# A Load Balancing Task Allocation Scheme in a Hard Real Time System

*Jean Louis LANET ELECMA - SNECMA*

**Abstract** : We address the problem of allocating real-time tasks subject to precedence constraints in a distributed system. The use of a list algorithm is a good trade-off between the complexity of the algorithm and the quality of the solution. The optimisation criterion is the load balancing which is linked with the fault tolerance requirements. Such a criterion guarantees that every processor will spend the same amount of time in performing self tests. The application is described by an acyclic graph giving the precedence constraints. To ensure that no deadline will be missed we have to build and check the whole sequence. Allocating and scheduling are handled together in a static approach.

**Keywords** : Distributed System, Real Time, Load Balancing, List Scheduling.

## 1. Introduction

The control of a turbo engine is a critical and complex task. During some phases, the engine reaches limits (pressure, temperature...) and the control computer has to meet all the real time constraints. Currently control computers are based on a dual lane architecture. Such an architecture is well adapted for recovering hardware failures but is unable to handle design errors and need self tests to locate an error. The use of a distributed architecture can be a solution to improve modularity, reliability and performances. But new difficulties will arise like checking the schedulability of the sequence before run time. This phase is often done off-line, and a set of schedulable sequences is stored into the memory. Due to a limited memory space, it is not any more possible to store all the configurations for every application mode and every hardware architecture. The schedulability checking must be done before run time, according to the current state of the system and the current operational mode. The schedulability test is performed during the fault treatment phase, which is scheduled within the idle times of the processor.

Without any fault, the processor idle time is used to perform hardware testing. All the processors must spend the same amount of time on the test during a period equal to the least common multiple period of the tasks. We have to balance the idle time among the sites, which is equivalent to balance the load. There are a lot of algorithms able to solve an allocation problem. The originality of our algorithm is to give a global solution in terms of allocation (both processors and network), scheduling and checking.

A number of methods has been developed to solve this problem, such as integer programming methods, graph theoretic methods, branch and bound search methods and numerous heuristic methods. The search for an exact solution is known to be a NP-hard problem. When dealing with hard real time applications, it is no longer possible to split the allocation and scheduling problems. Most of the solutions deals with the minimisation of a global cost including the communication and calculation costs. They are based upon the assumption that the shortest spanning time will lead to a valid sequence, which is not always true. The approaches proposed by Ramamritham [4] and Tindell *et al.* [5] are interesting because they consider the problems of allocation, scheduling and bus access in a global manner. Tindell uses a simulated annealing method to solve the problem but there are no precedence constraints among the tasks. Ramamritham uses a set of simple heuristics to solve the problem. In his architecture, the network access protocol is based on a TDMA scheme. There are precedence relationships between the tasks. He starts with the construction of the comprehensive graph. He makes a clustering phase to aggregate tasks with a high communication ratio and uses a list algorithm where the choice of the next task to be allocated is based on the smallest latest start time among the ready list. He shows that backtracking has little impact on the solution.

## 2. Characteristic of our system

The distributed system consists of a number of nodes, with a set of resources attached to each site : memory, local observation unit, network access circuit. They communicate through a point-to-point network employing a TDMA protocol providing a predictable communication time. A complete description of the system can be found in [1]. The application is represented by an acyclic graph. We use a static table driven scheduling which is motivated by the fact that resources needed to meet the deadlines of safety critical tasks must be preallocated in order to be guaranteed a priori. There are periodic and aperiodic tasks in the system. The aperiodic tasks are used for the fault treatment phase; they are dynamically scheduled and not taken into account within the feasible schedule. The feasible schedule must be computed for the least common multiple (LCM) of these periods.

## 3. The LDL algorithm

We have chosen a list algorithm which is a good trade-off between the complexity and the performance of the algorithm. Our algorithm operates by allocating tasks to nodes one by one without reconsidering previous decisions. As a result, the order in which the tasks are considered is the critical point of the algorithm. If the temporal characteristics of the tasks are

the only ones to be taken into account, a wrong choice may be done leading to an unfeasible solution. To avoid such a situation, the algorithm must use another parameter : the freedom degree. We called our algorithm LDL.

The first step consists in the construction of the complete task graph including all the redundant tasks. Using the critical path approach, we can derive the deadline of the tasks graph $d_t$ (activation of the actuators) into a set of earlier deadline $d_i$ attached to each tasks and the ready time (after sampling the sensors) $r_t$ into release time $r_i$. The second step is the allocation scheme. Two metrics are used for choosing the next task. The first one, the laxity $L_i$ of a task, will lead to a feasible solution. The second, the freedom degree $FD_i$ of a task will avoid an allocation leading to an incompatibility with the future constraints among the tasks. The freedom degree has been introduced by [3] in a data-base problem. It can be defined as the number of nodes to which the tasks can be allocated. It measures the flexibility of a task. If a task has a high freedom degree, its allocation can be postponed. The freedom degree must be updated throughout the allocation process.

A task becomes available only when all its predecessors have completed execution (the messages are available) and when its start time is reached. At a given time, only a subset of tasks is eligible for allocation. The ready list is ordered according to the increasing $L_i$ of the tasks in the list. If there is a tie, the task with the least freedom degree is chosen. Such a scheme is interesting if there are few constraints. By the time the task is ready, some nodes remain idle while others are executing previously scheduled tasks. According to the freedom degree of the selected task, only a subset of nodes should be eligible for the mapping. The load balancing can be achieved at this time. The list of the available nodes is ordered according to their loads. The task is then allocated to the first node in the list. If the mapping is valid, e.g. the deadline is met, the search path is extended. The load of the chosen node is updated with the execution time of the task. Otherwise the next node on the list is checked. If no processor matches, the algorithm stops and reaches an unsuccess state.

Having a non pre-emptive scheduling policy allows us to compute the completion time of the task. We search for the next event to occur, e.g. a completion time, an arrival of a message, or a task start time. For a completion time, we have to search for the next available time slot on the network giving us the arrival time on the other nodes. An arrival date updates the ready list of a task if the message was issued by the last predecessor of a task.

Several criteria can be used to select a task among the ready list. The ability of an algorithm to find a good solution is often linked with the

characteristics of the tasks. If the temporal constraints are sparse (i.e. high laxity) a basic algorithm like Heavy Task First is often enough. If the precedence constraints are tight, only a few heuristics are able to find a solution. In [2] we compared our algorithm with some other list algorithms with different selection criteria in a nominal mode without any fault. We showed that in this case with a high freedom degree for each task, we had not the best success ratio under heavy loads but the unbalance remained low under any load. The algorithm LDL has been improved (LDL*) by splitting the allocation scheme in two phases. The tasks have been split into two sets : independent tasks and tasks linked by precedence relationship. In the first phase, LDL* works like LDL with the dependent tasks and in a second phase, uses a Bin Packing algorithm (Best Fit) for allocating the set of independent tasks. The success ratio increases while the unbalance remains unchanged.

## 4. Conclusions

We have addressed the problem of allocating real-time tasks subject to precedence constraints in a distributed system. Our method provides both allocating and scheduling functions in a global manner. Under heavy load only the heuristics using temporal information give good results in term of success ratio. We have developed a heuristic which is a good compromise between complexity $O(n^2)$ and the quality of the solution for this graph. It is not possible to generalise these results for any random task graph. For a new application, it is necessary to verify which algorithm gives the best result. We offer more a methodology to solve the allocation problem in real time environment than a generic solution, which we believe doesn't exist.

## References


[1] **LANET**, *Placement statique de tâches dans un calculateur réparti de régulation moteur.* Thesis, University of Paris 6, 1995.

[2] **LANET**, *Task Allocation in a Hard Real Time Distributed System.* Real Time Systems'95, pp. 244-252, Sklarska Poreba, Poland, Sept.-95.

[3] **LU, CAREY**, *Load Balanced Task Allocation in Locally Distributed Computer Systems.* Computer Science Technical Report #633, University of Wisconsin, Madison, Feb.-86.

[4] **RAMAMRITHAM**, *Allocation and Scheduling of Complex Periodic Tasks.* IEEE 10th International Conference on Distributed Computing Systems, Jan.-90, pp. 108-115.

[5] **TINDELL, BURNS, WELLINGS,** *Allocating Hard Real Time Tasks : an NP-Hard Problem Made Easy.* The Journal of Real Time Systems, N°4, 1992, pp. 145-165.