The 27th IEEE Real-Time Systems Symposium, work-in-progress session
Rio de Janeiro - RJ, Brazil, 5-8 December 2006

Time-Interval Scheduling and its Applications to Real-Time Systems

Fábio Rodrigues de la Rocha Rômulo Silva de Oliveira

***Abstract*— Many embedded real-time applications have strict time constraints to perform computations and access I/O devices. However, aspects related to real-time scheduling algorithms may interfere with the execution of jobs reducing the benefit of a task. We assume that the execution of a job has a positive benefit if performed inside an ideal time-interval. Operations performed before or after this ideal time-interval are considered worthless to the system. We propose a new task model for dynamic task scheduling where tasks are composed by segments and one of them must execute inside an ideal time-interval. This time-interval represents a valid execution interval and has a *QoS* function to assign a benefit according to the time the job executes.**

I. Introduction

MANY real-time problems can be found in embedded control systems. A common scenario is a single processor that cyclically and concurrently interacts with the external environment acquiring data from sensors and controlling devices. An approach to deal with recurrent tasks is to use well-known real-time task models. In the periodic task model, every task τ_i has a fixed period P_i, a worst-case execution time C_i and a relative deadline D_i [1]. The meaning of a deadline for a task τ_i is a time-limit to finish its computation. If the computation has finished anytime before the deadline the result is always correct does not matter when it has finished.

Although many applications are suitable represented by this model, there are some situations in which tasks have special constraints not fully achieved using periodic task models and the concept of deadline [2]. In some applications, tasks demand that a segment of its running code runs inside a specific time-interval. We present the following scenarios as examples. ① In an automated factory, an industrial equipment performs two activities over products that arrive on a conveyor belt. After performing the first activity in a product it must perform the second activity (such as paint the product) in a time that is specified by a time-interval which gives the highest benefit (e.g: If the ink is applied earlier or later the result is not ideal). Even thought the first activity is periodic, the time-interval for the second activity is variable and only known after the first activity ends. ② In real-time tracking and shooting problems [3]. The tracking system must track the object under surveillance and shot a projectile to hit the moving target. The solution keeps track of the object's position, predicts its future location at a specific time and program a shot event to the predicted time. ③ In video tracking applications, different tasks can access the same shared video camera to acquire images. Lets assume task τ_1 wants to position the camera to (x, y) and acquire an image. Task τ_2 wants to position the same camera to $(x2, y2)$ and acquire another image. Even though the sensor is the same, the data acquisition is a CPU demanding I/O operation. The process demands the control of actuators, reading of sensors to be sure the camera is at the correct position and data has the minimum necessary quality. During this process, the device as well as the CPU are busy.

Clearly, none of these examples have a time-limit to complete part of their computations, at most they have a time-interval and a time with the highest benefit, therefore, the concept of deadline is not appropriated to model these applications. Also, it is necessary to run a task in a time determined at run-time, which is hard to achieve using periodic models. Furthermore, we must remember that when sensors/actuators are shared by concurrent tasks, tasks can be released at the same time resulting in queues to access the sensors and therefore deviations in the sensing and actuation times.

We assume an ideal time-interval to execute a job in a single processor system, which gives the highest contribution to the application. Task value decreases before and after the ideal time-interval. Computations performed before or after the specific time-interval might be useless for the application's purposes. Differently from previous work about task rewarding [4],[5],[6], our reward criterion is connected to the specific moment the job executes instead of its finish time or the amount of computation performed. We implement our model using *EDF* due to its capacity to exploit full processor bandwidth.

The paper is organized as follow. Section II summarizes related work. Section III presents the time-interval task model. Some approaches to implement to ideal time-interval model are presented in section IV and conclusions in section V.

II. Literature survey

The classic approach for a time-precise execution is a time-driven system [7]. However, time-driven systems are known by its inherent inflexibility when the schedule need be changed [8]. In [9] is shown an overview of value based-scheduling, their effectiveness to represent adaptive systems and present an framework for value-based scheduling. In [10] is presented a study about scheduling in overload situations in which tasks have a deadline and also a quality metric. The scheduler's performance is evaluated by the cumulative values of all tasks completed by their deadline and shows that in overload situations scheduling tasks by its value results in a better performance. In [4] is presented a model to schedule tasks composed by a mandatory and an optional part that increases the benefit of the task's execution. In [5] is presented

This research was supported by CNPq and CAPES.

The 27th IEEE Real-Time Systems Symposium, work-in-progress session
Rio de Janeiro - RJ, Brazil, 5-8 December 2006

a special case of imprecise computation where the reward increases as the task executes until its deadline. In [6] is presented the Time Utility Function model in which there is a function to assign a benefit obtained according to task's completion time. An extension of this work is presented in [11]. In that work is presented the concept of Joint Utility Function in which an activity utility is specified in terms of other activity's completion time. In [12] is presented an online interval scheduling problem in which a set of time-intervals are presented to a scheduling algorithm. The time-intervals are non-preemptive, have a start and end times and cannot be scheduled either early or late. As a future work, the authors discuss a different problem in which the release times are more general and a task could request a given time-interval to be delivered within "x" time units. The time-interval would be allowed to slide slightly to accommodate other tasks.

III. Time-Interval Model

We propose a new task model in which tasks τ_i, $i \in \{1 \dots n\}$ are described by a worst-case execution time C_i, period P_i and relative deadline D_i. We define by **segment** a sequential group of instructions of τ_i. Task τ_i can demand that segment of its computation B_i runs inside a time-interval $[s_{i,j}, e_{i,j}]$ (Figure 1). Instead of a deadline to conclude its computation, segment B_i has an ideal time-interval to execute. The ideal time-interval is determined by the previous segment A_i of task τ_i. The worst-case execution time of B_i is $C_{B_i}, C_{B_i} < C_i$. The execution of segment B is not always required i.e: in some activations A may not request its execution and so segment C will not execute either. In case segment B is required to execute, as soon as this segment concludes C is released to run. In figure 2 is shown the task benefit for a firm and hard real-time task. The benefit $v(t)$ is given by the equations in figure 2 and the *QoS* is given by $QoS(B_{i,j}, t) = \frac{\int_{start_{B_{i,j}}}^{end_{B_{i,j}}} v(t)dt}{end_{B_{i,j}} - start_{B_{i,j}}}$. The maximum benefit is 1 (100%) and the task activation has a benefit if executed inside the time-interval $[s_{i,j}, e_{i,j}]$, otherwise the benefit is zero. The *y* axis is the benefit and the *x* axis is the activation time. The execution of segment B is shown undisturbed by other tasks.

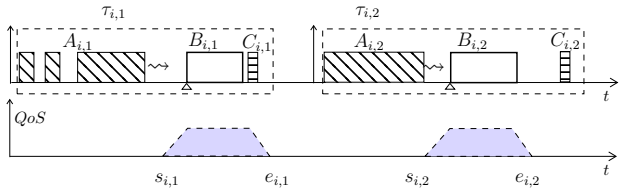

Fig. 1. Task model showing two jobs of task τ_i and a firm *QoS* function.

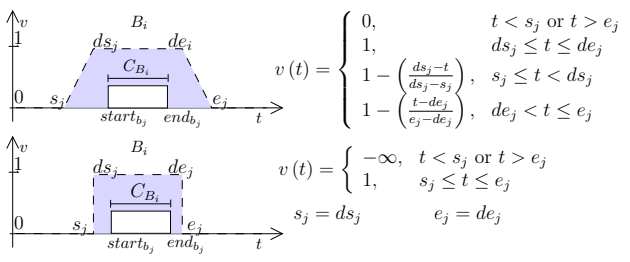

Fig. 2. *QoS* for a firm and a hard real-time task τ_i.

The ideal time-interval problem may also present constraints regarding exclusive access during segment B execution inside the ideal time-interval due to the nature of the devices under segment B control. Also, we assume that during the execution inside the ideal time-interval the CPU is busy controlling the devices. We define three **execution modes**:

Non-blocking execution mode – the execution of segment B_i from task τ_i can be disturbed by other segment B_j of task τ_j with greater urgency (priority). For instance, tasks τ_i and τ_j can share the same sensor as in a simple data acquisition operation. The QoS of τ_i is the cumulative QoS obtained by B_i while running inside the ideal time-interval. **Blocking execution mode** – the execution of B_i from task τ_i can be disturbed by other task τ_j. However, B_j cannot execute while B_i has not finished (i.e: the execution of segment B is serialized). This restriction is important to keep the consistence in cases where tasks τ_i and τ_j must access a directional sensor or other device in which is necessary to keep the device's mutual exclusive access and the cost/overhead of a rollback operation of all changes is very high. Therefore, the execution of segment B_i is similar to the execution of a task in a shared resource situation. **Strict execution mode** – the execution of B_i from task τ_i cannot be disturbed by other task τ_j due to both strict time requirements and device's mutual exclusive access. In this case, the start of B_i could be postponed, however, once started it cannot be disturbed. For instance, in real-time tracking problems and industrial equipment's control the sensors/actuators cannot be shared among tasks while there is an ongoing execution. Also, to ensure the correct timing behavior the operation cannot be preempted.

IV. Scheduling approach

Clearly, the strict execution mode is more restrictive and penalizes the schedulability bound. Moreover, when there are only non-blocking or blocking tasks, a simpler approach can be used. Thus, in this section the scheduling approaches are presented by their capacities to support the execution modes. As we consider an implicit separation inside a task it sounds natural to represent it as a set of subtasks. Therefore, we map all the segments of task τ_i into subtasks A_i, B_i and C_i.

A. *Non-blocking execution mode*

In this execution mode subtasks B_i are allowed to access non-blocking or private resources, which do not need a special treatment to concurrent access. Also, subtasks B_i can be disturbed by other subtasks with higher priorities. Therefore, using a preemptive approach (as shown in figure 3 on left) we can fulfill all requirements of this execution mode.

Preemptive approach – The synchronous task model considers all tasks released together at the first activation, however, it is clear that B_i cannot be scheduled before A_i ends neither C_i can be scheduled before B_i ends. A better schedulability test would consider the arrival of B_i and C_i only after the deadlines of their previous subtasks. In the asynchronous model, subtasks arrive and are released after an offset ϕ and a new schedulability test is necessary to deal with offsets. In this case, a necessary and sufficient schedulability test needs

The 27th IEEE Real-Time Systems Symposium, work-in-progress session
Rio de Janeiro - RJ, Brazil, 5-8 December 2006

an exponential time to run. In [13] the authors proved that the decision problem about the feasibility of an asynchronous task set with $\forall i, D_i \leq T_i$ is *NP-Hard*. Coffman [14] shown that for asynchronous tasks with deadline equal to period is still sufficient to test the schedulability using the test $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ [1]. In [15] Baruah argues that we can still use the processor demand approach for $D_i \leq T_i$ in case we set all offsets to zero. If the new task set is feasible, the task set with offsets is also feasible. Otherwise, if the new task set in not feasible no definitive answer can be given. This test is pessimistic and new sufficient tests considering the offsets in *EDF* are proposed in [16],[17]. These tests increase the schedulability of task sets that otherwise could be rejected by the previous tests. The drawback is their algorithmic complexity.

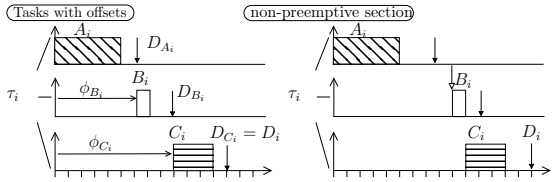

Fig. 3. Two approaches: subtasks with offsets and non-preemptive section.

Note that it is possible to arbitrate the deadlines for A and B and test the schedulability or use the Response-Time Analysis to compute the worst-case response-time of subtasks and use this value to set a tighter deadline.

B. ***Blocking*** *execution mode*

In this execution mode the execution of subtask B is modeled as a task running in a shared resource scenario. A critical section is used to guarantee the exclusive access to a resource following a protocol [18], [19], [20]. We assume the subtask B accesses a blocking resource Rb when a resource is protected by a critical section (e.g: using a semaphore). The execution of subtask $B_i, \forall i \in 1, \ldots, n$ holds a resource $R_j / j < n$ in exclusive mode (blocking resource). We assume B_i can hold only one resource and the duration of the critical section protecting this resource is the same $WCET(B_i)$. Therefore, B_i can be blocked in R_j for the length of the maximum critical section of the other subtask B. In ***PCP*** and ***SRP*** a task can be blocked by at most one critical section. The maximum blocking time E is defined by $\forall i, j | i = \{1, \ldots, n\}, j \neq i, E_i = max(WCET(B_j))$.

Therefore to account for blocking resources we extend the non-blocking approach adding the term E, the maximum blocking time in the schedulability test.

C. ***Strict*** *execution mode*

The strict execution mode demands a non-preemptive execution which cannot be achieved using the two previous approaches. Fortunately, in [21] is shown a useful schedulability condition in a model to ensure the schedulability of *EDF* in the presence of interruptions. Interruptions are modeled as tasks with higher priorities than application tasks and so cannot be preempted. This method works reducing the time length available for the execution of application tasks due to the time necessary for execute the interrupt handler. So, if tasks can finish before their deadline even suffering the interference from the interrupt handler, the task set is schedulable. Subtask B is modeled as an interrupt handler and the feasibility checked using [21].

Non-Preemptive approach – Subtasks A_i and C_i are preemptive and scheduled by *EDF*. On the other hand subtask B_i is non-preemptive and scheduled considering the *QoS* metric (figure 3, on right side). In this aspect, the solution presented in [21] is easily applied when tasks are independent and synchronous, however, the problem gets worst when tasks have precedence relations and different release times. In subsection 1 we detail the test and adapt our model to a synchronous system with jitters and in section 2 we expand the test to the case of asynchronous tasks. The theorem to account for interrupt handlers as expressed by Jeffay and Stone assumes a task set with deadlines equal to periods. Using the processor demand approach we can extend this theorem for deadlines less than periods as in:

Theorem *4.1: A set τ of n periodic or sporadic tasks and a set ι of m interrupt handlers is schedulable by EDF if and only if*

$\forall L \in \sigma \quad \sum_{i=1}^{n} \lfloor \frac{L+T_i-D_i}{T_i} \rfloor C_i \leq L - f(L)$ *where the upper bound $f(L)$ can be computed by:*

$$f(0) = 0$$

$$f(L) = \begin{cases} f(L-1) + 1 & if \sum_{i=1}^{m} \lceil \frac{L}{T_i^H} \rceil C_i^H > f(L-1) \\ f(L-1) & otherwise \end{cases}$$

Due to space constraints, the proofs are omitted in this paper and can be obtained in [15] and [21].

1) Synchronous system - jitters: In this case, subtasks A_i and C_i arrive at time zero. At a specific time an interrupt starts B_i. To ensure subtask C_i runs only after subtask B_i, we model subtask C_i with a release jitter $J_{C_i} = Bmax_i$. Where $Bmax_i$ is the upper bound to release subtask B_i. As the system is still synchronous we can modify theorem 4.1 to include the jitter effect. The new feasibility test in which all subtasks C have jitters and subtasks B are modeled as interruptions is:

$\forall L \geq 0 \quad \sum_{i=1}^{n} \lfloor \frac{L+T_i+J_i-D_i}{T_i} \rfloor C_i \leq L - f(L)$

We can extend this approach to support the non-blocking execution mode by adding subtasks A_j, B_j and C_j without consider B_j as an interruption handler. In this case we need a jitter to release B_j. Also, we can support blocking execution mode by adding the blocking time to the feasibility test. However, the maximum blocking time is much more pessimistic due to the non-preemptive subtasks.

2) Asynchronous system: The schedulability test provided by [21] is based on the processor demand criterion [15] and the same analysis is still valid if we consider the processor demand for asynchronous tasks. In this case, instead of checking the busy period L at most up to the H(hyper-period), we need to test all busy periods from 0 to $2.H + \Phi$ (where Φ is the largest offset in the task set). The drawback is the exponential time complexity due to the hyper-period limit which in the worst-case is $O(H^2)$ [22]. With this new test, we can assign properly the deadlines and offsets for subtasks A_i and C_i and verify the schedulability considering B_i as an interruption. We can extend this approach to support the non-blocking execution

mode if we add new subtasks A_j, B_j and C_j without consider B_j as an interruption handler. However, we need to adjust new offsets for B_j and C_j. Also, we can support the blocking execution mode by adding a pessimistic blocking term E in the feasibility test.

Increasing the *QoS* – Assuming a feasible task set with non-preemptive section, the QoS can be improved if the times to start subtasks B are ordered to increase the quality metric. This operation is performed by an heuristic server based on the algorithm presented in [23]. The heuristic server collects all requests from subtasks A and orders in run-time the future event queue targeting the subtasks B. Lets define :

s_k – start time of B_k, e_k – end time of B_k, $s_{k'}$ – new start time of B_k due to a slide, $e_{k'}$ – new end time of B_k due to a slide, r_k – ready time for a task B_k.

$\mathbf{A}$ is a set of subtasks B_j in non-decreasing order of u,

$$u = \begin{cases} r_j, & \text{if } r_j + WCET(B_j) \geq d_e \\ d_s, & \text{if } r_j + WCET(B_j) < d_s \\ r_j, & \text{if } d_s < r_j < d_{ej} \end{cases}$$

$\mathbf{S}$ is a partial schedule constructed using the subtasks from $\mathbf{A}$ in the following order. Pick each B_j subtask from $\mathbf{A}$ and insert in $\mathbf{S}$ to execute at u time. If the new subtask overlaps with a subtask B_k in $\mathbf{S}$ (shared area in figure 4), accommodate subtask B_j using one of the following four movements. The gain/lose factor Δ is the result of such movement in *QoS* and it is used to decide the new order of subtasks B.

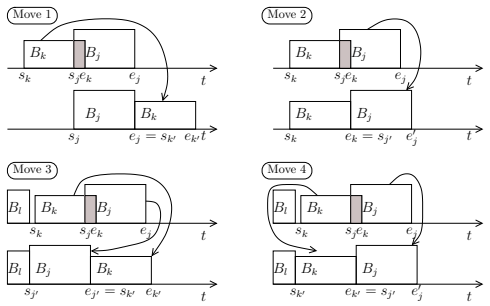

Fig. 4. Movements to remove the overlap.

In the first movement, we slide B_k to the right resulting in new start and end times for B_k. The factor Δ for this case is $\Delta_1 = QoS(B_k, s_{k'}) - QoS(B_k, s_k)$. Similarly, in the second movement $\Delta_2 = QoS(B_j, s_{j'}) - QoS(B_j, s_j)$. In the third movement, we slide B_k to the right and B_j to the left, resulting in new start and end times for both. However, it is necessary to compute the new start time for B_j considering the existence of a previous subtask B_l. The new start time is computed as $nst = max(r_j, e_l, s_k - WCET(B_j))$. We compare the ready time of j, the end time of l and the minimum slide of j necessary to remove the overlap. Then, the new end times of B_j and B_k are $e_{j'} = nst + WCET(B_j)$ and $e_{k'} = e_{j'} + WCET(B_k)$ and $\Delta_3 = QoS(B_k, s_{k'}) - QoS(B_k, s_k) + QoS(B_j, s_{j'}) - QoS(B_j, s_j)$. In the forth movement $nst = max(r_k, e_l, s_j - WCET(B_k))$ and $\Delta_4 = QoS(B_j, s_{j'}) - QoS(B_j, s_j) + QoS(B_k, s_{k'}) - QoS(B_k, s_k)$.

V. Conclusions and Future Work

In this preliminary paper we proposed a new task model where tasks have a benefit assigned with its execution inside an ideal time-interval. Our model is useful in many real-world problems which by the lack of theoretical study are implemented with conventional schedulers resulting in poor results and unknown schedulability bounds. We presented two approaches to implement the model and a technique to re-order the non-preemptive sections to increase the resulting *QoS*.

References

[1] J. Layland and C. Liu, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] B. Ravindran, E. D. Jensen, and P. Li, "On Recent Advances In Time/Utility Function Real-Time Scheduling And Resource Management," in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC 2005.*, 2005, pp. 55–60.

[3] T. Facchinetti and G. Buttazzo, "A Real-Time System for Tracking and Catching Moving Targets," in *Proc. of the* 5^{th} *Intl. Symposium on Intelligent Comp. and Instruments for Control Applications*, 2003.

[4] J. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computations," in *Proc. of the IEEE*, vol. 82, no. 1, 1994, pp. 83–94.

[5] J. Dey, K. J., and D. Towsley, "On-line scheduling policies for a class of IRIS (increasing rewardwith increasing service) real-time tasks," in *IEEE Transactions on Computer*, 1996.

[6] E. Jensen, C. Locke, and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems," 1985.

[7] C. D. Locke, "Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives," *Real-Time Systems*, vol. 4, no. 1, pp. 37–53, 1992.

[8] H. Tokuda, J. W. Wendorf, and H. Wan, "Implementation of a time-driven scheduler for real-time operating systems," in *Proc. of the IEEE Real-Time Systems Symposium*, 1987, pp. 271–280.

[9] A. Burns, D. Prasad, A. Bondavalli, F. D. Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini, "The Meaning and Role of Value in Scheduling Flexible Real-Time Systems," *Journal of Systems Architecture*, vol. 46, pp. 305–325, 2000.

[10] G. C. Buttazzo, M. Spuri, and F. Sensini, "Value vs. Deadline Scheduling in Overload Conditions," in *IEEE RTSS*, 1995, pp. 90–99.

[11] H. Wu, B. Ravindran, E. D. Jensen, and U. Balli, "Utility Accrual Scheduling under Arbitrary Time/Utility Functions and Multi-unit Resource Constraints," in *Proc. of the* 10^{th} *Real-Time and Embedded Computing Systems and Applications*, 2004.

[12] R. J. Lipton and A. Tomkins, "Online Interval Scheduling," in *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA, 1994, pp. 302–311.

[13] J. Leung and M. Merill, "A Note on the Preemptive Scheduling of Periodic, Real-Time Tasks," *Information Processing Letters*, vol. 11, no. 3, pp. 115–118, Nov 1980.

[14] E. C. Jr, "Introduction to Deterministic Scheduling Theory," in *Computer and Job-Shop Scheduling Theory*. John Wiley & Sons, 1976.

[15] S. k. Baruah, R. R. Howell, and L. E. Rosier, "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor," *Real-Time Systems*, vol. 2, pp. 301–324, 1990.

[16] J. P. Gutierrez and M. G. Harbour, "Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF," in *15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, 2003.

[17] R. Pellizzoni, "Efficient Feasibility Analysis of Real-Time Asynchronous Task Sets," Master's thesis, Università di Pisa and Scuola Superiore S. Anna, Pisa, Italy, 2003.

[18] M. Chen and K. Lin, "Dynamic priority ceiling: A concurrency control protocol for real-time systems," in *Journal of Real-Time Systems*, 1990.

[19] M. Spuri, "Efficient Deadline Scheduling in Real-Time Systems," Ph.D. dissertation, Scuela Superiore S. Anna, July 1995.

[20] T. P. Baker, "Stack-Based Scheduling of Real-Time Process," in *Real-Time Systems Journal*, 1991.

[21] K. Jeffay and D. L. Stone, "Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems," in *Proceedings of the* 14^{th} *IEEE Symposium on Real-Time Systems*, December 1993, pp. 212–221.

[22] J. Goossens, "Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints," Ph.D. dissertation, Université Libre de Bruxelles, 1999.

[23] R. Mazzini and V. A. Armentano, "A Heuristic For Single Machine Scheduling With Early And Tardy Costs," *European Journal of Operational Research*, vol. 1, pp. 129–146, 2001.