A new heuristic algorithm to assign priorities and resources to tasks with end-to-end deadlines

M.A. Moncusí, J.M. Banús
Dept Enginyeria Informàtica
Universitat Rovira i Virgili
Autovia de Salou sn,
43006 Tarragona, Spain

J. Labarta
Dept Arquitectura de Computadors
Universitat Politècnica de Catalunya
Jordi Girona, 1-3. D6 Campus Nord
08034 Barcelona, Spain

A. Arenas
Dept Enginyeria Informàtica
Universitat Rovira i Virgili
Autovia de Salou sn,
43006 Tarragona, Spain

Abstract *We propose a new off-line heuristic algorithm to allocate and assign priorities in distributed hard real-time systems when tasks have end-to-end deadlines. We have found solutions two order of magnitude faster than the simulated annealing technique applied to scheduling the same task sets, with maximal average loads around 90%.*

Keywords: Distributed real-time system, End-to-end Deadlines, Task allocation, Priority assignment.

1 Introduction

The correctness of the real time systems depends not only on the logical results of the computation, but also on the completion of all tasks before their deadlines. In a distributed real-time system, tasks may need the use of more than one resource and different processors to be executed. Then, tasks can be partitioned into a sequence of subtasks with precedence constraints among them. The timing constraints of these tasks are typically given as end-to-end deadlines, which is defined as the maximum response time allowed from the release of the first task in the sequence to the completion of the last task in the sequence.

A common approach to the end-to-end scheduling consist in decomposing an end-to-end system into a set of uniprocessor systems and then apply some existing real-time scheduling algorithms for uniprocessor. Network communication can be modeled as a fixed delay [1], or it can be considered as another resource where messages are represented as tasks [2]. Alternatively, a complete study of a bus protocol [3] can be done. The problem in those cases relies in assigning the tasks to the resources and choosing an uniprocessor scheduling algorithm.

Many researches have explored possible solutions to the problem of assigning tasks to resources. For example, bin-packing techniques with a preemptive fixed priority scheduling algorithm (Deadline Monotonic) [4], [5], [6] or a preemptive dynamic priority algorithm (Earliest Deadline First). Also, "*branch and bound*" techniques with a time driven algorithm to solve the problem of scheduling tasks with precedence constraints have been used [1]. Tindell et al [3] uses the simulated annealing algorithm as a global optimization problem, including the study of a bus protocol and the memory utilization of tasks.

In particular, Gutiérrez et al [2] deal with the priority assignment problem when tasks have an end-to-end deadline and all tasks are allocated to a specific resource. In their work, they use the Deadline Monotonic scheduling algorithm to schedule tasks and they consider the possible jitter between the tasks in the sequence [7]. As they used a deadline monotonic scheduling, they need to assign an artificial local deadline to tasks in the sequence in order to assign static priorities to these tasks. Initially, they assign a local deadline proportionally to the calculation tasks needs, and then, increase or decrease it to find a schedulable solution with a deadline monotonic algorithm.

We propose a new heuristic algorithm to assign tasks with end-to-end deadlines to resources and to assign priorities to tasks in order to schedule them with a preemptive fixed priority algorithm. The priorities we assign to tasks are not necessary related to deadlines, not to periods but they can be related to the importance of the task for the system, or just the ones needed to make the system schedulable. The tasks can be pre-allocated to a specific

resource [2], or be completely free to allocate anywhere [8], or a mixture, that is, pre-allocated tasks and non pre-allocated tasks. In order to compare the results of our algorithm, we also have implemented a simulated annealing technique to do the same.

The remainder of this paper is organized as follows: we present the framework in section 2. We then describe our heuristic algorithm in section 3. We describe briefly our implementation of the simulated annealing technique in section 4. Section 5 contains detailed experimental results and we conclude in Section 6.

2 Framework

We consider a distributed real-time system consisting of a set of periodic tasks Γ where $|\Gamma|=N$, a set of end-to-end deadlines Δ where $|\Delta|=M$ and a homogeneous distributed set of resources $\Re$ where $|\Re| = P$. We define $\Gamma=\{\tau_i(T_i, C_i, D_i, R_i) \mid (1 \le i \le N) \wedge (1 \le C_i \le D_i) \wedge (C_i \le T_i) \wedge (R_i \in \Re \vee R_i=\varnothing)\}$. That is, each task τ_i is being characterized by its period T_i, its "worst-case" computation time requirement C_i, its deadline D_i, and its specific resource R_i. The period T_i and the deadline D_i are arbitrary. If R_i is $\varnothing$, means that the tasks can be allocated to any resource, that is, the task is non pre-allocated. And we define $\Delta=\{Dee_i (\tau_0, \tau_1,...,\tau_K) \mid (1 \le i \le M) \wedge ((\tau_j \in \Gamma) \wedge (\Sigma C_j \le Dee_i) \wedge (T_j=T_i)) \wedge (0 \le j \le K)\}$.

The tasks of the end-to-end deadlines $\tau_j \in Dee_i$ do not have a defined local deadline, but have to satisfy an end-to-end deadline. We assign a local deadline to these tasks, just to prove that the system is schedulable in the usual way. The sum of the local deadlines must be less or equal to the end-to-end deadline Dee_j. And finally, all tasks must execute according to an established order in a sequence. This order can be guaranteed by releasing each task in the sequence after some time, equal to the sum of local deadlines of the all previous tasks in the end-to-end deadline. This time is called the offset and, in this paper, all offsets are static [9].

Our goal is to obtain a static partition from Γ, $\wp(\Gamma) = \{\Gamma_0, \Gamma_1,...,\Gamma_{P-1}\}$, where $\forall \tau_j \in \Gamma_i$ are executed in the resource $i \in \Re$ and $\forall \Gamma_i \in \wp(\Gamma)$, Γ_i is schedulable using a preemptive fixed priority scheduling algorithm.

The usual way to test the schedulability of a real time system uses the measure of the "worst-case" response time S_i of a certain task τ_i. If $\forall \tau_i \in \Gamma_i$, $S_i \le D_i$ then resource i is schedulable [10] [11] and consequently, if all $\Gamma_i \in \Gamma$ are schedulable, the system is schedulable. However in our framework, if some tasks τ_i are part of an end-to-end deadline Dee_j the prior schedulability condition is not sufficient. To test the end-to-end deadline schedulability, it is also necessary to satisfy

$$\sum_{\tau_i \in Dee_j} S_i \le Dee_j \quad \forall j \in \Delta$$

To calculate the "worst-case" response time S_i of task τ_i, we use the algorithms proposed by Joseph and Pandya [10], Liu and Layland [12] with the extension proposed by Lehoczky [11] in the scope of pre-emptive fixed-priority scheduling. All these algorithms are designed assuming uniprocessor systems and the extension assume arbitrary deadlines. In multiprocessor systems there are not well established analytical studies and it is a common practice to analyze each resource in the system as independent [2] [3] [13]. We also apply the uniprocessor scheduling to the multiprocessor system considering each resource in the system as independent, each communication network as a resource, and each message as a task, where C_i of these messages is the worst-case delay caused by the messages.

The worst-case response time we calculate are based on the assumption that all tasks are independent and this lead us to have a pessimistic results. If one resource has two tasks from the same Dee, the lower priority task τ_i will not have the interference of the higher priority task τ_{i-1}. This implies the calculated worst-case response time will be greater than the truth one.

To characterize each resource $k \in \Re$, its load U_k is defined, as usual, by the sum of ratios between the computation time and the period of the tasks $\tau_i \in \Gamma_k$ and must satisfy the total utilization limit

$$U_k = \sum_{\forall \tau_i \in \Gamma_k} \frac{C_i}{T_i} \le 1$$

3 The heuristic algorithm

We propose a new off-line heuristic algorithm to allocate and to assign priorities to a set of tasks in a set of resources in the framework explained before. The aim of the algorithm is a simple heuristics that looks for schedulability while harnessing the maximal capability of resources. The whole algorithm follows the heuristic of dealing first with the most restrictive problem, to end with the least restrictive one.

The methodology used in our algorithm decomposes the problem into two mainly steps:

a) Assignment of the relative priorities in each resource among the tasks that have been pre-assigned to it. (see algorithm in figure 1)
b) Allocation and assignment of the relative priorities in each resource among the non-preallocated tasks. (see algorithm in figure 2)

After run these two steps we have the relative order of all tasks in each resource.

Now, we just have to assign the final priorities among all the tasks in each resource in order to schedule them in a preemptive priority scheduler. Assign the rest of the end-to-end deadline to tasks of the deadline sequence, proportionally to the ones assigned in the two initial steps.

Let us describe the two main steps:

a) The tasks that must be executed in a specific resource are directly assigned to this resource. To prioritize these tasks, first it is necessary to know their worst response time, and for this, it is necessary to sort the resources by the most restrictive conditions of the end-to-end deadlines. The greater the number of tasks belonging to end-to-end deadlines the more restrictive the resource. Then we sort by the number of end-to-end tasks the resource has, in decreasing order. When different resources have the same number of tasks we decide the order by the number of the tasks belonging to the same end-to-end deadline or, if it is also the same, arbitrarily. This procedure shows to be irrelevant for the independent tasks but it is adequate for prioritizing first the end-to-end deadline restrictions.

For example, consider three resources $\Re=\{0,1,2\}$ and two end-to-end deadlines $|\Delta|=2$ with the following allocated tasks:

$\Gamma=\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8, \tau_9, \tau_{10}, \tau_{11}, \tau_{12}, \tau_{13}, \tau_{14}\}$. The end-to-end deadlines: $Dee_1=\{\tau_{10}, \tau_{11}, \tau_{12}\}$ and $Dee_2=\{\tau_{13}, \tau_{14}\}$. $\wp(\Gamma)=\{\Gamma_0,\Gamma_1,\Gamma_2\}$,where $\Gamma_0=\{\tau_1,\tau_2,\tau_3,\tau_{10},\tau_{12}\}$, $\Gamma_1=\{\tau_4, \tau_5,\tau_{11}, \tau_{13}\}$ and $\Gamma_2=\{\tau_6, \tau_7,\tau_8, \tau_9,\tau_{14}\}$

The resulting order for the resources in this particular configuration is 0, 1 and 2 after resolving the tie between the resources 0, 1. Once the resources are sorted, we start assigning priorities to schedule the tasks of this resource according to this order. For every resource the relative priority of each task P_i is calculated as follows

$$P_i = \frac{S_i}{D_i} \quad \forall \tau_i \notin Dee_k \ \forall k \quad \text{and}$$

$$P_i = \frac{\sum_{\forall j \in Dee_k} S_j}{Dee_k} \quad \forall \tau_i \in Dee_k$$

where S_j is the worst-response time.

This calculation assigns higher priorities to those tasks that are less “flexible”, being this flexibility a measure of the relative laxity of each task. The relative priorities we assign to the end-to-end deadline tasks are a simple extension of this flexibility. Once priorities are assigned, we test schedulability of the resource using the calculus of the worst-response time [10] and [11]. The process is sequential in the sense that the scheduling of the resource *i* in the sorted sequence is achieved before the scheduling of the resource *i+1*. This implies that worst response time S_i is updated every time a task is scheduled, initially $S_j=C_j$,$\forall\tau_i \in \Gamma$. This step is not iterative, that is, every resource is visited only once.

The whole process of sorting the resources and assigning priorities follows the heuristic of the dealing with the most restrictive case first. This heuristic search stops when all the resources are proved to be schedulable, or otherwise when one task of one resource does not pass the schedulability test.

b) For the non-allocated tasks, we first calculate the P_i's in the same way as before. We then use this prioritization to search the possible allocation of the tasks in a resource. The criteria to allocate is: i) to look for the more loaded resource that makes still possible to accept the task, i.e. that the sum of the maximum load of the resource plus the load of the task be less or equal than 1, and, ii) if the task can be allocated by load, it is assigned a relative priority depending on the rest of end-to-end deadline not still consumed, after that, it is necessary to check its schedulability (resource schedulability and end-to-end deadline schedulability). While the criteria are not totally fulfilled, we try to allocate the task in following most loaded resource.

When we have tried all the possible resources without succeeding, we assign the task to the less loaded resource. In this case, no schedulability is attained then, we try to re-allocate all the non-allocated tasks via a self-consistent algorithm.

The self-consistent algorithm works as follows:

a) Re-calculate the prioritization of all the non-allocated tasks using the S_i's obtained in the last step

b) We try to allocate the tasks according to the prior criteria

c) If the system is still not schedulable, there are changes in the prioritization order and we have not consumed the maximum number of iterations we allow in the self-consistent algorithm go to step i, otherwise finish the process.

Sort the resources by deadline restrictions.
$\forall\, Dee_k \in \Delta, S_{Dee_k} = \sum_{\forall \tau_i \in Dee_k} C_i$
for every resource R_i do
  Calculate the relative order of tasks
  Sort tasks by the prior order
  Calculate the worst execution time (S_j) for tasks$\in \Gamma_i$
  $\forall \tau_j \in \Gamma_i,\ \forall \tau_j \in Dee_k, D_j = \min(S_j, Dee_k - S_{Dee_k})$
  $S_{Dee_k} = S_{Dee_k} + D_j$
end for
If any resource is not schedulable then
  exit with no success
end if

Figure 1: Algorithm to Assign relative priorities among pre-assigned tasks

for every non pre - allocated task do
  Calculate the relative order of tasks
  Sort tasks
end for
Sort the resouces by load
for every non - preallocated task in the prior order τ_j do
  Find a max loaded resource k satisfying $U_k + \frac{C_j}{T_j} \le 1$
  while Γ_i is not schedulable and $i \le |\Re|$ do
    $\Gamma_i = \Gamma_i + \tau_j$
    calculate the worst execution time (S_j) for tasks$\in \Gamma_i$
    $\forall \tau_j \in \Gamma_i,\ \forall \tau_j \in Dee_k, D_j = \min(S_j, Dee_k - S_{Dee_k})$
    $S_{Dee_k} = S_{Dee_k} + D_j$
    if Γ_i is not schedulable and $i < |\Re|$ then
      $\Gamma_i = \Gamma_i - \tau_j; S_{Dee_k} = S_{Dee_k} - D_j$
      $\Gamma_i = \Gamma_{i+1}$
    end if
    Sort the resouces by load
  end while
  if $\forall \Gamma_i \in \Gamma, \Gamma_i$ is schedulable then
    exit with success
  else
    reset the partitions
    for every non - preallocated task do
      Calculate the relative order of tasks using the last S_i
      Sort tasks
    end for
  end if
end for

Figure 2: Algorithm to Allocate and Assign tasks and relative priorities among non-preallocated tasks

4 Simulated annealing algorithm

To compare the performance of our algorithm, we have implemented a simulated annealing algorithm [3]. Simulated annealing is a global optimization technique, which attempts to find the lowest point in an energy landscape. The algorithm requires a neighbor function, an energy function, and a cooling function.

The neighbor function that we used consisted of choosing two tasks from the non pre-allocated task set and exchanging their resources, if the two tasks are allocated on the same resource we exchange their priorities. After that, we test the resource schedulability applying a modification of the completion time test [10] to calculate the worst-case response time if deadlines are greater than periods [11]. We modify the assignment of the local

deadlines in order to make tasks schedulable if possible.

The energy function consisted on the sum of the cost of every resource if all of them are schedulable or the sum of the cost of the non-schedulable resources. The cost of the resource is calculated as follows:

$$\forall i \in \Re, \text{Cost}(i) = -\sum_{\forall \tau_j \in \Gamma_i} (D_j - S_j)$$

Note that if resource $i \in \Re$ is schedulable then the cost is negative otherwise the cost is positive. If $\exists\ i \in \Re$ such as $U_k > 1$ then cost(i)=∞, because the completion test may not converge.

The cooling function we have used is $T_{i+1}=\alpha T_i$, where $0 \leq \alpha \leq 1$.

5 Results

To test our algorithm we have performed several experiments using different task sets. Our task sets have been determined as follows:

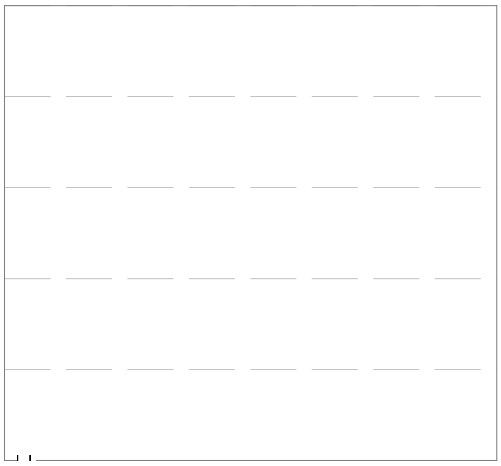
- The number of resources has been randomly chosen between min_R and max_R
- The number of tasks has been randomly chosen between min_t and max_t
- Tasks periods have been randomly chosen, between Tmin and Tmax
- The load of each task has been selected to accomplish with the desired requirements of total load in the system
- The number of end-to-end deadlines have been randomly chosen between min_D and max_D
- The number of tasks of each end-to-end deadline has been randomly chosen between 2 and the number of resources
- We have chosen two types of end-to-end deadlines: $Dee_i = T_i$ and $Dee_i > T_i$

We use our heuristic algorithm and the simulated annealing algorithm on the same task set looking for schedulability. We compare the performance of both algorithms in time and maximum average load of resources. To calculate the maximum average load of resources we gradually increment the workload of tasks.

In these firsts experiments (figures 3, 4, 5, 6), we have not considered pre-allocated task, and the end-to-end deadlines are $Dee_i=T_i$

outperforms simulated annealing in time by two orders of magnitude.

To check the influence of Tmax/Tmin in these experiments we have looked for schedulable solutions for task set where Tmax/Tmin=32 and 128 and no difference has been observed. See figure 5 and 6.



7 References

[1] Chao-Ju Hou and Kang G. Shin. “Allocation of periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems”, *IEEE Transactions on Computers*, Vol 46, No12, December 97.

[2] J.J. Gutiérrez García and Michael González Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems", *Proc 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara*, CA, pp. 124-132, 1995.

[3] K.W. Tindell, A.Burns and A.J. Wellings, "Allocating Real-Time Tasks. An NP-Hard Problem Made Easy", *Real-Time Systems Journal*, Vol. 4, No2, pp 145-166, May 1992

[4] M.F.Storch snd J.W.S. Liu. “Heuristic Algorithms for Periodic Job Assignment”, in *Proceedings of workshop on Parallel and Distributed Real Time Systems*, pp. 245-251, Apr. 1993

[5] A. Burchard, J. Liebeherr. Y. Oh and S. H. Son, "New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems. ", *IEEE Transactions in Computers*, Vol 44, No 12, pp 1429-1442. December 1995.

[6] Sergio Sáez, “Planificación en Sistemas Multiprocesadores de Tiempo Real”, *PhD tesis*, Universidad Politécnica de Valencia, March 2000.

[7] N.C.Audsley, A.Burns, M.F.Richardson and A.J.Wellings, “Hard Real-Time Scheduling: The Deadline Monotonic Approach”, *Procedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, USA, May 1991

[8] K.W. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", *Microprocessing & Microprogramming*, Vol 50, No 2-3, pp 117-134. April 1994.

[9] J.C. Palencia and M.Gonzalez Harbour, “Schedulability Analysis for tasks with Static and Dynamic Offset”, *IEEE Real Time System Symposium*, December 99.

[10] Joseph, M. and Pandya, P., "Finding Response Times in a Real-Time System", *British Computer Society Computer Journal*, 29(5): 390-395, Cambridge University Press, 1986.

[11] John P. Lehoczky. "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", IEEE *Real Time Systems Symposium*, 1990

[12] C.L.Liu and J.W.Layland, “Scheduling Algorithms for Multiprograming in a Hard Real-Time Environment”, *Journal of ACM* 20(1), pp. 46-61, 1973.

[13] Sheng-Tzong Cheng. ”Scheduling and Allocation in Multiprocessor Systems”, *PhD thesis*, University of Maryland, 1995.

[14] J.Y.T.Leung and J.Whitehead, “On The Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks”, *Performance Evaluation*, pp 237-250 Vol 2, Part 4, Dec 1982.