, a centralized implementation of all the controllers raises the problem of the schedulability of these *n* tasks. The tasks are generally considered as hard real-time tasks characterized by a fixed activation period (defined by the sampling period of the sub-system outputs) and a known worst-case execution times. Each instance of each task has to respect a deadline by which it is expected to complete its computation. However, due to the timing non-determinism of control application (non constant execution time, activation of new tasks, etc), using only worst-case execution time generally results in over sizing the necessary computing resource, and the overall control performance may not be optimal in the sense that they do not make a full use of the computing resource. Furthermore, many control systems are quite robust against variations in execution parameters such as a number of sample losses or a given variation in sampling period. Therefore, a scheduling approach that uses the current system configuration information to correctly adjust control task parameters in order to achieve higher resource utilization and better control performance is desirable.

The key to successful design of such a scheduling approach is the co-design of the controller and the resource management process which has attracted considerable attentions. In [15], an algorithm was proposed that selects the sampling periods for a set of control tasks so that the overall control performance was optimized under the schedulability constraints. The authors suppose that the cost function of each sub-system can be approximated by an exponential function. Due to the high computational cost of the algorithm, the proposed approach can only be used off-line. An exact off-line approach for sampling period selection was developed in [4] by supposing that the cost function is convex. To make on-line use of the proposed approach, the cost function is then supposed to be possibly approximated by a quadratic function so that the high computational costs can be reduced. In [3], a feedback-feedforward scheduling approach was proposed to improve the reaction speed of the feedback scheduling approach developed in [4] to a change in the computing resource load. Again, a linear function is proposed to approximate the cost function.

For all these above mentioned approaches, the cost functions of sub-systems are supposed convex (their theories also hold for the case where all the cost functions are concave) or can be approximated by a convex function. However, this assumption is restrict. In practice, one can not guarantee that the cost functions of sub-systems in the application are all convex, and since the cost functions of some control systems are not convex nor concave, they can not be well approximated by a convex function. Although it was shown in [1] that for some control systems, the cost function is quadratic for small sampling intervals, but the restriction on the choice of sampling period may lead to the non-schedulability of some tasks and therefore to the degradation of the overall control performance. Furthermore, these approaches maintain the control performance optimality and control task schedulability by the regulation of the sampling periods of sub-systems. However, changing the period of a task may necessitate a change in the periods of related tasks as task periods are often carefully selected for an efficient exchange of information between relative tasks; in addition, the change in sampling period alters dynamics of the sub-system and leads to an unavoidable additional study for the approaches based on the regulation of the sampling periods.

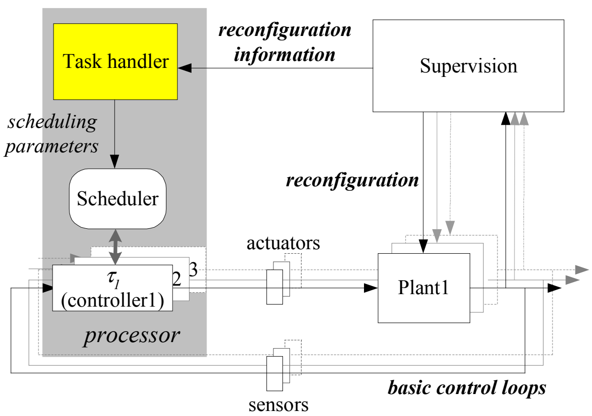

Figure 1. Overall system architecture

In this work, we consider a system architecture as shown in Figure 1. We suppose that a supervision function of all the controlled plants is implemented in a separate computer. The purpose of this function is to detect when the plants have to be controlled in a different way, meaning the change of control algorithm; it detects also when a plant is no more needed to be controlled or if a plant becomes operational and requires therefore the activation of a controller. Thanks to this supervision, the task handler can be notified when such a system configuration change leads to a new task set. Several controllers are implanted as real-time tasks in one processor, and each of them controls a physical plant. The number of control tasks and their execution times may change over time, for example when the overall system enters in a new working mode. At a system configuration change, the task handler is activated and it receives the information about these two execution parameters of control tasks from supervision component. Based on these information, it determines a set of scheduling parameters and transfer them to the scheduler. The scheduler then selectively discards the instances of control tasks according to these execution parameters so that the schedulability of control tasks is guaranteed and the overall performance of control application is maintained at a high level.

Specifically, the scheduling parameters transferred to scheduler are (m,k)-firm constraints [13][14], which indicate that the deadlines of at least m among any k consecutive instances of a control task must be met, where m and k are two positive integers with $m \le k$ (the case where m=k is equivalent to the ideal case, which is noted by (k,k)-firm). Since the discarded instances will not be executing the control law, this tends to degrade the control performance. However, if a control system is designed to accept a control performance degradation until k-m deadlines misses among k consecutive task instances (this can be justified by the observation that most control systems can tolerate misses of the control law updates to a certain extent), the system can then be conceived according to the (m,k)-firm approach to offer the varied levels of control performance between (k,k)-firm (ideal case) and (m,k)-firm (worst case) with as many intermediate levels as the possible values between k and m. This results in a control system with graceful degradation of control performance.

The control performance can be described by different performance criterion that could be cost function, state error, maximum overshoot, settling time, etc. The proposed scheduling solution does not make any assumption on the type and property of control performance function. That is, it will hold whatever the control performance functions of sub-systems are all convex or can be approximated by a convex function. Furthermore, since the original sampling periods of control tasks will not be changed at a system configuration change, no period adaptation of the related tasks will be needed and the dynamics of sub-system will not be altered.

Notice that in [17][18], a scheduling algorithm is presented, which uses feedback information about the current workload in processor to regulate the deadline miss-ratio for a set of tasks disregarding the specific purpose of these tasks. Our approach aims to control the scheduling of a set of tasks while taking into account the fact that these tasks execute specific control laws. For short in [17][18], the QoS is an objective while in our proposal both QoS and QoC (quality of control) are targeted. Therefore our strategy is also based on the performance of the plant control. The scheduling objective is to explicitly maintain the overall control performance at a high level rather than maintain the deadline miss-ration at certain level.

The paper is organized as follows. A survey of related work is contained in section 2. Section 3 gives the task model of the control application and the schedulability analysis of control tasks under (m,k)-firm constraint. A formal description of the problem is presented in section 4. Section 5 gives the heuristic algorithm for computing the sub-optimal (m,k)-firm constraint for each control task. A numerical example of the proposed scheduling approach is presented in section 6. Finally, we summarize our work and show the perspectives.

2. State of the art: (*m,k*)-firm model and its use for control application

The previous work falls into two categories. The first one is the field of real-time scheduling based on (*m,k*)-firm constraint model.

In [14], a scheduling approach is presented for the general (*m,k*)-firm constraint model. A simple algorithm is used to partition the instances of each task in the system into two sets: mandatory and optional. All mandatory instances are scheduled according to their fixed priorities, while all optional instances are assigned the low priority. It follows that if all mandatory instances meet their deadlines, the (*m,k*)-firm constraint is satisfied. A sufficient and necessary condition for determining the schedulability of the mandatory instances is also given. In [13], it's proved that in general case, the problem of determining the schedulability of a set of tasks under (*m,k*)-firm constraint is NP-hard. In [6], we show that if the task instances are partitioned using the approach in [14], the distribution in the instance sequence of the mandatory instances corresponds to a mechanical word [11]. A series of mathematical tools for the schedulability analysis are also given using the theories of mechanical word.

The second area of previous work is the analysis of impact of (*m,k*)-firm constraint model on the control performance of a single control system. In [7], we presented a formal analysis that derived, for a one-dimensional control system, the *m* and *k* values that guarantees the control system stability. We also proposed an approach for deriving the controller minimizing the variance of the process state in order to minimize the deterioration in the control system behavior due to the control low update misses. This work is extended in [5] and in [8] to a multiple dimension control system. We showed, in [5] how to determine the maximum value of *k* for a control system so that one can get as many varied levels of control performance as possible subject to the stability of control system. In [8], we gave a general method to derive the optimal LQ-controller under (*m,k*)-firm constraint, and based on the works in [6], we showed that for a single control task under a given (*m,k*)-firm constraint, the control performance of the corresponding control system is sub-optimal if the task instances are partitioned and scheduled using the scheduling approach in [14].

3. Task Model and Schedulability

In this section, we first give the task model of the real-time control application under study. Then, a sufficient condition for determining of schedulability of tasks set is given.

3.1 Task Model

Let an application be composed of *n* periodic control tasks, τ_1 τ_2 .. τ_n, arranged in decreased order of their priorities. The following timing parameters are defined for each task τ_i:

- T_i: the time interval between two consecutive instances of τ_i, referred to as its period;
- C_i: the maximum time needed for completing the execution of each instance of τ_i, referred to as its execution time;
- D_i: the deadline of each instance of τ_I; we consider that the deadline is equal to the period of the task;
- m_i and k_i: the (*m,k*)-firm constraint for τ_i with $m_i \le k_i$; it means that at least the deadlines of m_i out of k_i consecutive instances of the task must be met.

Furthermore, we assume that the algorithms of the control laws are independent in the sense that they do not share any resources except the processor; therefore the control tasks are assumed to be preemptive.

3.2 Schedulability under (*m,k*)-firm Constraint

The problem of determining the schedulability of a set of tasks under (*m,k*)-firm constraints has been proved in [13] to be NP-hard; however, when the tasks are scheduled using the algorithm proposed in [14], the schedulability of tasks can be explicitly judged. Furthermore, as stated formerly, if the instances of a control task under a given (*m,k*)-firm constraint are partitioned and scheduled by the scheduling approach in [14], the control performance of the corresponding control system is sub-optimal. Therefore we adopt in this work the scheduling algorithm proposed in [14] for the scheduling of the set of control tasks.

Concretely, the instances of each task are partitioned into two sets: the mandatory instances and the optional instances. The problem is to determine for given m and k which instances in a sequence of instances are mandatory. We propose in [6] to fit the distribution of mandatory instances using theory of mechanical word. Under this approach, an instance of τ_i, activated at time aT_i, for a = 0,1,… is classified as mandatory if the following condition is verified

$$a = \left\lfloor \left\lceil \frac{am_i}{k_i} \right\rceil \frac{k_i}{m_i} \right\rfloor \tag{1}$$

and as optional, otherwise. For example, if τ_i is under (3,5)-firm constraint, the condition (1) is verified for a=0+αk, 1+αk and 3+αk ($\alpha \in N$) and not verified for a=2+αk and 4+αk. Therefore the instances activated at 0,1,3,5, etc are mandatory while those activated 2,4,7, etc are optional. Notice that using the classification equation (1), there are exactly *m* mandatory instances and *k-m* optional instances among any *k* consecutive instances, and the mandatory instances are uniformly distributed in the instance sequence. The reader interested in could refer to [6] for a more detail information about the task instance classification theory.

The control tasks are scheduled using the fixed priority policy. The mandatory instances of all the tasks are assigned the rate-monotonic priorities [10]. That is, the mandatory instances of τ_i are assigned a higher priority than the mandatory instances of τ_j if $T_i<T_j$. The optional instances are assigned the lowest priority.

A sufficient and necessary schedulability condition for the above scheduling strategy is given in [14]. However, the condition contains a timing non-deterministic term, this prevents it from a application in our optimization routine presented below. We therefore give a sufficient schedulability condition:

Theorem 1. *Given a task set* ($\tau_1, \tau_2 \ldots \tau_n$) *such that* $T_1<T_2<..<T_n$. *Let:*

$$n_{ij} = \left\lceil \frac{m_j}{k_j} \left\lceil \frac{T_i}{T_j} \right\rceil \right\rceil \quad (2)$$

If $C_i + \sum_{j=1}^{i-1} n_{ij} C_j \le T_i$ *for all* $1 \le i \le n$, *then the* (m_i, k_i)*-firm constraint of each task* τ_i *is satisfied.*

Proof. In [6], we proved that the task instance partition algorithm results in the most mandatory instances from [0, t] compared with those in any other interval of the same length t. Therefore, to analyze the schedulability of a set of tasks, it's enough to study if the first instance of each task respects its deadline. From [6], we know that (2) gives the number of the mandatory instances of task τ_j before instant T_i, therefore, if $\left(C_i + \sum_{j=1}^{i-1} n_j C_j \right) / T_i \le 1$, then the first instance of τ_i will complete prior to its deadline. The theorem follows if the results holds for all i.

□

Note that the above schedulability condition is sufficient and necessary if the period of a task is multiple of the periods of all the lower priority tasks. In the other case, it degenerates to a sufficient condition.

4. Problem Formulation and scheduling architecture

Recall that we consider a real time control application composed of several control tasks that share the same processor. It's assumed that the control tasks in the application can switch between different modes. Going from one mode to another mode can lead to consider another task set configuration: the execution time can be different for some tasks, control tasks can be shutdown and new control tasks can be activated. We aim to implement a task handler that, at each change in the task set configuration, guarantees the schedulability of control tasks and keeps the overall control performance at a high level.

Based on the results in [5], we suppose that the value of k_i of each τ_i has been carefully chosen and is constant for each mode. The value of m_i is dynamically chosen in [1 .. k_i] on line. Specifically, for each control task τ_i, each possible m_i is associated with a level of control performance v_{ij}. We assume that a larger v_{ij} corresponds to a better control performance. As we do not make any assumption on the control performance property, the control performance is not necessarily improved when the value of m_i is increased [5]. Furthermore, theorem 1 shows that the increase of the value of m_i may increase the resource requirement. From task scheduling point of view, there is no reason to select a larger value for m_i with lower control performance. Therefore, only the values of m_i which give a better control performance are considered.

So, let us consider that m_i can take l_i different values, noted as m_{ij} for $j\in$ [1.. l_i] where $l_i \le k_i$. These values are arranged in increasing order, that is if $j' < j''$, then $m_{ij'} < m_{ij''}$ and therefore $v_{ij'} < v_{ij''}$. The aim of the task handler is to find, for each τ_i, a value m_{ij} for $j\in$ [1.. l_i] so that the sum of v_{ij} for $i\in$ [1.. n] is maximized, and the schedulability of control tasks is guaranteed. This is formulated as the following optimization problem:

to determine the sequence $x_{i1}, x_{i2}, .., x_{il_i}$ for each task τ_i, $i=1,..,n$

that maximizes $$\sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} v_{ij} \quad (3)$$

with $x_{ij} \in \{0,1\}$, $\sum_{j=1}^{l_i} x_{ij} = 1$, $i = 1,..,n$, $j = 1,..,l_i$

and such that

$$C_i + \sum_{j=1}^{i-1} \left\lceil \frac{\sum_{p=1}^{l_i} x_{jp} m_{jp}}{k_j} \left\lceil \frac{T_i}{T_j} \right\rceil \right\rceil C_j \le T_i,\ i = 1,..,n, \quad (4)$$

If the control performance is represented by the control cost (LQ cost for example), then a smaller v_{ij} gives a better control performance. The optimization problem thus becomes a minimization problem. However, it can be easily transformed as a maximization problem by take the additive inverse of v_{ij} for each τ_i.

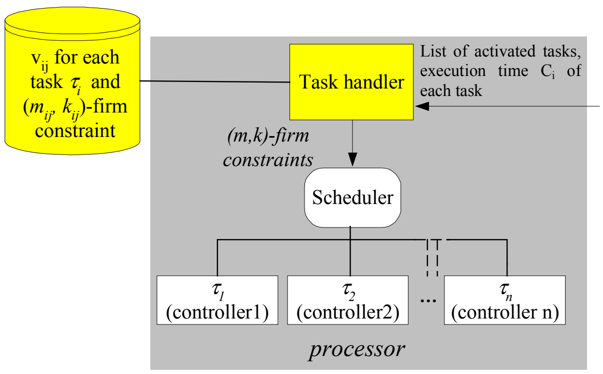

Figure 2. scheduling architecture

The scheduling architecture is given in Figure 2, which is a close-up of figure 1. At a task set configuration change, the task handler receives the information about the number of tasks sharing the processor and the actual execution time C_i of each task, and deduces the new (m_i,k_i)-firm constraint for each control task by resolving the optimization problem (3). The scheduler then schedules the tasks according to these (m,k)-firm constraints.

The control performance level v_{ij} corresponding to a (m_{ij},k_{ij})-firm constraint for task τ_i can be determined on-line or off-line. However, this determination is very time-consuming for some control performance criterion (e.g. the control performance criterion used in [8]). For this reason, The control performance level v_{ij} are calculated off-line and arranged in a table that is consulted by the task handler when solving the problem (3).

5. Performance Optimization

In this section, we first show that the optimization problem (3) is a NP-hard problem, then, based on the algorithm proposed in [9], a computationally cheaper heuristic algorithm is proposed for finding a sub-optimal solution.

5.1 Solution of the Optimization Problem

The optimization problem enounced in (3) is qualified as the multiple-choice multi-dimension knapsack problem (MMKP) which is defined as following:

Suppose there are n groups (stacks) of items. Group i has l_i items. Item j of group i has value v_{ij}, and requires resources given by vector $r_{ij}=(r_{ij1},r_{ij2},..,r_{ijm})$. The amounts of available resources are given by $R=(R_1,R_2,..,R_m)$. The MMKP is to pick exactly one item from each group in order to maximize the total value of the pick, subject to the resource constraints. Formally, the MMKP is expressed as follows:

$$\text{maximize} \sum_{i=1}^{n}\sum_{j=1}^{l_i} x_{ij}v_{ij}$$

such that

$$\sum_{i=1}^{n}\sum_{j=1}^{l_i} x_{ij}r_{ij} \le R_k, \quad k=1,..,m,$$

$$x_{ij}\in\{0,1\}, \sum_{j=1}^{l_i} x_{ij}=1,\ i=1,..,n,\ j=1,..,l_i$$

MMKP is one of the harder variants of the 0-1 knapsack problem [12][16]. The problem is proved to be NP-hard problem. Hence algorithms for finding the exact solution of MMDP are not suitable for application in real time decision-making application.

A computationally cheaper heuristic algorithm (HEU) for MMKP is presented in [9]. For our optimization problem, the proposed algorithm is to find a feasible solution at first, that is, select m_{ij} for each τ_i while satisfying the constraints enounced in (4), and then iteratively improve the solution by replacing, for each τ_i, m_{ij} by a $m_{ij'}$ corresponding to a better performance $v_{ij'}$ while keeping the constraints (4) satisfied. If no such solution can be found, the algorithm tries an iterative improvement of the solution which first replaces m_{ij} for a task τ_i which is not schedulable with the value m_{ij} (constraint (4) violated), and then replace $m_{i'j}$ of $\tau_{i'}$ for all $i\neq i'$ with $m_{i'j'}$ corresponding to a worse performance $v_{i'j'}$. The iteration finishes when no feasible solution can be find. It's shown in [9] that the algorithm HEU is efficient (the solution given by the algorithm is within 6% of the optimal value) and suitable for an on-line use for real-time application (the computation time is less than one millisecond if the number of tasks does not exceed 30 on a 700 MHz Pentium III processor).

As in our problem setup, the values m_{ij} and v_{ij} for $j\in$ [1.. l_i] are arranged in increasing order for each τ_i, therefore in contrast with HEU in which an infeasible solution may be selected at first and iterations are needed to make it feasible, we modify HEU by always picking the lowest value of m_i of each τ_i at first (if the solution is infeasible in this case, no other solution will be feasible). Furthermore, the algorithm HEU tries to find, after the first step, a better solution requiring less resource consummation which, however, does not exist in our model (augmentation of m increases the processor utilization), therefore this property also help us to delete a unprofitable researching procedure in HEU. These modifications can reduce the execution time of the algorithm. The modified algorithm is presented in Algorithm 1.

In this algorithm, replacing m_{ij} by $m_{ij'}$ is called an *exchange*. An exchange giving a better overall control performance is called *upgrade* while an exchange degrading the overall control performance is called *downgrade*. An exchange is called *feasible* if the solution after the exchange is feasible, otherwise it is called *infeasible*.

Algorithm 1. Algorithm for finding the value of the parameter m of (m,k)-firm constraint for all tasks

// Symbols and formalization:
// n : number of the tasks
// m_i and k_i: the (m_i,k_i)-firm constraint of task τ_i
// l_i : number of the possible values for m_i
// m_{ij}: the j^{th} possible value of m_i
//T_i: period of task τ_i; C_i : execution time of task τ_i
// $\rho=(\rho_1,\rho_2,..,\rho_n)$ denotes the current solution, where τ_i gives the index of the value of m_i in $[1..l_i]$
// $\rho|Z$: solution vector after exchange Z from ρ
// L: current resource requirement vector

// $U(\rho)$: total control performance with the solution vector ρ, with $U(\rho)=\sum_{i=1}^{n} v_{i\rho_i}$

// X = (i,j) denotes an exchange where the m_{ij} is selected instead of $m_{i\rho_i}$

1. Start with a feasible solution.

Set $\rho_i = 1$ for all $i \in [1,..,n]$.

Compute

$$L=\begin{pmatrix} C_1 \\ C_2+\sum_{j=1}^{1}\left\lceil \frac{m_{j1}}{k_j}\left\lceil \frac{T_2}{T_j}\right\rceil\right\rceil C_j \\ \vdots \\ C_n+\sum_{j=1}^{n-1}\left\lceil \frac{m_{j1}}{k_j}\left\lceil \frac{T_n}{T_j}\right\rceil\right\rceil C_j \end{pmatrix}$$

Find α such that $L_\alpha / T_\alpha = \max_{i=1,2,..n} \frac{L_i}{T_i}$.

if $L_\alpha / T_\alpha \le 1$, go to step 2, else exit the procedure with "no feasible solution".

2. Iterative improvement using feasible upgrades.

Define

$$\Delta\alpha(\rho,i,j)=\frac{\sum_{l=i+1}^{n}\left(\left\lceil \frac{m_{i\rho_i}}{k_i}\left\lceil \frac{T_l}{T_i}\right\rceil\right\rceil C_i-\left\lceil \frac{m_{ij}}{k_i}\left\lceil \frac{T_l}{T_i}\right\rceil\right\rceil C_i\right)L_l}{|L|}$$

and

$$\Delta p(\rho,i,j)=\frac{v_{i\rho_i}-v_{ij}}{\Delta\alpha(\rho,i,j)}$$

Find feasible upgrade $X^{'}=(\delta,\eta)$ that maximizes $\Delta p(\rho,i,j)$.

If X' is found and $\Delta p(\rho,i,j)>0$, set r = $(r|X')$ and repeat step 2.

3. Iterative improvement using upgrades followed by one or more downgrade(s).

3.1

Define

$$\Delta t(\rho,i,j)=\sum_{l=i+1}^{n}\frac{\left\lceil \frac{m_{i\rho_i}}{k_i}\left\lceil \frac{T_l}{T_i}\right\rceil\right\rceil C_i-\left\lceil \frac{m_{ij}}{k_i}\left\lceil \frac{T_l}{T_i}\right\rceil\right\rceil C_i}{T_l-L_l}$$

and

$$\Delta p'(\rho,i,j)=\frac{v_{i\rho_i}-v_{ij}}{\Delta t(\rho,i,j)}$$

Find an upgrade $Y=(\delta',\eta')$ maximizing $\Delta p'(\rho,i,j)$, set $\rho'=(\rho|Y)$

3.2

Define

$$\Delta t'(\rho,i,j)=\sum_{l=i+1}^{n}\frac{\left\lceil \frac{m_{i\rho_i}}{k_i}\left\lceil \frac{T_l}{T_i}\right\rceil\right\rceil C_i-\left\lceil \frac{m_{ij}}{k_i}\left\lceil \frac{T_l}{T_i}\right\rceil\right\rceil C_i}{L_l}$$

and

$$\Delta p''(\rho,i,j)=\frac{v_{i\rho_i}-v_{ij}}{\Delta t'(\rho,i,j)}$$

Find a downgrade $Y^{'}=(\delta'',\eta'')$ maximizing $\Delta p''(\rho',i,j)$ such that $U(\rho'|Y')>U(\rho)$.

If Y' is found and $(\rho'|Y')$ is feasible, set $\rho=(\rho'|Y')$ and go to step 2.

If Y' is found and $(\rho'|Y')$ is not feasible, set $\rho'=(\rho'|Y')$ and go to step 3.2.

5.2 Complexity analysis of Algorithm 1

As stated [9], we suppose $l_1=\ldots=l_n$ for simplifying of the complexity analysis. In the first step, the solution is set directly to be $\rho=(1,1,..,1)$ which allows to reduce the computational complexity of the step 1 to $O(n^2)$ instead of $O(n^2(l-1)^2(n-1))$ in HEU. In the other steps, the modification that we proposed doesn't change their worst-case computational complexities in the original algorithm that are $O(n^2(l-1)^2(n-1))$. As it is mentioned in [9], the combined complexity of steps 1 and 2 gives the computational behavior of the algorithm, the worst-case computational complexity of the algorithm 1 is thus $O(n^2(l-1)^2(n-1))$.

6. Numerical Example

In this section, we present a numerical example to illustrate the proposed scheduling approach. We consider a cart-control system whose objective is to control the position of a cart along a rail according to a position reference. We study the problem of simultaneously controlling four cart systems, $Cart_1$, $Cart_2$, $Cart_3$, $Cart_4$. To illustrate the benefit of our approach, both the traditional scheduling approach and ours are evaluated.

6.1 Plants and Controllers

The continuous model of the cart system is given by:

$$dx=\begin{bmatrix} 0 & 1 \\ 0 & \frac{-11.4662}{M} \end{bmatrix} xdt+\begin{bmatrix} 0 \\ \frac{1.7434}{M} \end{bmatrix} udt+dv_c$$

where M is the mass of the cart; v_c is the process noise with incremental covariance $R_{1c}=\begin{bmatrix} 3.24 & -1.8 \\ -1.8 & 1 \end{bmatrix}$. The four cart systems, $Cart_1$, $Cart_2$, $Cart_3$, $Cart_4$ have different masses: M_1 = 1.5, M_2 = 1.2, M_3 = 0.9, M_4 = 0.6, given in kg. In the following, the controller of $Cart_i$ is denoted $Controller_i$. Its sampling periods (second) is h_i (resp. *0.007, 0.0085, 0.01, 0.0115*).

The control performance is measured using a quadratic cost criterion:

$$J = \lim_{N \to \infty} \frac{1}{N} E\left(\int_0^N x^T(t)Qx(t) + u^T(t)Ru(t)dt \right) \qquad (5)$$

where

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \text{ and } R\text{=0.00006}$$

The value of k for each cart system is chosen using the approach proposed in [5] to guarantee the system stability. In [5], we identified that the maximum value of k calculated for $Cart_1$ and $Cart_2$ (masses 1.5kg and 1.2kg) is greater than 10; nevertheless, in this paper, we consider, for an easier demonstration, values of k that are less than 10 and for which, the systems remain stable. Furthermore, note that since the tasks are assigned the rate-monotonic priority, the task with the largest period has the lowest priority, its execution has no influence on the other tasks. Therefore the task will be executed without task instance drop, or in other words, it is executed under (k,k)-firm constraint. So the value of k for $Cart_4$ has no use. Finally, the parameters k for the other $Cart_1$, $Cart_2$ and $Cart_3$ are: k = [5,8,10]. For each cart system, the value of m may vary within [1..k].

The parameters of LQ-controller that minimizes the quadratic cost criterion (5) and the minimum cost obtained for each possible (m,k)-firm constraint are calculated using the approach presented in [8]. When the (m,k)-firm constraint is changed for a task during running time, the controller parameters are replaced by the ones corresponding to the new (m,k)-firm constraint. To allow for fast changes between different (m,k)-firm constraints, the parameters are calculated off-line for each (m,k)-firm constraint and stored in a table.

6.2 Experiment Setup

The simulation model is created using MATLAB/Simulink and the TrueTime toolbox [2].

The execution time of each control task is approximately 3 ms, and that of the task handler varies with the amount of tasks and the value of k for each tasks. For this example, the execution time of task handler is fixed as 2.5 ms. The task handler is assigned the highest priority, and the priorities of control tasks are assigned according to the rate-monotonic period assignment policy.

During the simulation, the release time of a task is always set to be the current release time plus the task period. Therefore, for a task, if a instance misses its deadlines, the release time of the following instance will have a release time back in time.

The experiment is done for both traditional scheduling approach which schedules the task instances without taking into account the processor overload and ours. At $t = 0$, $Controller_1$ and $Controller_2$ are on, while $Controller_3$ and $Controller_4$ are off. At t=1, $Controller_4$ switches on, and at t=2, $Controller_3$ also switches on. At starting of each controller, a position reference (0.5 cm from current position) is entered to each controller.

The accumulated cost for $Controller_i$ is used to measure the performance of a controller, which is given by:

$$J_i(t) = \int_0^t x^T(s)Qx(s) + u^T(s)Ru(s)ds \qquad (6)$$

The cost (6) is calculated at each time instant. A good control performance is therefore represented by a smooth increase in cost.

Finally, for comparing the scheduling results obtained with different scheduling approaches, the four plants are subjected to identical sequence of process noise in the two scheduling cases.

6.3 Simulation results

The simulation results in the two different scheduling approach are presented and discussed below.

6.3.1 Traditional scheduling approach

The accumulated costs of each controller (*J1, J2, J3,J4*) are shown in Figure 3. The close-up of schedule at t=1 and t=2 are shown in Figure 4 and Figure 5.

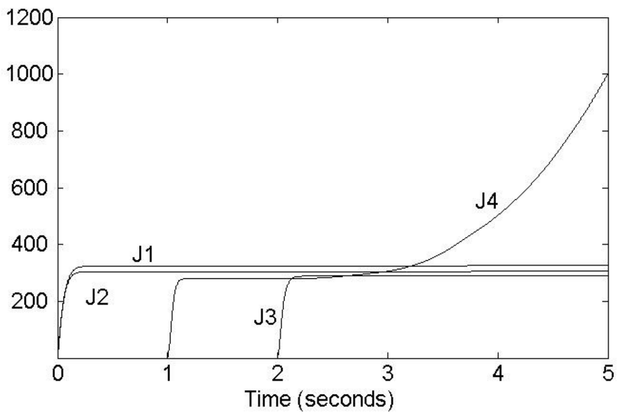

Figure 3. Accumulated costs under traditional scheduling

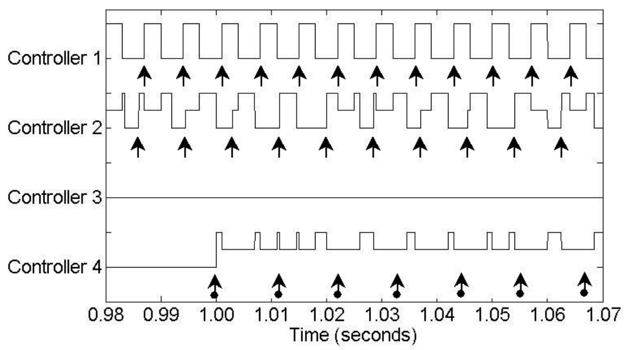

Figure 4. Close-up of schedule at *t*=1 under traditional scheduling

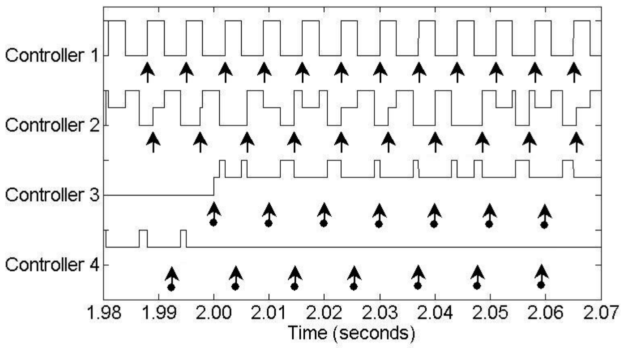

Figure 5. Close-up of schedule at *t*=2 under traditional scheduling

We will comment below the system evaluations represented on these figures. In figures 4 and 5, the state of each task is given. For each task, the state can take 3 values as represented in figure 6:

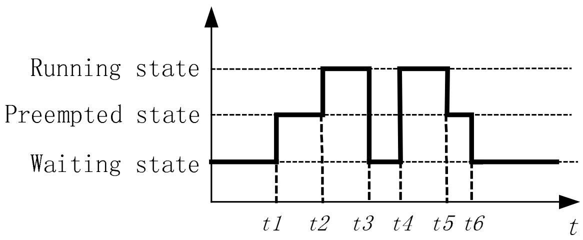

Figure 6. Task state representation

1. Running : being executed by the processor (e.g. state between *t2* and *t3*, *t4* and *t5*);
2. Preempted : the execution is preempted by other task (e.g. between *t1* and *t2*, *t5* and *t6*);
3. Waiting: the task waits for an activation : (e.g. state before *t1*, between *t3* and *t4,* after *t6*).

The arrows indicate the arrivals of task instances. Arrows with a solid point represent arrivals of task instances that miss their deadline.

During the observation, τ_1 and τ_2 are scheduled without any deadline violation. The accumulated costs increase steadily and the two systems perform well.

At t=1, τ_4 starts to execute. Under the preemption due to the execution of τ_1 and τ_2, the deadline of τ_4 is violated. However, the control performance is acceptable before the activation of τ_3 (at t=2) since the task instances are executed although their deadlines are all missed.

At t=2, τ_3 is turned on. Together with τ_1 and τ_2, the preemption due to these tasks makes the execution of τ_4 impossible. As a result, the cart system $Cart_4$ becomes instable (see *J4* in figure 3 for t > 2). Note that despite the execution preemption due to τ_1 and τ_2, the performance of $Controller_3$ does not decrease rapidly as for τ_4 because the task instances are executed although their deadlines are all missed.

6.3.2 Scheduling approach with (*m,k*)-firm constraint regulation

The simulation results obtained with the proposed approach are given in this subsection.

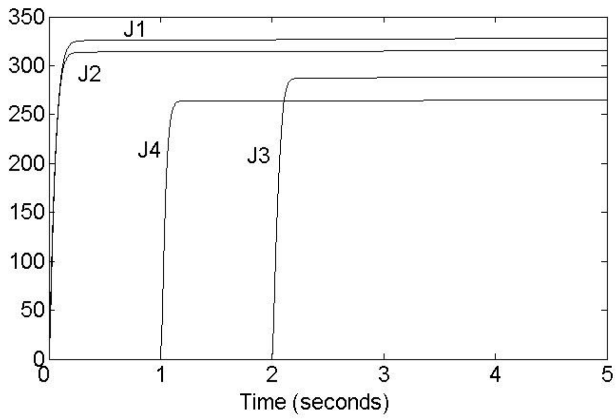

Figure 7. Accumulated costs under proposed scheduling approach

The accumulated costs for the four controllers are shown in Figure 7. The close-up of schedule at t=1 and t=2 are shown in Figure 8 and Figure 9. The signification of signs are the same as for the traditional scheduling case. Furthermore, the grey arrows mean that the task instances activated at the indicated instants are classified as optional instances and therefore they are not executed.

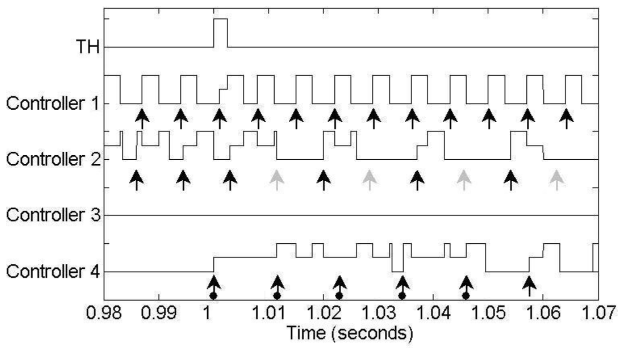

Figure 8. Close-up of schedule at *t*=1 under proposed scheduling approach

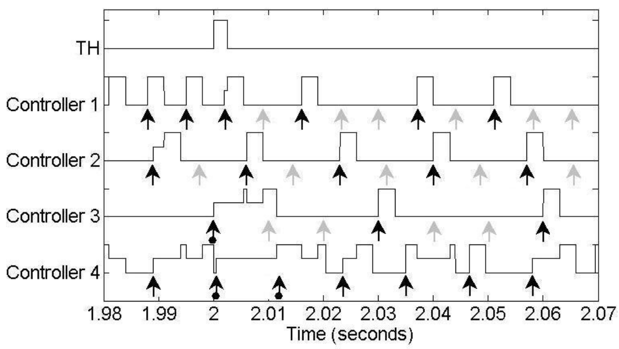

Figure 9. Close-up of schedule at *t*=2 under proposed scheduling approach

At the system starting, only τ_1, and τ_2 are activated. Their (*m,k*)-firm constraint are (*k,k*). Then, the system configuration change is detected at *t*=1 and the task handler is immediately activated. At *t*=1.025, the (*m,k*)-firm constraint of τ_2 are adjusted to (4,8)-firm constraint, and that of τ_1 is remained as (*k,k*). The overload condition is therefore avoided. The same thing is repeated at *t*=2, the (*m,k*)-firm constraints of the task τ_1, τ_2, τ_3 are adjusted to respectively (2,5), (4,8), and (3,10)-firm constraint; τ_4 is under (*k, k*)-firm constraint (see end of section 6.1). Note that the task deadline violations after the (*m,k*)-firm constraint adjustments in the two figures are due to the transient overload, however, the overload condition is removed rapidly.

Compared with the traditional scheduling approach, the controllers perform much better. The $Controller_4$ is stable and the control costs for $Controller_3$ and $Controller_4$ do not exceed 300 at *t*=5.

7. Conclusion

A scheduling architecture based on the (*m,k*)-firm constraint model is proposed. When a change in system configuration is detected, the task handler determines a strategy for selectively discarding task instances for each control task so that the schedulability of control tasks is guaranteed and the overall control performance is maintained at a high level.

Compared with formerly defined feedback scheduling approaches for control tasks, the proposed solution does not depend on the type and property of the control performance. That is, whatever the functions describing the control performance are convex, the proposed approach can always keep the overall control performance at high level while guarantying the schedulability of control tasks. Furthermore, at a system configuration change, the proposed solution avoids the change in the periods of related tasks, and does not alter the dynamics of the sub-system.

References

[1] Aström, K. J., "On the choice of sampling rates in optimal linear systems", Technical Report RJ-243, San José Research Laboratory, IBM, San José, California.

[2] Cervin, A., Eker, J., Bernhardsson. B., and Årzén, K.-E., "Feedback feedforward scheduling of control tasks," *Real-Time System.*, vol. 23, no. 1-2, pp. 25--53, 2002.

[3] Cervin, A., Henriksson, D., Lincoln, B., Eker, J., d Årzén, K.-E., "How does control timing affect performance" IEEE Control Systems Magazine, 23:3, pp. 16-30, June 2003

[4] Eker, J., Hagander, P., and Årzén, K.E., "A Feedback Scheduler for Real-Time Controller Tasks", *Control Engineering Practice*, vol. 12, no.8, p. 1369-1378, 2000.

[5] Felicioni. F., Jia. N., Song. Y.Q and Simonot-Lion. F., "Impact of a (*m,k*)-firm data dropouts policy on the quality of control", *6th IEEE International Workshop on Factory Communication Systems - WFCS'2006*, Torino, Italy, 2006.

[6] Jia, N., Hyon, E., Song, Y.Q., "Ordonnancement sous contraintes (*m,k*)-firm et combinatoire des mots", *13th International Conference on Real-Time Systems*, RTS'2005, Paris, France, 2005.

[7] Jia, N., Song, Y.Q., and Lin, R.Z., "Analysis of networked control system with packet drops governed by (m,k)-firm constraint". *Proc of the 6th IFAC international conference on fieldbus systems and their applications (FeT'2005)*, Puebla Mexico, 2005.

[8] Jia, N., Song, Y.Q., and Simonot-Lion, F., "Optimal LQ-controller design and data drop distribution under (*m,k*)-firm constraint", submitted to ECC'2007, available as Technical report at LORIA.

[9] Khan, S., LI, K.F., Manning, E.G. and Akbar, M. "Solving the knapsack problem for adaptive multimedia systems", *Studia Informatica universalis*, Vol. 2, No. 1, pp. 157-178, 2002.

[10] Liu. C.L., and Layland. J.W., "Scheduling algorithm for multi-programming in a hard real-time environment", *J.ACM*, vol. 20, pp.46-61, 1973.

[11] Lothaire., M. "Algebraic Combinatorics on Words", *Cambridge University Press*, 2002.

[12] Pisinger, D., "Algorithms for Knapsack Problems", Ph.D. thesis, DIKU, University of Copenhagen, Report 95/1, 1995.

[13] Quan, G., and Hu, X., "Enhanced Fixed-priority Scheduling with (*m,k*)-*firm* Guarantee ", *Proc. Of 21st IEEE Real-Time Systems Symposium*, pp.79-88, Orlando, Florida, USA, 2000.

[14] Ramanathan, P., "Overload management in Real-Time control applications using (m,k)-firm guarantee ", *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 549-559, 1999.

[15] Seto, D., Lehoczkyn, J. P., Sha, L. and Shin, K. G., "On task schedulability in real-time control systems", *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pp. 13-21, Washington, DC, USA, 1996.

[16] Silvano Martello, Paolo Toth (1990). Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons. ISBN 0-471-92420-2, 1990.

[17] Stankovic, J., Lu, C., Son, S. H., and Tao, G. "The case for feedback control real-time scheduling" In Proceedings of *the 11th Euromicro Conference on Real-Time Systems*, pp. 11-20.

[18] Stankovic, J., He, Tian., Abdelzaher, Tarek F., Marley, Mike., Tao, Gang., Son, Sang H and Lu Chenyang. "Feedback Control Scheduling in Distributed Systems". In *22nd IEEE Real-Time Systems Symposium(RTSS 2001)*, December 2001.