IMPROVING THE PERFORMANCE OF LOCAL REAL-TIME SCHEDULING

Gabriel A. Wainer

Computer Sciences Department.
Facultad de Ciencias Exactas y Naturales.
Universidad de Buenos Aires. Pabellón I - Ciudad Universitaria.
Buenos Aires - Argentina.
gabrielw@dc.uba.ar
http://www.angelfire.com/ga/gw

Abstract: In this work the results of two simple strategies to improve the performance of local real-time schedulers are presented. 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, traditional schedulers are 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 algorithms, stability, stochastic parameters, performance analysis.

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 ensure system predictability, but the restrictions in these systems are so diverse that this guarantee is an NP-hard problem.

There are different ways used to lower the complexity of the guarantee tests. Some solutions consider all the system's restrictions (including timeliness, criticality, precedence, concurrence, communication, etc.) and use heuristics to reduce the search time. Other 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 at the cost of extra complexity (and overhead). In this work low overhead techniques to improve the performance of local real-time schedulers are presented. The policies can be combined with different existing task models, and can be used with static or dynamic schedulers.

One goal is to improve stability allowing the execution of the most crucial tasks when the system is overloaded. Another one is to increase resource use when 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 sporadic (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) (Liu and Layland, 1973), and Least Laxity First (LLF) (Mok and Dertouzos, 1979).

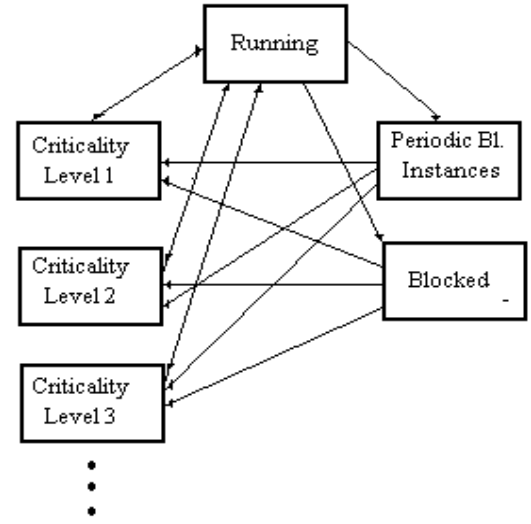
It is said that a transient overload occurs when no schedule can meet the timing constraints of every task in the system. Algorithms such as EDF or LLF have erratic behavior under transient overloads, because they choose arbitrary tasks, avoiding the completion of the most crucial ones. Other algorithms (for instance, RM) are stable, but presume that the most critical tasks are those with short activation period. In some cases this presumption is not adequate, as highly crucial tasks can have long activation frequency. Hence, the 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 deadline missing with low probability (Laplante, 1993).

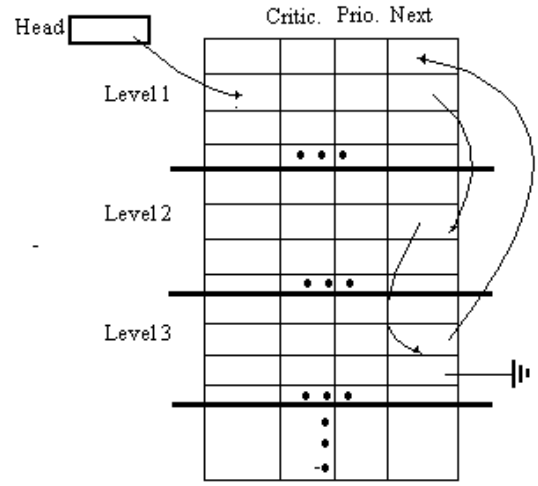
The strategies presented use a periodic task model, insuring timely execution of firm real-time tasks under transient overloads. The first step is to give a criticality level to each task, using, for instance, value functions. This kind of functions can also ensure predictable execution of hard real-time tasks. Then, a multiple queue algorithm is used to classify tasks according their criticality. Real-time tasks with hard deadlines are kept in the first queues, and the soft ones in the low levels, reducing the overhead for queue management.

The multiqueue is implemented as a container divided in levels, where tasks also are chained using a traditional basic algorithm. When the system is not overloaded, tasks are chosen using the traditional scheduler. When an overload is detected, the multiple queue is used, choosing tasks from the higher levels (lower levels are chosen when the upper are empty).

As the least critical tasks are kept in the lower level queues, they are automatically delayed. If the overload finishes, the base algorithm is restored (delayed tasks automatically wake up). The main advantage of this strategy is the little overhead introduced to lock the tasks to be delayed.



(a)



(b)

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

To insure predictability, the most critical tasks are chosen using the the base algorithm. Timely execution of the most critical tasks can be guaranteed if the critical task set must is kept on the high level queues. To do so, the guarantee tests of the basic algorithms can be used, testing different levels of predictability according with the tasks assigned to each queue.

System 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. Another approach is to

run alternative imprecise tasks with a more relaxed WCET. The early detection of timing failures allows to have enough time to execute these procedures, meeting a higher number of deadlines.

At present we are experimenting with different strategies to classify the system tasks, and to divide the ready queue in classes. We are also testing different approaches to choose the most critical tasks in the set.

3. SCHEDULING TASKS WITH VARIABLE COMPLETION TIME.

The periodic tasks model use the WCET to provide schedulability guarantees. At present there are several techniques that can be used to estimate this time accurately. In most cases, program structure and architectural factors can overstimate this time.

As many guarantee tests use the WCET to compute guarantees, processor time can be wasted. A task set could be highly underloaded and a new task would be rejected by the schedulability tests. Experimental studies of tasks' execution time were made, showing that this parameter behaves as a random process with Gamma distribution (similar results can be found in Tia, *et al.*, 1995). Processor use can be increased through 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** ∈ [T, T+**w**], the instant where the schedulability analysis is done;
. **τ** = {t1, ..., tn}, a task set consisting of n tasks;
. **Tj**, a task period (or deadline in the case of aperiodic tasks);
. **Cj** ≤ **Tj,** the WCET of the task (**Cij,** the WCET of the i-eth instance of task tj);
. e(t) = {j / instance tj finished in time **t** } = { j / j ∈ [0,⌊ **t / Tj**⌋ where tj is a periodic task} U { j / tj 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 tj (run before **t**).

Hence, if

$$\sum_{i \in \varepsilon(t),\, j \in [1,n]} Cij + \sum_{j \in [1,n]} qj \bullet Cj > \varpi$$

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

The first term represents the total time executed by instances of tj 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 (Wainer, 1996).

The test 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, this technique must be combined with other approaches. For instance, a probabilistic schedulability test, such as the one presented in Tia, *et al.* (1995), and reject the tasks with high chance to loose deadlines. In the tests made, the least critical tasks are locked 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 (Wainer, 1997). This tool was used to generate two load classes. 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 following the Hartstone benchmark (Weiderman, 1989). 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, some figures of the simulation results are presented. Here, 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).

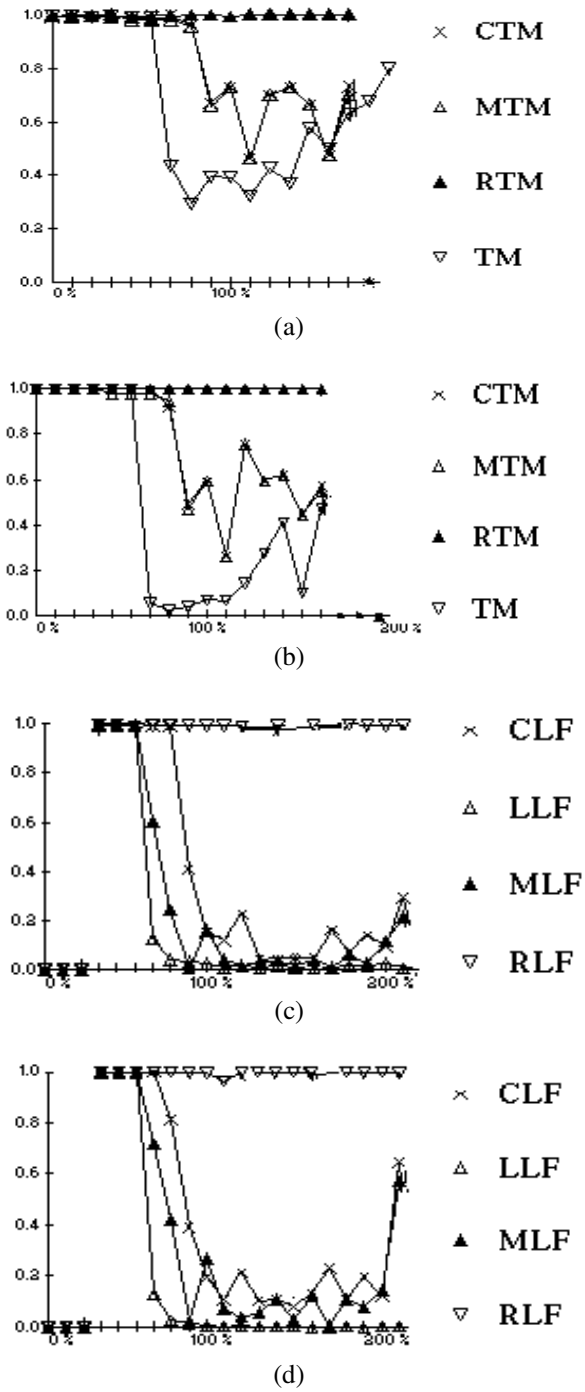


Figure 2. Guarantee ratio behavior. (a) Periodic Harmonic tasks. (b) Periodic Harmonic tasks (weighed G.R.). (c) Periodic Non Harmonic tasks. (d) Periodic Non Harmonic tasks (weighed G.R.). (TM: Rate Monotonic; LF: Least Laxity First).

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.

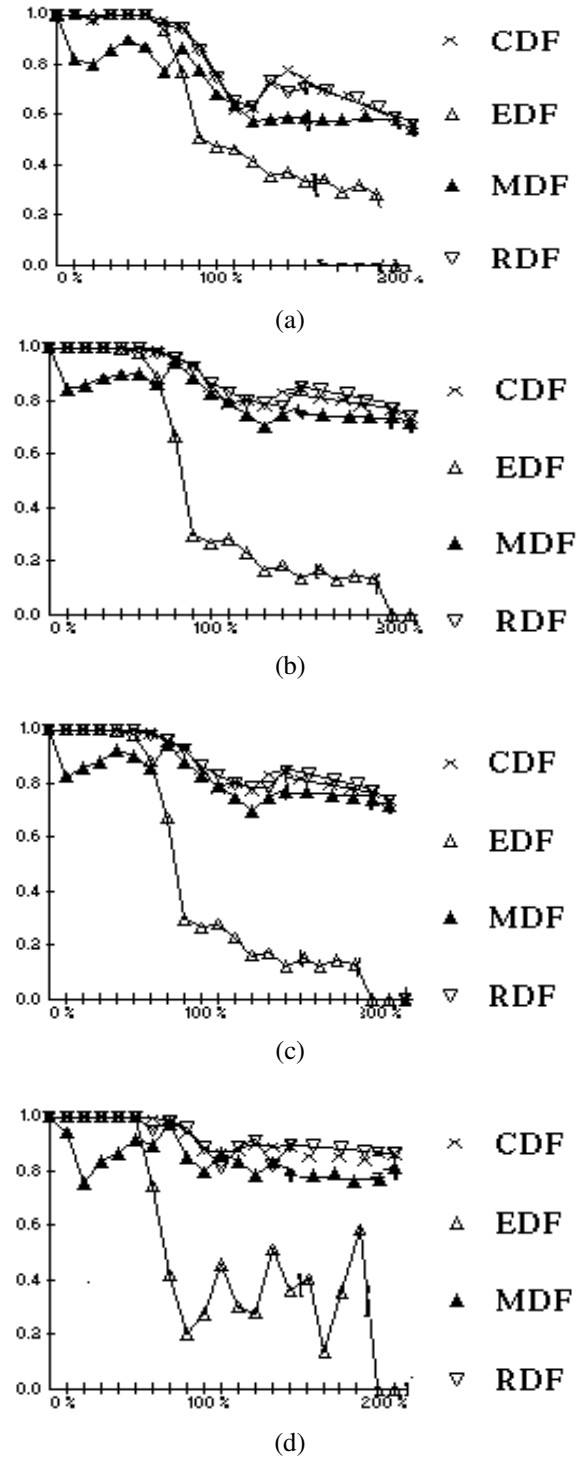


Figure 3. Guarantee ratio behavior. (a) Periodic and Aperiodic Non Harmonic tasks. (b) Periodic and Aperiodic Non Harmonic tasks (weighed G.R.). (c) Only Periodic Non Harmonic tasks (weighed G.R.). (d) Only Aperiodic Non Harmonic tasks (weighed G.R.). (DF: Earliest Deadline First).

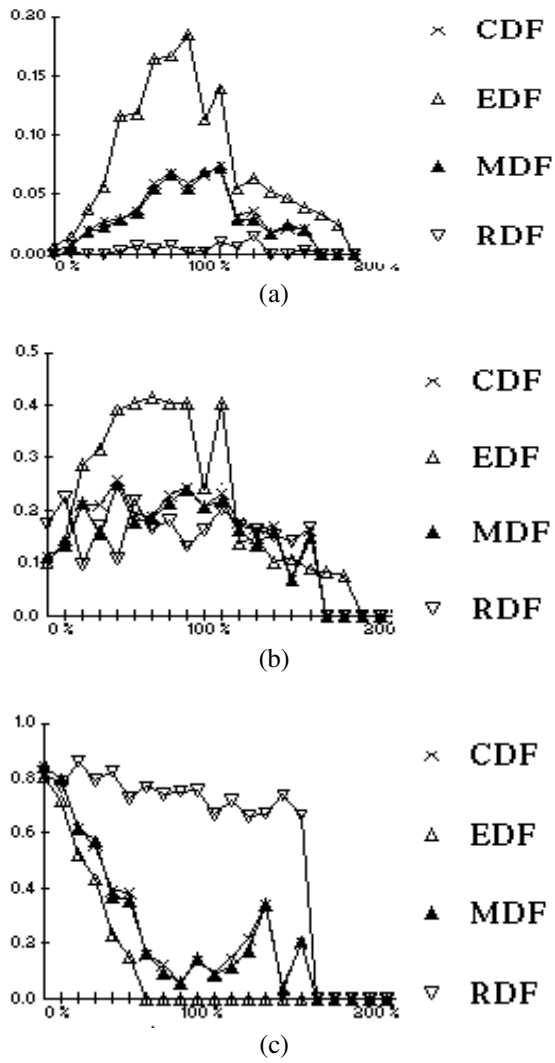


Figure 4. Overhead related measures. (a) Preemption percentage. (b) Context switch percentage. (c) Idle time. (DF: Earliest Deadline First).

As we can see, the R versions behave better in every case. This is an obvious result, as the tasks leave idle time. In this case, the new strategy allows to include a higher number of tasks and to meet their deadlines. It can also be seen that 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, there are noticeable differences when the weighed guarantee ratio is studied. This is another obvious result, as the solutions are oriented to increase this metric in overloaded systems. The differences are higher when the strategies are combined 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 pure criticality values have been used, without paying attention to task periods or deadlines. This problem can be avoided using techniques such as Sporadic Servers (Sprunt, *et al.*, 1989) or value functions (BEF, D*, and others - Martineau and Silly, 1995).

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, the idle time available was studied. 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 more time will be available to run mandatory tasks.

5. PRESENT WORK AND CONCLUSION.

In this work some schemes to improve the performance of traditional schedulers were presented. To test them, the new policies have been mixed with traditional algorithms, obtaining significant improvements.

The goal of these 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 the influence of running tasks with precedence constraints are being studied. The use of different value functions to select the most critical tasks combined with the stochastic task model is also being tested. Finally, the mixture of these strategies with fault-tolerant and probabilistic scheduling techniques have been started to being studied.

REFERENCES

Laplante, P. (1993). *Real-Time systems. Design and analysis. An engineer's handbook*. IEEE Press.

Liu, C. and Layland, J. (1973). "Scheduling algorithms for multiprogramming in a Hard Real Time System Environment". *Journal of the ACM,* **Vol. 20, No. 1**, 46-61.

Martineau, P. and Silly, M. (1995). "On-line scheduling algorithms for overloaded Real-Time Systems". In: *Algorithms and Architectures for Real-Time Control*. (P. Fleming, L. Boullart (Ed.)). 253-258. Elsevier Science Ltd. London, England.

Mok, A. and Dertouzos, M. (1978). "Multiprocessor scheduling in a hard real-time environment". In: *Proceedings of the Seventh Texas Conference on Computing System*.

Sprunt, B., Sha, L. and Lehoczky, J. (1989). "Aperiodic task scheduling for hard real-time systems". *J. Real-time systems,* **Vol. 1, No. 1**, 27-60.

Tia, T., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L. and Liu, J. (1995). "Probabilistic performance guarantee for real-time tasks with varying computation times". In *Proceedings of the 1995 IEEE Real-Time technology Applications Symposium.* Chicago, Illinois.

Wainer, G. (1997). "AgaPé-TR: a tool to simulate real-time scheduling algorithms". *To be published in Proceedings of Algorithms and Architectures for Real-Time Control. Algarve, Portugal. 1997.*

Wainer, G. (1996). "Two simple strategies to improve the performance of local real-time scheduling". *Technical Report No. 96-007, Computer Sciences Department.* Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires.

Weiderman, N. (1989). "Hartstone: synthetic benchmark requirements for Hard Real-Time applications". *Technical Report CMU/SEI-89-TR-23.* Carnegie Mellon University. Software Engineering Institute.