A GA-Based Approach to Dynamic Reconfiguration of Real-Time Systems*

Marco A. C. Simões
Bahia State University
ACSO/UNEB, Brazil
msimoes@uneb.br

George Lima
Federal University of
Bahia - LaSiD/UFBA, Brazil
gmlima@ufba.br

Eduardo Camponogara
Federal University of
Santa Catarina (DAS/UFSC), Brazil
camponog@das.ufsc.br

Abstract

Modern real-time systems have become increasingly complex since they have been required to operate in modern architectures and environments with high level of temporal uncertainty, where the actual execution demand is determined only during execution and is valid only for a given period of time. Thus such systems require both temporal isolation and dynamic reconfiguration. In this paper we deal with both these requirements. By assuming that the system is structured such that there are several modes of operation to be chosen at runtime, we formulate a reconfiguration problem. Temporal isolation is ensured by CBS. A solution to the reconfiguration problem is based on Genetic Algorithms, which provide the graceful adaption of the system so that it copes with new demands of the environment. The proposed solution is evaluated by simulation and the results found show the suitability of the approach.

1 Introduction

The design of real-time systems has become increasingly complex. Modern hardware and software architectures and/or the support to open environments, for example, make it difficult or even impossible to estimate task worst-case execution times accurately [13]. A usual way of dealing with this problem is by providing *temporal isolation* in the system. Indeed, if a certain task runs more than what was expected, the effect of this timing fault should not propagate to other application tasks. There are several approaches to temporal isolation and for EDF-scheduled systems the Constant Bandwidth Server (CBS) has received special attention recently [1, 9, 5, 4].

Although temporal isolation is an important aspect, for certain kinds of modern applications, the system should also provide support for *dynamic reconfiguration*. Indeed, modern systems may be structured so that their tasks have one or more modes of operation. For example, a control task can execute one among several control functions and/or with one among several sampling periods previously set. In other words, a task that has more than one operation mode may have alternative pieces of code to execute or alternative release periods. Assuming that each operation mode gives a different benefit for the system, the problem is then to select, at runtime, the set of task modes that maximizes the system benefit subject to the system schedulability.

Applications such as autonomous mobile robots, which embed several complex functions into off-the-shelf complex hardware components, are among those that require support to both temporal isolation and dynamic reconfiguration. For example, the robot computer vision subsystem may experiment different operational modes due to environment changes. Light conditions, obstacles, vision angle and other unpredictable environmental characteristics may generate strong variation on execution times of vision subsystem tasks. Also, modifications in the robot goals during its lifetime may well require dynamic reconfiguration. Further timing faults in one of the system components, even during reconfiguration, should not jeopardize the system performance as a whole.

Some approaches to dynamic reconfiguration have been proposed. For example, Feedback Scheduling (FS) [11], based on control theory, treats the system scheduling as a control plant and uses a controller to adjust the scheduling parameters at runtime. This approach offers no temporal isolation. FS-CBS [14] uses FS. It is assumed that the system is composed of Model Predictive Control (MPC) tasks served by CBS. Although this approach provides temporal isolation, its applicability is restricted to MPC tasks. Further, FS-CBS does not deal with reconfiguration of server periods. Other approaches aim at providing server capacity sharing mechanisms [4, 5, 9]. However, they do not provide dynamic reconfiguration. If a system has no idle time, for example, there is no slack time to be shared.

In this paper we deal with dynamic reconfiguration in a new and challenging scenario, as will be seen in Section 2. Temporal isolation is ensured by the use of CBS. Differ-

*This work is funded by CNPq (grant number: 475851/2006-4) and CAPES/PROCAD (AST project)

ent server modes are previously defined, each one giving a benefit for the system. Dynamically reconfiguring the system is then seen as choosing the operation modes of each server so that the system benefit is maximized. Clearly, this is a reasonably complex optimization problem, which may require too much computational resources to be solved online. However, as will be explained in Section 3, instead of searching for the optimal solution, the approach provided here is capable of progressively reconfiguring the system toward the optimal solution. To do that, we use Genetic Algorithms (GA). The results of experiments, given in Section 4, highlight the suitability of the described approach for the kind of system we are considering. Final comments on this work and on the challenges it brings about are given in Section 5.

2 System Model and Problem Definition

We are considering a system with one processor supporting n Constant Bandwidth Servers (CBS) [1] $S = \{S_1, S_2, ..., S_n\}$ that are scheduled by the Earliest Deadline First algorithm (EDF) [10]. Each server $S_i \in S$ has $\kappa(i) \geq 1$ operation modes and we denote $K_i = \{1, 2, \ldots, \kappa(i)\}$. The k-th operation mode of S_i is denoted by the tuple (Q_{ik}, T_{ik}), where Q_{ik} represents the server mode capacity and T_{ik} is its period. Thus, each server S_i operating in mode k has a maximum processor utilization $U_{ik} = \frac{Q_{ik}}{T_{ik}}$. In other words, the system allocates, to each task served by S_i, a constant bandwidth defined by U_{ik} so that temporal isolation is ensured [1]. As the system is scheduled by EDF, it is possible to use 100% of processor resources. Clearly, by the CBS schedulability properties [1], if the parameters Q_{ik} and T_{ik} are appropriately set for each server, tasks meet their deadlines or have an acceptable lateness (in case of soft tasks) provided that there are no timing faults.

We assume that the system may require that predefined values of Q_{ik} and T_{ik} are assigned to each server S_i at runtime so that the system can adapt or switch itself to attend to new demands by means of dynamic reconfiguration. This is performed by a system call, say `reconfig`$(U_{1k_1}, v_1, U_{2k_2}, v_2, \ldots, U_{nk_n}, v_n)$, where U_{ik_i} is the new desired processor utilization for server S_i and v_i is the associated benefit to this new configuration. Thus, if the system can allocate at least U_{ik_i} for server S_i, there will be a benefit v_i for the system. Without loss of generality, we assume that the benefit function associated to S_i running in mode k is defined by

$$A_{ik}(U_{ik_i}, u_i, v_i) = \frac{min(u_i, U_{ik_i})}{U_{ik_i}} v_i \,, \qquad (1)$$

where u_i represents the processor utilization effectively allocated to S_i. Other benefit functions are possible and we do not impose any restriction on them. Interesting discussion on benefit functions can be found elsewhere [3] and is beyond the scope of this paper. The benefits v_i are defined by the application when it uses the `reconfig` system call.

The execution of `reconfig` solves the optimization problem R defined as follows:

$$R : f = Maximize \sum_{S_i \in S} \sum_{k \in K_i} A_{ik} x_{ik} \qquad (2a)$$

$$\sum_{S_i \in S} \sum_{k \in K_i} U_{ik} x_{ik} \leq 1 \qquad (2b)$$

$$U_{ik} = \frac{Q_{ik}}{T_{ik}} \qquad (2c)$$

$$\sum_{k \in K_i} x_{ik} = 1,\ S_i \in S \qquad (2d)$$

$$x_{ik} \in \{0, 1\},\ S_i \in S,\ k \in K_i \qquad (2e)$$

Equation (2a) defines the objective function, which is based on equation (1). Variable x_{ik}, defined by equation (2e), represents the choice of one of the $\kappa(i)$ server operation modes of S_i. Only one configuration must be selected by `reconfig`, which is ensured by equation (2d). Restriction (2b) guarantees the schedulability of S according to the EDF policy.

It is not difficult to see that the classical knapsack problem can be reduced to R and so it is a $NP-Hard$ problem. One can solve R by using some standard optimization techniques. In particular, we have used dynamic programming to get optimal results, which will be used to assess our GA-based reconfiguration approach, as will be seen in Section 4. There is a recursive formulation for R which leads to a solution via dynamic programming, which is not shown here for the sake of space.

3 GA-Based Dynamic Reconfiguration

In this section we explain how the reconfiguration procedure was set up. The parameters of the procedure were empirically adjusted using an example of 10 servers with 15 operation modes each. The fitness function is given by equation (2a). The individuals (represented by their chromosomes) that have the highest value returned by the fitness function represent an optimal solution for the problem. In the context of problem R, a solution can be defined by a list $(k_1, k_2, \ldots, k_n)$, where $k_i \in K_i$. Let $\kappa^* = \max_{i=1}^{n} \kappa(i)$. Thus, a binary encoding [12] can be used so that $\lceil n \log_2 \kappa^* \rceil$ bits are needed to represent a possible solution (an individual). For example, a system with 10 servers each of which with 15 operation modes would require a 40-bit chromosome.

At the start of the `reconfig` procedure, we set the initial population by randomly generating $5n$ individuals.

These initial parameters are based on the De Jong's test suite [8] with a few adjustments. At each iteration of the algorithm a new population is generated from the current population. To do that, the operations of selection, crossover, mutation and migration are performed:

Selection. First, the individuals are ordered according to their fitness values. Two of them (the best fit ones) are selected by elitism [8] and copied to the new population. Selection probability values are assigned to the other $n-2$ individuals by the stochastic uniform method [2] according to their fitness values.

Crossover. The crossover operation is carried out $\lfloor \frac{5n-2}{4} \rfloor$ times to generate 50% of remaining offsprings. Each operation combines two individuals of the current population, generating two new offsprings to be added in the new population.

Mutation. This operation is carried out for $\lfloor \frac{5n-2}{2} \rfloor$ selected individuals, following some usual recomendations [12]. Each selected individual can have its bits changed with a probability of 0.1 per bit. The new generated individuals after mutation are added to the new population.

Migration. We have divided our population into two subpopulations of $\frac{5n}{2}$ individuals each. Then the 20% best-fit individuals from each subpopulation are cloned and they replace the 20% worst-fit individuals of the other subpopulation. This migration operation [6] takes place every 10 generations.

4 Assessment

We implemented the GA-based reconfiguration using Matlab/Simulink and the TrueTime toolbox [7]. We simulated 100 systems and the results presented below correspond to the average values found for these systems. Each simulated system had 10 servers with 15 operation modes each. The server parameters were randomly generated as follows. First, the values of U_{ik} and Q_{ik} for all $S_i \in S$ were generated according to a uniform distribution in the interval $(0, 0.2]$ and $[10, 100]$, respectively. Time is measured in time units (tu).

The first evaluated parameter was the average performance ratio (APR) achieved by the proposed approach. APR is defined here as $\frac{1}{m}\sum_{j=1}^{m}\frac{V_j}{OPT_j}$, where $m = 100$ is the number of simulated systems, V_j is the value achieved by the proposed approach as for the simulated system j and OPT_j is its optimum value given by a dynamic programming based solution (as said before, not shown here). As can be seen from Table 1, the APR achieved by the proposed approach gives very good results, which lies very close to 80% to the optimum value. It is interesting to note that the performance ratio does not change significantly as a function of the number of generations used to reconfigure the system, indicating a rapid convergence of the reconfiguration mechanism. As can be observed, satisfatory solutions can be found using as few as 10 generations at the expense of about 73 tu.

Generations	APR	Std. Dev.	Av. Exec. Time (tu)
10	0.74	0.02	73.13
25	0.76	0.02	162.45
50	0.77	0.02	311.97
100	0.78	0.01	610.22
250	0.79	0.01	1505.70
500	0.79	0.01	3001.63
1000	0.79	0.01	6016.20

Table 1. Performance ratio.

We have set up a simulation experiment as follows. A specific server with a single mode, given by $(20, 100)$, was added to the system and reserved for carrying out the reconfiguration. This means that the other 10 servers have now 80% of processor resources available. During the simulation, the average performance ratio was measured for 100 simulated systems. The time interval of observation corresponds to the first 100 activations of the reconfiguration server after the time the `reconfig` system call is called. Each reconfiguration server instance manages to deal with 3.44 generations on average. The average values for all systems are shown in Figure 1. As can be seen, there is a fast convergence of the reconfiguration mechanism. More than 72% to the optimum is achieved after the third instance of the reconfiguration server while 76% is reached after its 15th instance. However, the reconfiguration procedure tends to converge to some local optimum. As we managed to set up the GA parameters so that the global optimum was reached for some specific simulated systems, we believe that the local optimum convergence is due to the fact that the GA parameters used are not suitable for all simulated systems, since they were generated at random. This suggests that a more systematic approach to setting up such parameters is needed.

It is important to emphasize the adaptive nature of the proposed approach. While each reconfiguration instance is able to manage only a few generations, it is producing partial results. For example, when the third instance of the server finishes executing, it gives 72% of APR. This partial solution can be immediatelly used and is progressively improved as long as the reconfiguration server is running. In the case of this simulation, 80 time units on average were needed to process the three first instances of the server. The 15th instance finishes after 1190 time units. It is also worth mentioning that we did not address the problem of mode

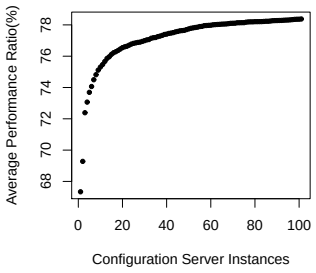

Figure 1. Average performance ratio growth.

change necessary to pass from one configuration to another.

5 Conclusions

We have described an approach to dynamic reconfiguration of real-time systems structured as a set of Constant Bandwidth Servers each of which can operate in one of the previously defined modes. Using a specific objective function, we have formulated a specific reconfiguration problem. We have shown that approximate solutions to this problem can be found in an effective way by using Genetic Algorithms. The described approach is evaluated by simulation and the results found have indicated the suitability of the described approach. For example, the proposed approach can be used for gradually adapting the system to environment changes. This is particularly interesting to deal with modern real-time systems operating in environments with high level of uncertainty.

Most parameters used in the reconfiguration approach were defined empirically. It would be interesting to derive mechanisms to adjust such parameters in a mechanized way by using, for instance, Neural Networks. More complex objective functions can also be investigated. For example, one may be interested in considering parameters such as lateness, deadline miss ratio, etc., which will give rise to multi-objective optimization problems. Another challenging goal would be to design self-adjustable scheduling policies so that the system itself decides when to carry out reconfigurations. Certainly, these and other research topics can use the proposed approach as a starting point.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. *Real Time Systems Symposium(RTSS), The 19th IEEE*, pages 4–13, 1998.

[2] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In J. J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*. Erlbaum, 1987.

[3] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46:305–325, 2000.

[4] M. Caccamo, G. Buttazzo, and Lui Sha. Capacity sharing for overrun control. *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, pages 295–304, 2000.

[5] M. Caccamo, G.C. Buttazzo, and D.C. Thomas. Efficient reclaiming in reservation-based real-time systems with variable execution times. *IEEE Transactions on Computers*, 54(2):198–213, Feb. 2005.

[6] E. Cantu-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. *J. Heurist*, 7:311–334, 1999.

[7] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Arzen. How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime. *Control Systems Magazine, IEEE*, 23(3):16–30, June 2003.

[8] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[9] Caixue Lin and Scott A. Brandt. Improving soft real-time performance through better slack reclaiming. *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, 0:410–421, 2005.

[10] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[11] Chenyang Lu, John A. Stankovic, Sang H. Son, and Gang Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst.*, 23(1-2):85–126, 2002.

[12] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[13] Xiaorui Wang. *Adaptive Quality of Service Control in Distributed Real-Time Embedded Systems*. PhD thesis, Washington University, August 2006.

[14] Pingfang Zhou, Jianying Xie, and Xiaolong Deng. Optimal feedback scheduling of model predictive controllers. *Journal of Control Theory and Applications*, 4(2):175–180, Feb 2006.