Heuristic for the Assignment of Tasks in a Real-Time Distributed Environment

Luis Alberto Gutiérrez Díaz de León[1], Carlos Alberto Franco Reboreda[1], Carlos A. Gutiérrez[2]

[1]Universidad de Guadalajara, Av. Juárez 976, Col. Centro, C.P. 44100, Guadalajara, Jal, México
[2]Universidad Panamericana Aguascalientes, Josemaría Escrivá de Balaguer 101, C.P. 20290, Aguascalientes, Ags., México
luis.gutierrez@redudg.udg.mx, carlos.franco@cuaad.udg.mx, cagutierrez@up.edu.mx

Abstract. This work consider the scheduling of periodic tasks or processes with real-time constraints in a distributed environment. Each task must be executed meeting deadlines, precedence relationships and resources constraints. The problem of scheduling tasks on a distributed environment can be viewed as a problem of assigning processes to the processors but keeping the schedulability in local environments. Thus, the problem can be divided in two phases: the first phase is assigning processes to processors and the second is to schedule assigned processes in each processor in the distributed environment. This paper focuses in the first phase. It introduces a heuristic mechanism for assigning processes in a distributed environment with real-time constraints. The heuristic mechanism has two alternative ways to perform: one takes into account the utilization factor and the other considers period. The performance of both alternatives proposed is evaluated via simulation on a high performance-computing platform.

Keywords: Real-time, scheduling, tasks, heuristic.

1 Introduction

Real-time systems are systems that must react to events in the environment within the confines of a stringent deadline. As a consequence, correctness of real-time systems does not exclusively depend on the accuracy of obtained results, but also the time these results are calculated. In hard real-time systems when there is a delay in the calculation of the results, it produces a decrease in the system performance with potentially catastrophic consequences. In hard real-time systems there are three kinds of constraints involving tasks: precedence, time and mutual exclusion on shared resources. Time constraints -known as deadlines- represent the time when a task must end its execution to prevent potential damage to the system. The tasks are classified depending on potential damage they could cause to the system in case they do not meet its deadline. For high risk of damage, the task is considered a critical task, otherwise it is non-critical. Tasks can -or not- be present periodically (in regular time

Received 23/01/11
Accepted 20/02/11
Final version 07/03/11

periods). Under this perspective, tasks can be classified in periodic and aperiodic. Periodic tasks are the most common and critical in real-time systems, but aperiodic tasks can also be critical or non-critical [5].

So, hard real-time systems must provide correct results (logically and within time constraints). In order to do this, they must consider and solve the problem of tasks constraints, which is known as a NP-complete problem. It means that it can not be assured that a practical solution exists in a reasonable time. However, approximate and quasi-optimal solutions are feasible and frequent enough to solve problems commonly found in practice. Distributed Hard Real-Time Systems increased its application domain to multimedia systems, virtual reality, process control, avionics, robotics, nuclear plants, quality of service, defense, military applications and so on, which would not be possible to look after outside from parallel and distributed processing environments [16].

In the distributed environment the complexity of the system increases proportionally to its application domain because it is necessary to increase the considerations (to a distributed environment) to keep the predictability level that is required in real-time. Besides, a new problem arise concerning the assignment of tasks or processes into processing entities keeping at all time the schedulability of the system, that is also a NP-complete problem [10]. However, as it is known, there are several techniques or ways to find solutions to NP-complete problems, in order to offer acceptable solutions even though they are not optimal. Regarding hard real-time distributed systems there has been development of several solutions with bounded models, less dynamic and with different scheduling criteria, but most of the proposals are for very restricted applications [4].

In this work it is proposed an algorithm for the beginning of the construction of an ordered execution plan of a set of periodic tasks with precedence relationships, time constraints and shared resource requests in a hard real-time distributed environment. This is the process assignment, which consists in the assignment of tasks or processes into processing entities that can execute those tasks or processes in such way that restrictions are met. The rest of the paper is organized as follows: description of terminology and presentation of the model, proposal presentation, and validation of the results via simulation environment.

2 Task Assignment in Distributed Environments

The problem of scheduling tasks in distributed environments or in multiple processors can be understood as the search for an execution plan that fulfill all restrictions related to tasks [3]. In this case, the search space could be represented with a tree where each node is an assignment of a task to a processor. Any path from the root to any leave of the tree is an execution plan, but not necessarily feasible, from the satisfaction of constraints perspective. Most of the algorithms perform the search trying to generate a complete path (from root to leave) that meets all restrictions. In case of several solutions in the tree, algorithms look for an optimal solution, trying to perform their search trough an heuristic function.

The search for generating a processor-task assignment can be classified in two ways: oriented to assignment and oriented to sequence. This truly depend on the semantics of edges in the search tree. When an algorithm selects a task and then tries to assign it to a processor, it is called an oriented to assignment algorithm. If the algorithm selects a processor and then tries to find a task to execute on it, the algorithm is oriented to sequence. Both algorithms traverse the tree in-depth searching for a feasible execution plan, are executed in an exponential time and produce identical results when exploring the tree in a single iteration [7].

When algorithms are static and there are no time constraints it is possible to explore the tree in a single iteration and find all possible solutions. When we consider dynamic algorithms in an environment where it is possible to accept new tasks during execution time, the search must be incremental, which increases complexity. Obtained results would be partial to specific intervals, and new tasks would be added among them.

Both options in systems with a single processor are identical, but there is a change in multiprocessor or distributed environment. In previous works it has been proved that oriented to sequence approach is not easily scalable because in each iteration it tries to select a task to execute it on the next processor. Sequence oriented schedulers are based on load balancing strategies instead of focusing on satisfy all time constraints [8]. In the other side, assignment oriented schedulers are always trying to execute the most urgent or critical task at the moment. The proposal presented in this paper is consistent with this approach, this is, oriented to assignment.

3 Terminology and System Model

Let **E** = { $E_1, E_2, .., E_n$ }, be a processing environment, where E_i is the i-th processing entity, which is comprised of a processing resource (processor) and a set R_i of resources assigned to entity E_i and are different to processing resources. So, E_i generates a local processing environment. In such a way, the distributed environment **Dist=E+T** is comprised of the set **E** of processing entities and the communication paths **T** that are in charge of connecting processing entities. The relationship established in **Dist** can be defined by a fully connected graph.

The communication cost between any two processing entities is considered constant and is defined by variable **k**. It is constant because k is equal to the longest communication time between two processing entities, from here $k = \max\{ T_1, T_2, .., T_m\}$. This allows to keep a predictable environment in execution and makes simpler the synchronization and schedulability analysis. **R** is the total resources in the distributed environment, from this **R** = $\{R_1, R_2, .., R_n\}$. Processing resources are not part of R, this is to provide more detail in the problem definition. Intersection between any two resource sets by processing entity is empty set. The distributed environment will have a set **P** of processes or tasks to be executed. The environment offers as a scheduling criteria the one defined by Earliest Deadline First [15]. The general set of processes is comprised of:

A set **t** □of n periodic preemptive tasks, where:

- Each task t_i Є t□□has an execution time C_i and a period T_i in which its instances are to be activated.
- Activation of periodic tasks can be done in t=0 to for purposes of schedulability analysis. Activation of aperiodic tasks is not necessarily in t = 0 for purposes of execution.
- Periodic tasks have a deadline for its execution, known as deadline D_i.
- Deadlines of periodic tasks can be different from period.
- Periodic tasks can be run causal relationship, known as precedence constraints. Causal relationships are resolved by a proposed scheme in [12].
- The precedence relationships are represented by directed rooted trees, where the destination nodes must be executed after the source nodes.
- Periodic tasks belonging to the same precedence tree have equal periods.
- Periodic tasks are independent only if they are in different trees.
- Periodic tasks can access shared system resources. In general and in order to simplify the analysis, it is assumed that they only access a single resource (cases where access to more than one resource is required are not left out and they can be modeled as different processes serialized by precedence constraints).

A set **J** of m aperiodic tasks, where:

- Are independent.
- Do not have a specified time for execution, but are given a deadline according to MDAS [11], in order to be handled by the scheduler under EDF.
- Aperiodic requests are serviced according how they (FCFS).
- Aperiodic tasks have an execution time, known when they arrive to the system.
- Aperiodic tasks have unknown arrival time.

4 Formal Definition of the Problem

Given a set of processes **P** = **J** U **t** with time constraints (C, T, D), precedence relationships and resource restrictions. And given a distributed environment **Dist** comprised by a set **E** of processing entities and communications paths **T**. In this context the scheduling problem is to allocate resources from **R** and start times for the execution of tasks, to tasks in **(t U J)**, in order to complete all tasks with the imposed constraints. This is, maintaining schedulability property [4]. The definition of the problem includes local scheduling because in this model processors are not in **R**.

To define the distributed scheduling problem, it is divided in two stages:

a) Allocation of responsibilities for processing with guarantees of schedulability (Process Allocation).

b) Generation of local schedule schemes.

For point b, in [11] and [12] was proposed a solution that generates a local scheduling environment, this solution is consistent with current needs. The fundamental problem in this work is the full assignment of all elements of **P** to all processing entities of **E**, in a way that preserves the schedulability for every element of **E**. With the assignment of processes to the processing entities in conjunction with local scheduling, it is generated the distributed scheduling environment. The proposal for assigning processes to the processing entities is not done dynamically because the foundation is not load balancing the elements of **E**, but to make an allocation of processes to maintain the schedulability in these elements [1] [14]. Therefore in the process of assigning processes is evaluated the schedulability of the processing entity before assigning each process. The allocation strategy applies only to **t** processes because the processes of **J** are events produced in each processing entity, and therefore they must execute in the processing entity with the same aperiodic tasks dispatching scheme proposed in [11].

5 Proposed Solution

The proposed allocation of the elements of **P** to **E** processing entities is performed in three steps and in three different ways of assigning processes. Actually, when the assignment is performed, the execution plans of the processes are generated as shown in figure 1. The steps for the process assignment proposed in this paper are: I Clusters generation, II Neighbors process allocation, and III Disappearance of islands.

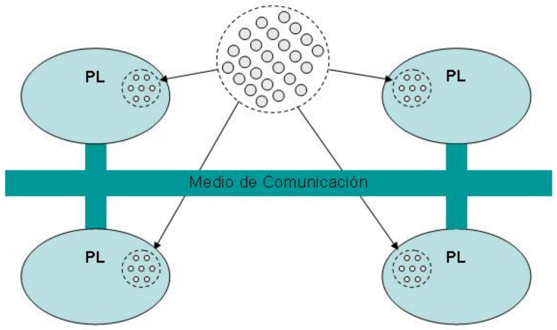

Fig. 1. Process allocation and production of local schedule plans.

5.1 Clusters Generation

With the set of processes in the initial form (fig. 2), the first step is to assign only those processes that have resource requirements from **R**.

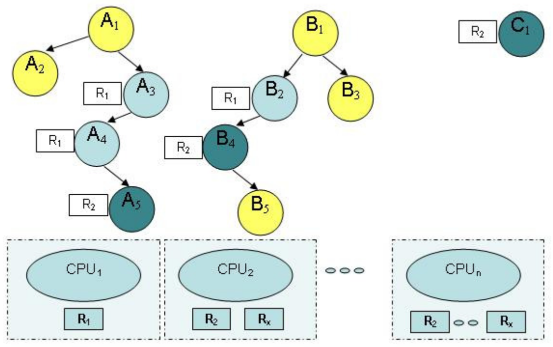

Fig. 2. Set of processes to be allocated.

Thus, any process t_i Є t, which requires a resource R_j has to be assigned to the processing entity E_k if and only if this entity contains the resource, as shown in figure 3. After the allocation of processes in this step, it is performed the schedulability analysis; if the set of processes is schedulable we can proceed to the next steps, otherwise we can not proceed with the assignment because **Dist** does not have enough resources to perform the whole set of processes meeting the restrictions imposed.

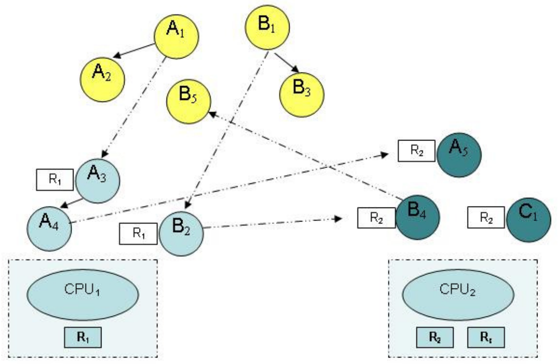

Fig. 3. Clusters generation.

5.2 Allocation of Neighbors

In this step the processes that do not have requirements from **R** are allocated. The processes that are still not allocated into processing entities may have causal relationships with processes already allocated to a processing entity of **E**. If a process that is going to be allocated has relationship wit one already allocated, the goal is that they both are allocated to the same processing entity to avoid delays in their communication due to synchronization signals. This step is known as allocation of neighbors.

To allocate processes in this step it is considered a heuristic function that leads the selection in the allocation of processes, considering reduce the communication costs (represented by constant k). However the heuristic function can be determined by two variations with the combination of factors that define the communication cost, among these factors are:

k: communication cost, o: the number of relationships that have a process when it is allocated with another process already allocated in the processing entity where it is being evaluated. (also known as occurrences), C: Execution time of the process in evaluation, T: Period of the process in evaluation.

So, the heuristic functions proposed and that are evaluated in this paper are described as follows:

$$f_1(k, o, T) = k * o / T \quad (1)$$

$$f_2(k, o, C, T) = k * o / (C/T) \quad (2)$$

These heuristics are intended to express the weight or cost that generate periodic tasks, and reduce it on their use. For this reason the first function express the cost directly related with the period or the frequency that each synchronizing task brings. The second function is a slight modification to involve the use of the processor that a task requires for its execution, involving by percentage of use the communication cost.

With these features the allocation of neighbors is conducted. In case we can not allocate a neighbor due to the lack of processing capacity, it will be evaluated and reallocated to the next processing entity where exists a relationship. In case there are no relationships with other element of **E**, then it is assigned to the processing entity with smaller utilization factor and where the allocation can keep schedulability conditions, this according to Earliest Deadline First Best Fit (EDF-BF) allocation criteria, defined in [13].

In the absence of processing entities that can withstand the process that is being assigned, then it is enabled a new processing entity and it is evaluated its allocation. In case of a new failure, it is determined the infeasibility to schedule the set of tasks in the system. This strategy allows controlling the number of processors that is actually required for system implementation.

One option might have been to enable processing entities and assign resources to processes as they were needed, but no matter how seductive this possibility may seem, there is the potential risk of obtaining a system design that does not correspond to reality by separating resources where they are really necessary.

For instance, consider the case of interaction with an actuator; the actuator must be physically in a place determined by the environment of the system. Seeking to change this assignment in theory can lead to produce unrealistic systems, which is beyond the scope of this paper.

5.3 Disappearance of Islands

In this step there may be present processes that still have not been allocated because they do not have relationship with processes with need of resources or processes that have been already assigned. So these processes remain isolated in the allocation until this step, generating what is known as process islands.

Once processes have been allocated in the previous two steps, we check the existence of islands.

In case of the existence of any island, the root process of the island is assigned to the processing entity with higher residual utilization factor and where the allocation keep the schedulability conditions, and we continue in the step of allocation of neighbors. This step is consistent with the Earliest Deadline First – Worst Fit (EDF-WF) allocation criteria, whose definition is found in [6]. The reason for using it is because root processes can have children that in the ideal case will be allocated in the same processing entity; this is to propose that precedences are local as far as possible to reduce communication costs. In case of using EDF-BF the possibility to generate remote precedences increases.

In the absence of processing entities that can withstand the process that is being assigned, then it is enabled a new processing entity and is evaluated for a possible allocation. In case of a new failure, it is determined the infeasibility to schedule the set of tasks in the system. In the absence of any island, then the allocation process has completed successfully and the next step is the preparation for execution, this is, the preset parameters in local schedulers (considering the phase of synchronization of phases in tasks addressed in the previous section).

The allocation criteria EDF-BF y EDF-WF were selected in this proposal considering their performance evaluated in the studies published in [17] [2].

6 Simulation Environment

The simulator implemented to evaluate the proposal of this work, it is really an execution environment that consists of a microkernel, which incorporates the properties of a minimal real-time system, it also includes a pitcher of execution modules that will be considered as the process to analyze their behavior, this last one was developed in this work. A MicroKernel is to add the Linux system kernel, a second kernel that acts as an interface layer between the hardware and the standard kernel. The microkernel layer controls the execution of real-time tasks and all of the conventional system. It considers the standard kernel like one more task without time constraints. The architectural design of the simulator, has a module to generate tasks as a running module. This running module and all generated by it, depend on the micro kernel, which generates real-time environment under EDF. Each of the experiments contained 100 periodic tasks whose periods were generated as a uniformly distributed variable from 10 to 1000 units of time. Each C_i/T_i was evaluated as a variable uniformly distributed in [0,1] and was standarized to provide U_P of each experiment whereas U_{Ptotal} vary from 4 to 7 units. From this, the experiment considered up to 10 processors. To fully characterize the environment, we generated a set of activities or events (aperiodic load) per processor to be varied in each graph from one point considered to $1-U_P$, where for each load were carried out 1000 simulations. The arrival times for aperiodic tasks were modeled using the exponential distribution, with interarrival average calculated to match the aperiodic load. The execution times of aperiodic tasks were modeled using a uniform distribution [10,100].

7 Results

The results were obtained by 1000 runs with a length of 50,000 units over time. The simulation is run to compare the performance of the heuristic function in the two alternatives defined in (1) and (2). The first one considers the period to minimize communication costs; the second considers the utilization factor. The scope where the alternatives were evaluated with the same sets of tasks (varying processing loads) was in terms of number of processors used, percentage of schedulability, communication cost and response time. The results are shown in Figures 4, 5, 6 y 7.

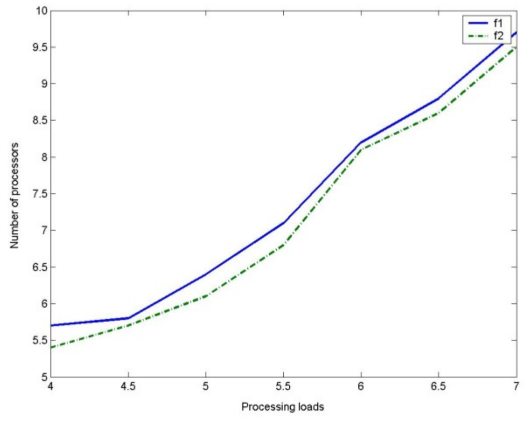

Fig. 4. Number of processors.

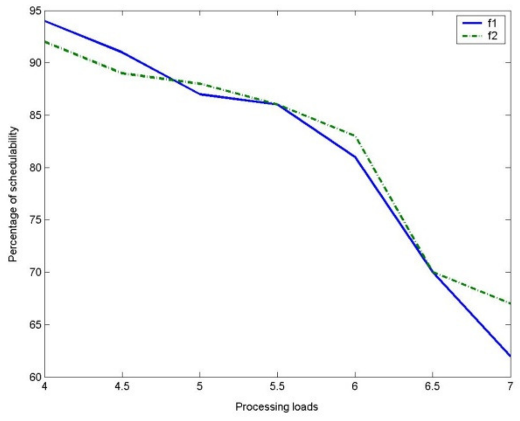

Fig. 5. Schedulability.

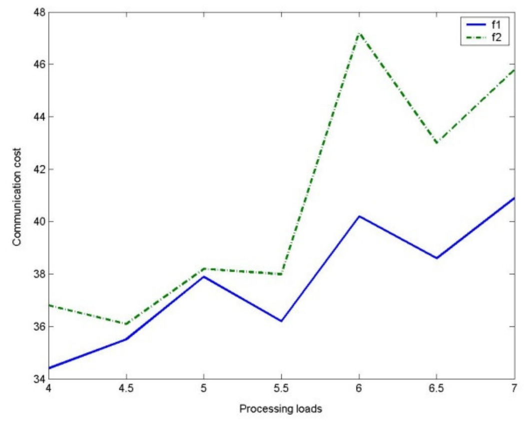

Fig. 6. Communication costs.

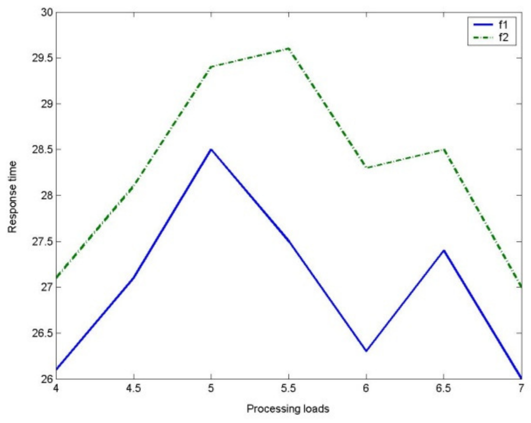

Fig. 7. Response time.

The results show that when considering the utilization factor in the heuristic function (2) it reduces the average number of processors to use, but in terms of schedulability does not generate any trend because the two heuristics work similarly (random) on a common combinatorial problem. With regard to reducing communication costs or reducing synchronization signals, the heuristic function that considers the timing of the processes (1) shows an improvement compared to the function (2) because the trend shows that there are fewer synchronization signals with function (1); this aspect correlates with response times, since these are also improved by reducing the number of synchronization signals.

8 Conclusions

In distributed real-time environments is essential the process allocation activity. This is because we now have to take care of the schedulability not only in a single processing entity but in a set of them communicated by message passing. In general the problem is classified as NP-complete. In this paper, we presented a scheme of allocation of processes to establish a real-time distributed environment trough local schedulers. The allocation scheme has a heuristic function to conduct the search assignment, trying to reduce communication costs. For the definition of the heuristic function we had two alternatives that were evaluated in a simulation environment. After analyzing the simulation results, we conclude that the function which considers the utilization factor for the allocation of processes, showed better results in terms of number of processors used, however with the function that considers only the frequency of processes there were less synchronization signals and thus response times are lower. Therefore, $f_1(k, o, T) = k * o / T$ gives better results by reducing response times due to reduction of communication costs. This work will continue with its validation through mathematical analysis of what we concluded and the evaluation of the performance of the entire distributed environment, considering the implementation of local schedulers.

9 Bibliography

[1] Atif, Y., & Hamidzadeh, B. (1998). A Scalable Scheduling Algorithm for Real-Time Distributed Systems. *Proceedings of the 18th International Conference on Distributed Computing Systems* , (págs. 352-359).

[2] Beitollahi, H., & Miremadi, S. (2005). Systems, Performance Evaluation of Fault-Tolerant Scheduling Algorithms in Real-Time Multiprocessor. En M. H. T. Fahringer (Ed.)., (pág. 738). Innsbruck, Austria.

[3] Burns, A., & Wellings, A. (2001). *Real-Time Systems and Programming Languages.* England: Addison Wesley .

[4] Buttazzo, G. C. (2002). *Hard Real-Time Computing Systems.* . United States of America: Kluwer Academic Publishers.

[5] Buttazzo, G., & Sensini, F. (1999). Optimal Deadline assignment for Scheduling Soft Aperiodic Tasks in hard real-time environments. *In IEEE Transactions On Computer* , 76.

[6] Coffman, E. G., Galambos, G., Martello, S., & Vigo, D. (1998). *Bin Packing Approximation Algorithms: Combinatorial Analysis.* Kluwer Academic Publishers.

[7] Gantman, A., Gui, P.-N., Lewis, J., & Rashid, F. (1998). *Scheduling Real-Time Tasks in Distributed Systems* . California : University of California, San Diego.

[8] Gantman, A., Guo, P., Lewis, J., & Rashid, F. (1998). *Scheduling Real-Time Tasks in Distributed Systems: A Survey.* Recuperado el 20 de enero de 2007, de Internet Draft,

www.cs.ucsd.edu/classes/fa98/sce221/OSSurveF98/index.htm
www.cs.ucsd.edu/classes/fa98/sce221/OSSurveF98/index.htm

[9] Garey, M., & Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company.

[10] Graham, R. (1969). Bounds on multiprocessing timing anomalies. *SIAM J. Appli. Math.* , 416-429.

[11] Gutiérrez, L., Franco, C., Jacinto, R., & Gutiérrez, C. (2006). Minimizing the Response Times of Aperiodic Tasks in Hard Real-time. *Electronics, Robotics and Automotive Mechanics Conference* (págs. 268-273). Cuernavaca, Moreles: IEEE Computer Society.

[12] Gutiérrez, L., Jacinto, R., & Rodríguez, F. (2002). Algoritmo para la creación de un plan ordenado de ejecución de tareas periódicas con relaciones de precedencia bajo EDF. *Memorias del Congreso Internacional de Computación 2002* (pág. 240). México, D.F.: CIC.

[13] Johnson, M. R. (1979). *computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman.

[14] Khemka, A., & Shyamasundar, K. (1997). An Optimal Multiprocessor Real-Time Scheduling Algorithm. *Journal of Parallel and Distributed Computing, vol. 43* , 37-45.

[15] Liu, C., & Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery* , 139.

[16] Masters, M. (1998). Challenges in designing future command and control systems. *Workshop on Parallel and Distributed Real-Time systems.*

[17] Pereira Zapata, O. U., & Mejia Alvarez, P. (2002). EDF and RM Multiprocessor Scheduling Algorithms: Survey and Performance Evaluation. *Report No. CINVESTAV-CS-RTG-02. CINVESTAV-IPN, Sección de Computación.*