EDZL scheduling analysis

Michele Cirinei
Scuola Superiore Sant'Anna
Pisa, Italy
e-mail: cirinei@gandalf.sssup.it

Theodore P. Baker*
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4530
e-mail: baker@cs.fsu.edu

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.

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 a test that is sufficient to guarantee that a system of independent sporadic tasks with arbitrary deadlines will not miss any deadlines if scheduled by global EDZL on a platform with m identical processors. We also show through experiments how the new EDZL schedulability test compares to known schedulability tests for global Earliest-Deadline-First (EDF) scheduling, on large numbers of pseudo-randomly generated task sets.

EDZL has been studied by Cho *et al.* [8], who showed 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, and out-performs global EDF in particular, while not incurring significantly more context switches compared to global EDF.

The latter property, of incurring few context switches, makes EDZL an attractive alternative to known optimal global algorithms that require fine-grained time slicing, such as the Pfair[5] family. The ability to operate on-line, without exact knowledge of future job arrival times, also makes it more attractive than algorithms that are restricted to strictly periodic task sets, such as the optimal LLREF algorithm[7].

However, for systems with hard deadlines, good or even optimal scheduling performance is not enough, if there is no practical way to verify that a particular task set will be scheduled to meet all its deadlines. In this paper, we address that need, by deriving a sufficient test of schedulability for sporadic task sets under global EDZL scheduling on m identical processors.

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 $e_i \leq \min(d_i, p_i)$, since otherwise a job would never be able

*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.

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 $d_i \leq p_i$ 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, a job cannot start its execution 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 $\tau_i \neq \tau_k$ 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 $W^J(a, b)/(b - a)$. For notational simplicity, the superscript will be omitted if the identity of the job J is clear from context.

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 [8]. Moreover, it is quite easy to show that EDZL strictly dominates EDF (see Theorem 2 in [12]), 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 and al. [8], EDZL is actually the EDF algorithm with a "*safety rule*" (the zero laxity rule) to be applied

[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.

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, Funk and Baruah [9] 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.

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 [11, 10] 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.* [13] 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 [1, 6]: 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 call the job the *problem job*, its task the *problem task*, and the interval between its release time and deadline the *problem window*. Let τ_k always denote the problem task. Moreover, $\bar{t}$ denotes the missed deadline (i.e. the deadline of the problem job), and $[\bar{t} - d_k, \bar{t})$ 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.

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 the upper bound of (3) is less than the lower bound of (1), that would be a contraction, so there can be no problem job; that is, the task is schedulable.

The main difference between the EDZL test we propose, and the EDF tests explained in [1, 6], is that with EDF it is sufficient to find a possibly unschedulable task to conclude that the task set could be unschedulable, while for EDZL it is necessary to find at least $m + 1$ possibly unschedulable tasks, 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 unit 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 [1, 6]. However, in the case of EDZL a scheduling failure provides additional information.

Under EDZL, once the problem job 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 the problem job 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) with zero laxity at the same moment.

The problem job can reach zero laxity only if it is blocked for at least $d_k - e_k$ in the problem interval. Since EDZL is work-conserving, a released job can be blocked for only two reasons:

- precedence, by an older job of the same task;

- 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 the problem job. In this case the problem job 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 problem window. If $[\bar{t} - d_k, \bar{t})$ is the problem window, it follows that

$$\sum_{i \neq k} W_i(\bar{t} - d_k, \bar{t}) > m(d_k - e_k)$$

$$\sum_{i \neq k} W_i(\bar{t} - d_k, \bar{t})/d_k > m(1 - \lambda_k)$$

In the other case, if $d_k > p_k$, the precedence constraint could contribute to the blocking. However, we can still find a lower bound for the competing work and load. The only job of τ_k that can execute in the interval $[\bar{t} - p_k, \bar{t})$ is the problem job, since the preceding job of τ_k would have missed its deadline at $\bar{t} - p_k$. In the worst case scenario that we are considering all the m processors must be working on jobs other than the problem job for $p_k - e_k$ time units in the interval $[\bar{t} - p_k, \bar{t})$[2]. From this upper bound we obtain

$$\sum_{i \neq k} W_i(\bar{t} - p_k, \bar{t}) > m(p_k - e_k)$$

$$\sum_{i \neq k} W_i(\bar{t} - p_k, \bar{t})/p_k > m(1 - \lambda_k)$$

[2]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 problem job at time $\bar{t} - p_k$. Unfortunately, it is difficult to estimate the remaining computation time of a job without simulating the system.

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 definition of λ_k differs for the two cases, the expression for the load bound is the same. Therefore, both bounds can be unified in one lemma.

Lemma 1 *If EDZL is used to schedule a sporadic task system $\tau = \{\tau_1, ..., \tau_n\}$ on m identical processors then a problem job J of task τ_k with deadline $\bar{t}$ can reach zero laxity only if*

$$\sum_{i \neq k} W_i(\bar{t} - \Delta_k, \bar{t})/\Delta_k \geq m(1 - \lambda_k) \qquad (1)$$

and can reach negative laxity only if the $>$ strictly holds.

Proof. The lemma follows from the above discussion. □

From now on we define the *overload window* of the problem job to be the interval under analysis above, *i.e.*, $[\bar{t} - \Delta_k, \bar{t})$, noting that the length of this interval is always equal to Δ_k.

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 1, 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 ($>$).

6 Upper bound

In this section we derive an upper bound for the contribution of a task τ_i to the competing work of the problem job in the overload window. We first determine the worst case release times of the jobs of τ_i in the overload 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 problem 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 decrease the competing work.

As contributors to the competing work, we do not need to consider jobs that have a deadline before the overload window. Since there are no missed deadlines before the end of the overload window, such jobs must complete before it.

We do need to consider as contributors to the competing work every job of τ_i that has its deadline in the overload window. 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 deadline. Jobs with both release time and deadline in the overload window are not influenced by this assumption, while it can only increase (and cannot decrease) the contribution of jobs with release time before the window.

We also need to consider jobs released in or before the overlead window and with deadline after the overload window. Such a job can compete with the problem job 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 overload window 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 the problem interval 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 $\bar{t}$ of the problem job, as shown in Figure 1.

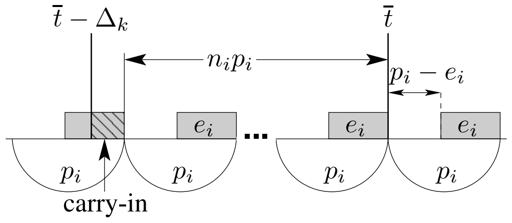

Figure 1. Upper bound on carry-in

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 problem window. 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 overload window 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 Figure 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 before $\bar{t} - \Delta_k$). 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 of the problem interval is achieved when the jobs of τ_i are released periodically and one deadline of τ_i coincides with the deadline of the problem job.

6.2 Worst case competing work

It is now easy to compute an upper bound for the competing work of a task τ_i in the overload window. In the worst-case scenario that we are considering, when all the jobs are released periodically and execute exactly before their deadline, the competing work is composed of two different contributions:

1. The contribution of a number n_i of jobs of τ_i that have both start time and deadline in the overload window. Each of these contributes exactly e_i.
2. The contribution of at most one job, called the carried-in job, whose execution starts before the starting time of the window, and ends inside the window. This contribution, called the *carry-in*, is clearly no more than e_i.

The maximum number n_i of jobs completely executed inside the overload window depends on Δ_k (the length of the overload window) and can be computed as

$$n_i = \left\lfloor \frac{\Delta_k}{p_i} \right\rfloor$$

A carried-in job exists only if the starting time of the overload window falls between the release and completion time of some job of τ_i (which is the carried-in job), and the contribution of the carried-in job to the competing work is no greater than the length of the interval between the start of the overload window and the completion time of the carried-in job. If $[\bar{t} - \Delta_k, \bar{t})$ is the overload window and $n_i > 0$, the deadline of the last of the n_i jobs is at time $\bar{t}$ and the deadline of the first is at time $\bar{t} - n_i p_i$. The length of the interval during which the carried-in job can execute is $\max(0, \Delta_k - n_i p_i)$, so the size of the carry-in cannot be greater than $\min(e_i, \max(0, \Delta_k - n_i p_i))$. This upper bound clearly remains valid if $n_i = 0$. Note that the max function is necessary for tasks with constrained deadline ($d_i < p_i$), because in some cases $\Delta_k - n_i p_i$ could be negative, while clearly the carry-in cannot.

Lemma 2 *If EDZL is used to schedule a sporadic task system $\tau = \{\tau_1, ..., \tau_n\}$ on m identical processors, the competing work contributed by task τ_i in the overload window $[\bar{t} - \Delta_k, \bar{t})$ of a job J of task τ_k is subject to the bound*

$$W_i(\bar{t} - \Delta_k, \bar{t}) \leq n_i e_i + \min(e_i, \max(0, \Delta_k - n_i p_i))$$

Proof. The proof follows from the preceding discussion. □

Note that the upper bound depends only on the length of the overload window, and not on the specific start and end points of the interval. Moreover, once the task τ_k under analysis is selected, the length of the overload window is fixed. So, we can define an upper bound for the load of τ_i in the overload window of τ_k as

$$\beta_k^i = \frac{n_i e_i + \min(e_i, \max(0, \Delta_k - n_i p_i))}{\Delta_k} \qquad (2)$$

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 2 (First EDZL test) *A sporadic task system $\tau = \{\tau_1 ... \tau_n\}$ 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:*

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

where β_k^i is defined as in Equation 2.

Proof. According to Lemma 1, a job J can reach zero laxity only if the competing work of the other tasks in its overload window is greater or equal to $m(1 - \lambda_k)$. 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 (3). □

8 Second schedulability test

To improve the precision of Theorem 2, we now reconsider the above definitions and lemmas, verifying and adapting them to deal with interference, a concept introduced by Bertogna, Cirinei and Lipari in [6]. Some of the following results can be found, only with a sligthly different notation, in [6], 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 *interference* $I_i^J(a,b)$ of a task τ_i on a job J over an interval $[a,b)$ is the cumulative length of all the intervals in which J is priority-blocked and a job of τ_i is one of the m jobs blocking the problem job J.

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, if we focalize on the overload window $[\bar{t}-\Delta_k,\bar{t})$ of task τ_k, in no case there can be precedence-blocking, so we can avoid to distinguish the two cases. For this reason, from now on we always consider the overload window (this is the main difference with the analysis in [6], where no particular interval was selected).

By the definition, it is clear that in the overload window of every job J of τ_k we have

$$I_i^J(\bar{t}-\Delta_k,\bar{t}) \leq I^J(\bar{t}-\Delta_k,\bar{t}) \quad \forall i.$$

and

$$I_i^J(\bar{t}-\Delta_k,\bar{t}) \leq W_i(\bar{t}-\Delta_k,\bar{t}) \leq \beta_k^i\Delta_k \leq \Delta_k \quad \forall i.$$

Moreover, 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 interference are increased. From this descends that

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

The above results can be used to prove the following

Lemma 3 *(Lemma 4 in [6])* $I^J(\bar{t}-\Delta,\bar{t}) \geq x \iff \sum_{i\neq k}\min(I_i^J(\bar{t}-\Delta_k,\bar{t}),x) \geq mx.$

Proof.

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

$$\sum_{i\neq k}\min(I_i^J(\bar{t}-\Delta_k,\bar{t}),x) = \xi x + \sum_{\tau_i\notin\tau'} I_i^J(\bar{t}-\Delta_k,\bar{t}) =$$
$$= \xi x + mI^J(\bar{t}-\Delta_k,\bar{t}) - \sum_{\tau_i\in\tau'} I_i^J(\bar{t}-\Delta_k,\bar{t}) \geq$$
$$\geq \xi x + mI^J(\bar{t}-\Delta_k,\bar{t}) - \xi I^J(\bar{t}-\Delta_k,\bar{t}) =$$
$$= \xi x + (m-\xi)I^J(\bar{t}-\Delta_k,\bar{t}) \geq \xi x + (m-\xi)x = mx.$$

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

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

□

Considering the definition of *interference*, it is clear that a job of τ_k (i.e., the problem job) can reach zero laxity only if $I^J(\bar{t}-\Delta_k,\bar{t}) \geq (\Delta_k - e_k)$. Note that this is again a worst-case assumption which introduces some pessimism in the analysis. Applying Lemma 3, we have that the problem job can reach zero laxity only if

$$\sum_i \min(I_i^J(\bar{t}-\Delta_k,\bar{t}),\Delta_k - e_k) \geq m(\Delta_k - e_k)$$

and so

$$\sum_i \min(I_i^J(\bar{t}-\Delta_k,\bar{t})/\Delta_k, 1-\lambda_k) \geq m(1-\lambda_k). \quad (4)$$

It is very difficult to correctly compute the interference $I_i^J(\bar{t}-\Delta_k,\bar{t})$. However, we can use the above upper bounds, and in particular introduce β_k^i in Equation 4. We obtain the following Lemma (compare with Lemma1).

Lemma 4 *If EDZL is used to schedule a sporadic task system $\tau = \{\tau_1,...,\tau_n\}$ on m identical processors then a problem job J of task τ_k with deadline $\bar{t}$ can reach zero laxity only if*

$$\sum_i \min(\beta_k^i, 1-\lambda_k) \geq m(1-\lambda_k) \quad (5)$$

and can reach negative laxity only if the $>$ strictly holds.

Proof. The lemma follows from the above discussion. □

Thanks to this result we can now formulate the following refined version of Theorem 2. The proof remains identical, with the only difference that Lemma 4 is used instead of Lemma 4.

Theorem 3 (Refined EDZL test) *A sporadic task system $\tau = \{\tau_1,...,\tau_n\}$ is schedulable by EDZL on m identical processors unless the following inequality holds for least $m+1$ different tasks τ_k, and it holds strictly ($>$) for at least one of them:*

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

where β_k^i is defined as in Equation 2.

9 Experimental Evaluation

In order to see how well the EDZL algorithm and the above schedulability test perform, a series of experiments were conducted. In the first set of experiments the EDZL test of Theorem 3 was 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, Funk and Baruah [9] to density (called GFB in [6]) and the test of Bertogna, Cirinei and Lipari (called BCL in [6]). 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 $1, \ldots, m$ 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 [2].

- EDZL – pure EDZL scheduling, with the schedulability test of Theorem 3.

- EDZL or EDF – pure EDZL scheduling, using the schedulability test of Theorem 3, and also the two EDF schedulability tests, using the fact that every task set that is schedulable by global EDF is also schedulable by EDZL.

Figures 2-4 show the result of experiments on 1,000,000 pseudo-randomly generated task sets with periods uniformly distributed in the range 1..1000, utilization exponentially distributed with mean 0.25, and deadlines uniformly distributed in the range $[u_i p_i, p_i]$, for $m = 4, 8, 16$ processors. Task systems that were trivially schedulable ($n \leq m$ or total density ≤ 1) were thrown out, as were task systems with total utilization greater than m. Task sets that were duplicates of those previously tested, regardless of task order, were also thrown out. All tasks with 100% utilization, regardless of period, were considered identical.

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 criterion.

For the top line, which is unadorned, there is no additional criterion. That is, the Y value is simply the number

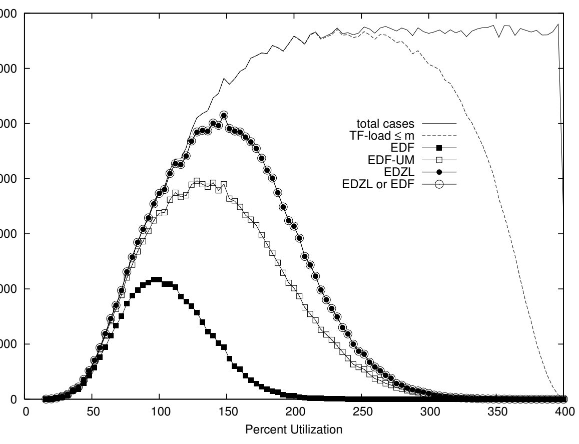

Figure 2. Comparison of EDZL and EDF schedulability tests on 4 processors

of task sets with $X \leq U_{sum} < X + 0.01$. For the second line, which is dashed, the additional criterion is that the task set was not found to be entirely infeasible by the test throw-forward load $\leq m$[3]). For the other lines, the criteria are the EDF, EDZL, EDF-UM, and the combined EDF and EDZL criteria as described above.

Global EDZL with our schedulability test is able to verifiably schedule more task sets than the pure global EDF or hybrid EDF-UM scheduling policies with the available schedulability tests. Since the EDF-UM criteria were able to verify schedulability for many more task sets than the pure EDF criteria, we also experimented with a hybrid of EDZL and utilization-monotonic (EDZL-UM). The results are not shown here because there was virtually no difference between pure EDZL and EDZL-UM. We believe this is a property of the zero-laxity scheduling rule, which is 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.

Many additional tests were run, with the individual task utilizations generated according to an exponential distribution with mean 0.15, a uniform distribution, and a bimodal distribution, and with unconstrained as well as constrained deadlines. The results were very similar. One example with unconstrained deadlines is shown in Figure 5. The task sets were generated as for Figure 4, except that the deadlines were uniformly distributed in the range $[u_i p_i, 4p_i]$.

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,

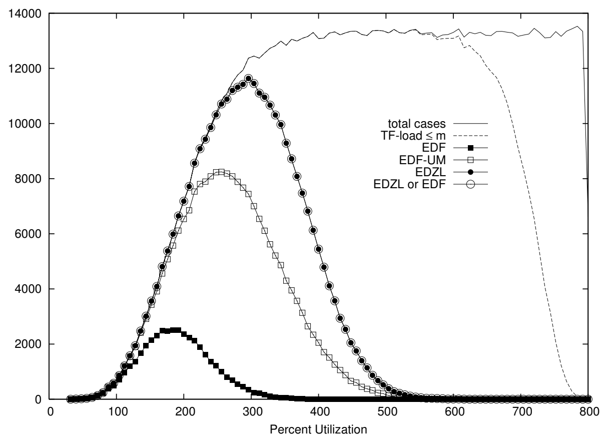

Figure 3. Comparison of EDZL and EDF schedulability tests on 8 processors

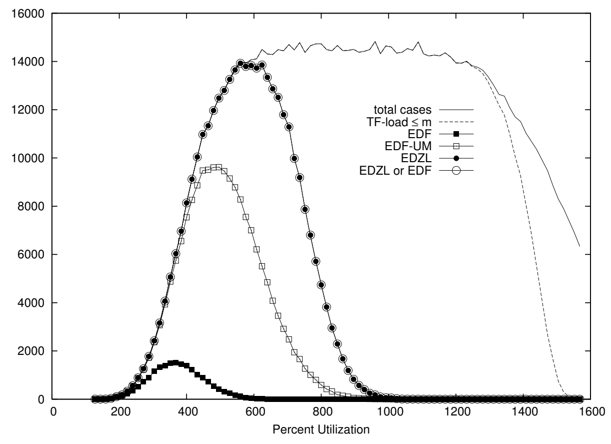

Figure 4. Comparison of EDZL and EDF schedulability tests on 16 processors

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. Figure 6 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.

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 "TF-load $\leq$ m" have the same meaning as in Figures Figures 2-4.

The graph is more jagged in appearance than those in Figures 2-4, because of the limitation of periods and execution times to 1..5 made some 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.

10 Conclusions and Future Work

Theorem 3 is the first known schedulability test for EDZL on a multiprocessor platform. The empirical tests indicate that EDZL with this sufficient schedulability test is not only superior in performance to pure global EDF, but 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 [1, 6]. 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 by Baker in [1].

We also hope to verify that our EDZL schedulability tests are *sustainable*, as the term is defined by Baruah and Burns [4].

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 [9] 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

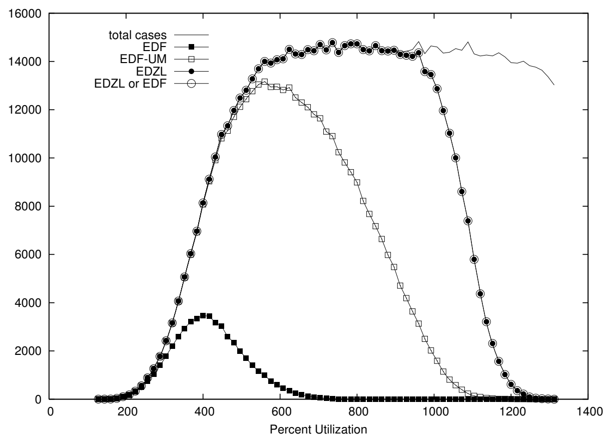

Figure 5. Comparison of EDZL and EDF schedulability tests on 16 processors with some post-period deadlines.

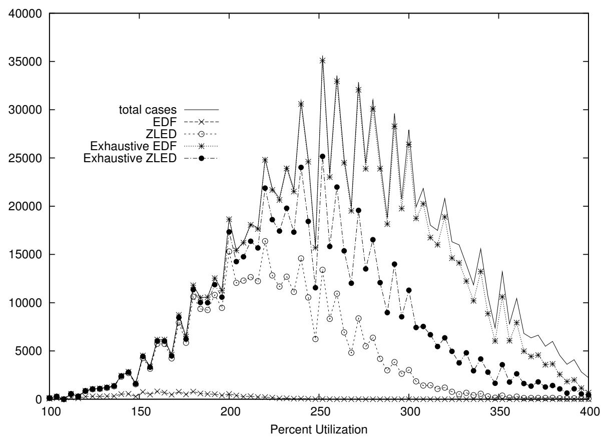

Figure 6. Comparison of EDZL and EDF algorithms on 2 processors, using exhaustive schedulability tests.

a new proof technique may be required.

References

[1] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proc. 24th IEEE Real-Time Systems Symposium*, pages 120–129, Cancun, Mexico, 2003.

[2] T. P. Baker. A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *International Conf. on Real-Time and Network Systems*, pages 119–127, Poitiers, France, June 2006.

[3] T. P. Baker and M. Cirinei. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In *Proc. 27th IEEE Real-Time Systems Symposium*, Rio de Janeiro, Brazil, Dec. 2006.

[4] S. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the* 27^{th} *IEEE Real-Time Systems Symposium, RTSS '06*, pages 159–168, Rio de Janeiro, Brasil, Dec. 2006.

[5] S. K. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel. Proportionate progress: a notion of fairness in resource allocation. In *Proc. ACM Symposium on the Theory of Computing*, pages 345–354, May 1993.

[6] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proc. 17th Euromicro Conference on Real-Time Systems*, pages 209–218, Palma de Mallorca, Spain, July 2005.

[7] H. Cho, B. Ravindran, and E. D. Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proc. 27th IEEE International Real-Time Systems Symposium*, Rio de Janeiro, Brazil, Dec. 2006.

[8] S. Cho, S.-K. Lee, A. Han, and K.-J. Lin. Efficient real-time scheduling algorithms for multiprocessor systems. *IEICE Trans. Communications*, E85-B(12):2859–2867, Dec. 2002.

[9] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real Time Systems*, 25(2–3):187–205, Sept. 2003.

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

[11] R. Ha and J. W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proc. 14th IEEE International Conf. Distributed Computing Systems*, pages 162–171, Poznan, Poland, June 1994. IEEE Computer Society.

[12] M. Park, S. Han, H. Kim, S. Cho, and Y. Cho. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. *IEICE Trans. on Information and Systems*, E88-D(3):658–661, Mar. 2005.

[13] X. Piao, S. Han, H. Kim, M. Park, Y. Cho, and S. Cho. Predictability of earliest deadline zero laxity algorithm for multiprocessor real time systems. In *Proc. 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, Gjeongju, Korea, Apr. 2006.