A Dynamic Scheduling Algorithm for Real-Time Expert Systems

A. M. Campos

Department of Informatics
University of Oviedo
Campus de Viesques, 1.2.12. Asturias 33271, SPAIN
Phone 985182518, Fax 985181986
campos@atc.uniovi.es

D. F. García

Department of Informatics
University of Oviedo
Campus de Viesques 1.2.14, Asturias 33271, SPAIN
Phone 985182066, Fax 985181986
daniel@atc.uniovi.es

Abstract. Computational characteristics of real-time expert systems have been the subject of research for more than a decade. The computation time required to complete inferences carried out by expert systems present high variability, which usually leads to severe under-utilization of resources when the design of the schedule of inferences is based on their worst computation times. Moreover, the event-based aperiodic activation of inferences increases the risk of transient overloads, as during critical conditions of the controlled or monitored environment the arrival rate of events increases. The dynamic scheduling algorithm presented in this article obtains statistical bounds of the time required to complete inferences on-line, and uses these bounds to schedule inferences achieving highly effective utilization of resources. In addition, this algorithm handles transient overloads using a robust approach. During overloads our algorithm completes nearly as many inferences as other dynamic scheduling algorithms, but shows significantly better effective utilization of resources.

Keywords: real-time systems, scheduling algorithms, performance evaluation.

Track: paper track.

Topics: real-time systems.

A Dynamic Scheduling Algorithm for Real-Time Expert Systems

A. M. Campos, D. F. García

Department of Informatics
University of Oviedo
Campus de Viesques, Asturias 33271, SPAIN
{campos, daniel}@atc.uniovi.es

Abstract. Computational characteristics of real-time expert systems have been the subject of research for more than a decade. The computation time required to complete inferences carried out by expert systems present high variability, which usually leads to severe under-utilization of resources when the design of the schedule of inferences is based on their worst computation times. Moreover, the event-based aperiodic activation of inferences increases the risk of transient overloads, as during critical conditions of the controlled or monitored environment the arrival rate of events increases. The dynamic scheduling algorithm presented in this article obtains statistical bounds of the time required to complete inferences on-line, and uses these bounds to schedule inferences achieving highly effective utilization of resources. In addition, this algorithm handles transient overloads using a robust approach. During overloads our algorithm completes nearly as many inferences as other dynamic scheduling algorithms, but shows significantly better effective utilization of resources.

1. Introducción

The potential benefits of using expert systems in real-time environments has sparked research on algorithms and techniques to fulfill the temporal requirements and overcome the limitations of using expert systems in these environments. Various research projects have addressed the high variability of time required to complete inference processes, and the inexistence of tight bounds for inference time [1], [2], [3]. In addition to this formal research, the demand by industry for adequate software development tools has motivated the appearance of fast inference engines and complete development environments, such as G2 (by Gensym Corp.), Cogsys KBS (by Cogsys Ltd.) or Rtie (by Talarian Corp.). Some of these tools are suitable for building expert systems for on-line continuous operation or soft real-time expert systems.

The applicability of some of these tools for building real-time expert systems for automating or monitoring complex industrial processes has been evaluated. Although some of the tools claim to be the definitive solution for developing real-time expert systems, most of them do not recognize the concept of a real-time task as a computation that must satisfy a time restriction. So, management of temporal

requirements in design decisions is complicated. Furthermore, the scheduling policy is a key element of the design to fulfill the temporal requirements in real-time systems. Few of the commercial tools provide the system designer with any scheduling mechanisms. Some of them only allow the application of static priorities to the rules, which does not provide sufficient predictability of the system. In addition, when a real-time expert application developed with commercial tools must co-exist with other applications in the same computer, there is no a clear way of knowing how the expert application is using computer resources. This makes it impossible to control CPU assignment among all running applications. Clearly, there is a need for a specific real-time expert system based on a compact (embeddable) and easily connectable architecture, which is perfectly integrated in the operating system, and allows scheduling inferences following a well-defined policy.

The solution presented in this work involves serializing the use of a single inference engine by all the tasks in the system. This approach allows the use of a fast, general-purpose inference engine without truth maintenance support, because the working memory content will not be corrupted by the preemption of one running inference by another. In real-time expert systems, data input/output duration can usually be considered negligible when compared to inference duration. Thus, the behavior of our real-time expert system can be analyzed as the problem of scheduling a set of inference tasks, whose main characteristic is a highly variable computation time, in a non-preemptive way. The arrival of tasks, associated to changes in the environment, is aperiodic. This characteristic, added to the highly variable computation time of tasks, makes dynamic scheduling necessary. In addition, event-based aperiodic arrival of tasks can easily lead to transient overload conditions, related to critical conditions of the environment to be monitored or controlled. During these transient overloads the performance of a real-time expert system must be predictable. In summary, a dynamic, non-preemptive scheduling algorithm for tasks with aperiodic arrival and highly variable computation times, with graceful degradation during transient overloads, has been developed and evaluated.

The rest of the paper is structured as follows. In the next section the system model is presented. Section 3 presents the scheduling algorithm. The final sections describe the load and performance metrics used and include the results obtained.

2. System Model

The real-time expert system is composed of a set of n tasks, called intelligent tasks, $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Intelligent tasks are invoked aperiodically by events (stimulus) received from the environment, and produce a corrective reaction to this environment through the adequate interface. Arrival times, a_i, of events are not known in advance (the system is *non-clairvoyant*), and there is no lower bound on the duration between occurrences of the same event. Worst-case computation times, c_i, of the intelligent tasks are not known in advance (the high variability of execution time makes bounds obtained by static analysis very pessimistic), but an on-line estimation of the worst-case computation time, $\hat{c}_i$, will be calculated, as is described later. Each

intelligent task must meet a time restriction defined by its relative deadline, d_i. If a task does not meet this requirement, the task is said to have failed. Missing a deadline would not jeopardize the behavior of the system, although the benefit of executing a task that misses its deadline is zero (*firm* tasks). In this first approach, the benefit (the utility) of executing a task before its deadline has been considered constant for each task. Thus, an intelligent task, τ_i, is defined by the 4-tuple $(a_i, \hat{c}_i, d_i, u_i)$, where a_i is the arrival time of the event that invokes the task, $\hat{c}_i$ is a estimation of the worst-case computation time, d_i is the relative deadline of the task and u_i is the benefit obtained if the task execution finishes prior to its deadline.

Intelligent tasks are broken down into three activities: *data acquisition*, *inference* and *actuation*. During *data acquisition*, data from the environment are obtained and pre-processed. During *inference*, conclusions about the environment are obtained sharing a single unit resource in exclusive mode, the *inference kernel*. Only the inference activity of one task can be executed by the system at a time. *Actuation* applies corrective actions over the environment if necessary. In general, the resources used by acquisition and actuation activities of tasks do not require exclusive access. So two or more of these activities of different tasks can use resources concurrently. The usage of resources by both data acquisition activities and actuation activities are negligible compared with the consumption of resources of the inference activity (in particular, the highest consumption of resources, up to 90%, takes place during the pattern-matching phase [4] of the inference activity).

In summary, at a given moment, t, the expert control system can be modeled as a set of n firm aperiodic tasks $\{\tau_1, \tau_1, \ldots, \tau_n\}$, defined by 4-tuples $(a_i, \hat{c}_i, d_i, u_i)$, which compete for the use of an exclusively usable single unit resource, the *inference kernel*.

The sum of the utilities of all the tasks in the set is the total value of the task set. The task set is said to be feasible when all the tasks of the set can be completed before their deadlines. If one or more tasks fail, the system is said to be overloaded.

3. Scheduling Algorithm

The goal of an algorithm that schedules the execution of intelligent tasks is to find a schedule for a given task set which obtains as high a value as possible, while fulfilling the restrictions imposed by the environment and the internal architecture of the expert system itself. In this paper we assume that the expert system runs in a uniprocessor machine.

Exclusive access imposed by the inference kernel makes the problem of finding a feasible schedule NP-Hard [5]. It is well known that optimal pre-emptive scheduling algorithms for uniprocessor systems are not optimal when preemption is not allowed, as with Lowest Laxity First algorithm [6], or remain optimal but under more restrictive conditions, as with Earliest Deadline First (EDF) algorithm. EDF was proven to be optimal in a non-preemptive task model if the processor does not remain idle while there are tasks waiting to be executed [7][8]. As the expert system is non-clairvoyant, there is no reason to remain idle while there are tasks waiting in the

system, so the EDF algorithm is optimal in the sense of feasibility. In addition, deadline scheduling means that it is not necessary to know computation times of tasks in advance, ratifying the EDF selection. Unfortunately, the performance of EDF is dramatically reduced in overload conditions, so the effect of a transient overload can be catastrophic [9]. This risk is unacceptable for most real-time applications, and in particular for real-time expert systems, as overload conditions are normally related to critical conditions in the environment

Overloads can be handled using two basic techniques [10]. The first technique, *guarantee*, handles overloads by using acceptance tests, and rejecting tasks that makes the task set unfeasible. The second technique, *robust*, is an extension of the first technique, in which rejected tasks enter a reject queue from where they can be rescued for execution or finally discarded.

The best effort technique is not adequate for real time expert systems with non-preemptive restrictions, as it does not predict overloads. If EDF is used running without overload prediction, catastrophic situations will result. Nor are guarantee techniques adequate because of the high variability of the execution time of intelligent tasks. Consequently, using guarantee techniques there is a risk of under-utilization of resources. So, only robust-like approaches are able to achieve adequate scheduling for this kind of system.

Our algorithm uses statistical information, obtained on-line, about the intelligent tasks in order to detect a system overload. If the system is not overloaded, the intelligent task to run is selected following EDF policy. If, on the contrary, an overload is detected, the task to execute is selected as a function of the probability of success and the expected utility. Tasks remain in the system until they have missed their deadlines.

3.1. Overload detection

The high variability of the computation time of intelligent tasks reduces the validity of the feasibility analysis of the complete set of tasks in the expert system. It is quite normal to have time after executing one intelligent task or to miss a deadline due to an unexpectedly long computation, so feasibility analysis loses validity as the number of executed tasks of the analyzed set increases. Also, as intelligent tasks can share data stored in the working memory of the real-time expert system, the computation time of intelligent tasks can be influenced by the computation of previous tasks. The aperiodic arrival of new intelligent tasks can also make long feasibility analysis invalid. Thus, this work predicts overloads after the execution of every task using all the tasks in the system, but considering only one-task-ahead execution, as is explained below.

Let $\alpha \in \Re$, $0 \le \alpha < 1$ be the maximum admissible probability of missing a deadline, chosen at design time. If $\hat{c}_i$ is an estimator of the worst-case computation time of task τ_i with a *confidence level* of $1-\alpha$ (that is $P(c_i \le \hat{c}_i) \ge 1-\alpha \quad 0 \le \alpha < 1$), the laxity, L_i, of the task τ_i, defined in (1), can be obtained at any instant, t.

$$L_i = a_i + d_i - \hat{c}_i - t \tag{1}$$

If the laxity of the task is negative, the probability that task τ_i will fail is given by (2).

$$L_i < 0 \Rightarrow P(\tau_i \ fails) > \alpha \tag{2}$$

Before executing a task, the laxities of all the tasks in the system are calculated. If the laxity of even one task, i.e. τ_i, is negative, the expert system is considered overloaded because the probability of failing when executing τ_i is inadmissible.

Tchebychef's inequality [11] allows us to estimate the worst-case execution time of intelligent tasks using the sample mean and the sample variance of the past execution times of the tasks.

Let $\hat{\mu}_i$ and $\hat{\sigma}_i^2$ be the sample mean and the sample variance of task τ_i after $n+1$ samples of c_i, obtained recursively using (3) and (4).

$$\hat{\mu}_i = \hat{\mu}_{i,n+1} = \hat{\mu}_{i,n} + \frac{c_{i,n+1} - \hat{\mu}_{i,n}}{n+1} \tag{3}$$

$$\hat{\sigma}_i^2 = \hat{\sigma}_{i,n+1}^2 = \left(1 - \frac{1}{n}\right)\hat{\sigma}_{i,n}^2 + (n+1)(\hat{\mu}_{i,n+1} - \hat{\mu}_{i,n})^2 \tag{4}$$

If both $\hat{\mu}_i$ and $\hat{\sigma}_i^2$ are finites, given $k \in \Re, k \geq 1$ Tchebychef's inequality gives a lower bound of the probability that c_i belongs to an interval centered in $\hat{\mu}_i$ and with radius $k\hat{\sigma}_i$, as holds (5).

$$P(\hat{\mu}_i - k\hat{\sigma}_i \leq c_i \leq \hat{\mu}_i + k\hat{\sigma}_i) \geq 1 - \frac{1}{k^2} \tag{5}$$

From (5), (6) and (7) immediately follow.

$$P(c_i \leq \hat{\mu}_i + k\hat{\sigma}_i) - P(\hat{\mu}_i - k\hat{\sigma}_i \leq c_i) \geq 1 - \frac{1}{k^2} \tag{6}$$

$$P(c_i \leq \hat{\mu}_i + k\hat{\sigma}_i) \geq 1 - \frac{1}{k^2} + P(\hat{\mu}_i - k\hat{\sigma}_i \leq c_i) \tag{7}$$

As $P(\hat{\mu}_i - k\hat{\sigma}_i \leq c_i) \geq 0$, (8) can be obtained from (7).

$$P(c_i \leq \hat{\mu}_i + k\hat{\sigma}_i) \geq 1 - \frac{1}{k^2} \tag{8}$$

That is, for each value of $k \in \Re, k \geq 1$, computation time c_i of task τ_i is known to be lower than $\hat{\mu}_i + k\hat{\sigma}_i$ with a probability of $1-1/k^2$. From (8) we also get (9), where the value $1-\alpha$ is the confidence level of the bound.

$$P(c_i \leq \hat{\mu}_i + k\hat{\sigma}_i) \geq 1 - \frac{1}{k^2} = 1-\alpha \quad k \geq 1,\ 0 \leq \alpha < 1 \tag{9}$$

So, the worst-case computation time, $\hat{c}_i$, of task τ_i can be estimated using (10) with a confidence level of $1-\alpha$.

$$\hat{c}_i = \hat{\mu}_i + k\hat{\sigma}_i = \hat{\mu}_i + \alpha^{-1/2}\hat{\sigma}_i \tag{10}$$

Tchebychef's inequality makes no suppositions about probability distribution of computation time, so bounds obtained are usually pessimistic. As (8) also neglects the value of $P(\hat{\mu}_i - k\hat{\sigma}_i \leq c_i)$, bounds are even more pessimistic. Thus, it is not necessary to choose high values of k to obtain bounds of adequate confidence. We have determined empirically that a value of $k = 2$ or, what is the same, a confidence level of $1-\alpha = 0.75$ are adequate for most cases.

3.2. Scheduling policy during overloads

If the system is overloaded, EDF must be replaced by a more adequate scheduling policy. We have observed that during overloads the result of executing a task with low probability of success is even worse for non-preemptive than for preemptive systems. If a task cannot be preempted once scheduled, all the tasks of the system may miss their deadlines if their laxity is small. For this reason, during overloads tasks are run taking their probability of success into account.

The benefit of executing tasks must also be taken into account, as it can not be the same for all tasks. Thus, in some situations it is preferable to execute a task with a lower probability of success but which will provide greater benefit if it succeeds. Locke [12] observed experimentally that running the tasks with the greatest values of the ratio u_i / c_i allows the system to achieve a utility at least as high as with any other policy.

The dynamic priority of the tasks, p_i, obtained using (11) summarizes both factors, and so was chosen as the policy to select tasks to be run during overloads.

$$p_i = \frac{L_i + \hat{c}_i}{\hat{c}_i} \cdot \frac{u_i}{\hat{c}_i} \tag{11}$$

Tasks with high utility and high (positive) laxity have high values of p_i. Tasks with low utility and low (negative) laxity have values of p_i near zero: a task remains in the system until its deadline is missed, and at that moment $L_i = a_i + d_i - \hat{c}_i - t = -\hat{c}_i \Rightarrow L_i + \hat{c}_i = 0$.

4. Results

The results achieved using the scheduling policy presented in this work are measured using two performance metrics, the *completed task ratio* CTR (the count of tasks completed before their deadlines divided by the total number of task arrivals) and the *effective processor utilization* EPU (the time consumed executing tasks completed before their deadlines, divided by the total time consumed). Only these two measurements have been included as in most cases the number of tasks completed and the value of the effective processor utilization is enough to analyze the behavior of a system during overloads [16]. The results presented are obtained by assigning the same constant utility of 1 to all the tasks in the system. So another broadly used metric, the *hit value ratio* [10], is not included as it coincides with the completed task ratio.

Results are obtained for various sets of tasks, composed of a number of tasks, n, of 10, 25 or 50 intelligent tasks. Only results for the case of $n = 10$ are presented (see Fig. 2), as they can be safely extrapolated to the cases of 25 and 50 tasks. Computation time of tasks are random variables distributed following an Erlang distribution [17], whose probability density function is shown in (12), with $k = 2$ and variable mean value $\overline{c}_i = k / \lambda_i$.

$$f(c_i) = \frac{\lambda_i^{\,k} c_i^{\,k-1} e^{-\lambda x}}{(k-1)!} \quad c_i \geq 0 \tag{12}$$

Mean computation time of tasks, $\overline{c}_i$, varies uniformly from a value of 1 to a value of $\overline{c}_{\max}$, where $\overline{c}_{\max}$ takes the values 1, 10, 25 or 50. Relative deadlines of tasks, d_i, are defined by (13).

$$d_i = \overline{c}_i + 4 \cdot \overline{c}_i \tag{13}$$

The mean load of the system, $\overline{L}$, (see (14)), ranges from a value of 0.5 to a value of 1.5, where f_i is the arrival rate of each intelligent task.

$$\overline{L} = \sum_i \overline{c}_i f_i \tag{14}$$

For each value of n, $\overline{c}_i$ and $\overline{L}$ arrival rates of tasks, f_i, are obtained using (15).

$$f_i = \frac{\overline{L}}{\overline{c}_i \cdot n} \tag{15}$$

Each intelligent tasks contributes equally to the load of the system with a demand value of $\overline{L} / n$.

The results are obtained from simulations of 180000 time units in length, divided in 30 batches of 6000 time units each, which assures a significance level of the measurements presented greater than 90%. Figures include results obtained by scheduling tasks following our scheduling policy, EDF policy and maximum expected

$u_i / \hat{c}_i$ policy (or maximum value density, MVD). EDF has been included for reference, and it is very important to take into account the fact that tasks are removed from the system when they miss their deadline, which dramatically improves EDF performance during overloads.

Comparing the behavior of the new algorithm with the behavior of the EDF algorithm, the validity of overload detection can be affirmed, as the new algorithm behaves as well as EDF when the mean load is low, and degrades gracefully when the mean load increases. It can also be observed that the new algorithm achieves nearly the same value of CTR as the MVD algorithm, while the EPU is always better.

The better behavior of the MVD can be explained by analyzing the CTR of tasks individually for high values of the mean load, shown in Fig. 1. With $n = 10$, $\overline{c}_{max} = 10$, and values of mean load $\overline{L} = 1.25$ and $\overline{L} = 1.5$, MVD shows preference for tasks with short computation time (dashed lines), which improves the count on completed tasks with respect to the new algorithm (continuous lines), which improves the balance between the number of tasks with long and short computation time scheduled.

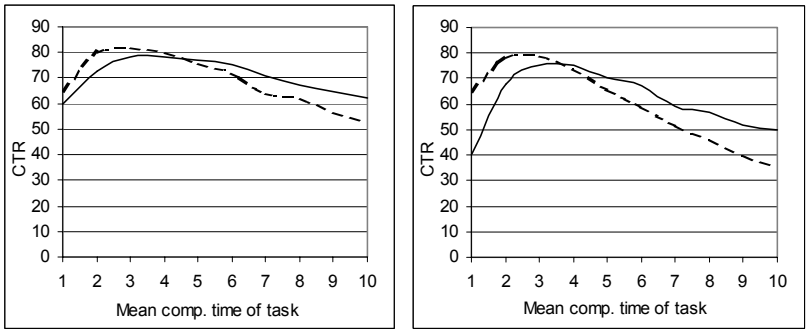

Fig. 1. CTR for each task of the set with $\overline{L} = 1.25$ (left) and $\overline{L} = 1.5$ (right).

Conclusions

This work presents a new scheduling algorithm designed to be coupled with a specific real-time expert system architecture, whose main characteristic is the serialization of inferences. The algorithm exploits the observation that in non-preemptive systems deadline misses cause high performance degradation and the well-known properties of the MVD scheduling. As its performance analysis shows, the algorithm achieves highly effective utilization of resources, and nearly as high a task completion ratio as the MVD algorithm.

References

1. Barachini, F., "Frontiers in Run-Time Prediction for the Production-System Paradigm", *AI Magazine*, Vol. 15, No. 3, pp. 47-61, Fall 1994.
2. Wang, R., H. and Mok, A. K., "Response-Time Bounds of Rule-Based Programs under Rule Priority Structure", *Proc. IEEE Real-Time Systems Symposium*, pp. 142-151, 1994.
3. Cheng, A. M. K and Chen, J., "Response Time Analysis of OPS5 Production Systems", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 12, No. 3, pp. 391-498, May-Jun 2000.
4. Forgy, C. L., "On the Efficient Implementacion of Production Systems", *PhD Thesis, Carnegie-Mellon University*, 1979.
5. Lenstra, J. K. and Rinnooy Kan, A. H. G., "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey", *Annals of Discrete Mathematics*, Vol. 5, pp. 343-362, 1977.
6. George, L., Rivierre, N. and Spuri, M., "Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling", *Rapport de Reserche RR-2966, INRIA*, Le Chesnay Cedex, France, 1996.
7. Jeffay, K., Stanat, D. F. and Martel, C. U., "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks with Varying Execution Priority", *Proc. IEEE Real-Time Systems Symposium*, pp. 129-139, 1991.
8. George, L., Muhlethaler, P. and Rivierre, N., "Optimality and Non-Preemptive Scheduling Revisited", *Rapport de Reserche RR-2516, INRIA*, Le Chesnay Cedex, France, 1995.
9. Locke, C. D., "Best Effort Decision Making for Real-Time Scheduling", *PhD Thesis, Carnegie-Mellon University, Computer Science Department*, Pittsburgh, PA, 1986.
10. Butazzo, G., *Hard Real-Time Computer Systems, Predictable Scheduling Algorithms and Applications*, Chapter 8, Kluwer Academic Publishers, 1997. ISBN: 0-7923-9994-3.
11. Hardy, G., Littlewood, J. E. and Pólya, G., "Tchebychef's Inequality", *Inequalities, Second Edition*, *Cambridge Mathematical Library*, pp. 43-45 and 123, Feb. 1998. ISBN 0-521-35880-9.
12. Jensen, E., D., Locke, C. D. And Tokuda, H., "A Time-Driven Scheduling Model for Real-Time Operating Systems", *Proc. IEEE Real-Time Systems Symposium*, pp. 112-122, 1985.
13. Gordon, A., *The COM and COM+ Programming Primer*, Prentice Hall, 2000. ISBN: 0130850322.
14. Shepherd, G., "COM Apartments", *Visual C++ Developers Journal*, Vol. 2, N. 1, February/March 1999.
15. Gärdenfors, P., and Rott, H., "Belief Revision", in Gabbay, D., Hogger, C. J. and Robinson, J. A., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 4, pp. 35-132. Clarendon Press, Oxford.
16. Baruah, S. K., Haritsa, J. and Sharma, N., "On-Line Scheduling to Maximize Task Completions", *Proc. IEEE Real-Time Systems Symposium*, pp. 228-236, 1994.
17. Erlang, A. K., "The Theory of Probabilities and Telephone Conversations", *Nyt Tidsskrift for Matematik B*, Vol. 20, 1909.

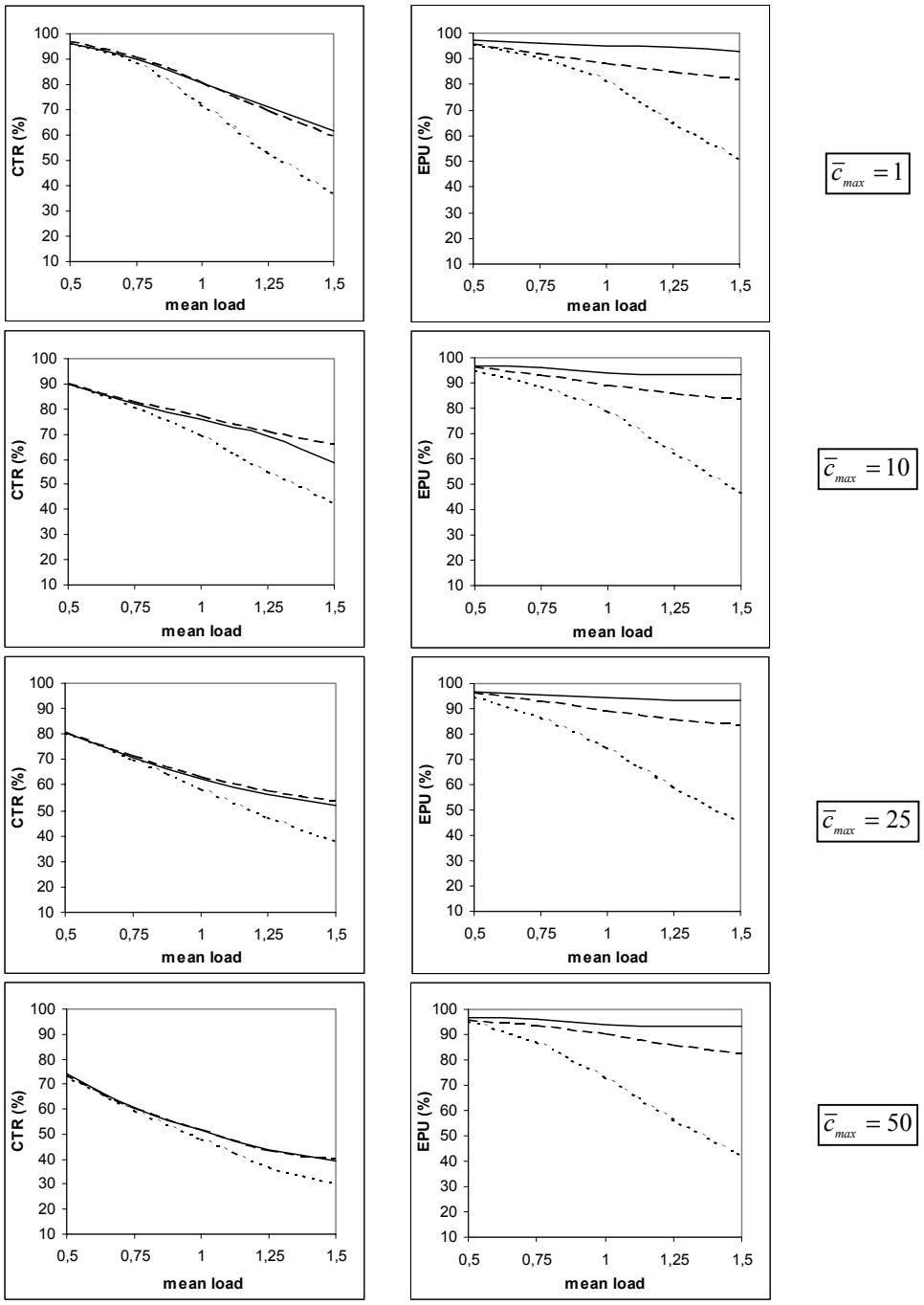

Fig. 2. Performance comparison of the new scheduling algorithm (continuous lines), EDF dotted lines) and MVD (dashed lines) for $n = 10$ tasks.