Improving the performance of local real-time scheduling

Gabriel A. Wainer

Computer Sciences Department.
Facultad de Ciencias Exactas y Naturales.
Universidad de Buenos Aires.
Charcas 3960 7mo. 43 (1425).
Buenos Aires. Argentina.
PHONE: (054)1-832-2023. FAX: (054)1-783-0729.
E-MAIL: gabrielw@dc.uba.ar

Abstract

In this work we show the results of two simple strategies to improve the performance of local real-time schedulers. The main goals are to increase system stability under transient overloads and to improve the use of system resources when tasks have stochastic execution times. To do so, we use traditional schedulers combined with a multiple queue algorithm and an on-line guarantee test. Results obtained through simulation allowed to detect improvements in system time loading and stability.

Keywords: Real-Time scheduling.

1. INTRODUCTION

In Real-Time systems the moment when a result is computed is as important as its logical correctness. One way to meet systems' timing constraints is to rely on a real-time scheduler. The scheduler should study system predictability, but the diversity of restrictions in these systems makes it an NP-hard problem.

There are different ways used to lower the complexity of the guarantee tests. Several solutions consider all the system's restrictions (including timeliness, criticality, precedence, concurrence, communication, and so on) and employ heuristics to reduce the search time. Another approaches use simpler task models to solve less generic problems (most of the local schedulers use this policy). These simple task models are usually improved to solve new problems, but the extra complexity (and overhead) makes the solutions impractical. In this work we present simple techniques to improve the performance of traditional local real-time schedulers without adding much overhead. The policies can be combined with different existing task models, and can be used with static or dynamic schedulers, but the schedulability analysis must be on-line.

One goal is to improve stability allowing the execution of the most crucial tasks when the system is overloaded. We also want to increase resource use when the execution times are below the worst cases.

2. IMPROVING STABILITY

Several scheduling algorithms rely on a well-known task model, usually called the **periodic task's** model. It considers the existence of two different kinds of tasks to schedule: the **periodic** and **spo-**

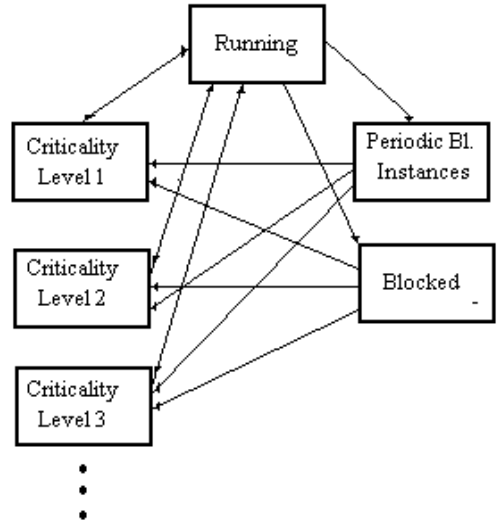
radic (aperiodic) ones. Periodic tasks have a continuous series of regular invocations (whose time is called the **period**), and a **worst case execution time** (WCET). The task **deadline** is at the beginning of the next period. Aperiodic tasks have arbitrary **arrival time** and deadline, and a known WCET. When they are invoked, they execute only one instance. Some widely used algorithms with this model include Rate Monotonic (RM), Earliest Deadline First (EDF) [Liu73], and Least Laxity First (LLF) [Mok79].

A transient overload exists when no schedule can meet the timing constraints of every task in the system. Algorithms such as EDF or LLF have erratic behavior under transient overloads, because they choose arbitrary tasks, avoiding the completion of the most crucial ones. Other algorithms (for instance, RM) are stable, but presume that the most critical tasks are those with short activation period. There are cases where this is not adequate, as we need to run highly crucial tasks with long activation frequency. Hence, our first goal is to selectively lock tasks with little criticality to improve the success of the most crucial tasks.

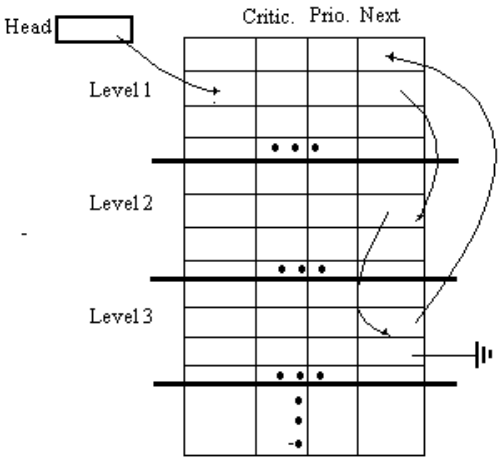
Usually, real-time tasks are classified in **hard** and **soft** real-time. For hard real-time tasks, to meet the timing constraints is crucial (loosing deadlines may lead to catastrophe). Soft real-time deadlines can be missed occasionally with only a degradation in performance (the system is still operative). Recently, the term **firm** real-time has been defined to include hard real-time systems that can tolerate to miss deadlines with low probability [Lap93]. Our goal will be to insure the timely execution of the harder real-time tasks under transient overloads, using a periodic task model.

First, we use any of the existing strategies (including, for instance, value functions) to give a criticality level to each task. Then, we use a multiple queue algorithm to classify tasks according their criticality. Hard real-time tasks are kept in the first queues, and the soft ones, in the low levels, reducing the overhead for queue management. When the system is not overloaded, tasks are chosen using a traditional scheduler. The multiqueue is implemented as a container divided in levels, where tasks also are chained using a traditional basic algorithm.

If the system is normally loaded, we use the base scheduler. When an overload is detected, we use the multiple queue, choosing tasks from the higher levels. Tasks in the lower levels are chosen when the upper levels are empty. As the least critical tasks are kept in the lower level queues, they are automatically delayed. When the overload finishes, we return to the base algorithm, and the delayed tasks are restored. The main advantage of this strategy is the little overhead introduced to lock the tasks to delay.



(a)



(b)

Figure 1. Basic structure of the scheduler. (a) Multiqueue structure (used when there is an overload); (b) Chained Process Control Blocks divided in multiple queues.

To insure predictability, we choose the most critical tasks using the basic guarantee tests, and keep them on the high level queues. In this way, the critical task set can run predictably. System time loading is kept to the maximum, because if there are no tasks in the high level queues, those in the lower levels are chosen. When there is an overload, a new

task will be accepted only if it fits in the critical class and the new critical set is still schedulable.

The least critical tasks could be sent to other processors (if available) to insure their execution. We also could run alternative imprecise tasks using less processor time. The early detection of timing failures allows to have enough time for these procedures, meeting a higher number of deadlines.

One problem is how to divide the ready queue in classes. The simplest way is to leave this decision to the designer. At present we are experimenting with different strategies to classify the system tasks (they will be included in the final version).

3. SCHEDULING TASKS WITH VARIABLE COMPLETION TIME.

The periodic task models use the WCET to study the schedulability of the task sets. Even there are techniques to estimate this time accurately due to program structure the time can be over the actual execution time.

Hence, if we use the WCET, processor time can be wasted. A task set can be highly underloaded but a new task can be rejected by the guarantee tests. We made experimental studies of tasks' execution time and could see that it behaves as a random process with Gamma distribution (similar results can be found in [Tia95]). We will increase processor use an on-line schedulability test considering stochastic execution times.

Let us call

. w, to the total time of a time window, equal to the l.c.m. of the periodic task's periods;
. **T**, the beginning of the time window;
. **t** Î [T, T+w], the instant where the schedulability analysis is done;
. *t* = {τ1, ..., τn}, a task set consisting of n tasks;
. **Tj**, a task period (or deadline in the case of aperiodic tasks);
. **Cj** ≤ **Tj,** a WCET (**Cij,** the execution time of the i-eth instance of task τj);
. e**(t)** = {j / instance τj finished in time **t** } = { j / j ∈ [0, ⌊ **t** / **Tj**⌋ where τj is a periodic task} U { j / τj is a sporadic task }, set of instances finished on time **t**; and
. **Cij'** ≤ **Cij**, the actual execution time of the i-eth instance of task τj (run before **t**).

Hence, if

$$\sum_{i \in e(\mathbf{t}),\ j \in [1,n]} Cij' \ + \ \sum_{j \in [1,n]} qj \bullet Cj \ > \ \varpi$$

(where qj = 1 if τj is an aperiodic task to run in the window but not yet completed, and qj = ⌈ w / **Tj** ⌉ - ⌊ **t** / **Tj** ⌋ for periodic task τj in instant t), the task set is not schedulable.

The first term represents the total time executed by instances of τj up to the instant **t**. The second term is total of WCETs from **t** up to the end of the window. If this inequality holds, there is at least one instance (the last in the window) that cannot meet its deadline. Hence, the task set is not schedulable. For Earliest Deadline algorithms, this is a sufficient and necessary condition for schedulability [Wai96].

The routine can be used for both sporadic and periodic tasks. The strategy is specially useful to include sporadic tasks in overloaded systems. When a sporadic instance arrives, the computations made using the actual processor time could admit it and run it predictably.

To allow predictability for hard real-time tasks, we must combine this technique with other approaches. For instance, we could use a probabilistic schedulability test, such as the one presented in [Tia95], and reject the tasks with high chance to lose deadlines. In the present case, we will lock the least critical tasks using the strategies of section 2. When both strategies were compound, the system achieved higher resource use and number of deadlines met.

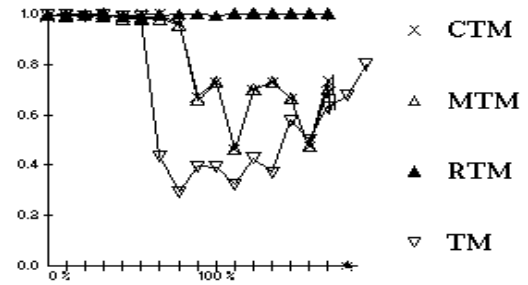
4. RESULTS

The stated strategies were applied to the three traditional algorithms mentioned in section 2, using the schedulability tests provided by those approaches. The algorithms were tested using AgaPé-TR, a tool to analyze real-time schedulers [Wai96]. We used this tool to generate two kinds of load. The first one used a Gamma distribution for execution times with a mean close to zero. The second one used a Gamma distribution with a mean close to the WCET.

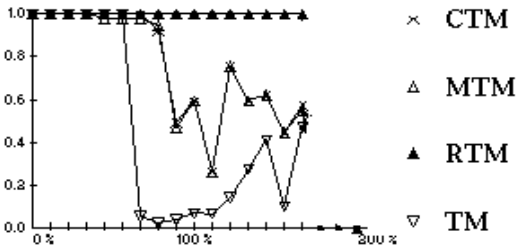
The tool simulates different task loads and collects information about several metrics. To do so, it uses an experimental frame generating task loads using the Hartstone benchmark [Wei89]. The metrics include the relative success ratio (relationship between the successful instances and total instances), time loading (percentage of useful processing), preemptions, context switches and idle

processor time. The stability is analyzed computing the relative guarantee ratio weighted by the tasks' criticality.

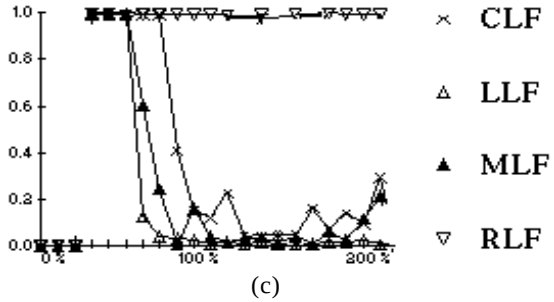
Below, we can see some figures related with the results obtained by simulation. M is the simple Multiqueue algorithm and C is the M one with stochastic execution times close to the WCET. R is the combination between M and random execution times close to zero. Each figure represents a different metric with a variable degree of offered system loading (represented on the X axis). The offered load measures the load as declared by the system designer. We studied the worst case values for each case. When considering the guarantee ratios, we use the minimum ones; when studying measures related with the system overhead, the maximum cases are presented.



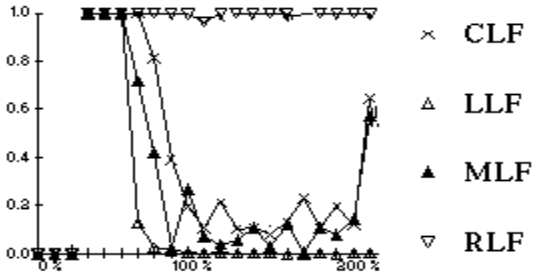
(a)



(b)

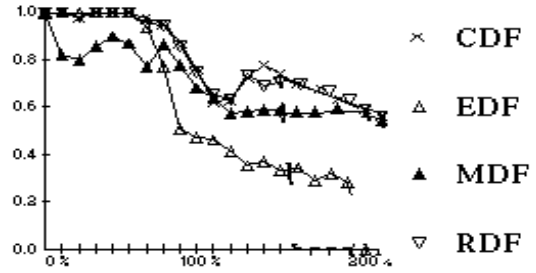


(c)

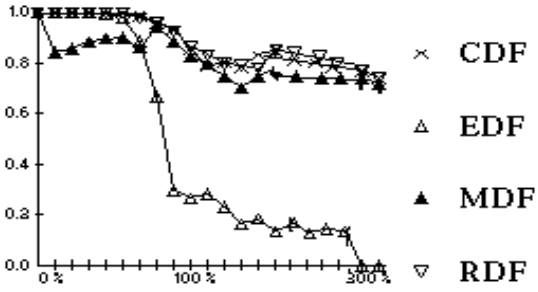


(d)

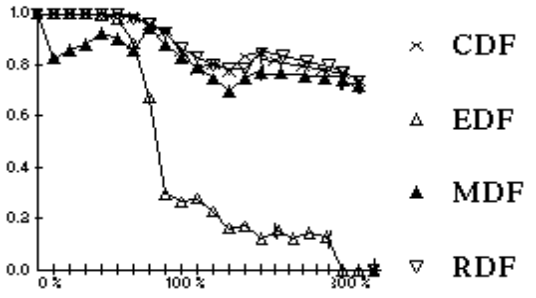
Figure 7. Guarantee ratio behavior. (a) Periodic Harmonic tasks. (b) Periodic Harmonic tasks (weighed G.R.). (c) Periodic Non Harmonic tas ks. (d) Periodic Non Harmonic tasks (weighed G.R.). (TM: Rate Monotonic; LF: Least Laxity First).



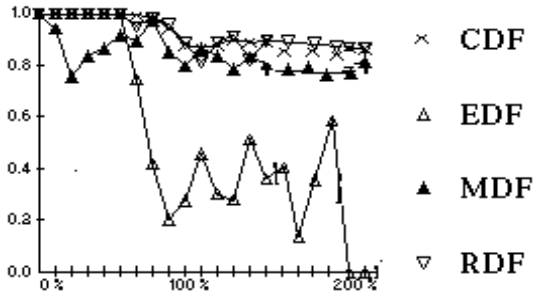
(a)



(b)

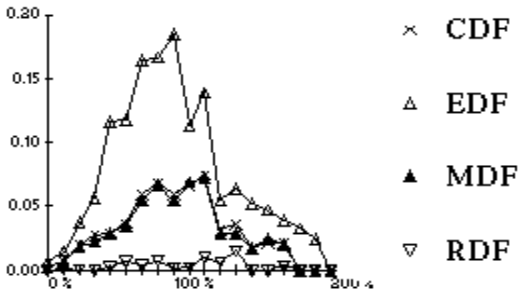


(c)

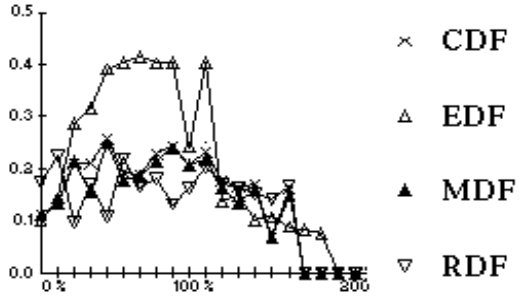


(d)

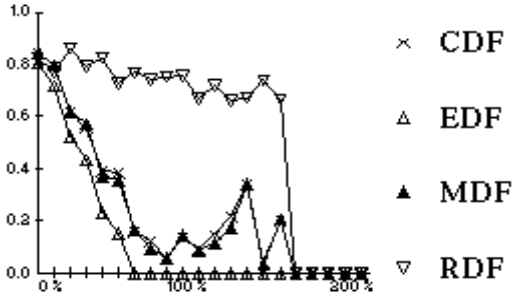
Figure 8. Guarantee ratio behavior. (a) Periodic and Aperiodic Non Harmonic tasks. (b) Periodic and Aperiodic Non Harmonic tasks (weighed G.R.). (c) Only Periodic Non Harmonic tasks (weighed G.R.). (d) Only Aperiodic Non Harmonic tasks (weighed G.R.). (DF: Earliest Deadline First).



(a)



(b)



(c)

Figure 9. Overhead related measures. (a) Pre-emption percentage. (b) Context switch percentage. (c) Idle time. (DF: Earliest Deadline First).

As we can see, the R versions behave better in every case. This is an obvious result, as we run tasks leaving idle time. In this case, the new strategy allows to include a higher number of tasks and to meet their deadlines. We also can see there is almost no difference between C and M versions, specially in cases when running harmonic periodic tasks. The difference is more noticeable when running non harmonic tasks. This happens because when we run periodic harmonic tasks a 100% of guarantee ratio can be achieved. As the M algorithm selects the most critical tasks, the processor is loaded up to 100%. This is not true for the non harmonic cases where there are more tasks delayed (and several deadlines missed).

In every case, the difference increases when we study the weighed guarantee ratio. This is another obvious result, as the solutions are oriented to increase this metric in overloaded systems. The differences are higher when we combine our strategies with unstable algorithms such as EDF or LLF.

In some cases the guarantee ratio of the M version behaves worse than the base one (this is not true for the weighed ratio). This problem occurs because we have used pure criticality values without paying attention to task periods or deadlines. This problem can be avoided using techniques such as Sporadic Servers [Spr89] or value functions (BEF, D*, and others [Mar95]).

Context switches and preemptions have been highly reduced. A singular behavior can be seen in Figure 9: task preemptions and context switches reduce when the overload increases. This happens because when the task set is highly overloaded there are several tasks that cannot even start its execution, reducing the number of preemptions and context switches.

Finally, we have studied the idle time available. Again the R version provided good results and the behavior of M and C solutions are alike. They also produced higher idle time than the original algorithms. When techniques to run alternate routines are employed, the overhead introduced by these approaches can be reduced as we will have more time to run mandatory tasks.

5. PRESENT WORK AND CONCLUSION.

In this work we have presented schemes to improve the performance of traditional schedulers. To test them, we have mixed the new policies with

traditional algorithms with significant improvements.

The goal of our policies was to increase processor allocation, enhancing the merit of local algorithms. System predictability could be increased, attaining to a higher number of time critical tasks meeting their deadlines. Besides, the time loading also increased, permitting the completion of task sets that, otherwise, could not have been executed. Finally, system stability was also enhanced. The association of both strategies presented allowed to meet these goals.

At present we are studying the influence of running tasks with precedence constraints. We are also simulating the use of different value functions to select the most critical tasks combined with the stochastic task model. We also have started to study how to mix the strategies with fault-tolerant scheduling techniques.

6. REFERENCES

[Lap93] LAPLANTE, P. "Real-Time systems. Design and analysis. An engineer's handbook". IEEE Press. 1993.

[Liu73] LIU, C.; LAYLAND, J. "Scheduling algorithms for multiprogramming in a Hard Real Time System Environment". Journal of the ACM, Vol. 20, No. 1, 1973, pp. 46-61.

[Mar95] MARTINEAU, P.; SILLY, M. "On-line scheduling algorithms for overloaded Real-Time Systems". Proceedings of the 3rd. IFAC/IFIP Workshop of Algorithms and Architectures for Real-Time Control. (5) 1995.

[Mok78] MOK, A.; DERTOUZOS, M. "Multiprocessor scheduling in a hard real-time environment". *Proceedings of the Seventh Texas Conference on Computing System*. 1978.

[Spr89] SPRUNT, B.; SHA, L.; LEHOCZKY, J. "Aperiodic task scheduling for hard real-time systems". J. Real-time systems, Vol. 1, No. 1, 1989. pp. 27-60.

[Tia95] TIA, T.; DENG, Z.; SHANKAR, M.; STORCH, M.; SUN, J.; WU, L.; LIU, J. "Probabilistic performance guarantee for real-time tasks with varying computation times". Proceedings of the 1995 IEEE Real-Time technology Applications Symposium. Chicago, Illinois. May 1995.

[Wai96a] WAINER, G. "AgaPé-TR: a tool to simulate real-time scheduling algorithms". Submitted to AARTC '97. (1996).

[Wai96b] WAINER, G. "Techniques to improve local real-time scheduling". Internal report. FCEN-UBA. (1996).

[Wei89] WEIDERMAN, N. "Hartstone: synthetic benchmark requirements for Hard Real-Time applications". Technical Report CMU/SEI-89-TR-23. Carnegie Mellon University. Software Engineering Institute.