Dynamic Reconfiguration of *Constant Bandwidth Servers*

Augusto Born de Oliveira[1], **Eduardo Camponogara**[1], **George Lima**[2]

[1]Departamento de Automação e Sistemas – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis, SC – Brazil

[2]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
40170-110 – Salvador, BA – Brazil

{augustob,camponog}@das.ufsc.br, gmlima@ufba.br

Abstract. *This paper presents a model of real-time systems composed of a set of Constant Bandwidth Servers (CBS) and a reconfiguration mechanism that allows the redistribution of processor time between tasks at runtime. In our model, the developer attributes a value to each reconfiguration mode of the CBS servers and an optimization problem is solved to maximize the overall value of the system.*

1. Introduction

The research in the area of real-time scheduling has produced several approaches capable of dealing with several classes of applications [5]. Some applications, characterized by high variability of execution times, such as encoding or decoding of media frames, usually require special scheduling mechanisms so that their time constraints are satisfactorily dealt with. A usual way to schedule the tasks of this kind of application is by using aperiodic task servers. These servers can be informally defined as virtual tasks whose execution time is dedicated to execute application tasks. Among the server approaches, the Constant Bandwidth Server (CBS) [1] has received special attention since it is capable of providing temporal isolation, which is specifically desirable for applications with high variability in their execution times. Indeed, according to CBS rules, the fraction of processor time allotted for each server is never surpassed even if its tasks overrun. The CBS approach is also simple to implement since it is based on the well known EDF policy to schedule the servers [8].

A CBS server is described in terms of two parameters, a period and a maximum budget, and a set of rules that are used to restore the budget of the servers so that schedulability and temporal isolation are ensured. Usually, the server parameters are static and defined during design time. However, in a system with various applications, there can be events for which processor time needs to be redistributed between the tasks during runtime. If, for instance, a surveillance system detects motion in the room it monitors, it might be needed to shift computing power to the video recording system to capture better images of a possible intruder. Such systems need support for dynamic reconfiguration. Indeed, the issue of dynamically changing application requirements, due to external sensory data or low-level architecture features has been recently raised [3]. This work clearly shows that there is the need for flexible scheduling, in which algorithms might change during runtime to better adapt to environment variations.

This paper describes a model of a system with various concurrent CBS servers, each with various levels of utilization and benefit values. If the state of the system changes

in a way that alters those values, a reconfiguration interface is used to redistribute processor time between the servers with the objective of maximizing the total benefit. It is important to point out that in order to provide dynamic reconfiguration, usually one has to solve two other problems: (i) choosing among several possible modes of operation the one that is more suitable to the current system requirements; (ii) providing a mode change strategy once a new mode of operation is chosen. In this paper we focus only on the first problem, which is usually a fundamental step towards dynamic reconfiguration.

Dynamic reconfiguration has been addressed by other researchers [4, 2, 7, 6, 10] but most of them do not deal with CBS-based systems. Thus, these and other related approaches do not offer support to temporal isolation. Recently, a reconfiguration mechanism for CBS-based systems has been proposed [13]. Although this approach provides temporal isolation, its applicability is restricted to a specific task model for control systems. Further, it does not deal with reconfiguration of server periods.

Unlike most of the above approaches, our goal is to dynamically reconfigure both server parameters of CBS-based real-time systems. The optimization problem seen in our formulation has recently been solved by a Genetic Algorithm [11]. However, due to its slow convergence rate, this solution is more appropriate to define meta-heuristics and must not be used when the time to reconfigure the system is an issue.

After defining the computational model and formalizing the reconfiguration problem addressed in this paper in section 2, we describe our solution in section 3. This solution is based on an approximation approach since the computational complexity of the problem prevents the use of optimal solutions. Results from extensive simulation, presented in section 4, show the effectiveness of our approach in terms of both its running time and its achieved approximation rate. Our final comments are given in section 5.

2. Model and Problem Definition

We consider a system with a single processor composed of n CBS servers $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ [1]. Each server $S_i \in \mathcal{S}$ is defined by the tuple (Q_i, T_i), where Q_i represents its maximum budget and T_i represents its period. Each server S_i's utilization is at most $u_i = Q_i/T_i$ and serves a specific set of tasks. In other words, a system constructed in this manner allocates, for each set of tasks attributed to server S_i, a constant bandwidth determined by u_i, providing temporal isolation between servers even in overload conditions. The budget of each server S_i is depleted as its jobs are executed. To guarantee that the server obeys its utilization limit, the deadline of S_i is postponed by T_i every time its budget is depleted. At the same time, full budget Q_i is restored. When a new job arrives in an idle CBS, budget restoration and deadline postponement depend on a condition calculated at run time. For more details refer to [1].

We assume that there is no precedence relation nor shared resources between tasks. Thus, since every server is scheduled according to EDF, 100% utilization can be attained when scheduling the servers in $\mathcal{S}$. That is, the system is schedulable if and only if

$$\sum_{S_i \in \mathcal{S}} u_i \leq 1 \tag{1}$$

In the context of this work, we assume that the application can, at any time, request a reconfiguration of the parameters of the servers in $\mathcal{S}$. To accomplish this, the application

indicates what processor time percentages should be allocated to each server S_i through a system call `reconfig`$(U_1, v_1, U_2, v_2, \ldots, U_n, v_n)$. If the system is able to allocate at least U_i to the S_i server, a benefit value of $v_i \geq 0$ is attained. The value v_i is assumed to be derived from the knowledge of the system designer and represent one of the parameters used by the reconfiguration mechanism. Generally, we assume that the benefit function associated to S_i has the following form:

$$A_i(U_i, u_i, v_i) = \frac{\min(u_i, U_i)}{U_i} v_i \,, \tag{2}$$

where u_i represents the processor time allocated to S_i.

As it can be noted, the execution of `reconfig`$(U_1, v_1, U_2, v_2, \ldots, U_n, v_n)$ requires solving an optimization problem, where equation (1) is one of the constraints and function (2) must be maximized. We assume that, for each server S_i, $\kappa(i)$ values for U_i exist. In other words, let $K_i = \{1, 2, \ldots, \kappa(i)\}$ and $U_{ik} = Q_{ik}/T_{ik}$, $k \in K_i$, define the kth configuration of S_i for each of which there is a benefit A_{ik} as given by (2). More formally, the optimization problem that must be solved by the reconfiguration mechanism is given as follows:

$$P: \; f = \text{Maximize} \sum_{S_i \in \mathcal{S}} \sum_{k \in K_i} A_{ik} x_{ik} \tag{3a}$$

Subject to :

$$\sum_{S_i \in \mathcal{S}} \sum_{k \in K_i} u_{ik} x_{ik} \leq 1 \tag{3b}$$

$$u_{ik} = \frac{Q_{ik}}{T_{ik}} \tag{3c}$$

$$\sum_{k \in K_i} x_{ik} = 1, \; S_i \in \mathcal{S} \tag{3d}$$

$$x_{ik} \in \{0, 1\}, \; S_i \in \mathcal{S}, k \in K_i \tag{3e}$$

The parameter A_{ik} defines the benefit attained by allocating u_{ik} units of computational resources to the S_i server. The x_{ik} variable, defined by equation (3e), indicates which configuration $k \in K_i$ is chosen for the S_i server. Exactly one configuration must be selected, which is reflected in restriction (3d). Restriction (3b) guarantees schedulability of $\mathcal{S}$ according to the EDF policy. With no loss of generality, we assume that $u_{ik} \leq u_{ik-1}$ for every $S_i \in \mathcal{S}$ and $k \geq 2$. To model the possibility of cancellation of a S_i server, it is sufficient to define $A_{i1} = 0$ and $u_{i1} = 0$. We also assume that $\sum_{S_i \in \mathcal{S}} u_{i1} < 1$, otherwise the servers would not be schedulable or there would not be potential to optimize the applications.

The classic knapsack problem is trivially reducible to the reconfiguration problem, therefore P is NP-Hard [12]. Thus, approximation algorithms to solve the problem are necessary.

3. Approximation Algorithms

An algorithm that returns a solution that approximates the optimum is said to the an approximation algorithm [9]. These algorithms can be useful in the search for a good solu-

tion when computation time is restricted, specially when dealing with NP-Hard problems. What follows is a generalization of approximation algorithms for the knapsack problem to the reconfiguration problem defined in the previous section. Both approximation algorithms described follow a greedy heuristic.

3.1. Density Greedy Algorithm

The density greedy algorithm has two distinct steps. First, it allocates the minimum resources possible to each server to ensure the feasibility of the reconfiguration. This resource pre-allocation is made necessary by equation (3d), that requires that each server receives a share of the system's resources. Second, it distributes the remaining resources to the servers folowing a non-increasing order of additional relative benefit $\hat{A}_{ik} = (A_{ik} - A_{i1})/(u_{ik} - u_{i1})$ as a primary ordering key and decreasing order of relative resource demand $\hat{u}_{ik} = (u_{ik} - u_{i1})$ as a secondary ordering key.

By pre-allocating u_{i1} resource units to each server $S_i \in \mathcal{S}$, problem P can be reformulated as:

$$P: \ f = \text{Maximize} \sum_{S_i \in \mathcal{S}} \sum_{k \in K_i - \{1\}} (A_{ik} - A_{i1}) x_{ik}^{ad} + \sum_{S_i \in \mathcal{S}} A_{i1} \tag{4a}$$

Subject to :

$$\sum_{S_i \in \mathcal{S}} \sum_{k \in K_i - \{1\}} (u_{ik} - u_{i1}) x_{ik}^{ad} \leq 1 - \sum_{S_i \in \mathcal{S}} u_{i1} \tag{4b}$$

$$\sum_{k \in K_i - \{1\}} x_{ik}^{ad} \leq 1, \ S_i \in \mathcal{S} \tag{4c}$$

$$x_{ik}^{ad} \in \{0, 1\}, \ S_i \in \mathcal{S}, k \in K_i - \{1\} \tag{4d}$$

Note that a one-to-one relation exists between the solutions of formulation (3a)–(3e) and formulation (4a)–(4d). Particularly, for each server $S_i \in \mathcal{S}$, if $\sum_{k \in K_i - \{1\}} x_{ik}^{ad} = 0$ then $x_{i1} = 1$ e $x_{ik} = 0$ for every $k \in K_i - \{1\}$. Otherwise, if $\sum_{k \in K_i - \{1\}} x_{ik}^{ad} = 1$, then $x_{i1} = 0$ and $x_{ik} = x_{ik}^{ad}$ for every $k \in K_i - \{1\}$.

Let $\Omega = \{(i, k) : S_i \in \mathcal{S}, \ k \in K_i\}$. Then, for a subset of pairs $\omega \subseteq \Omega$, let $S(\omega) = \{S_i : (i, k) \in \omega\}$ be the set of servers present in ω.

DGA$(\mathcal{S}, \{A\}, \{u\})$

1: Order the pairs of $\Omega - \{(i, 1) : S_i \in \mathcal{S}\}$ in the sequence $\langle (i_1, k_1), \ldots, (i_{|\Omega|-|\mathcal{S}|}, k_{|\Omega|-|\mathcal{S}|}) \rangle$ so that $\hat{A}_{i_p k_p} > \hat{A}_{i_q k_q}$ or $\hat{A}_{i_p k_p} = \hat{A}_{i_q k_q}$ and $\hat{u}_{i_p k_p} \geq \hat{u}_{i_q k_q}$ for every $p < q$
2: $\omega \leftarrow \emptyset$
3: $b \leftarrow 1 - \sum_{S_i \in \mathcal{S}} u_{i1}$
4: $t \leftarrow 1$
5: **while** $t \leq |\Omega| - |\mathcal{S}| \wedge |\omega| < |\mathcal{S}|$ **do**
6: **if** $S_{i_t} \notin S(\omega)$ and $(u_{i_t k_t} - u_{i_t 1}) \leq b$ **then**
7: $\omega \leftarrow \omega \cup \{(i_t, k_t)\}$
8: $b \leftarrow b - (u_{i_t k_t} - u_{i_t 1})$
9: **end if**

10: $t \leftarrow t + 1$
11: **end while**
12: **for** $S_i \in \mathcal{S} - S(\omega)$ **do**
13: $\omega \leftarrow \omega \cup \{(i, 1)\}$
14: **end for**
15: return ω

3.2. Modified Density Greedy Algorithm

The modified density greedy algorithm (M-DGA) returns the reconfiguration produced by DGA, ω, unless the reconfiguration of a server $S_{i'}$ in an execution level k', (i', k'), induces an objective function with a larger value than that produced by the density greedy algorithm. In this case, M-DGA returns $\omega' = \{(i, 1) : S_i \in \mathcal{S}, i \neq i'\} \cup \{(i', k')\}$.

M-DGA$(\mathcal{S}, \{A\}, \{u\})$
1: $(i', k') \leftarrow \arg \max_{(i,k)\in\Omega:k>1} \{A_{ik} - A_{i1} : u_{ik} - u_{i1} \leq 1 - \sum_{S_t\in\mathcal{S}} u_{t1}\}$
2: $\omega' \leftarrow \{(i, 1) : S_i \in \mathcal{S}, i \neq i'\} \cup \{(i', k')\}$
3: $\omega \leftarrow DGA(\mathcal{S}, \{A\}, \{u\})$
4: **if** $f(\omega') > f(\omega)$ **then**
5: return ω'
6: **else**
7: return ω
8: **end if**

The quality of an approximation algorithm is usually stated as the performance ratio regarding the optimum solution as formalized below.

Theorem 1. *Let I be an instance of the reconfiguration problem P. The performance of the modified density greedy algorithm, denoted by M-$DGA(I)$, and the optimum performance, denoted by $OPT(I)$, are related by the following expression:*

$$M\text{-}DGA(I) \geq \frac{OPT(I) + \sum_{S_i\in\mathcal{S}} A_{i1}}{2} \tag{5}$$

Proof: Let $\omega^\star$ be the solution produced by M-DGA, where $\omega^\star = \omega$ if $f(\omega) \geq f(\omega')$ and $\omega^\star = \omega'$ otherwise. Let $(i_\omega, k_\omega) = \arg \max_{(i,k)\in\Omega:k>1} \{(A_{ik} - A_{i1})/(u_{ik} - u_{i1}) : (i, 1) \in \omega\}$. Clearly,

$$OPT(I) \leq f(\omega) + \frac{A_{i_\omega k_\omega} - A_{i_\omega 1}}{u_{i_\omega k_\omega} - u_{i_\omega 1}}(1 - \sum_{(i,k)\in\omega} u_{ik})$$

There are two possible cases. If $\omega^\star = \omega$, then:

$$OPT(I) \leq 2f(\omega) - \sum_{S_i\in\mathcal{S}} A_{i1} = 2M\text{-}DGA(I) - \sum_{S_i\in\mathcal{S}} A_{i1}$$
$$\implies M\text{-}DGA(I) \geq \frac{OPT(I) + \sum_{S_i\in\mathcal{S}} A_{i1}}{2}$$

Otherwise, if $\omega^\star = \omega'$, then:

$$OPT(I) \leq 2f(\omega') - \sum_{S_i \in \mathcal{S}} A_{i1} = 2M\text{-}DGA(I) - \sum_{S_i \in \mathcal{S}} A_{i1}$$
$$\implies M\text{-}DGA(I) \geq \frac{OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i1}}{2}$$

Thus demonstrating the relation (5) between $OPT(I)$ and $M\text{-}DGA(I)$.

Corollary 1. *The modified density greedy algorithm has relative performance* $R_{M-DGA} = 2$.

Proof: From the theorem above, we have that:

$$\frac{OPT(I)}{M\text{-}DGA(I)} \leq \frac{OPT(I)}{(OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i1})/2} \leq 2 \tag{6}$$

Demonstrating that $R_{M-DGA} = 2$. □

4. Performance Analysis

The methods described above and a problem instance generator were implemented to evaluate their practical application. A dynamic programming (DP) algorithm was implemented to produce optimal results against which the approximate solutions were compared. All algorithms were implemented in C++ and executed on Linux on an Intel Core2Duo 2.20GHz CPU with 2GB of RAM.

A set of problems of varying cardinality and number of configurations was generated to assess the performance of the methods, namely $25 \leq |\mathcal{S}| \leq 250$ and $3 \leq \kappa(i) \leq 10$. Configurations were generated using a uniform distribution for both utilization and benefit values. The mean solution time of five problems of the same size is considered here to minimize time measuring errors.

Figure 1 shows the execution times of both methods, DGA and $M\text{-}DGA$. A comparison with the exact method showed that these have an execution time approximately twenty thousand times shorter.

Figures 2 and 3 show the quality of the solution returned by the approximation methods in absolute and relative forms, respectively. The heuristic used in $M\text{-}DGA$ was not activated in the generated instances, therefore the results of the two methods were identical. The optimal solution showed in figure 2 was calculated using dynamic programming, while the lower bound for the approximation and the upper bound of the optimal solver were calculated using inequality (5) of Theorem 1.

Similarly, figure 3 shows that, in practice, the solutions returned by the approximation methods are closer to the optimum (represented by the function $f(x) = 1$) than the theoretic lower bound, calculated with inequality (6).

5. Conclusion

This paper presented a model of real-time systems composed of a set of Constant Bandwidth Servers, and a reconfiguration system that allows the redistribution of processor time between tasks at runtime. In our model, the designer attributes a value to each reconfiguration mode of the servers. When trying to maximize the overall value of the

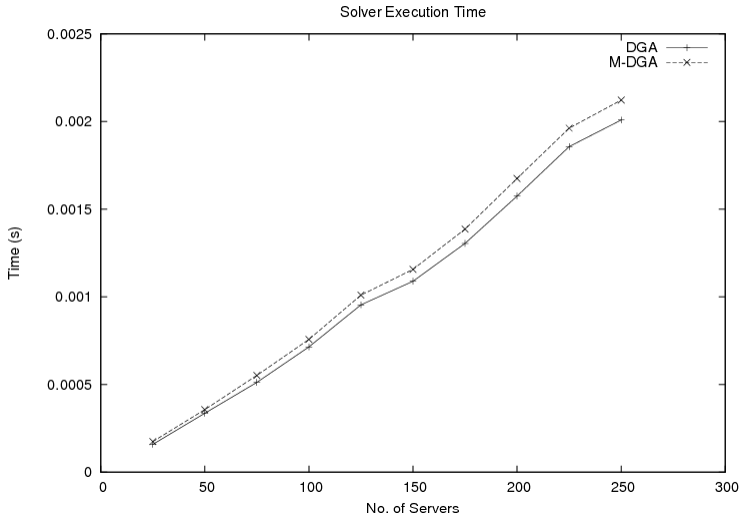

Figure 1. Solver execution time.

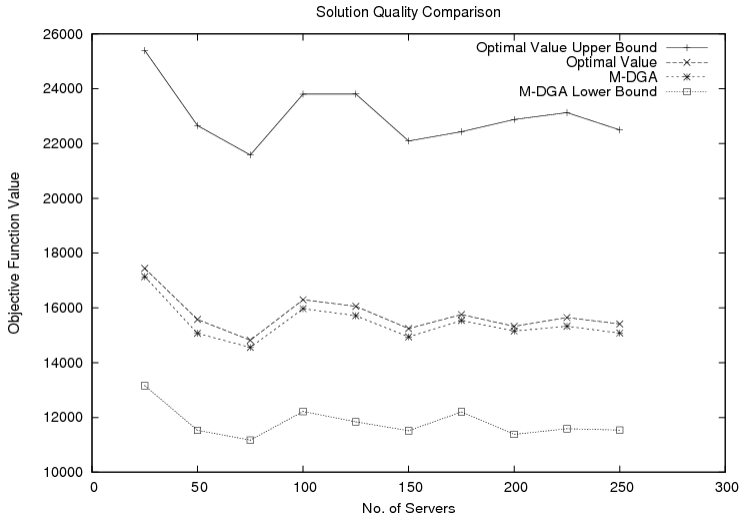

Figure 2. Absolute performance of the approximation algorithms.

system and maintain utilization under the schedulable limit, an NP-Hard optimization problem is encountered. Two approximation algorithms for solving this problem were presented. Their running time and their solution quality were compared to those of an optimal solver. This showed that while their solutions approximate the optimum by a considerably smaller margin than the theoretical bounds, their execution time is approximately twenty thousand times shorter than an exact solver based on dynamic programming.

The integration of the reconfiguration mechanism described here with a mode change strategy can be considered for future research. Less restrictive task models, which include resource sharing and/or precedence relations among tasks, should also be investigated. The results presented in this paper may well serve as a foundation for these and other possible developments in the field of scheduling for modern and adaptive real-time systems.

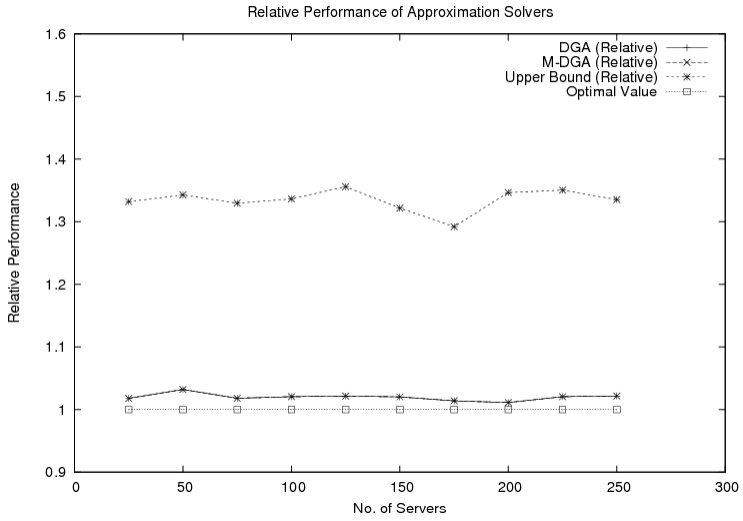

Figure 3. Relative performance of the approximation algorithms.

References

[1] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

[2] G. Beccari, S. Caselli, and F. Zanichelli. A technique for adaptive scheduling of soft real-time tasks. *Real-Time Systems*, 30(3):187–215, July 2005.

[3] G. Buttazzo. Research trends in real-time computing for embedded systems. *ACM SIGBED Review*, 3(3), 2006.

[4] G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real-Time Systems*, 23(1-2):7–24, 2002.

[5] J. M. Farines, J. Fraga, and R. S. de Oliveira. *Sistemas de Tempo Real*. Escola de Computação, first edition, 2000.

[6] J. Jehuda and A. Israeli. Automated meta-control for adaptable real-time software. *Real-Time Systems*, 14(2):107–134, 1998.

[7] T.-W. Kuo and A. K. Mok. Incremental reconfiguration and load adjustment in adaptive real-time systems. *IEEE Transactions on Computers*, 46(12):1313–1324, 1997.

[8] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogram in a hard real-time environment. *Journal of ACM*, 20(1):40–61, 1973.

[9] R. Motwani. Approximation algorithms. Book in preparation, 1992.

[10] C. Rusu, R. Melhem, and D. Mossé. Multi-version scheduling in rechargeable energy-aware real-time systems. *J. Embedded Comput.*, 1(2):271–283, 2005.

[11] M. Simões, G. Lima, and E. Camponogara. A GA-based approach to dynamic reconfiguration of real-time systems. In *Proc. of the Workshop on Adaptive and Reconfigurable Embedded Systems (APRES'08) - to appear*, 2008.

[12] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, NY, 1998.

[13] P. Zhou, J. Xie, and X. Deng. Optimal feedback scheduling of model predictive controllers. *Journal of Control Theory and Applications*, 4(2):175–180, Feb 2006.