AN HEURISTIC APPROACH TO ALLOCATING AND SCHEDULING NP-HARD REAL-TIME SYSTEMS

J. Santos* and E. Ferro
Dep. Ing. Eléctrica. Universidad Nacional del Sur, Bahía Blanca, Argentina

Abstract: An heuristic approach to the problem of scheduling a set of preemptible real-time tasks to be executed in a set of heterogeneous processors communicated through an interprocessor network is presented. The problem is NP-hard. Results obtained are compared to those obtained by simulated annealing.

Keywords: Heuristic, allocating, scheduling, NP-hard, real time systems.

INTRODUCTION

Real-time scheduling theory is one of the areas in which efforts must be concentrated in order to develop a science of large scale real-time systems [1].

Conventional real-time systems are mainly concerned with tasks that do not change their fundamental timing parameters during execution (static tasks). They may be sporadic (e.g. alarms, aperiodic but with a definite deadline) or periodic (with definite period and deadline). On top of the real-time load, non real-time aperiodic tasks may be also present.

The deadline and maximum repetiton rate of each sporadic task as well as the period, deadline and execution time of each periodic task must be known in advance in order to make possible the pre-run scheduling computations guaranteeing that no deadline is missed. This is particularly important in the case of hard real-time systems in which missing one deadline may have catastrophic effects including the loss of human lives. Examples of such systems are airplane avionics, smart robots, space vehicles and many industrial applications [2].

With all the information about timing constraints, the system schedulability may be tested beforehand, with the added advantage of reducing the use of precious resources required for scheduling and context switching during actual run-time [3, 4].

Some of the main earlier and recent contributions cover the following problems: a) Scheduling a set of independent preemptible tasks on a single processor without [5,6,7,8] or with [9] restricted granularity. b) Scheduling preemptible tasks under time and resource constraints [10]. c) Scheduling a set of non preemptible tasks with release-times, deadlines, precedence and exclusion relations on a set of identical processors [4]. d) Scheduling a set of preemptible tasks on a set of heterogeneous processors with allocation, processing, memory and communications constraints [11].
The computational complexity of the first problem is \( O(nT_m) \), where \( n \) denotes the number of processors and \( T_m \) the maximum task period in the system [9]. The last problem, on the other hand, is NP-hard. It has been treated by heuristic methods [10] and by the use of the global optimization technique known as simulated annealing [11].

In this paper an heuristic method is proposed to attack the problem. The results obtained are compared with those obtained by simulated annealing using the same example presented in [11].

THE ASSIGNMENT PROBLEM

The system is a set of preemptible tasks to be executed in a set of processors in such a way that each task will be completely executed in one processor. It is said then that the task is allocated to that processor. A set of allocations distributing all the tasks on all or part of the processors is called an assignment. If there are no tasks/processors allocation constraints, the number of possible different assignments is \( m^n \), which makes it NP-hard. For example, 43 tasks on 8 processors (the numbers used in the example of [11]) produce different assignments in the order of \( 10^{13} \).

The assignement problem to be treated here may be defined in the following terms:

1. There are \( m \) different real-time preemptible tasks to be assigned to \( n \) processors, each one with a definite memory capacity.

2. The tasks are periodic and each one has a definite period, deadline and execution-time. The system is hard real-time and therefore no deadline may be missed.

3. In order to be executed, each task requires a known amount of memory.

4. The tasks also have communications requirements: Certain pairs of tasks must interchange definite amounts of information. Pairs of tasks allocated to different processors use the interprocessor communication network. Pairs of tasks allocated to the same processor communicate by some internal mechanism and do not load the network.

5. There are some tasks/processors allocation constraints:

   5.1. Some tasks must be executed in a certain definite processor and therefore there is a unique feasible allocation.

   5.2. Because of fault tolerance the processing of some tasks must be duplicated. Obviously, the pair (original and replica) must not be executed in the same processor.
After several tasks have been allocated to a certain processor, the incorporation of another task is feasible if it passes the placement test, the memory test, the schedulability test and the communications test on that processor.

The placement test consists in verifying that the task is not preallocated to another processor or that it is not a replica of an original task already allocated to that processor. The memory test consists in verifying that the memory required by the candidate task is equal to or less than the residual memory left free by tasks already allocated to the processor. The schedulability test consists in verifying, for example by using the empty-slot method [9], that the set obtained by adding the new task to the set of tasks already allocated to that processor is executable in real-time. The communication test consists in verifying that by allocating the task to that processor, the transfer capacity and time constraints of the network are not surpassed.

The system will be said to be absolutely non-schedulable if none of the $m^n$ possible assignments meets all the preallocation, memory, scheduling and communication requirements within the specified time constraints.

The sets of tasks and processors ordered by some relation will be called the tasks-stack and the processor-stack, respectively.

The set of pairs of tasks that interchange information ordered by some relation will be called the communication-stack.

THE HEURISTIC APPROACH

Step I. The task-stack is assembled by randomly ordering the tasks.

Step II. The processor-stack is assembled by randomly ordering the processors.

Step III. The communication-stack is assembled ordering the pairs by monotonically decreasing times of communication.

Step IV. The first processor in the processor-stack is selected.

Step V. The memory and schedulability tests are performed on the subset of tasks preallocated to this processor. If some of the tasks fails any of the tests, the system is absolutely non-schedulable.

Step VI. The communication stack is polled top-down until finding the first pair containing a task allocated to this processor. The companion task is tested for placement, memory and schedulability; if it passes the three tests, it is allocated to this
processor. Each time a task is allocated to this processor, the top-down polling is repeated and candidate companion tasks tested in the same way.

Steps V to VI are performed on the rest of the processors. When allocating a task that communicates through the network, the communication test is performed. If it fails and the communication is between preallocated tasks, the system is absolutely non-schedulable. If it fails but only one or none of the tasks is preallocated, the system is non-schedulable by applying this method to the processor stack under consideration. After dealing with the last processor, the tasks-stack has been depopulated and only a subset of free (non allocated) tasks remains.

**Step VII.** The placement, schedulability, memory and communication tests are performed on the first free task for allocation to the first, second, ..., processor until finding one to which it can be allocated. The tests are made easier and faster by the fact that, from past calculations, data about empty slots and residual memory in each processor as well as total load in the communications network are known.

Step VII is repeated for second, third, ..., free tasks until finding: a) that the whole set of tasks has been allocated or b) that the system is absolutely non-schedulable or c) that the system is non-schedulable for that particular processor stack. If a) is true, one solution of the assignment problem has been found. If b) is true, the assignment problem has no solution. If c) is true, Step VIII follows.

**Step VIII.** A permutation of the processor stack is generated and Steps IV to VII are performed on it. If all possible n! permutations have been tried with no success but the system has not been found to be absolutely non-schedulable, the system is deemed to be non-schedulable by this method.

![Fig. 1](image-url)

It must be noted that although all possible permutations of the processor stack are tried, the tasks-stack is invariant. In Fig. 1 the set A of all possible tasks/processors assignements (cardinality n!), the subset S of the assignements leading to a schedulable system and the subset P of the n! assignements produced by the proposed method are represented. Obviously, S and S\(\cap\)P may be empty. If S is empty, the system is absolutely non-schedulable. If S\(\cap\)P is empty but S is not, the method has been unable to find a solution although it exists and would have been found with the appropriate tasks-stack.
If the system is found to be non-schedulable although not absolutely, the method gives configurations that can be used as starting configurations to try the simulated annealing technique. In that way, all the preliminary trials, necessary for instance to separate an original and a replica that have been randomly put in the same processor, are saved.

Example: Tables I and II show a system of 8 processors and 43 real-time periodic tasks used as illustrative example in [11]. Table I lists the memory capacity of each processor. In Table II the period, worst case execution-time, memory requirement, communication load and, if it exists, the preallocated processor for each task are shown. 50/1 and 50/2 in the first line mean that task 0, while executing, must send messages of 50 bytes to task 1 and of 50 bytes to task 2.

### Table I

<table>
<thead>
<tr>
<th>Processor</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10,000</td>
</tr>
<tr>
<td>1</td>
<td>10,000</td>
</tr>
<tr>
<td>2</td>
<td>10,000</td>
</tr>
<tr>
<td>3</td>
<td>12,000</td>
</tr>
<tr>
<td>4</td>
<td>7,000</td>
</tr>
<tr>
<td>5</td>
<td>7,000</td>
</tr>
<tr>
<td>6</td>
<td>12,000</td>
</tr>
<tr>
<td>7</td>
<td>10,000</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>WCE</th>
<th>Memory</th>
<th>Messages</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>60</td>
<td>4</td>
<td>3000</td>
<td>50/1, 50/2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>60</td>
<td>4</td>
<td>1500</td>
<td>60/3, 70/4, 30/5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>2</td>
<td>1200</td>
<td>20/3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>2</td>
<td>1700</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>2</td>
<td>3000</td>
<td>60/6</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>4</td>
<td>3000</td>
<td>80/6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>6</td>
<td>1100</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>2</td>
<td>500</td>
<td>40/8</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>35</td>
<td>2</td>
<td>700</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>35</td>
<td>8</td>
<td>900</td>
<td>90/11</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>35</td>
<td>14</td>
<td>2200</td>
<td>250/11</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>35</td>
<td>4</td>
<td>1000</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td>2</td>
<td>1000</td>
<td>150/13, 150/14</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>2</td>
<td>1500</td>
<td>50/15</td>
<td></td>
</tr>
</tbody>
</table>

743
The simulated annealing method starts with an arbitrary energy and temperature associated to a random assignment in the system. The energy is a measure of the unschedulability and the high temperatures are associated to high energies. Jumps are performed from one assignment to other trying to lower the energy.

Although the computing equipment of [11] is not described and the programming language and computation time are not stated, jumps from one energy point to another are performed in the order of thousands for each temperature until reaching one solution. Table III shows the solution found, with tasks allocated to processors and the processor and memory utilization factors. The heuristic method proposed here produced the first solution on the first trial. It is shown in Table IV.
In the schedulable assignment found with the heuristics here proposed, the average processor utilization was 56 % and the average memory utilization was 78.4 %. The assignment obtained by simulated annealing as presented in Table III produced 56 % and 73 %, respectively.

As can be seen the clusterings of communicating tasks are similar in both cases. With a 386 PC, 40 MHz, the method, programmed in Visual Basic, took about 3 seconds to find the solution. To test all possible 40,320 permutations it took about 33 hours. 38,304 permutations produced schedulable assignments although only 110 of them were different. For instance, the solution of Table IV was also produced by the next generated permutation in which processors were ordered \{1,2,3,4,5,6,8,7\}. Curiously, the solution found by simulated annealing was none of the 110 found by the heuristic method.
References


4. J. Xu, "Multiprocessor scheduling of processes with release times, deadlines, precedence and exclusion relations", IEEE TSE, 19, 2, Feb'93, pp. 139-154.


