Real-Time Syst (2008) 40: 264–289
DOI 10.1007/s11241-008-9061-6

EDZL scheduling analysis

Theodore P. Baker · Michele Cirinei · Marko Bertogna

Published online: 17 September 2008

Abstract A schedulability test is derived for the global Earliest Deadline Zero Laxity (EDZL) scheduling algorithm on a platform with multiple identical processors. The test is sufficient, but not necessary, to guarantee that a system of independent sporadic tasks with arbitrary deadlines will be successfully scheduled, with no missed deadlines, by the multiprocessor EDZL algorithm. Global EDZL is known to be at least as effective as global Earliest-Deadline-First (EDF) in scheduling task sets to meet deadlines. It is shown, by testing on large numbers of pseudo-randomly generated task sets, that the combination of EDZL and the new schedulability test is able to guarantee that far more task sets meet deadlines than the combination of EDF and known EDF schedulability tests.

In the second part of the paper, an improved version of the EDZL-schedulability test is presented. This new algorithm is able to efficiently exploit information on the slack values of interfering tasks, to iteratively refine the estimation of the interference a task can be subjected to. This iterative algorithm is shown to have better performance than the initial test, in terms of schedulable task sets detected.

This is an extended version of the ECRTS'07 paper of Cirinei and Baker (2007), with corrections of some flaws in that original paper and a new iterative schedulability test based on Cirinei (2007).
This material is based upon work supported in part by the National Science Foundation under Grant No. 0509131, and a DURIP grant from the Army Research Office.

T.P. Baker (✉)
Department of Computer Science, Florida State University, Tallahassee, FL 32306-4530, USA
e-mail: baker@cs.fsu.edu

M. Cirinei · M. Bertogna
ReTIS Lab, Scuola Superiore Sant'Anna, Pisa, Italy

M. Cirinei
e-mail: cirinei@gandalf.sssup.it

M. Bertogna
e-mail: marko@sssup.it

Keywords EDF · Earliest deadline first · EDZL · Earliest deadline zero laxity · Multiprocessor · Real time · Scheduling · Symmetric multiprocessing · Schedulability

1 Introduction

EDZL is a hybrid preemptive priority scheduling scheme in which jobs with zero laxity are given highest priority and other jobs are ranked by deadline. In this report we apply demand analysis to Earliest Deadline Zero Laxity (EDZL) scheduling, and derive conditions that are sufficient to guarantee that a system of independent sporadic tasks with arbitrary deadlines will not miss any deadline if scheduled by global EDZL on a platform with m identical processors. We also show through experiments how the new EDZL schedulability tests compare to known schedulability tests for global Earliest-Deadline-First (EDF) scheduling, on large numbers of pseudo-randomly generated task sets.

It was previously shown by Cho et al. (2002) that when EDZL is applied as a global scheduling algorithm for a platform with m identical processors its ability to meet deadlines is never worse than pure global EDF scheduling, and that it is "suboptimal" for the two processor case, meaning that "every feasible set of ready tasks is schedulable" by the algorithm. They propose this weak definition of optimality as being appropriate for on-line scheduling algorithms, which cannot take into account future task arrivals. Cho et al. also provide experimental data, showing that even though EDZL is not "suboptimal" for $m > 2$, it still performs very well. EDZL is a proper extension of EDF, in the sense that both EDZL and EDF make the same scheduling decisions up to a point where a job with zero laxity would not be scheduled by EDF. It follows that EDZL incurs at most one additional preemption overhead over EDF per job, and the additional preemption occurs only for jobs that would fail to meet their deadlines under EDF scheduling.

However, for systems with hard deadlines even optimal scheduling performance is not enough if there is no practical algorithmic way to verify that a task set will be scheduled to meet all its deadlines. In this paper, we address that need, by deriving tests that are sufficient to verify schedulability for sets of independent sporadic task sets under global EDZL scheduling on m identical processors. We describe and prove the correctness of several tests, starting from a simple one and improving the tightness of the schedulability conditions by progressively refining the estimation of the mutual interferences imposed among the various tasks. This allows us to derive schedulability testing algorithms that are proved to detect many more schedulable task sets than existing techniques.

2 Task system model

A *sporadic task* $\tau_i = (e_i, d_i, p_i)$ is an abstraction of a process that generates a potentially infinite sequence of *jobs*. Each job has a *release time*, an *execution time*, and an *absolute deadline*. The execution time of every job of a task τ_i is bounded by the maximum (worst case) execution time requirement e_i. The release times of successive jobs of each task τ_i are separated by the minimum inter-release time (period) p_i. The absolute deadline for the completion of each job of τ_i is $r + d_i$, where r is the release time and d_i is the *relative deadline* of τ_i. It is required that $e_i \le \min(d_i, p_i)$, since otherwise a job would never be able to complete within its deadline. The *scheduling window* of a job is the interval between its release time and absolute deadline. A task system τ has *constrained deadlines* if $d_i \le p_i$ for every $\tau_i \in \tau$. Otherwise, the task deadlines are *unconstrained*, and for every task d_i can be indifferently greater or lesser than p_i. In order to generalize the description, it is useful to define $\Delta_i \overset{\text{def}}{=} \min(d_i, p_i)$, noting that for constrained deadlines $\Delta_i = d_i$. In this paper we always consider unconstrained deadlines task systems. The utilization of a task is defined as $u_i \overset{\text{def}}{=} \frac{e_i}{p_i}$ and the density as $\lambda_i \overset{\text{def}}{=} \frac{e_i}{\Delta_i}$.

An m-processor *schedule* for a set of jobs is a partial mapping of time instants and processors to jobs. It specifies the job, if any, that is scheduled on each processor at each time instant. For consistency, a schedule is required not to assign more than one processor to a job, not to assign more than one job to a processor in the same time instant, and not to assign a processor to a job before the job's release time or after the job completes. For a job released at time a, the *accumulated execution time* at time b is the number of time units in the interval $[a, b)$ for which the job is assigned to a processor, and the *remaining execution time* is the difference between the total execution time and the accumulated execution time. A job is *backlogged* if it has nonzero remaining execution time. The *completion time* is the first instant at which the remaining execution time reaches zero. The *response time* is the elapsed time between the job's release time and its completion time. A job misses its absolute deadline if the response time exceeds its relative deadline.

The *laxity* (sometimes also known as slack time) of a job at any instant in time is the amount of time that the job can wait, not executing, and still be able to complete by its deadline. At any time t, if job J has remaining execution time e and absolute deadline d, its laxity is $d - e$.

The jobs of each task are required to be executed sequentially. That is, the start time of a job (the instant in which the job starts its execution) cannot be before the completion time of the preceding job of the same task. If a job has been released but is not able to start executing because the preceding job of the same task has not yet completed, we say that the job is *precedence-blocked*. If a job has been released and its predecessor in the same task has completed, the job is *ready*. If a job is ready, but m jobs of other tasks are scheduled to execute, we say that the job is *priority-blocked*.

Let J be any job, and let τ_k be the corresponding task. The *competing work* $W_i^J(a, b)$ contributed by any task $\tau_i \ne \tau_k$ in an interval $[a, b)$ is the sum of the lengths of all the subintervals of $[a, b)$ during which a job of τ_i is scheduled to execute while job J is priority-blocked. The *total competing work* $W^J(a, b)$ in the interval $[a, b)$ is defined to be the sum of $W_i^J(a, b)$ over all the tasks, and the *competing load* is defined to be the ratio $W^J(a, b)/(b - a)$.

Feasibility and schedulability A given schedule is *feasible* for a given task system if it assigns each job sufficient processor time to complete execution within its scheduling window; that is, if the response time of each job is less than or equal to its relative deadline. A given job set is *feasible* if there exists a feasible schedule for it. In practice, feasibility does not mean much unless there is an algorithm to compute a feasible schedule. A job set is *schedulable* by a given algorithm if the algorithm produces a feasible schedule.

A sporadic task system is feasible if there is a feasible schedule for every set of jobs that is consistent with the minimum inter-release time, deadline, and worst-case execution time constraints of the task system, and it is schedulable by a given algorithm if the algorithm finds a feasible schedule for every such set of jobs.

A *schedulability test* for a given scheduling algorithm is an algorithm that takes as input a description of a task system and provides as output an answer to whether the system is schedulable by the given scheduling algorithm. A schedulability test is *tight* if it always provides a simple answer of "yes" or "no". It is *sufficient* if the algorithm answers "maybe" in some cases.

For any scheduling algorithm to be useful for hard-deadline real-time applications it must have at least a sufficient schedulability test, that can verify that a given job system is schedulable. The quality of the scheduling algorithm and the schedulability test are inseparable, since there is no practical difference between a job system that is not schedulable and one that cannot be proven to be schedulable.

EDZL VS EDF EDZL scheduling is a variant of the well-known preemptive Earliest-Deadline-First (EDF) scheduling algorithm. The difference is the *zero laxity rule*: jobs with zero laxity are given the highest priority. Other jobs are ranked as in EDF. Ties between jobs with equal priority are assumed to be broken arbitrarily.[1] The priority scheduling policy is applied globally, so that if there are m processors and m or more ready jobs then m of the jobs with highest priority will be executing. Like EDF, EDZL is *work conserving*, meaning that a processor is never idle if there is a ready job that is not executing.

Simulation studies have shown that EDZL scheduling performs well (Cho et al. 2002). Moreover, it is quite easy to show that EDZL strictly dominates EDF (see Theorem 2 in Park et al. 2005), with the meaning that if a task set is schedulable by EDF on a platform composed of m processors, it is also schedulable by EDZL on the same platform, and there exist task sets schedulable by EDZL and not by EDF. In fact, intuitively, as noted by Cho et al. (2002), EDZL is actually the EDF algorithm with a "*safety rule*" (the zero laxity rule) to be applied in critical situations. It means that the scheduling of the two algorithm differs only in cases in which EDF fails scheduling some tasks.

It follows that all the sufficient EDF schedulability tests are also sufficient for EDZL, including the EDF *density bound test*, which was proposed for implicit-deadline systems by Goossens et al. (2003) and subsequently shown to extend to constrained and unconstrained deadline systems. However, one would expect that the addition of the safety rule might permit a stronger schedulability test, that is able to verify the schedulability of task sets that are not schedulable by global EDF. To the best of our knowledge, no such schedulability test for EDZL has been published. Our objective is to find such a test.

[1]This is a worst-case assumption. In practice one would have a specific tie breaking rule, such as to give priority to a job that is already executing on a given processor, to avoid wasteful task switches.

3 Predictability

An important subtlety in schedulability testing is that the so-called "worst-case" execution time e_i of each task is just an upper bound; the execution times of different jobs of a task can vary. This leaves open the possibility that the upper bound, or even the actual maximum execution time of task, may not actually be the worst situation with respect to total system schedulability. For multiprocessor scheduling, there are well known anomalies, where a job set is schedulable by a given algorithm, but if the execution time of one or more jobs is *shortened*, the job set becomes unschedulable.

Ha and Liu (1994) and Ha (1995) studied this problem, and were able to identify certain families of scheduling algorithms that are *predictable* with respect to variations in job execution time. A scheduling algorithm is defined to be *completion-time predictable* if, for every pair of sets $\mathcal{J}$ and $\mathcal{J}'$ of jobs that differ only in the execution times of the jobs, and such that the execution times of jobs in $\mathcal{J}'$ are less than or equal to the execution times of the corresponding jobs in $\mathcal{J}$, then the completion time of each job in $\mathcal{J}'$ is no later than the completion time of the corresponding job in $\mathcal{J}$. That is, with a completion-time predictable scheduling algorithm it is sufficient, for the purpose of bounding the worst-case response time of a task or proving schedulability of a task set, to look just at the jobs of each task whose actual execution times are equal to the task's worst-case execution time.

An important class of scheduling algorithms for which Ha and Liu were able to prove completion-time predictability is the *preemptive migratable fixed job-priority scheduling algorithms*. One such algorithm is global preemptive EDF scheduling. Unfortunately, while EDZL is preemptive and migratable, it does not have fixed job priorities. Therefore, while one might suspect that EDZL could be predictable with respect to execution time variations, a necessary first step in looking for a EDZL schedulability test is to verify that. Piao et al. (2006) addressed this question and showed that EDZL is completion-time predictable on the domain of integer time values. The result clearly also applies to any other discrete time domain. We give a somewhat more self-contained and direct proof below.

Theorem 1 (Predictability of EDZL) *The EDZL scheduling algorithm is completion-time predictable, with respect to variations in execution time.*

Proof We actually prove a stronger hypothesis; that is, if the only difference between $\mathcal{J}$ and $\mathcal{J}'$ is that some of the actual job execution times are shorter in $\mathcal{J}'$ than in $\mathcal{J}$, then the accumulated execution time of every uncompleted job in the EDZL schedule for $\mathcal{J}'$ is greater than or equal to the accumulated execution time of the same job in the EDZL schedule for $\mathcal{J}$ at every instant in time. It will follow that no job can have an earlier completion time in $\mathcal{J}$ than in $\mathcal{J}'$, since the actual execution times in $\mathcal{J}$ are at least as long as in $\mathcal{J}'$.

Suppose the above hypothesis is false. That is, there exist job sets $\mathcal{J}$ and $\mathcal{J}'$ whose only difference is that some of the actual job execution times are shorter in $\mathcal{J}'$ than in $\mathcal{J}$, and such that at some time t the accumulated execution time of some uncompleted job is less with $\mathcal{J}'$ than with $\mathcal{J}$. We will show that this leads to a contradiction, and the theorem will follow.

Without loss of generality, we can restrict attention to the case where $\mathcal{J}$ and $\mathcal{J}'$ differ only in the actual execution time of one job. To see this, observe that between $\mathcal{J}$ and $\mathcal{J}'$ there is a finite sequence of sets of jobs such that the only difference between one set and the next is that the actual execution time of one job is decreased. Let $\mathcal{J}$ and $\mathcal{J}'$ be the first pair of successive jobs in such a sequence such that at some time t the accumulated execution time of some uncompleted job J is less with $\mathcal{J}'$ than with $\mathcal{J}$.

Let t be the earliest instant in time after which the accumulated execution time of some uncompleted job is less with $\mathcal{J}'$ than with $\mathcal{J}$, and let J be such a job. That is, up through t the accumulated execution time of each uncompleted job in the schedule for $\mathcal{J}$ is less than or equal to the accumulated execution time of the same job in the schedule for $\mathcal{J}'$, and after time t the accumulated execution time of job J is greater with $\mathcal{J}$ than with $\mathcal{J}'$.

Job J must be scheduled to execute starting at time t with $\mathcal{J}$ and not with $\mathcal{J}'$. This means some other job J' is scheduled to execute in place of J with $\mathcal{J}'$. That choice cannot be based on deadline, since the deadlines of corresponding jobs are the same with $\mathcal{J}$ and $\mathcal{J}'$, so it must be based on the zero-laxity rule. That is, J' has zero laxity at time t with $\mathcal{J}'$ but not with $\mathcal{J}$. However, that would require that J' has greater accumulated execution time at time t with $\mathcal{J}$ than it does with $\mathcal{J}'$. This is a contradiction of the choice of t. Therefore, the theorem must be true. □

4 Sketch of the test

The schedulability test we propose is based on the same core idea as Baker (2003), Bertogna et al. (2005): with a work-conserving scheduling algorithm a job can miss its deadline only if competing jobs of other tasks priority-block it for a sufficient amount of time.

In order to analyze the conditions that are necessary for a job to miss its deadline, we focus on the earliest point in a given schedule where any job misses a deadline, on a specific job that misses its deadline at that point, and on the time interval between the release of that job and its missed deadline. We let $\bar{t}$ denote the time of the first missed deadline, call a job that misses its deadline at $\bar{t}$ the *problem job, and call the interval between its release time and deadline the* problem window.

The analysis is done in the following steps:

1. Determine a lower bound on the total competing work that is needed in the problem window to cause the problem job to miss its deadline, or to cause another job to reach zero laxity within the problem window.
2. Determine an upper bound on the competing work that can be contributed by each individual task.
3. Combine the per-task bounds to obtain an upper bound on the total competing work in the problem window.

The schedulability test amounts to a comparison of the results of steps 1 and 3. If the upper bound of step 3 is less than the lower bound of step 1, that would be a contradiction, so there can be no problem job; that is, the task is schedulable.

The main difference between the EDZL tests we propose here and the EDF tests explained in Baker (2003) and Bertogna et al. (2005) is that with EDF it is sufficient to find a possibly unschedulable task to conclude that the task set might not be schedulable, while for EDZL it is necessary to also find at least $m+1$ tasks that may reach zero laxity, since EDZL would give maximum priority to the first m tasks which reach zero laxity, and only the $(m+1)$th task that reaches zero laxity can force a deadline miss.

5 Lower bound

Recall that the laxity of a job, if positive, represents the amount of time that the job can wait, without executing, and still have enough real time left that it could complete execution within its deadline, if the schedule allowed it to execute for all of that remaining time.

Whenever a job is blocked and does not execute, its laxity decreases, and whenever the job executes, the laxity remains constant. When a job is released, its initial laxity is equal to its relative deadline minus its execution time, $d_i - e_i$, which is non-negative. With both EDF and EDZL scheduling, the laxity of the problem job must become negative at or before the missed deadline. That is, other jobs must block the problem job for enough time within the problem window to consume all of its initial laxity, plus at least one more instant of time. This is a necessary and sufficient condition for a deadline miss, and is the sole basis of the analysis of EDF scheduling failures in Baker (2003) and Bertogna et al. (2005). However, in the case of EDZL a scheduling failure provides additional information.

Under EDZL, once any job J reaches zero laxity its priority will be raised to the top and will stay at that level continuously up to the job's finish time, which coincides with its deadline. In this situation, only other jobs with zero laxity are able to force J to wait (and so miss its deadline).

This means that in order for the problem job to miss its deadline (that is, reach negative laxity) there must be at least $m+1$ jobs (including the problem job itself) that all have zero laxity together at some instant inside the problem window.

The following lemma provides a lower bound on the competing load of any job that reaches zero laxity prior to the first missed deadline of the system.

Lemma 2 *If EDZL is used to schedule a sporadic task system $\tau = \{\tau_1, \ldots, \tau_n\}$ on m identical processors and $\bar{t}$ is the first missed deadline of the system, then a job J of task τ_k with deadline t can reach zero laxity before $\bar{t}$ only if the following inequality holds, and can reach negative laxity before $\bar{t}$ only if it holds strictly* ($>$):

$$\frac{\sum_{i \neq k} W_i^J(t-\Delta_k, t)}{\Delta_k} \geq m(1-\lambda_k) \tag{1}$$

Proof J can reach zero laxity only if it is blocked for at least $d_k - e_k$ in its scheduling window. Since EDZL is work-conserving, a released job can be blocked for only two reasons:

1. precedence, by an older job of the same task;
2. priority, by jobs of equal or higher priority belonging to other tasks.

If $d_k \leq p_k$, only one job of τ_k can be active at a time, so precedence never blocks a job of such a task. In this case the job J can reach zero laxity only if jobs of other tasks, with higher or equal priority, can occupy all m processors for at least $d_k - e_k$ time units in the scheduling window $[t - d_k, t)$. It follows that

$$\sum_{i \neq k} W_i^J(t - d_k, t) \geq m(d_k - e_k)$$

$$\sum_{i \neq k} \frac{W_i^J(t - d_k, t)}{d_k} \geq m(1 - \lambda_k)$$

and if the job reaches negative laxity the above inequalities must be strict ($>$).

In the other case, if $d_k > p_k$, the precedence constraint could block J. However, we can still find a lower bound for the competing work and load by considering a subinterval of $[t - d_k, t)$ over which J cannot be precedence-blocked. Because J reaches zero laxity before $\bar{t}$ it must be that $t - e_k < \bar{t}$. Since we are assuming that $e_k \leq p_k$, it follows that $t - p_k \leq \bar{t}$. Because $\bar{t}$ is the first missed deadline, any job of τ_k released prior to $t - p_k$ must complete within its deadline, and so J cannot be precedence blocked in the interval $[t - p_k, t)$. Since J reaches zero laxity, all the m processors must be working on jobs other than J for $p_k - e_k$ time units in the interval $[t - p_k, t)$. (Of course this is a worst case upper bound, and a more exact estimate would be $p_k - e$ where e is the remaining time execution time of the job J at time $t - p_k$. Unfortunately, it is difficult to estimate the remaining computation time of a job without simulating the system.)

From this upper bound we obtain

$$\sum_{i \neq j} W_i^J(t - p_k, t) \geq m(p_k - e_k)$$

$$\sum_{i \neq j} \frac{W_i^J(t - p_k, t)}{p_k} \geq m(1 - \lambda_k)$$

and if the job reaches negative laxity the above inequalities must be strict ($<$).

The intervals and the bounds on competing work differ between the two cases above, but because load is normalized by the interval length and because the definitions of λ_k and Δ_k differ for the two cases, the expression for the load bound is the same. Therefore, both bounds can be unified as in the statement of the lemma. $\square$

Note again that with EDZL scheduling, considering the zero laxity rule, a job can miss its deadline only if in a certain time instant both of the two following conditions hold:

– the laxity of the job is zero;
– the laxity of at least m other jobs is zero.

So, for a deadline to be missed there must exist at least $m+1$ different tasks whose jobs can be blocked by the others for a sufficient amount of time for each of them to reach zero laxity, and at least one to reach negative laxity. By Lemma 2, there must be at least $m+1$ jobs for which the condition (1) above holds, and for at least one of them the equation must hold strictly ($>$).

From this point on, we use the term *terminal window to refer to the interval* $[t-\Delta_k, t)$*, for any job of task* τ_k *with deadline* t*. The significance of the terminal window of a job is that it is a sub-interval of the scheduling window in which no precedence blocking can occur, and hence the only interference experienced by the job is due to priority-blocking by jobs of other tasks.*

6 Upper bound

In this section we derive an upper bound for the contribution of a task τ_i to the competing work of any job J of task τ_k in its terminal window, $[t-\Delta_k, t)$, provided J reaches zero laxity before $\bar{t}$. We first determine the worst case release times of the jobs of τ_i in J's terminal window, and then compute an upper bound on the amount of competing work that τ_i can contribute with that set of release times.

6.1 Worst case release times

It is clear that the competing work $W_i^J(a,b)$ contributed by a task τ_i for any job J in any interval $[a,b)$ cannot be larger than when the release times of τ_i are exactly periodic. That is, moving the release times of τ_i farther apart cannot increase the competing work.

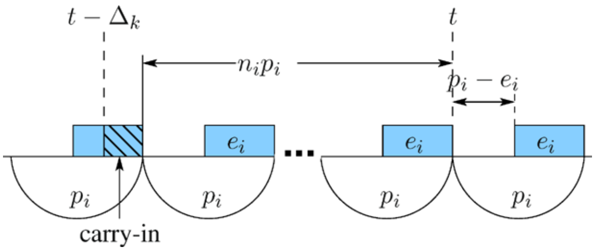
As contributors to the competing work, we do not need to consider any jobs that have a deadline on or before the start of the terminal window $[t-\Delta_k, t)$. Such jobs must have completed by time $t-\Delta_k$. (This follows from the assumption that J reaches zero laxity before the first missed deadline of the system, by the same argument as in the proof of Lemma 2.)

We do need to consider as contributors to the competing work every job of τ_i that has its deadline in the terminal window of J. In order to maximize the competing work of these jobs we can assume without loss of generality that they each execute as late as possible, that is, exactly in the interval of length e_i just before their deadlines. Any job with both release time and deadline in the terminal window is not influenced by this assumption, while the assumption can only increase (and cannot decrease) the contribution of jobs with release times before the window.

We also need to consider any job of τ_i released in or before the terminal window of J and with deadline after the window. Such a job can compete with J only when its laxity is zero, which can happen no earlier than $d_i - e_i$ after its release time. Note that this is equivalent to considering the job to execute as late as possible.

In all cases, whether a job has a deadline in the terminal window of J or after, the worst case competing work contributed by that job cannot be greater than the amount of time that the job would execute if it were scheduled to run for exactly the e_i time units before its deadline. So, the competing work contributed by a task τ_i in

Fig. 1 Upper bound on carry-in



the terminal window of J cannot be greater than the amount of time the task could execute if it were released periodically, at intervals of exactly p_i, and each job were scheduled to run in the last e_i time units before its deadline.

We will next argue that the competing work contributed by τ_i cannot be greater than if a deadline of τ_i is aligned with the deadline t of job J, as shown in Fig. 1.

The argument will consider what happens to the demand if we simultaneously shift all the release times and deadlines of τ_i either forward or backward from that alignment. The maximum shift we need to consider in either direction is p_i, since for longer shifts the effect is periodic:

– Forward movement: if we shift forward (meaning later in time) all the release times by a quantity $x \leq p_i$, the maximum contribution of τ_i to the competing work in the interval is decreased by $\min(x, e_i)$, which is the amount of its work shifted out of the terminal window $[t - \Delta_k, t)$. The shift may increase the contribution of a job at the start of the interval, but by at most $\min(x, e_i)$. Therefore, a forward shift of the release times cannot increase the maximum contribution of τ_i to the competing work, though it can decrease it.
– Backward movement: if we shift backward all the release times by $x \leq p_i$, the first job of τ_i after the terminal window of J cannot achieve higher priority than the problem job until it has reached zero laxity, so the maximum contribution of τ_i to the competing work in the interval does not increase for $x < p_i - e_i$, while for greater values of shift the increase is $x - (p_i - e_i)$ (see Fig. 1). We obtain an increase of $\max(0, x - (p_i - e_i))$. However, the shift also decreases the contribution to the competing work by the first job of τ_i by at least $\max(0, x - (p_i - e_i))$ (which happens when the carried-in job of τ_i has its release time exactly at $t - \Delta_k$). Again, the net change in the maximum contribution of τ_i to the competing work cannot increase, though it can decrease.

Taking the two cases together, it is clear that an upper bound on the contribution of τ_i to the competing work in any interval is achieved when the jobs of τ_i are released periodically and one deadline of τ_i coincides with the end of the interval.

6.2 Worst case competing work

The following lemma establishes an upper bound on the contribution of any task τ_i to the competing work of any job that is released before the first missed deadline of the system.

Lemma 3 *If EDZL is used to schedule a sporadic task system $\tau = \{\tau_1, \dots, \tau_n\}$ on m identical processors and $\bar{t}$ is the first missed deadline, and if J is a job of task τ_k with deadline at time t that reaches zero laxity before time $\bar{t}$, then the competing work contributed by task τ_i in the terminal window of J is subject to the bound*

$$W_i^J(t - \Delta_k, t) \le n_i e_i + \min(e_i, \Delta_k - n_i p_i)$$

where $n_i \overset{\text{def}}{=} \lfloor \Delta_k / p_i \rfloor$.

Proof In the worst-case scenario, all the jobs are released periodically and execute exactly before their deadlines, as depicted in Fig. 1. The competing work of task τ_i is then composed of two different contributions:

1. *Body jobs.* The contributions of the $n_i = \lfloor \Delta_k / p_i \rfloor$ jobs of τ_i for which the interval $[t' - p_i, t')$ immediately preceding the job's deadline is completely contained inside J's terminal window, $[t - \Delta_k, t)$. Each one of these jobs contributes exactly e_i time units.
2. *Carry-in.* The contribution of one job, called the carried-in job, for which the start of the interval $[t' - p_i, t')$ preceding the job's deadline occurs before the start of J's terminal window $[t - \Delta_k, t)$. This contribution is clearly less than or equal to the worst-case execution time e_i. The carry-in also cannot be greater than the length of the interval between the start of the window $[t - \Delta_k, t)$ and the completion time of the carried-in job. The deadline of the last of the n_i jobs is at time t, and the deadline of the first one is at time $t - n_i p_i$ (they coincide if $n_i = 0$). The length of the interval during which the carried-in job can execute is $\Delta_k - n_i p_i$, so the size of the carry-in cannot be greater than $\min(e_i, \Delta_k - n_i p_i)$.

□

Note that the upper bound of Lemma 3 depends only on the length of J's terminal window, and not on the specific start and end points of the interval. Moreover, for any fixed task τ_k, the length of the terminal window is fixed. Therefore, it follows from Lemma 3 that the following is an upper bound on the contribution $W_i^J(t - \Delta_k, t) \wedge \Delta_k$ of task τ_i to the competing load for any job of τ_k that reaches zero laxity before the first missed deadline of the system:

$$\beta_k^i = \frac{n_i e_i + \min(e_i, \Delta_k - n_i p_i)}{\Delta_k} \tag{2}$$

7 First schedulability test

Based on the above lemmas, and considering that $m + 1$ tasks must have zero laxity at the same time in order for a task to miss a deadline, one derives the following first schedulability test for EDZL on a multiprocessor.

Theorem 4 (First EDZL test) *A sporadic task system $\tau = \{\tau_1, \dots, \tau_n\}$ is schedulable by EDZL on m identical processors unless the following condition holds for at least*

$m+1$ different tasks τ_k, and it holds strictly ($>$) for at least one of them:

$$\sum_{i\neq k} \beta_k^i \geq m(1-\lambda_k) \tag{3}$$

where β_k^i is defined as in (2).

Proof It was shown above that a job can miss its deadline only if it and at least m other tasks reach zero laxity. According to Lemma 2, a job J of a task τ_k can reach zero laxity only if the competing work of the other tasks in its terminal window is greater than or equal to $m(1-\lambda_k)$. By (2) the total competing load is no greater than $\sum_{i\neq k} \beta_k^i$. It follows that τ is schedulable by EDZL unless (3) holds for at least $m+1$ tasks.

We also know that a job can miss its deadline only if it reaches negative laxity, which Lemma 2 says can only happen if the competing work is strictly greater than $m(1-\lambda_k)$. It follows that τ is schedulable by EDZL unless (3) holds strictly for at least one task. □

8 Second schedulability test

To improve the precision of Theorem 4, we now reconsider the above definitions and lemmas, verifying and adapting them to deal with interference, a concept introduced by Bertogna et al. (2005). Some of the following results can be found, only with a slightly different notation, in Bertogna et al. (2005), but we repeat them here in order to help the reader.

The *interference* $I^J(a,b)$ on a job J of task τ_k over an interval $[a,b)$ is the cumulative length of all the intervals in which J is *priority-blocked*.

The above definition, like that of competing load, does not include in the interference cases of precedence-blocking. If job J belongs to a task τ_k with $d_k \leq p_k$, precedence-blocking cannot occur, but that is not true if τ_k has $d_k > p_k$. However, in the terminal window $[t-\Delta_k, t)$ of a job J there cannot be any precedence-blocking, so we can avoid distinguishing the two cases. For this reason, from now on we always consider the terminal window $[t-\Delta_k, t)$. (This is the main difference with the analysis in Bertogna et al. (2005), where only constrained deadlines were considered, and so no particular interval was selected.)

In every time instant in which job J of τ_k is priority-blocked, the m processors must be occupied by exactly m jobs of tasks other than the task τ_k of job J. Consequently, the respective m values of competing work increase. From this it follows that

$$I^J(t-\Delta_k, t) \overset{\text{def}}{=} \frac{\sum_{i\neq k} W_i^J(t-\Delta_k, t)}{m}.$$

The above results can be used to prove the following:

Lemma 5 (Lemma 4 in Bertogna et al. 2005)

$$I^J(t-\Delta_k,t) \geq x \quad \Longleftrightarrow \quad \sum_{i\neq k} \min(W_i^J(t-\Delta_k,t),x) \geq mx.$$

Proof Only if. Let $\tau' \subseteq \tau$ be the set of tasks τ_i for which $W_i^J(t-\Delta_k,t) \geq x$, and ξ is the cardinality of τ'. If $\xi \geq m$ the lemma directly follows, so we consider only $\xi < m$.

$$\begin{aligned}\sum_{i\neq k} \min(W_i^J(t-\Delta_k,t),x) &= \xi x + \sum_{\tau_i \notin \tau'} W_i^J(t-\Delta_k,t)\\ &= \xi x + mI^J(t-\Delta_k,t) - \sum_{\tau_i \in \tau'} W_i^J(t-\Delta_k,t)\\ &\geq \xi x + mI^J(t-\Delta_k,t) - \xi I^J(t-\Delta_k,t)\\ &= \xi x + (m-\xi)I^J(t-\Delta_k,t) \geq \xi x + (m-\xi)x = mx\end{aligned}$$

If. Note that if $\sum_{i\neq k} \min(W_i^J(t-\Delta_k,t),x) \geq mx$, it follows that

$$I^J(t-\Delta_k,t) = \sum_{i\neq k} \frac{W_i^J(t-\Delta_k,t)}{m} \geq \sum_{i\neq k} \frac{\min(W_i^J(t-\Delta_k,t),x)}{m} \geq \frac{mx}{m} = x$$

□

Considering the definition of *interference*, it is clear that a job J of τ_k can reach zero laxity only if $I^J(t-\Delta_k,t) \geq \Delta_k - e_k$. Note that this is again a worst-case assumption which introduces some pessimism in the analysis. Applying Lemma 5, we have that J can reach zero laxity only if

$$\sum_{i\neq k} \min(W_i^J(t-\Delta_k,t), \Delta_k - e_k) \geq m(\Delta_k - e_k)$$

and so

$$\sum_{i\neq k} \min(W_i^J(t-\Delta_k,t)/\Delta_k, 1-\lambda_k) \geq m(1-\lambda_k) \tag{4}$$

It is very difficult to accurately compute the competing workload $W_i^J(t-\Delta_k,t)$. However, we can use the above upper bounds, and in particular introduce β_k^i in (4). Using the above lemma, we can prove the following (compare with Lemma 2).

Lemma 6 *If EDZL is used to schedule sporadic task system $\tau = \{\tau_1,\ldots,\tau_n\}$ on m identical processors then a job J of task τ_k with deadline t can reach zero laxity only if*

$$\sum_{i\neq k} \min(\beta_k^i, 1-\lambda_k) \geq m(1-\lambda_k) \tag{5}$$

Proof The lemma follows directly by substitution of $W_i^J(t-\Delta_k, t)/\Delta_k$ for β_k^i in inequality (4). □

Thanks to this result we can now formulate the following refined version of Theorem 4.

Theorem 7 (Refined EDZL test) *A sporadic task system $\tau = \{\tau_1, \ldots, \tau_n\}$ is schedulable by EDZL on m identical processors unless the following inequality holds for at least $m+1$ different tasks τ_k:*

$$\sum_{i\neq k} \min(\beta_k^i, 1-\lambda_k) \geq m(1-\lambda_k) \tag{6}$$

and the following inequality holds for at least one task

$$\sum_{i\neq k} \min(\beta_k^i, 1-\lambda_k) > m(1-\lambda_k) \tag{7}$$

where β_k^i is defined as in (2).

Proof According to Lemma 6, a job J can reach zero laxity only if inequality (6) is satisfied. Once J has reached zero laxity, as we say above, it can miss its deadline only if at least m other tasks reach zero laxity. This can happen only if at least $m+1$ tasks satisfy inequality (6).

It must also be the case that (7) applies for at least one task. □

9 Iterative test

It is possible to improve the schedulability analysis of the previous sections by using a tighter bound on the interference a task can impose in a given window. Both schedulability tests of Theorems 4 and 7 suffer from the gross overestimation of the carry-in of an interfering task. For this reason, improvement is possible if we can give a better estimation of the interfering contributions. We will improve this estimation by taking into account an upper bound of the maximum finishing time of an interfering task. If we know that a task τ_i cannot receive enough interference to execute close to its deadline, we can use this information to improve the estimation of the interference $W_i^J(t-\Delta_k, t)$ imposed by τ_i on a job J of a different task τ_k. To better explain this technique, we introduce the *slack* terms into our analysis. The slack s_i^J of a job $J \in \tau_i$ is defined as the minimum time interval between the finishing time f^J and the deadline $(r^J + d_i)$ of the job. The slack s_i of a task τ_i is the minimum slack among all jobs of τ_i:

$$s_i \overset{\text{def}}{=} \min_{J\in\tau_i}(r^J + d_i - f^J)$$

Clearly, a positive slack implies that the task will always complete the execution of each of its jobs before the job's deadline. Exactly computing the slack is not an easy task. If we would be able to give a lower bound on the slack of a task, we could refine the β_k^i terms leading to upper bounds for the load of τ_i in the terminal window $[t-\Delta_k, t)$ of a job of τ_k. The next theorem shows a way to provide a safe lower bound on the slack.

Theorem 8 *Every job J of task τ_k has a slack greater than or equal to*

$$s_k^{lb} = \Delta_k - e_k - \frac{1}{m}\sum_{i\neq k} \min(W_i^J(t-\Delta_k, t), \Delta_k - e_k) \tag{8}$$

where $t = r^J + d_k$ is the deadline of the job, when this term is strictly positive.

Proof Suppose there is a job of τ_k with a slack s^J less than s_k^{lb}, and $s_k^{lb} > 0$. Let J be the first such job. Since s_k^{lb} is strictly positive, it follows from (8) that

$$\sum_{i\neq k} \min(W_i^J(t-\Delta_k, t), \Delta_k - e_k) < m(\Delta_k - e_k)$$

Applying Lemma 5, in the contra-positive, we have $I^J(t-\Delta_k, t) < (\Delta_k - e_k)$. Since $W_i^J(t-\Delta_k, t) \le I^J(t-\Delta_k, t) < \Delta_k - e_k$, it follows that

$$\min(W_i^J(t-\Delta_k, t), \Delta_k - e_k) = W_i^J(t-\Delta_k, t)$$

We now consider separately the constrained and arbitrary deadline cases.

If job J belongs to a task with $d_k \le p_k$, the slack can be rewritten as

$$\begin{aligned} s^J &= (r^J + d_k) - f^J \\ &= (r^J + \Delta_k) - (r^J + e_k + I^J(t-\Delta_k, t)) \\ &= \Delta_k - e_k - \frac{1}{m}\sum_{i\neq k} W_i^J(t-\Delta_k, t) \\ &= \Delta_k - e_k - \frac{1}{m}\sum_{i\neq k} \min(W_i^J(t-\Delta_k, t), \Delta_k - e_k) = s_k^{lb} \end{aligned}$$

contradicting the hypothesis.

Otherwise, it must be true that $d_k > p_k$. Since J is, by hypothesis, the first job having a slack less than s_k^{lb}, all previous jobs of τ_k complete their execution at least $s_k^{lb} > 0$ time units before their deadlines. Therefore, there is no precedence blocking in interval $[t-\Delta_k, t)$. Job J can then have a slack lower than s_k^{lb} only if $I^J(t-\Delta_k, t) > \Delta_k - e_k - s_k^{lb}$. This happens when

$$\begin{aligned} s_k^{lb} &> \Delta_k - e_k - I^J(t-\Delta_k, t) \\ &= \Delta_k - e_k - \frac{1}{m}\sum_{i\neq k} W_i^J(t-\Delta_k, t) \\ &= \Delta_k - e_k - \frac{1}{m}\sum_{i\neq k} \min(W_i^J(t-\Delta_k, t), \Delta_k - e_k) \end{aligned}$$

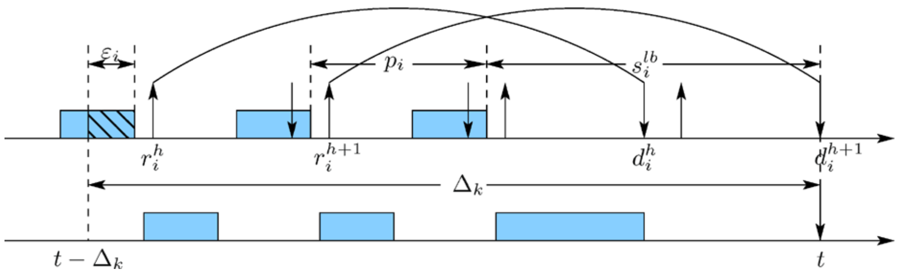


Fig. 2 Worst-case interference of a task τ_i on a job of task τ_k when s_i^{lb} is a safe lower bound on the slack of τ_i

which contradicts (8). The theorem is proved. □

By replacing the terms for competing workload in (8) with the upper bound on the workload given by Lemma 3, we obtain the following lower bound on the slack of τ_k, when this term is strictly positive:

$$s_k^{lb} = \Delta_k - e_k - \frac{1}{m}\sum_{i\neq k}\min\big(n_i e_i + \min(e_i, \Delta_k - n_i p_i), \Delta_k - e_k\big) \tag{9}$$

Now, suppose a bound s_i^{lb} on the slack of a task τ_i is known, for instance by using (9). We would like to exploit this information to tighten the estimation of the workload of τ_i in the terminal window $[t-\Delta_k, t)$ of a job of task τ_k. Consider the situation depicted in Fig. 2. We can use the known lower bound on the slack of τ_i to improve the estimation of the body and the carry-in contributions of τ_i in the window.

Improving the terminal window Recall that Δ_k was chosen to be the minimum of d_k and p_k because in the case that $d_k > p_k$ we could not rule out the possibility of J being precedence-blocked by an earlier job of τ_k. The bound s_i^{lb} allows us to replace p_k by $p_k + s_i^{lb}$ in the definition of Δ_k for all of the above analyses. Therefore, from this point on, we replace Δ_k by $\Delta_k^* \overset{\text{def}}{=} \min(d_k, p_k + s_k^{lb})$.

Improving the body We want to account for a new job in the body of the terminal window only if the window is large enough to include *the latest possible finishing time of the previous job*. Since the worst-case interference of a task τ_i on a job J of task τ_k can still be found when a job of τ_i has an absolute deadline aligned with the deadline t of J, the last interfering job of τ_i will complete after at most $\Delta_k^* - s_i^{lb}$ time instants from the beginning of the interval $[t-\Delta_k^*, t)$. The formula to compute the number of jobs included in the body, given any global lower bound s_i^{lb} on slack of τ_i, then becomes

$$n_i^*(s_i^{lb}) \overset{\text{def}}{=} \left\lfloor \frac{\max(0, \Delta_k^* - s_i^{lb})}{p_i} \right\rfloor \tag{10}$$

and is valid for both arbitrary and constrained deadline systems. The maximum term is needed because s_i^{lb} can be greater than Δ_k^*.

Improving the carry-in The carry-in represents the contribution of the first interfering job that is preceded by a job whose finishing time is before the start of the terminal window $[t - \Delta_k^*, t)$. In other words, it denotes the interference not included in the body. Since the carried-in job of τ_i finishes at least s_i^{lb} time instants before its deadline, the interval in which it can execute inside the window $[t - \Delta_k^*, t)$ is reduced by the same amount. As a consequence, the following is an upper bound on the carry-in:

$$\varepsilon_i(s_i^{lb}) = \min\bigl(e_i, \max(0, \Delta_k^* - s_i^{lb}) - n_i^*(s_i^{lb}) p_i\bigr) \tag{11}$$

Note that the definition of $n_i^*(s_i^{lb})$ implies $\varepsilon_i(s_i^{lb}) \geq 0$.

Combining the results for the body and the carry-in, the following result is proved.

Lemma 9 *If EDZL is used to schedule sporadic task system $\tau = \{\tau_1, \dots, \tau_n\}$ on m identical processors, the competing work contributed by task τ_i in the window $[t - \Delta_k^*, t)$ of a job J of task τ_k is subject to the bound*

$$W_i^J(t - \Delta_k^*, t) \leq n_i^*(s_i^{lb}) e_i + \varepsilon_i(s_i^{lb})$$

where s_i^{lb} is a valid lower bound on the slack of a task τ_i, and $n_i^*(s_i^{lb})$ and $\varepsilon_i(s_i^{lb})$ are given by (10) *and* (11).

Using the bound on the competing workload given by Lemma 9, we can then refine the estimation of the interference imposed by a task τ_i on a task τ_k. This suggests an efficient way to improve our analysis by iteratively refining—i.e., increasing—the lower bounds on the slacks. Before describing this method, we first need the following theorem.

Theorem 10 *Suppose a set of slack lower bounds $s_i^{lb} \geq 0$ is known for a task system scheduled with EDZL, taking $s_i^{lb} = 0$ for the tasks without a known bound. A lower bound on the slack of a task τ_k is then given by*

$$s_k^{lb} = \Delta_k^* - e_k - \frac{1}{m} \sum_{i \neq k} \min\bigl(n_i^*(s_i^{lb}) e_i + \varepsilon_i(s_i^{lb}), \Delta_k^* - e_k\bigr) \tag{12}$$

when this term is strictly positive.

Proof Follows from Theorem 8, replacing the interfering terms with the lower bound on the workload $W_i^J(t - \Delta_k^*, t)$ given by Lemma 9, or—when a slack lower bound s_i^{lb} isn't available—by Lemma 3. □

We are now ready to state an iterative method to check the schedulability of a task system composed by n sporadic tasks and scheduled with EDZL on a multiprocessor platform. It basically consists in checking if it is possible to guarantee that at least $n - m$ tasks have a positive slack lower bound, so that no more than m tasks could reach a zero-laxity condition, assuring EDZL-schedulability. To do that, we initially set every slack lower bound to zero, and apply Theorem 10 to compute a lower bound

```
EDZL-I($\tau$)
1  $s^{lb} \leftarrow (0,\ldots,0)$
2  $Converged \leftarrow True$
3  repeat
4    $Infeasible \leftarrow 0$
5    for $k \in \{1\ldots n\}$ do {
6      $NewSlack \leftarrow \Delta_k^* - e_k - \frac{1}{m}\sum_{i\neq k}\min(n_i^*(s_i^{lb})\cdot e_i + \varepsilon_i(s_i^{lb}), \Delta_k^* - e_k)$
7      if $s_k^{lb} < NewSlack$ then {
8        $s_k^{lb} \leftarrow NewSlack$
9        $Converged \leftarrow False$
10     }
11     if $s_k^{lb} \leq 0$ then $Infeasible \leftarrow Infeasible + 1$
12   }
13 until $Converged$ or $Infeasible \leq m$
14 return $Infeasible \leq m$
```

Fig. 3 Pseudo-code for iterative EDZL schedulability test

for a task τ_k. If (12) returns a negative value, the slack bound value is not updated. Otherwise the tighter slack bound is used to compute a lower bound on the slack of another task, and so on. In this way, we can iteratively refine the slack estimate for every task, using at each round the tighter values derived at preceding steps. When $n-m$ tasks have a positive slack lower bound value, we can declare the task system schedulable. Otherwise, we continue iterating until no further slack update is possible. The function of $(s_1^{lb},\ldots,s_n^{lb})$ defined by the expression on the right-hand side of (12) is non-decreasing with respect to each of the values (s_i^{lb}), the slack of each task cannot exceed $d_i - e_i$, and we are dealing with integer time values, so this iterative process must converge.

The pseudo-code for this algorithm, which we call EDZL-I, is given in Fig. 3.

Note that this algorithm is slightly more conservative than the theorem on which it is based, since it treats zero slack as a scheduling failure rather than success.

Each iteration of the outer loop that does not result in *Converged* $=$ *True* increases at least one of the s_i^{lb} values, and none of those values can exceed $d_i - e_i$. Therefore, the number of iterations of the outer loop cannot exceed $\sum_{i=1,\ldots,n}(d_i - e_i)$. The inner loop has complexity $\mathcal{O}(n^2)$, figured as $\mathcal{O}(n)$ for the number iterations of the loop, times $\mathcal{O}(n)$ for the sum in the computation of *NewSlack*. Thus, the complexity of the entire algorithm is at most $\mathcal{O}(n^2\sum_{i=1,\ldots,n}(d_i - e_i))$. This is clearly a gross over-estimate. For the several million task sets examined in the experiments described below, we never required more than six (6) iterations of the outer loop. If slower convergence of the algorithm should become a concern for larger task sets, an arbitrary bound can be imposed on the number of iterations of the loop, or a minimum can be imposed on the amount by which the slack of a task must decrease to set *Converged* to *False*.

10 Experimental evaluation

In order to see how well the EDZL algorithm and the above schedulability tests perform, a series of experiments were conducted. In the first set of experiments the EDZL test of Theorem 7 and the EDZL-I test were applied to pseudo-randomly chosen task systems. For comparison, the following four combinations of a global multiprocessor scheduling algorithm and schedulability test were tested:

- EDF—pure global earliest-deadline-first scheduling, using the generalization of the utilization-based test of Goossens et al. (2003) to density, called GFB in Bertogna et al. (2005), and the test called BCL in Bertogna et al. (2005). Since each of these tests is able to recognize some cases of schedulable task sets that the other cannot, the combination was chosen to represent the currently most accurate sufficient schedulability test for pure global EDF scheduling.
- EDF-UM—a hybrid between EDF and utilization-monotonic scheduling. It assigns top priority to jobs of the $k-1$ tasks that have the highest utilizations, and assigns priorities according to deadline to jobs generated by all the other tasks, where k is the minimum value in the range $1,\dots,m$ for which the remaining $n-k$ tasks can be shown to be schedulable on $m-k$ processors using either the GFB or BCL test. A similar algorithm was found to be top performer among several global scheduling algorithms studied in Baker (2006).
- EDZL—pure EDZL scheduling, with the schedulability test of Theorem 7.
- EDZL-I—the iterative schedulability test for EDZL presented in Sect. 9.

Tables 1, 2 and Figs. 4–7 show the results of some experiments on collections of pseudo-randomly generated task sets. Several batches of such experiments were run, with collections of task sets generated according to various rules. Task systems that were trivially schedulable ($n \le m$ or total density ≤ 1) were thrown out, as were task systems that were obviously infeasible according to the tests $U_{\text{sum}} > m$ and throw-forward load $> m$ (Baker and Cirinei 2006). Task sets that were duplicates of those previously tested, regardless of task order, were also thrown out.

Table 1 Verifiably schedulable cases out of 1,000,000 task sets, with $d_i \le p_i$

Processors	EDF	EDF-UM	EDZL	EDZL-I	EDF/EDZL	EDF-UM only
4	82085	290750	327729	408809	408818	4989
8	48126	244222	320602	387780	387780	474
16	20291	199459	317067	380185	380185	7

Table 2 Verifiably schedulable cases out of 1,000,000 task sets, with $P(d_i > p_i) = 1/3$

Processors	EDF	EDF-UM	EDZL	EDZL-I	EDF/EDZL	EDF-UM only
4	169748	428785	479511	527693	527987	10533
8	88532	377018	493968	532721	532721	928
16	28200	339815	537945	576180	576180	9

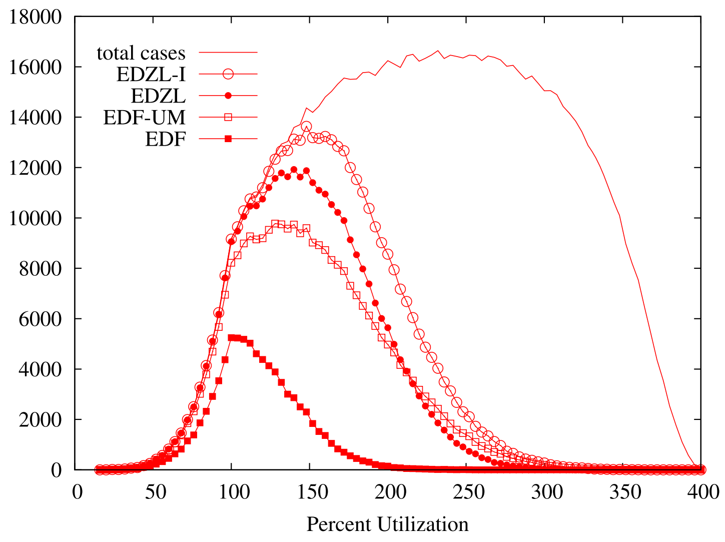


Fig. 4 Comparison of EDZL and EDF schedulability tests on 4 processors

The results of all the experiments were very similar in flavor, so the data from just a few are reported here. Those are for two collections of 1,000,000 task sets which had periods uniformly distributed in the range 1..1000 and utilization exponentially distributed with mean 0.25, for $m = 4, 8, 16$ processors. For the first group the deadlines were constrained to be uniformly distributed in the range $[u_i p_i, p_i]$. The results of these experiments are shown in Table 1 and Figs. 4–6. For the second group, deadlines were allowed to exceed the task periods, as follows:

- with probability $1/3$, $d_i = p_i$ (periodic);
- with probability $1/3$, d_i was uniformly chosen from the range $[u_i p_i, p_i - 1]$ (pre-period deadlines);
- with probability $1/3$, d_i was uniformly chosen from the discrete values $2p_i, \dots, 5p_i$ (post-period deadlines).

Columns 2–5 of Tables 1 and 2 show the total numbers of task sets (out of 1,000,000) that could be verified as schedulable by each of the tests. Clearly, EDZL with the available tests is effective in verifying schedulability of more task sets than EDF with the available tests, and there were very few cases where combining the EDF and EDZL tests was of any benefit. While EDZL is effective in many more cases than EDF-UM, overall, there were a few task sets that could be verified as schedulable by EDF-UM but not verified as EDZL-schedulable; those numbers are shown in the last column of the tables.

Since the EDF-UM criteria were able to verify schedulability for many more task sets than the pure EDF criteria (and in a few cases could verify schedulability of task sets for which the EDZL and EDZL-I tests failed), we also experimented with a hybrid of EDZL and utilization-monotonic (EDZL-UM). The results are not shown here because there was no difference between pure EDZL and EDZL with the iterative test. We believe this is a property of the zero-laxity scheduling rule, which is

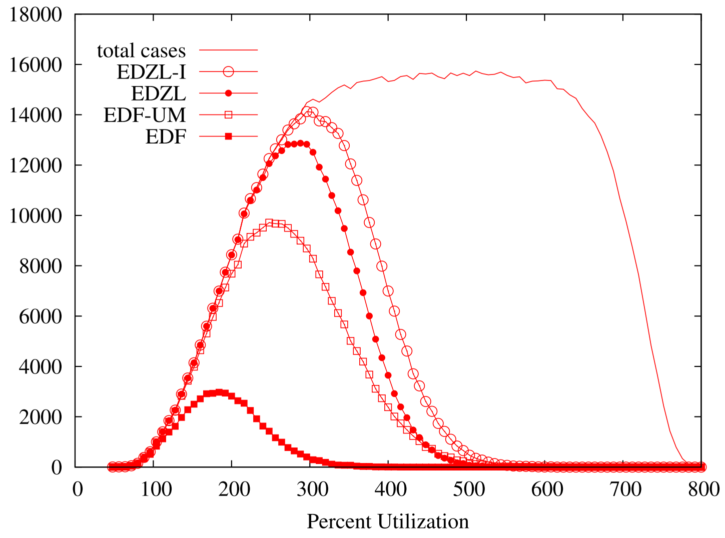


Fig. 5 Comparison of EDZL and EDF schedulability tests on 8 processors

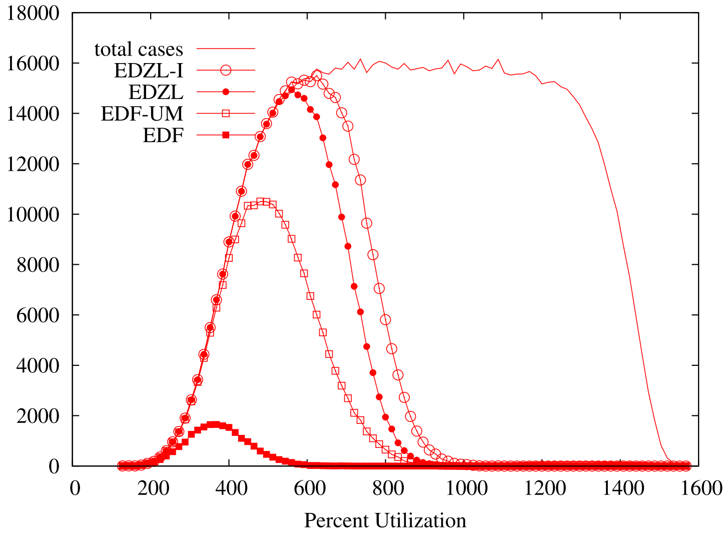


Fig. 6 Comparison of EDZL and EDF schedulability tests on 16 processors

already a hybrid with EDF of a different kind; EDZL gives top priority to tasks that are in danger of missing their deadlines; this cannot be improved upon by giving top priority to any other tasks.

Figures 4–7 provide a more detailed view. Each graph is a histogram in which the X axis corresponds to the total processor utilization U_{sum} and the Y axis corresponds to the number of task sets with U_{sum} in the range $[X, X + 0.01)$ that satisfy a given

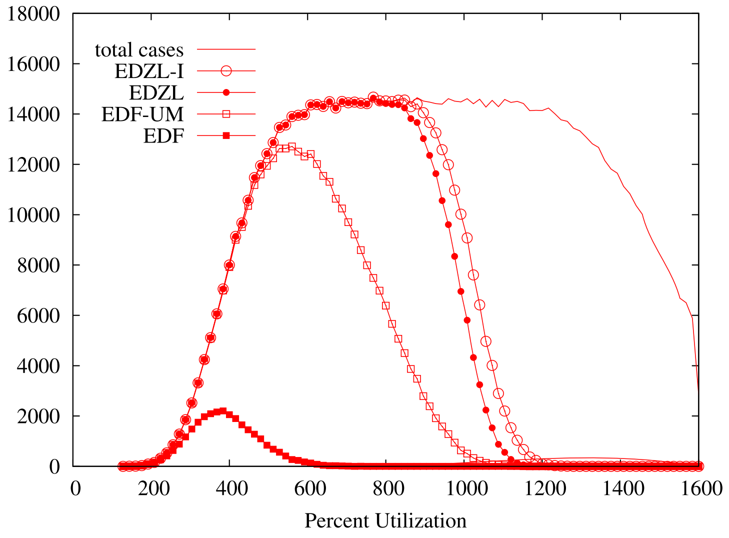


Fig. 7 Comparison of EDZL and EDF schedulability tests on 16 processors with some post-period deadlines

criterion. For the top line, which is unadorned, there is no additional criterion. That is, the Y value is simply the number of task sets with $X \leq U_{\text{sum}} < X + 0.01$. For the other lines, the criteria are the EDF, EDF-UM, EDZL, and EDZL-I as described above.

The graphs show how the performances of the tests vary with respect to the total utilization of the task sets. At most utilization levels the EDZL-I test dominates the EDZL test, and EDZL dominates EDF. However, EDF-UM does better than EDZL for smaller numbers of processors and task sets with high utilization.

Note that the above experiments do not distinguish performance differences due to differences in accuracy of the schedulability tests from differences in ability of the scheduling algorithms. That is, there is no distinction between (1) a task set that is schedulable by the given algorithm but cannot be verified as schedulable by the given test, and (2) a task set that is not schedulable by the given algorithm.

To the best of our knowledge, there are no known algorithms other than "brute force" exhaustive state enumeration that can distinguish the above two cases. However, it is practical to perform an exhaustive verification of pure EDF and pure EDZL schedulability for tasks sets with very short periods, using the brute-force algorithm presented in Baker and Cirinei (2007). Figure 8 shows the result of experiments using such a necessary-and-sufficient test of schedulability for pure EDF and pure EDZL schedulability for 4 processors on a collection of 1,000,000 sets, without repetitions, of tasks with periods in the range 1..5. All tasks with 100% utilization were considered equivalent to the task with unit period, deadline, and execution time. Task sets that differed only in the order of tasks were considered repetitions. So were task sets that were only "scaled up" by a constant factor from a prior task in the enumeration. Tests on larger task systems were not practical, due to the exponential growth in time and storage requirements of the necessary-and-sufficient algorithm.

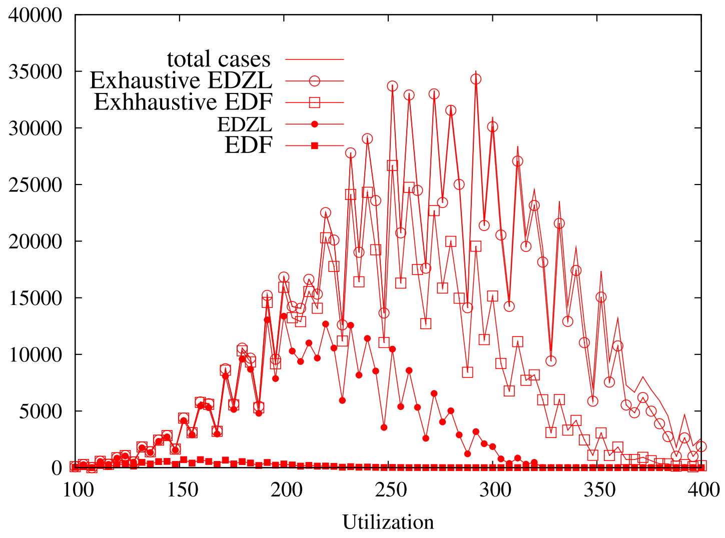


Fig. 8 Comparison of EDZL and EDF algorithms on 4 processors, using exhaustive schedulability tests

The lines labeled "Exhaustive EDZL" and "Exhaustive EDF" show the number of task sets that were schedulable using the necessary-and-sufficient (brute force, exhaustive) tests of sporadic schedulability according to global EDF and global EDZL algorithms, respectively. The lines labeled "EDZL", "EDF", and "total cases" have the same meaning as in Figs. 4–6.

The graph is more jagged in appearance than those in Figs. 4–6, because limitation of periods and execution times to 1..5 made many utilization values impossible or very improbable.

For this collection of task sets, it is clear that there is much room for improvement in the sufficient schedulability tests for both EDF and EDZL, which fail to recognize most of the schedulable task sets. It is also clear that EDZL outperforms EDF by a significant margin, both in combination with the conservative sufficient schedulability tests and with the necessary-and-sufficient schedulability test. In fact, EDZL was able to schedule virtually all of the task sets. Of course, it remains to be seen whether the behavior of the algorithms on such simple, small task sets generalizes to larger task sets and other numbers of processors.

11 Conclusions and future work

The tests reported here are the first known schedulability tests for global EDZL scheduling on a multiprocessor platform. They provide two different ways to check the schedulability of a task set. Theorem 7 is a simple test with linear complexity. The iterative algorithm EDZL-I allows one to detect more schedulable task sets at a slightly higher computational cost. The experiments with pseudo-randomly generated task sets indicate that EDZL with these sufficient schedulability tests is not only superior in performance to pure global EDF, but in most cases is also superior to an alternate EDF hybrid global scheduling that is known to outperform pure EDF.

The approach followed in this analysis is very similar to that followed for EDF in Baker (2003) and Bertogna et al. (2005). Therefore, we hope to be able to continue to extend the analysis of EDZL along similar lines. In particular, we plan to introduce a tighter bound for the carry-in, using the technique proposed for EDF in Baker (2003).

Another aspect that needs attention is the assumption that if $m + 1$ tasks can reach zero laxity individually (what we test for), they can reach zero laxity *at the same time* (what is required for a job to miss a deadline). This assumption clearly introduces some pessimism in the analysis, and should be addressed in future extensions.

A more ambitious, but for the moment very distant, goal is the extension of the whole analysis, in order to find a density bound for EDZL on a multiprocessor similar to the EDF density bound for implicit deadline systems. However, the proof of the density bound in Goossens et al. (2003) is based on a "resource augmentation" argument, which relates how long it takes to complete a set of jobs on m processors to how long it takes to complete them on a single processor. Since EDF is already optimal on one processor, it does not seem that this technique can derive any tighter bound with EDZL, so a new proof technique may be required.

Empirical study is needed of the implementation of EDZL scheduling, including the comparative overhead of EDZL versus simple EDF scheduling. The main functional difference is that the EDZL scheduler needs to detect when a task reaches zero laxity, in order to raise its priority. This seems similar, but a bit simpler, than what is required for least-laxity-first scheduling. It can be done using a single timer if one maintains an auxiliary queue of the non-running tasks ordered by laxity. Rather than directly tracking laxity, which will be decreasing over time for the non-running tasks, one can maintain the queue in increasing order of the absolute time at which the laxity is predicted to reach zero. The predicted zero-laxity time remains static for each job that is not running. The timer is set to go off at the earliest predicted zero-laxity time. The timer needs to be reset whenever a task with earliest predicted zero-laxity time is allowed to execute, and whenever a new task is released that is not chosen to run immediately and has an earlier zero-laxity time than the other non-running tasks. It seems likely that the overhead of this timer management will not be greater than, for example, managing budget replenishments for algorithms like Sporadic Server (Sprunt et al. 1989).

References

Baker TP (2003) Multiprocessor EDF and deadline monotonic schedulability analysis. In: Proc of the 24th IEEE real-time systems symposium, Cancun, Mexico, pp 120–129

Baker TP (2006) A comparison of global and partitioned EDF schedulability tests for multiprocessors. In: International conf on real-time and network systems, Poitiers, France, pp 119–127

Baker TP, Cirinei M (2006) A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In: Proc of the 27th IEEE real-time systems symposium, Rio de Janeiro, Brazil

Baker TP, Cirinei M (2007) Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In: Principles of distributed systems, 11th int conf, OPODIS 2007, Guadeloupe, French West Indies. Springer, Berlin, pp 62–75

Bertogna M, Cirinei M, Lipari G (2005) Improved schedulability analysis of EDF on multiprocessor platforms. In: Proc of the 17th Euromicro conference on real-time systems, Palma de Mallorca, Spain, pp 209–218

Cho S, Lee S-K, Han A, Lin K-J (2002) Efficient real-time scheduling algorithms for multiprocessor systems. IEICE Trans Commun E 85-B(12):2859–2867

Cirinei M (2007) Exploiting the power of multiprocessors for real-time systems. PhD thesis, Scuola Superiore S Anna, Pisa, Italy

Cirinei M, Baker TP (2007) EDZL scheduling analysis. In: Proc EuroMicro conference on real-time systems, to appear

Goossens J, Funk S, Baruah S (2003) Priority-driven scheduling of periodic task systems on multiprocessors. Real Time Syst 25(2–3):187–205

Ha R (1995) Validating timing constraints in multiprocessor and distributed systems. PhD thesis, Dept of Computer Science, University of Illinois, Urbana-Champaign, IL

Ha R, Liu JWS (1994) Validating timing constraints in multiprocessor and distributed real-time systems. In: Proc of the 14th IEEE international conf on distributed computing systems, Poznan, Poland. IEEE Computer Society, Los Alamitos, pp 162–171

Park M, Han S, Kim H, Cho S, Cho Y (2005) Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. IEICE Trans Inf Syst E 88-D(3):658–661

Piao X, Han S, Kim H, Park M, Cho Y, Cho S (2006) Predictability of earliest deadline zero laxity algorithm for multiprocessor real time systems. In: Proc of the 9th IEEE international symposium on object and component-oriented real-time distributed computing, Gjeongju, Korea

Sprunt B, Sha L, Lehoczky L (1989) Aperiodic task scheduling for hard real-time systems. Real-Time Syst 1(1):27–60

Theodore P. Baker is a professor in the Department of Computer Science at the Florida State University, and served four years as chair of that department. He earned his Ph.D. degree in computer science from Cornell University in 1974, for research on relative computability and computational complexity. Starting in 1979, Professor Baker became involved with the development of the Ada programming language. The group he organized at FSU produced one of the first validated Ada cross-compilers for embedded systems. Since then he has done research, development, and consulting related to real-time embedded computing, from basic research on scheduling and concurrency control through development of kernels and run-time system support for real-time programming languages. He has also been active in IEEE (POSIX) and ISO standards work related to real-time systems. Professor Baker was a member of the SEI Rate Monotonic Analysis group, served as real-time area expert for the Ada 9X language mapping and revision team, and was a member of the 1997 National Research Council panel on Software Policies for the Department of Defense. He directed the FSU teams that developed several software artifacts, including the FSU POSIX threads library, the Florist implementation of the POSIX.5 API, a validation suite for the same, and the multitasking run-time system for the Gnu Ada (GNAT) compiler. He directed the porting of the latter to several environments, including the Java Virtual Machine and RTLinux. Professor Baker's current research focii are multiprocessor scheduling theory and experimentation, and real-time device driver architecture.

Michele Cirinei received the Laurea degree (summa cum laude) in computer science engineering from the University of Pisa, Italy, in 2004. In 2007 he received his Ph.D. degree in computer science from the Scuola Superiore Sant'Anna in Pisa, Italy, for his research on the use of multiprocessor platforms for real-time systems. His research particularly focused on how to improve both computational power and fault tolerance of systems with strict time constraints. Since July 2007, he moved to Arezzo, where he has been employed as software designer by SECO, an European designer, manufacturer and marketer of high integrated board computers and systems for embedded applications.

Marko Bertogna graduated (summa cum laude) in Telecommunications Engineering at the University of Bologna (Italy), in 2002. In 2008, he received the Ph.D. in Computer Science from Scuola Superiore Sant'Anna in Pisa (Italy), where he currently has a post-doc position. In 2002 he visited TU Delft (Netherlands) working on optical devices. In 2006 he visited the University of North Carolina at Chapel Hill (USA), working with prof. Sanjoy Baruah on scheduling algorithms for single and multi-core real-time systems. His research interests include scheduling and schedulability analysis of real-time multiprocessor systems, protocols for the exclusive access to shared resources, reconfigurable devices. He received the 2005 IEEE/Euromicro Conference on Real-Time Systems Best Paper Award. He is author of 10 papers in peer-reviewed international journals and conferences.