Schedulability Checking in Real-Time Systems using Neural Networks

Renzo Davoli
Fabio Tamburini
Luigi-Alberto Giachini
Franca Fiumana
University of Bologna - Italy

In distributed Hard-Real-Time systems the correctness of a computation strictly depends on the results and on the computing time to produce them. We introduce a mathematical model to check the schedulability of a set of parallel programs represented through Data Flow precedence graphs. Schedulability checking involves a mapping subproblem, which is NP-complete. We transform the schedulability problem into an optimization one and present a solution based on Stochastic Neural Networks.

1 INTRODUCTION

One of the principal aims of an Operating System (OS) for general purpose computing is to minimize the program average-response times. In Real-Time (RT) systems each program has its own time requirements and the main goal of the OS is to fulfill all the timing constraints.

In this paper we focus our attention on the class of Hard-Real-Time (HRT) systems, those which may cause catastrophic effects in case of a missed timing deadline. In HRT program tailored systems the scheduler clearly plays a major role: the RT correctness of the system (i.e. the ability to be consistent with the program timing constraints) follows from good scheduling specifications [1],[10],[12]. RT schedulers have been classified [11] with respect to several features or characteristics depending on the method of operation and according to the types of programs they can handle. The processing environment we assume is multiprocessor and loosely coupled;

[0]Correspondence and request for reprints should be sent to Renzo Davoli, Dept. of Computer Science, Piazza di Porta S.Donato 5, I-40127, Bologna, Italy. E-mail: renzo@cs.unibo.it

a distributed system is a well-known example of this sort of architecture. We will show how the schedulability problem can be expressed in terms of function minimization: Neural Networks give the opportunity to perform the minimum search in situations where classical Newton (or quasi-Newton) methods fail due to the presence of several local minima. At the end of the paper we present an experimental result section. Some instances and combinations of classical programs are used to test the schedulability algorithm in various ways.

2 PROBLEM STATEMENT

The schedulability problem we are addressing, if applied to independent programs, is clearly an instance of the N-knapsack problem: the solution is a mapping function which assigns each program to a processor such that no processor is overloaded. Each program contributes to its assigned processor load in terms of the ratio between its computational cost and requested response time. If there is no overloaded processor, the use of the preemptive scheduling policy Earliest Deadline First guarantees that all the programs will meet their deadlines [7].

In the model we propose, programs are not atomic entities: they consist of nodes that may be executed concurrently (hopefully in parallel by different processors). The data communications and precedence relations between the nodes of a program are expressed by the deterministic Data Flow (DF) paradigm. In the DF paradigm, programs are represented by acyclic graphs whose arcs embody both data communication and timing precedence relationships and whose nodes are procedures (having functional-stateless behavior). This paradigm makes all the sequential and concurrent parts of the problem explicit, allowing the construction of RT schedules on the basis of the whole-program timing specification and the computational cost of its nodes.

Checking the schedulability of a set of DF programs means not only providing a suitable mapping function but also evaluating how to distribute the response time of each program among its nodes. Changing the time assignment of a node affects the processor load for its execution: therefore, even if the mapping had been fixed a priori, a suitable time assignment would have to be computed.

Our method applies only to a specific set of graphs (SG graph) with precise characteristics. Programs are obtained using two operators recursively: node (or SG graph) concatenation and node (or SG graph) scattering with successive compulsory gathering. The principal advantage of this restriction is the better minimizing or totally elimination of interference between the different concurrent components. In addition, we elaborated a method for treating complete meshes through our operators.

We regard mapping as a function having a logical set of nodes as its domain and a physical set of processors as its co-domain. This is a search problem on all subsets of the set of nodes.

3 A FORMAL VIEW

In Appendix B we introduce a language describing the graph structure of SG programs. It is based on two primary operators: concurrent execution and sequence. This formal definition embodies all previously defined DF program graphs. Each program has a string S associated with its structure, in which each node appears once. Appendix B shows some examples of DF graphs and the corresponding strings for the problems analyzed in this paper.

Our schedulability problem can be described as a search (i) to build a mapping function between all nodes and the set of available processors and (ii) to assign an execution time to each node in the system, such that:

$\otimes$

1. Load specifications are positive.
2. All programs complete their jobs within their respective deadlines.
3. No processor is overloaded.
4. Each node is assigned exactly to one processor.

Without losing generality, all nodes in the system are named $n_1, ..., n_{NNODE}$ with computational load $c_1, ..., c_{NNODE}$. c_k represents the time for node n_k to be executed on a single dedicated processor, regarding its algorithm's worst case. System processors are numbered as $P_1, ..., P_{NPROC}$. The DF programs are named $J_1, ..., J_{NPGMS}$: each program J_i exhibits a related string S_i which respects the grammar defined above and a deadline D_i. Each activation of J_i has to be completed within D_i time units; moreover, D_i is also the minimum time between two sequential activations of J_i. To complete the necessary definition set, we have to introduce the notion of restriction. Having a string S satisfying the grammar G, the restriction of S to a subset $T = \{n_1, ...n_{NSUB}\}$, $NSUB \leq NNODE$, is the result of the substitution of all node names not belonging to T by ϵ, then reduced using the rules:

$$\alpha\ '|'\ \epsilon \rightarrow \alpha, \quad \epsilon\ '|'\ \alpha \rightarrow \alpha, \quad \alpha\epsilon \rightarrow \alpha, \quad \epsilon\alpha \rightarrow \alpha, \quad \epsilon\ '|'\ \epsilon \rightarrow \epsilon, \quad \epsilon\epsilon \rightarrow \epsilon, \quad '('\ \epsilon\ ')' \rightarrow \epsilon.$$

For example the restriction of $a(b|c)(d|ef)g$ to the set $T = \{a, b, d, e, g\}$ is $ab(d|e)g$. This operator is used to make explicit, which all the nodes of a program that are mapped on the same processor are concurrent and which are in a sequential chain and therefore mutually exclusive.

The goal ($\otimes$) is to find (if any) a vector $\overline{\lambda}$ of load-assignment values $\lambda_1, ..., \lambda_{NNODE}$ and a mapping vector $\overline{\mu} : \mu_1, ..., \mu_{NNODE} \in [1, ..., NPROC]$ that binds each node to a specific processor, such that:

$\odot$

$$\begin{cases} \lambda_k > 0 & \forall k = 1, ..., NNODE \\ Time(\overline{\lambda}, S_i) \leq D_i & \forall i = 1, ..., NPGMS \\ \sum_{i=1}^{NPGMS} Load(\overline{\lambda}, rmap(\overline{\mu}, S_i, P_j)) \leq 1 & \forall j = 1, ..., NPROC \end{cases}$$

The $rmap(\overline{\mu}, S, P_j)$ function is the restriction of S to the set $\{n_k \mid \mu_k = j\}$; it selects the program substructure consisting of the nodes that are mapped on the same processor P_j. The $Time$ function computes the maximum execution time for program J_i given a vector $\overline{\lambda}$ of node-load assignments: each node deadline is obtained as the ratio c_k/λ_k. The $Load$ function computes the load for processor P_j executing the nodes of program J_i mapped on it. It adds the concurrent components and takes the maximum value of all the sequential ones. Formal definitions of $Time$ and $Load$ functions can be found in appendix A.

The approach we propose is to solve $\odot$ as an optimization problem. We present a solution using Stochastic Neural Networks (SNN) with Adaptive Simulated Annealing.

4 STOCHASTIC NEURAL NETWORKS

NN is a widely used method to cope with NP optimization problems. There are several works that use this approach to deal with classical NP-complete problems, such as TSP [3], graph partitioning for chip design [4] and N-knapsack [5].

Making the neuron dynamics behave stochastically [2] [8], an implicit concept of temperature is introduced into the SNN dynamics function, so an appropriate annealing method must be chosen to decrease the temperature properly. Ingber [6] and Rosen [9] have introduced an efficient method called Adaptive Simulated Annealing (ASA).

4.1 Network structure

In the general formulation ($\odot$) we have used two kinds of unknowns:

- the $\overline{\lambda}$ vector containing the processr load assigned to each node,
- the $\overline{\mu}$ mapping vector between nodes and processors.

We have to represent these data as neuron outputs to build a proper SNN; we use continuous valued neuron units with output ranging from 0 to 1. Then we define a net composed of $2 \times NNODE$ neurons. The first $NNODE$ neurons represent the processor ratios assigned to each node (the $\overline{\lambda}$ vector), where 0 means no work and 1 means maximum load. The second $NNODE$ neurons represent the mapping function (the $\overline{\mu}$ vector). Given the above definitions of problem unknowns, we can derive the energy function to define our problem. Calling $\overline{\eta}$ the neuron output vector, $\overline{\eta} = \overline{\lambda} \cdot \overline{\mu}$ (where $\cdot$ means concatenation of vectors), the energy function can be written as:

$$E(\overline{\eta}) = E(\overline{\lambda} \cdot \overline{\mu}) = \alpha \sum_{i=1}^{NPGMS} \Phi(Time(\overline{\lambda}, i)) + \beta \sum_{j=1}^{NPROC} \Phi\left(\left(\sum_{i=1}^{NPGMS} Load(\overline{\lambda}, rmap(\overline{\mu}, S_i, P_j)) \right) - 1 \right)$$

where:

- $\Phi(z) = \begin{cases} 0 & for\ z \leq 0 \\ e^z - 1 & for\ z > 0 \end{cases}$
- α, β are constants used to balance the strength of constraints.
- *Time* and *Load* are the function previously defined.

The function E has its minimum when all the constraints of the general problem $\odot$ are satisfied. (Note that the first constraint in $\otimes$ is implicitly met by our unknown definitions, while the fourth is fulfilled by the definition of the object *map* as a functional relationship).

4.2 Annealing

Adaptive simulated annealing [6] is a global optimization algorithm. The result of the algorithm is the discovery of the best global fit of a non-linear, non-convex cost-function over a δ-dimensional space. ASA is an algorithm designed to search for optimal function value in a constrained state space. New states $\overline{\eta}$ are generated using the random variable y. y is derived from a uniform distribution $u \in U[0,1]$ using the function $y = sgn(u - 1/2)\ T_k[(1 + 1/T_k)^{|2u-1|} - 1]$, where T_k is the current temperature. New η_i are then derived using the relation $\Delta\eta_i = y(B_i - A_i)$, where A_i and B_i are the limits of the i-th dimensional space and newly generated values η_i are constrained by $\eta_i \in [A_i, B_i]$. Deriving new parameters according to this method leads to an exponential temperature scheduling, in particular:

$$T_k = T_0\ \exp(-\gamma\ k^{1/\delta})$$

where T_0 is the starting temperature and γ is a user-defined parameter to help system tuning. This annealing schedule ensures that a global minimum of energy function can statistically be obtained (proven in [6]). Newly generated states $\overline{\eta}$ are subjected to an acceptance probability function $h(\Delta E, k) = 1/(1 + \exp(\Delta E/T_k))$. In our scheduling problem η_i represents each single neuron, so the parameter space has dimension $\delta = 2 \times NNODE$, and each variable is bounded into the interval $[0,1]$, so that $A_i = 0$ and $B_i = 1,\ i = 1, ..., \delta$. At each step a neuron η_i is randomly chosen, a new value $\Delta\eta_i$ is computed, according to the previous rules, and accepted using the probability function h.

If the dimension of the sampled space becomes too high, the convergence process may be prohibitively slow and one cannot commit resources to properly sample the search space ergodically. Therefore Ingber introduces a new parameter, called quenching factor Q to speed up annealing. Introducing this new factor we lose the guarantee of finding the solution given in the ASA proof, but, for practical purposes, tuning the system using Q and γ parameters is quite easy and gives good results. Using quenching means modifying the annealing schedule as follows:

$$T_k = T_0\ \exp(-\gamma\ k^{Q/\delta}).$$

n_k	c_k	λ_k	n_k	c_k	λ_k	n_k	c_k	λ_k
q0	1.0	0.9961	f0	1.0	0.9961	w0	1.0	0.9961
q1	1.0	0.9961	f1	4.0	0.5000	w1	5.0	0.5000
q2	1.0	0.9961	f2	4.0	0.5000	w2	5.0	0.5000
q3	3.0	0.5000	f3	4.0	0.5000	w3	5.0	0.5000
q4	3.0	0.5000	f4	4.0	0.5000	w4	5.0	0.5000
q5	3.0	0.5000	f5	4.0	0.5000	w5	5.0	0.9961
q6	3.0	0.5000	f6	4.0	0.5000	w6	5.0	0.9961
q7	1.0	0.9961	f7	4.0	0.5000	w7	5.0	0.9961
			f8	4.0	0.5000			
			f9	1.0	0.9961			
Total Time		9.0118	*Total Time*		18.0078	*Total Time*		21.0431
Qsort Deadline		9.0200	*FFT Deadline*		18.0200	*WT Deadline*		21.0450

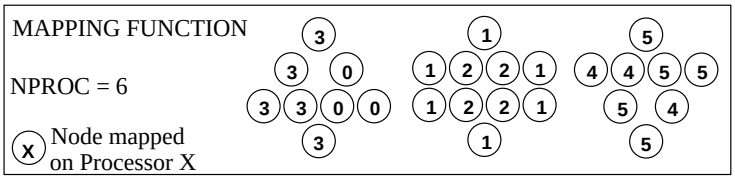

Figure 1: Test with multiple different programs (an instance of QSort, an FFT and a Wavelet Transform.)

The results obtained, applying the ASA method with the quenching correction, are outlined in the following section.

5 SIMULATION RESULTS

We verified our method in two different ways: checking that the method has found an acceptable solution to $\odot$ and investigating the computational complexity exhibited by our algorithm. To test the effectiveness of our methods in term of solution goodness we made several simulations with different degrees of complexity. Some DF programs, such as QSort, FFT and Wavelet Transform, were used in various combinations and with different deadlines (see appendix B for their SG definitions). Figure 1 shows a quite difficult test involving all three programs at the same time. This table outlines the problem parameters and the resulting $\overline{\lambda}$ vector and mapping function. The deadlines used are very close to the theoretical limits imposed by node costs. The other test direction involves a heuristic evaluation of algorithm complexity. We investigated the increase in convergence time as a function of the number of programs involved in the system and the number of nodes composing the programs. These tests are focused mainly on QSort-like programs. The results show that the convergence time grows as a quadratic function with respect to node number and program number. Figure 2 graphs this behavior: the test points were

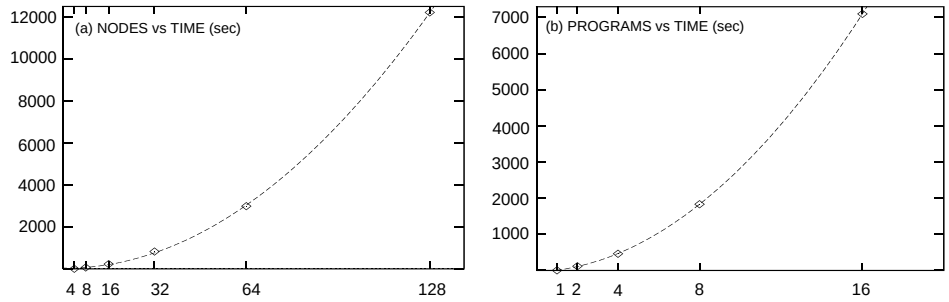

Figure 2: Result graphs: number of nodes (a) and number of programs (b) vs. convergence time.

fitted to a second-degree polynomial function, obtaining very good interpolations. Figure 2a presents the schedule computation for a single QSort-like program on two processors. Figure 2b shows the correlation between the number of programs and the time to compute a schedule. Each program was an eight-node QSort; the system had four processors.

We should point out that it is statistically possible to find local minima of the optimization problem. Clearly, the probability of failures in the solution search increases with problem size, i.e. it depends on the number of nodes, programs and processors. In our test set, involving narrow solution spaces, SNN always reached the solution for all the simulations. For significantly larger problems, either a local minimum can be found or the annealing factors have to be modified, which affects the complexity.

6 CONCLUSIONS

The schedulability problem for a set of SG programs can be expressed either as a non-linear set of equations or as an equivalent, non-linear, non-convex function to be minimized. The independent variables, or the domain variables for the functional specification, are the timing (or load) and the mapping information of each node. Although this specification does not change the hard nature of the problem, it allows a number of non-deterministic techniques to be applied. We showed that SNN can be used to give solutions. The SNN approach, when applied with ASA method, has led to encouraging results both in terms of accuracy and complexity growth. We plan to further improve the non-deterministic methodology to address the schedulability problem for very-large sets of programs and to extend the model to minimize resources.

APPENDIX A - $TIME$ AND $LOAD$ FUNCTIONS

$Time(\overline{\lambda}, S)$ and $Load(\overline{\lambda}, S)$ functions are defined as the numerical results, respectively, $E.time$ and $E.load$ of the following Syntax Directed Definition (SDD) applied to S:

$$
\begin{array}{ll}
E_1 \rightarrow E_2\ '|'\ F & \{E_1.time = \max\{E_2.time, F.time\}; E_1.load = E_2.load + F.load\} \\
F_1 \rightarrow F_2 T & \{F_1.time = F_2.time + T.time; F_1.load = \max\{F_2.load, T.load\}\} \\
E \rightarrow F & \{E.time = F.time; E.load = F.load\} \\
F \rightarrow T & \{F.time = T.time; F.load = T.load\} \\
T \rightarrow n_k & \{T.time = c_k/\lambda_k; T.load = \lambda_k\} \\
T \rightarrow'('\ E\ ')' & \{T.time = E.time; T.load = E.load\}
\end{array}
$$

APPENDIX B - DF GRAPHS AND RELATED STRINGS

All nodes are indicated by their unique name; concurrent execution is a binary operator $'|'$ and sequence is indicated by an implicit operator, i.e. sequential elements are simply juxtaposed. The language uses round parentheses for grouping. The DF Graph language has the following grammar G:

$$
\begin{array}{l}
G = (\Sigma_t, \Sigma_n, P, E) \\
\Sigma_t = \{'(', ')', '|'\} \cup node\_identifiers \\
\Sigma_n = \{E, F, T\}
\end{array}
\qquad
P = \begin{cases}
E \rightarrow E\ '|'\ F \\
E \rightarrow F \\
F \rightarrow FT \\
F \rightarrow T \\
T \rightarrow node \\
T \rightarrow'('\ E\ ')'
\end{cases}
$$

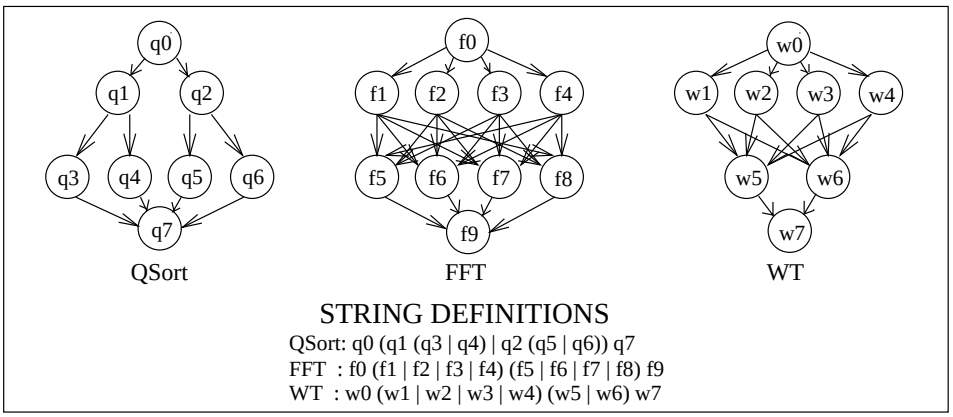

Figure 3: Program graphs used in the paper as examples of DF tasks.

References

[1] M. L. Dertouzos and A. K. Mok. Multiprocessor On-line Scheduling of Hard-Real Time Tasks. *IEEE Transactions on Software Engineering*, vol. 15(12), 1989.

[2] G.E. Hinton and T.J. Sejnosky. Optimal Perceptual Inference. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 448–453, Washington, 1983.

[3] J.J. Hopfield and D.W. Tank. "Neural" Computation of Decision in Optimization Problems. *Biological Cybernetics*, 52, pp. 141–152, 1985.

[4] Y. Fu and P.W. Anderson. Application of Statistical Mechanics to NP-Complete Problems in Combinatorial Optimization. *Journal of Physics*, A 19, pp. 1605–1620, 1986.

[5] M. Ohlsson, C. Peterson and B. Söderberg. Neural Network for optimization Problems with Inequality Constraints: the Knapsack Problem. *Neural Computation*, 5(2), pp. 331–339, 1993.

[6] L. Ingber. Very Fast Simulated Re-Annealing. *Mathl. Comput. Modelling*, 12(8), pp. 967–973, 1989.

[7] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in Hard-Real Time Environment. *Journal of ACM*, vol. 20(1), pp. 46–61, 1973.

[8] P. Peretto. Collective Properties of Neural Networks: A Statistical Physics Approach. *Biological Cybernetics*, 50, pp. 51–62, 1984.

[9] B. Rosen. Function Optimization based on Advanced Simulated Annealing. In *IEEE Workshop on Physics and Computation - PhysComp 92*, pp. 289–293, 1992.

[10] K. Schwan and H. Zhou. Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Thread. *IEEE Transactions on Software Engineering*, vol. 18(8), 1992.

[11] J.A. Stankovic, M. Spuri, M. Di Natale and G.C. Buttazzo. Implications of Classical Scheduling Results for Real-Time Systems. *IEEE Computer*, pp. 16–25, 1995.

[12] J. Zhu, T. Lewis, W. Jackson and R. Wilson. Scheduling In Hard Real-Time Applications. *IEEE Software*, vol. 12(3), pp. 54–63, 1995.