Online Schedulability Tests for Real-Time Systems [*]

George Lima[1] **, Ana Carolina Sokolonski**[1] **, Eduardo Camponogara**[2]

[1]Department of Computer Science (DCC)
Federal University of Bahia (UFBA)
Salvador, BA, Brazil

[2]Department of Automation and Systems Engineering (DAS)
Federal University of Santa Catarina (UFSC)
Florianópolis, SC, Brazil

gmlima@ufba.br,anaanton@ufba.br,camponog@das.ufsc.br

Abstract. *Modern real-time systems have to be adaptable and flexible to deal with non-periodic events, requiring a means of checking for schedulability during execution time. Thus, online schedulability tests are a fundamental building block in the design of such systems. Usually, when such tests are carried out at time t they either consider or not future events that occur after t. Those that take such events into consideration are based on previously derived offline scheduling. Otherwise, they focus only on the jobs that are active at t. Unlike these usual approaches, we describe in this paper two online tests that verify schedulability within a given time interval $[t, t')$ so that future event occurrences are considered but bounded within the interval. The tests are proved correct and are evaluated by simulation.*

1. Introduction

Real-time systems, in the past restricted to the domain of a strictly periodic and highly predictable task model [Liu and Layland 1973], nowadays demand high levels of adaptation and flexibility. Indeed, fault tolerance, system mode changes during execution time, multiversion or energy-aware task models are some examples of the current requirements that make the incorporation of aperiodic tasks into the system increasingly necessary. However, aperiodic tasks must be carefully dealt with since they may cause overload scenarios making the system unpredictable [Liu 2000]. Scheduling approaches to avoiding such a side-effect are of paramount importance.

When aperiodic tasks are soft, i.e. they may miss deadlines, the usual scheduling approach is based on sever mechanisms [Liu 2000]. If such tasks are hard or firm, the system must be equipped with some sort of adaptation technique. This, in turn, requires the use of online schedulability tests that must be carried out upon the release of aperiodic tasks. The goal of these kinds of test is to provide information about the system schedulability so that some means of adaptation can be implemented. For example, if it is detected that some task may miss its deadline upon the activation of an aperiodic task, the system

[*]This work has been funded by CNPq (grant 475851/2006-4), CAPES (PROCAD) and FAPESB (grants 7630/2006 and 3381/2005).

may change to a safe mode of operation or may decide to cancel non-important task according to some definition of importance. In any case, efficient online schedulability tests are fundamental to subsidize such adaptation techniques.

In this paper we describe two online schedulability tests that can be used for supporting uniprocessor real-time systems scheduled according to the EDF policy [Liu and Layland 1973]. These tests take into account a given time interval $[t, t')$ and are designed to ensure that all tasks active in this interval finish their execution within it without missing their deadlines. Both tests are proved correct and are evaluated by simulation.

The remainder of this paper is structured as follows. Section 2 summarizes related work while Section 3 defines the system model and the notation used throughout the paper. The proposed schedulability tests are described in Section 4. Some results of simulation are given in Section 5. Finally, conclusions are drawn in Section 6.

2. Related Work

The design of online schedulability test is a fundamental problem in real-time systems when the support to adaptation and flexibility is taken into consideration. Thus, research on this issue has produced results for several decades. Here we briefly comment on some relevant work that is related to uniprocessor EDF-based systems.

When the system is composed of strictly periodic and non-periodic tasks, one can determine offline the knowledge on slack time available after executing periodic tasks. Then this knowledge can be used to derive online schedulability tests. Several online schedulability tests based on this strategy have been proposed. For example, [Chetto and Chetto 1989] have shown a way of determining slack times offline and have proposed the use of this information for schedulability tests. [Isovic and Fohler 2000] have considered the offline generation of the scheduling for periodic tasks. Then they have shown how to combine the produced schedule with EDF rules for online schedulability tests to deal with aperiodic and sporadic tasks. Since these approaches are based on offline scheduling, the stored information may be large for certain systems.

Some online schedulability tests have also been proposed by some researchers. [Buttazzo and Stankovic 1993] have described a method based on computing residual times upon the release of tasks. If the residual time is positive for all tasks after inserting a newly arriving task, schedulability is ensured. Otherwise, some deadlines may be missed. Based on this information, the authors have proposed strategies to rejecting tasks according to their importance, an attribute that may be given by the user. Recently, [Andersson and Ekelin 2007] have developed an exact online scheduablity test for non-periodic tasks. Their test is derived by calculating the accumulated execution time upon each arrival of aperiodic tasks. When periodic tasks are taken into consideration, slack time information, derived offline, is needed, similar to the approach by Chetto and Chetto.

The schedulability tests presented in this paper are based only on information derived online. Also, unlike the approaches previously mentioned, the problem addressed here focus on determining the schedulability of a system within a given time interval $[t, t')$. Thus, although we consider future periodic and sporadic task arrivals, like it was done by some other approaches [Chetto and Chetto 1989, Andersson and Ekelin 2007], the number of tasks is bounded within $[t, t')$. By doing so, no offline scheduling needs to be derived and sporadic task activation can be considered by assuming their minimum inter-release time.

The problem addressed here should not be confused with the work by other researchers on feasibility intervals [Baruah et al. 1990, Goossens and Devillers 1999]. These are intervals that should be analyzed to ensure that a given real-time system is feasible. Instead, the work described in this paper is to check the system feasibility within a given interval.

3. System Model and Notation

We consider a preemptive uniprocessor system composed of a set of tasks scheduled by EDF [Liu and Layland 1973]. Tasks do not share resources nor have precedence constraints. Each task generates one or more jobs during the system execution. Jobs are uniquely identified, that is, J_i and J_j are distinct jobs of tasks iff $i \neq j$. The release instant of J_i is denoted r_i. If J_i and J_j are released at r_i and r_j, then they have to be executed within the time intervals $[r_i, d_i)$ and $[r_j, d_j)$, respectively, where $d_i = r_i + D_i$ and $d_j = r_j + D_j$. The term D_i represents the relative *deadline* of J_i while d_i is its absolute *deadline*. If J_i and J_j are jobs of the same task, $D_i = D_j$. The scheduling tests described in this paper take the absolute *deadlines* of jobs, and so hereafter we refer to them as *deadlines*. Tasks can be periodic, sporadic or aperiodic. There is a fixed and known inter-release time of consecutive jobs of periodic tasks. For sporadic tasks, only their minimum inter-release times are known while aperiodic tasks may be released at any instant. Each job J_i has a known maximum execution cost, represented by C_i. It is assumed that if J_i is an aperiodic job C_i is known by time r_i.

The set $\Gamma(t, t') = \{J_1, \ldots, J_n\}$ represents the jobs active in the interval $[t, t')$. A job J_i is said to be active in this interval if it was released at $r_i < t'$ and has not yet finished its execution by time t. Note that there may exist more than a job of the same task in $\Gamma(t, t')$. We define $s_i = \max(t, r_i)$ as the time from which the execution of J_i is considered by the scheduling test. Without loss of generality, we assume that $\Gamma(t, t')$ is ordered such that $\forall i < n : d_i < d_{i+1}$ or $d_i = d_{i+1}$ and $s_i \leq s_{i+1}$.

We assume that the system keeps track of the processing time of each active job J_i. Note that this assumption is in line with modern operating systems. Further, we define:

Definition 1. *Let $c_i(t)$ be the time already executed of J_i up to time instant t. The maximum effective execution time of J_i is*

$$C_i(t) = \begin{cases} C_i - c_i(t) & \textit{if } J_i \in \Gamma(t,t) \\ 0 & \textit{otherwise} \end{cases} \tag{1}$$

We assume that the system is schedulable in normal operating conditions, i.e. when no aperiodic tasks are released. However, there may be overload scenarios during some intervals $[t, t')$ upon the release of some aperiodic tasks, caused by, say, error-recovery routines that have to be executed to keep the system correct or in a consistent state, mode change operations etc. Online scheduling tests must verify the system schedulability of $\Gamma(t, t')$ to ensure its predictability.

4. Online Scheduling Tests

The scheduling tests described in this paper give sufficient schedulability conditions to verify that the set of n jobs $\Gamma(t,t')$ is schedulable in the interval $[t,t')$. Schedulability in a time interval is stated as follows:

Definition 2. *A system is schedulable in a time interval $[t,t')$ if all jobs $J_i \in \Gamma(t,t')$ finish their execution by $e_i = \min(d_i,t')$.*

Clearly, verifying the schedulability within $[t,t')$ can be pessimistic if $t' < d_i$ for some job $J_i \in \Gamma(t,t')$. Even so, we think that solving this schedulability problem is important to provide some means of adapting the system when hard real-time aperiodic tasks are active in $[t,t')$ as we have shown elsewhere [Lima et al. 2008].

The first scheduling test, presented in Section 4.1, is based on sequencing the jobs in $\Gamma(t,t')$ as if the system was not preemptive. Thus, we call this test Job Sequencing Test, JST for short. The second test, described in Section 4.2, takes preemptions into consideration. The main idea of this test is to compute the accumulated workload that may interfere in the execution of each job $J_i \in \Gamma(t,t')$. If this workload is not greater than $e_i - s_i$, J_i is considered schedulable. We name this test AWT that stands for Accummulated Workload Test. Both tests have time complexity $O(n)$, where $n = |\Gamma(t,t')|$.

4.1. Job Sequencing Test

Let us define t_i, $i = 1, 2 \ldots, n$, as follows:

$$t_i = \begin{cases} t' & \text{if } i = n \\ \min(t_{i+1}, d_{i+1}) - C_{i+1}(s_{i+1}) & \text{if } 1 \leq i < n \end{cases} \tag{2}$$

The schedulability condition derived in this section is based on sequencing the jobs in $\Gamma(t,t')$ in the interval $[t,t')$ one after another as if there was no preemption. For example, J_n must be scheduled not before s_n and must finish by $\min(t', d_n)$. In turn, J_{n-1} should start executing not before s_{n-1} but must finish by $\min(t_{n-1}, d_{n-1})$. Since the definition of t_i for $i < n$ takes into consideration the time to execute J_{i+1}, i.e. $C_{i+1}(s_{i+1})$, the schedulability of all jobs in $\Gamma(t,t')$ is ensured, as the following theorem states.

Theorem 1. *The jobs in $\Gamma(t,t') = \{J_1, J_2, \ldots, J_n\}$ are schedulable in the interval $[t,t')$ if*

$$\forall J_i \in \Gamma(t,t') : s_i + C_i(s_i) \leq \min(d_i, t_i)\,, \tag{3}$$

Proof. The proof will be by constructing a non-preemptive schedule during $[t,t')$. Consider job J_n first. It is clear that J_n is schedulable if $s_n + C_n(s_n) \leq \min(d_n, t_n)$ since by equation (2), higher priority jobs must have finished their execution by time $t_{n-1} = \min(t_n, d_n) - C_n(s_n)$. In other words, there is a time interval $I_n = [\min(d_n, t_n) - C_n(s_n), \min(d_n, t_n))$ reserved for the exclusive execution of J_n. Now remove J_n from the set of jobs $\Gamma(t,t')$ and let $t' = t_{n-1}$. The same reasoning applies for $\{J_1, J_2, \ldots, J_{n-1}\}$, which leads to the definition of time intervals $I_i = [\min(d_i, t_i) - C_i(s_i), \min(d_i, t_i))$ that are reserved for execution of J_i, $i = 1, 2, \ldots, n$, as if there was no preemption. As I_i is enough to execute J_i, schedulability holds, as required. □

It is important to emphasize that we are not considering non-preemptive systems. Non-preemption is assumed only for deriving a schedulability condition. Since EDF is an optimal scheduling policy, if a system is schedulable within an interval $[t, t')$ according to Theorem 1, then the system is schedulable by preemptive EDF. The opposite is not necessarily true. Figure 1 illustrates this by showing a set of jobs that are schedulable in the interval $[0, 7)$ by EDF but JST is not able to verify so. Vertical arrows in the figure represent job releases.

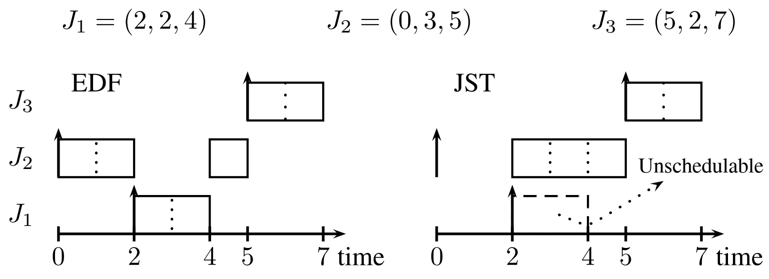


Figure 1. An illustration of JST schedulability condition. $\Gamma(0, 7) = \{J_1, J_2, J_3\}$ **and** $J_i = (r_i, C_i, d_i)$ **is given for** $i = 1, 2, 3$.

4.2. Accumulated Workload Test

We define the accumulated workload of a job $J_i \in \Gamma(t, t')$ as the maximum time the processor will be executing J_i or other jobs that may interfere in its execution. The goal is then to compute such a maximum interference, denoted w_i. If the computed workload is not greater than $e_i - s_i$, one knows that there is enough resource to finish J_i successfully, i.e. by e_i. It should be noted that: (a) jobs that may interfere in the execution of J_i cannot have lower priorities than that of J_i's; and (b) some of those higher priority jobs may start executing before s_i, which should be accounted for. Taking observations (a) and (b) into consideration, we define the following:

$$w_i^1 = \sum_{J_j \in \Gamma(t,t') : d_j \leq d_i} C_j(s_j) \tag{4}$$

$$w_i^2 = \sum_{J_j \in \Gamma(t,t') : d_j \leq d_i, s_j < s_i} C_j(s_j) \tag{5}$$

By definition, $w_i^1 \geq w_i^2$. Also, it should be noticed that $w_i \leq w_i^1$. Thus, $\Gamma(t, t')$ can be considered schedulable in the interval $[t, t')$ if $w_i^1 < e_i - s_i$. However, this may be too pessimistic since observation (b) is not taken into account. In order to reduce such a pessimism, we need to consider possible executions of jobs that contribute to w_i^2 and start executing before s_i, see Figure 2. Time $s_i' \geq t$ is the earliest time the processor is busy executing higher priority jobs that contribute to w_i^2. More formally,

$$s_i' = \min_{J_j \in \Gamma(t,t') : d_j \leq d_i, s_j < s_i} (s_j) \tag{6}$$

Now, the accumulated workload can be defined more precisely.

Definition 3. *The accumulated workload w_i regarding a job $J_i \in \Gamma(t, t')$ for a given interval $[t, t')$ is the maximum interference J_i may suffer by other jobs and is given by*

$$w_i = w_i^1 - \min(s_i - s_i', w_i^2) \tag{7}$$

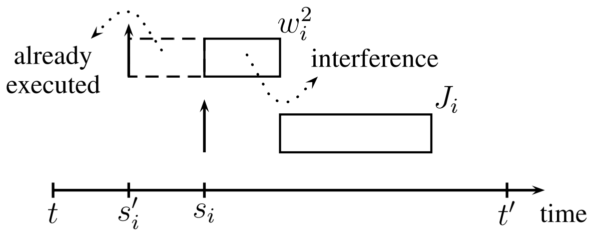


Figure 2. Interference w_i^2 in the execution of J_i.

Based on the definition 3, the following theorem states the schedulability condition of the system in an interval $[t, t')$.

Theorem 2. *A set of jobs $\Gamma(t, t')$ is schedulable in the interval $[t, t')$ if $w_i \leq e_i - s_i$ for all jobs $J_i \in \Gamma(t, t')$.*

Proof. First, assume a particular case where $w_i^2 = 0$. Thus, by the definition of s_i', $w_i = w_i^1$. Since by equation (4) there is no other task in $\Gamma(t, t')$ that may interfere in the execution of J_i, it is clear the theorem holds. Now, consider that $w_i^2 > 0$. This means that there is some $J_j \in \Gamma(t, t')$ which starts executing before s_i. In this case, one must consider the time interval $[s_i', s_i)$, as illustrated in Figure 2, during which the processor is busy executing such tasks. There are two scenarios to be considered:

Scenario (a): $w_i^2 \leq s_i - s_i'$. Obviously, in this scenario one should not compute w_i^2 as if it was contributing to w_i since the time interval $[s_i', s_i)$ is big enough to complete any job J_j before starting J_i. Therefore, $w_i = w_i^1 - w_i^2$.

Scenario (b): $w_i^2 > s_i - s_i'$. Since there is not enough time to execute all jobs J_j that contribute to w_i^2, the time the processor is busy executing such jobs before s_i is equal to $s_i - s_i'$. Thus, $w_i = w_i^1 - (s_i - s_i')$

Scenarios (a) and (b) above can be unified by equation (7), which accounts for the execution of J_i and higher priority jobs that interfere in its execution. It is clear that this interference should not exceed $e_i - s_i$, which is the time within which J_i has to finish. Following this argument for all jobs in $\Gamma(t, t')$, the theorem follows. □

It is interesting to observe that AWT considers the example in Figure 1 schedulable in $[0, 7)$. However, it is not difficult to see that by making $t' < 7$ the system is considered unschedulable in $[t, t')$ by AWT, which makes it a sufficient schedulability test. Differences between both tests are better evaluated by simulation.

5. Evaluation

Extensive simulation to evaluate the proposed schedulability tests was carried out. This section presents some of the relevant results found during the simulation. Task sets with 8 periodic tasks each were randomly generated as follows. For each periodic task, its worst-case execution task was generated according to an exponential distribution with parameter $u_p/10$, where $u_p = 40\%, 50\%, \ldots, 90\%$ is the processor load (utilization) for periodic tasks. Task periods were generated according to a uniform distribution in the interval 80 to 500 time units and their deadlines were considered to be equal to their periods. The simulation time was defined as $100,000$ time units and the first jobs of all periodic tasks were assumed to be released at the beginning of the simulation. Since $u_p \leq 90\%$, if there were only periodic tasks, the system would be schedulable.

In order to bring about overload scenarios aperiodic jobs were generated during the simulation according to a Poisson distribution with parameter $1/1000$. This means that there were on average 100 aperiodic jobs during each simulation run. Each release of aperiodic job determines an interval $[t, t')$. Instant t corresponds to the release time of the aperiodic job while t' is computed as will be explained shortly. The worst-case execution times of the aperiodic jobs were generated according to an exponential distribution with parameter 0.2. The deadline of each generated aperiodic job J_i was defined as $d_i = t + C_i/u_a$, where u_a is the computing demand of J_i within $[t, d_i)$. Several values of u_a were considered during the simulation. Aperiodic tasks could be rejected if the schedulability test used pointed out that there was some risk of missing deadlines of periodic jobs. Thus, task rejection rate was used as an evaluation parameter so that higher rejections indicate higher levels of pessimism in the tests.

As for the value of t', for some evaluations, several interval sizes were generated. The following procedure was used. Consider an aperiodic job J_j released at t. Time t' was set so that it was equal to the release time r_i of the first job J_i released at or after d_j. This procedure defines standard intervals of size of $I = t' - t$. In order to evaluate the behavior of the schedulability tests for other interval sizes, other values of t' were generated by inflating the standard interval by a factor $\alpha \geq 1$ so that $t' = t + I\alpha$. For example, if $\alpha = 1.5$, a 50% increase in the size of the interval is considered.

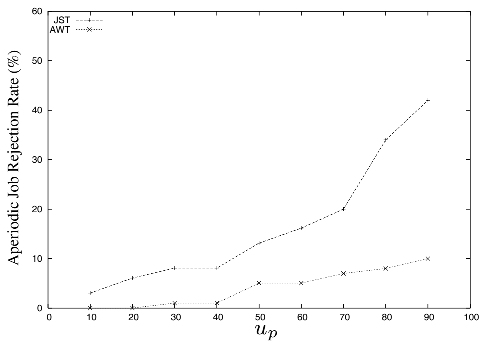
Figure 3 shows simulation results that illustrate the performance of JST and AWT. Each point in the graphs corresponds to the average value found regarding all aperiodic jobs generated during simulation runs. Ten simulation runs were observed, each one for a distinct randomly generated task set. As can be seen from the figure, the aperiodic rejection rates for JST are significantly higher when compared with AWT. Also, note that the performance of JST worsens when the interval size increases. This behavior is expected since JST is more restrictive and its pessimism tends to be higher for larger number of jobs in $\Gamma(t, t')$. Figures 3 (b) and (d) illustrate the behavior of both tests when the aperiodic jobs were not considered. Ideally both tests would indicate that the system is schedulable in all generated intervals since by the EDF policy no job misses its deadline when $u_p \leq 100\%$. Once again, AWT performs much better than JST. Nevertheless, due to its simplicity, JST has successfully been used in the design of reconfiguration mechanism [Lima et al. 2008].

6. Conclusions

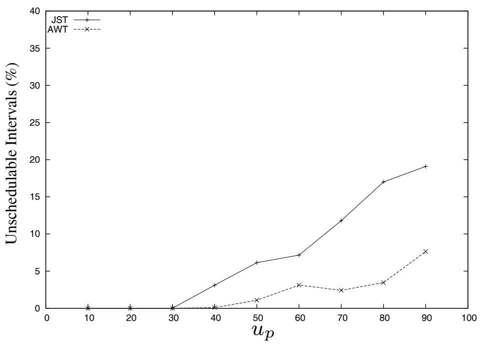
The problem of verifying whether a set of jobs can be successfully scheduled within a given interval has been defined and two sufficient schedulability tests able to carry out such a verification have been described. Both tests are evaluated by simulation for different scenarios. The next step of our work is focused on integrating these tests in modern real-time systems that demand high levels of adaptation at run-time.

References

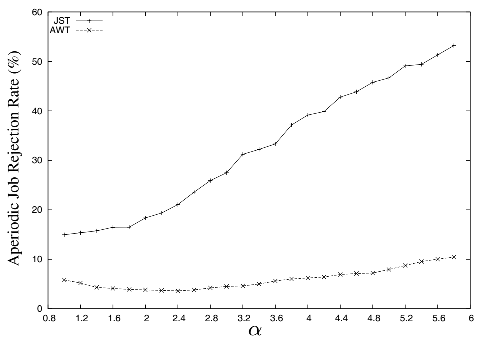
Andersson, B. and Ekelin, C. (2007). "Exact Admission-control for Integrated Aperiodic and Periodic Tasks". *J. Comput. Syst. Sci.*, 73(2):225–241.



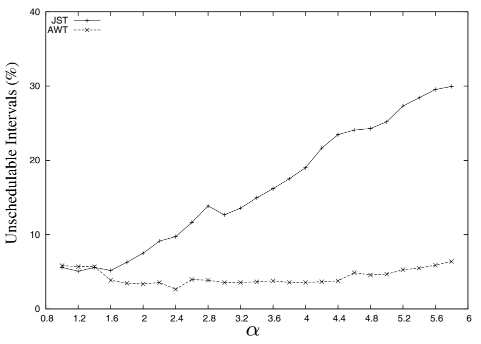
(a) $u_a = 0.5$; $\alpha = 1$



(b) $u_a = 0$; $\alpha = 1$



(c) $u_a = 0.5$; $u_p = 0.6$



(d) $u_a = 0$; $u_p = 0.6$

Figure 3. Simulation results.

Baruah, S. K., Howell, R. R., and Rosier, L. E. (1990). "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-time Tasks on one Processor". *Real-Time Systems*, 2(4):301–324.

Buttazzo, G. and Stankovic, J. (1993). RED: Robust Earliest Deadline Scheduling. In *Proc. of The 3rd International Workshop on Responsive Computing Systems*, pages 100–111.

Chetto, H. and Chetto, M. (1989). "Some Results of the Earliest Deadline Scheduling Algorithm". *IEEE Transactions on Software Engineering*, 15(10):1261–1269.

Goossens, J. and Devillers, R. (1999). Feasibility Intervals for the Deadline Driven Scheduler with Arbitrary Deadlines. In *Proc. of the 6th IEEE Intl. Conf. on Real-Time Computing Systems and Applications (RTCSA'99)*, pages 54–61.

Isovic, D. and Fohler, G. (2000). Efficient Scheduling of Sporadic, Aperiodic, and Periodic Tasks with Complex Constraints. In *Proc. of the 21st IEEE Real-Time Systems Symposium (RTSS 2000)*, pages 207–216.

Lima, G., Camponogara, E., and Sokolonski, A. C. (2008). "Dynamic Reconfiguration for Adaptive Multiversion Real-Time Systems (to appear)". In *Proc. of the 20th IEEE Euromicro Conference on Real-Time Systems (ECRTS 08).*

Liu, C. L. and Layland, J. W. (1973). "Scheduling Algorithms for Multiprogram in a Hard Real-Time Environment". *Journal of ACM*, 20(1):40–61.

Liu, J. W. S. (2000). *Real-Time Systems*. Prentice-Hall.