Efficient Scheduling of Reliable Automotive Multi-core Systems with PD^2 by Weakening ERfair Task System Requirements

Michael Deubzer[1], Ulrich Margull[2], Jürgen Mottok[1], Michael Niemetz[3], Gerhard Wirrer[3]
[1]University of Applied Sciences Regensburg, LaS^3 - Laboratory for Safe and Secure Systems, Faculty of Electrical Engineering and Information Technology, P.O. Box 12 03 27, D-93025 Regensburg,{michael.deubzer,juergen.mottok}@hs-regensburg.de
[2]1 mal 1 Software GmbH, Maxstraße 31, D-90762 Fürth, ulrich.margull@1mal1.com
[3]Continental Automotive GmbH, P.O. Box 100943, D-93009 Regensburg
{michael.niemetz,gerhard.wirrer}@continental-corporation.com

Abstract

Partly Proportionate fair (*Partly-Pfair*) scheduling, which allows task migration at runtime and assigns each task processing time with regard to its weight, makes it possible to build highly efficient embedded multi-core systems. Due to its non-work-conserving behavior, which might leave the CPU idle even when tasks are ready to execute, tasks finish only shortly before their deadlines are reached. Benefits are lower task jitter, but additional workload, e.g. through interrupts, can lead to deadline violations.
In this paper we present a work-conserving extension of *Partly-Pfair* scheduling, called *P-ERfair* scheduling and the algorithm *P-ERfair-PD*^{2} which applies *Pfair* modifications used for *Partly-Pfair* on the concept of *ERfair*ness and *PD*^{2} policies. With a simulation based schedulability examination we show for multiple time base (MTB) task sets that *P-ERfair-PD*^{2} has the same performance as *Partly-Pfair-PD*^{2}. Additionally, we show that *P-ERfair-PD*^{2} has a much higher robustness against perturbations, and therefore it is well suited for embedded domains, especially for the Automotive domain.

1. Introduction

Scheduling a task set[1] $\tau = \{T_i\}$ $(i = 1, ..., n;\ n \in \mathbb{N})$, with independent and hard real-time constrained tasks T_i, on a multiprocessor[2] $M = \{P_x\}$ $(x = 1, ..., m;\ m \in \mathbb{N})$ with m identical processing resources P_x, has been widely studied in the last two decades. Scheduling algorithms for these systems can be classified in two groups.
One group consists of algorithms using the partitioning scheduling approach, which allocates tasks before runtime. The benefit of this approach is that the scheduling problem can be treated like in uniprocessor systems, using well studied algorithms like Earliest Deadline First (EDF) or Rate Monotonic (RM) [1] together with partitioning heuristics, e.g. bin-packing variants [2], applied before runtime. One major drawback of the partitioning approach is a low maximal system utilization [3].
The other group consists of algorithms using the dynamic scheduling approach, which allocates tasks during runtime. Partly Proportionate fair (*Partly-Pfair*) Scheduling [4], a modification of Proportionate fair (*Pfair*) Scheduling [5], is a highly efficient mechanism of dy-

[1]A *task set* is the description of timing relevant embedded software properties and has to fulfill the restrictions given by the underlying task system.
[2]coll. Multi-Core

namic scheduling. In [4] we presented the algorithm *Partly-Pfair-PD*^{2}, which is based on *PD*^{2} policies [6]. The algorithm *Pfair-PD*^{2} is optimal[3] for a system with m processors.

In this paper we show how work-conserving scheduling can be applied to multiple time base (MTB) task systems [4] which allow to model a wide variety of automotive control systems. The resulting model is called partly early release fairness (*P-ERfair*) and is deduced from [8]. Using a simulation based analysis technique, we examine the performance of *P-ERfair*.

The remainder of this paper is organized as follows. In the next chapter we give a brief review of the properties of *ERfair* scheduling and *PD*^{2} policies as introduced by [8] and [6]. Then we describe the multiple time base (MTB) task system and the restrictions required for the application of *P-ERfair*ness to MTB task systems before we introduce in Chapter 5 the new scheduling algorithm *P-ERfair-PD*^{2} which is based on *P-ERfair* and *PD*^{2}. After a brief introduction to the simulation-based scheduling analysis we give a performance examination of *P-ERfair-PD*^{2}. Finally we give a conclusion of our contribution to dynamic multiprocessor scheduling.

2. Early Release Fair Scheduling

Early Release Fair scheduling was introduced by Anderson et al. [8]. It is an extension of proportionate fair (*Pfair*) scheduling, which was introduced by Baruah et al. [5] for periodic task systems, as defined by Liu and Layland [1]. *Pfair* implies that for each task T_i processing time is assigned according to its weight $\text{wt}(T_i)$,

$$\text{wt}(T_i) = \frac{T_i.e}{T_i.p} \tag{1}$$

with the worst case task execution time $T_i.e$ and the task activation period $T_i.p$. Anderson et al. [6] proved that the *Pfair* scheduling algorithm *PD*^{2} is optimal for scheduling a task set τ in a multiprocessor system with m processors iff formula 2 holds.

$$\sum_{i=1}^{n} \text{wt}(T_i) \leq m \tag{2}$$

Pfair is deduced from the *fluid scheduling* model. The fluid schedule *fluid*(T, t_1, t_2) represents the processing time, which has to be assigned to a task T_i during a time interval between t_1 and t_2 with regard to its weight $\text{wt}(T_i)$. Figure 1-a shows an example with two tasks T_1 and T_2. The fluid schedule graph of both tasks is shown as a thick line.

$$\textit{fluid}(T_i, t_1, t_2) = \text{wt}(T_i)(t_2 - t_1) \qquad (t_1 < t_2)$$

Theory tells that if each task is executed with an individual processing speed according to the fluid schedule, all task deadlines are held. This theorem is valid as long as system utilization does not exceed m. Physically, fluid scheduling is not applicable with current processor architectures and approximations are needed.

[3]Optimality, defined by Buttazzo [7]: *[...] an algorithm is said to be optimal if[f] it always finds a feasible schedule whenever there exists one. [...] A schedule is said to be feasible, if[f] all tasks can be completed according to a set of specified constraints.*

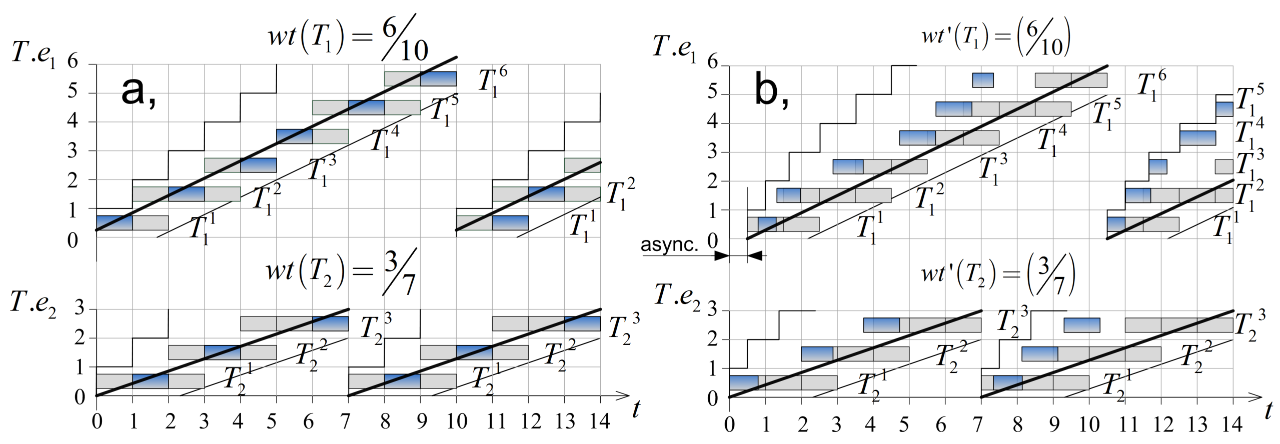

Figure 1: a, *ERfair* Scheduling with *ERfair-PD*^{2}; b, *P-ERfair* Scheduling with *P-ERfair-PD*^{2}: Windows $w(T_i^k)$ of a heavy task T_1 with weight wt(T_1) and quantized weight wt'(T_1) = wt(T_1) = $(6/10)$ and light task T_2 with weight wt(T_2) = wt'(T_2) = $(3/7)$. Subtasks $T_1^1, ..., T_1^6$ and $T_2^1, ..., T_2^3$ (black) have to be executed until end of their windows to guarantee *ERfair*ness. (Example shows subtask execution on a uniprocessor.)

The difference between the received processing time and the fluid schedule for a Task T_i at time t is defined as *Lag*:

$$\mathit{Lag}(T_i, t) = \text{fluid}(T_i, 0, t) - \text{received}(T_i, 0, t)$$

Pfair scheduling is an adjustment of fluid scheduling theory to physical processors and a discrete time model. The resolution of the discrete time is called *quantum*. Let Q denote the duration of a quantum, then *Pfair* implies that the minimal *Lag* is $-Q$ and the maximal *Lag* is $+Q$.

$$-Q \le \mathit{Lag}(T_i, t) \le +Q \quad \forall\, T_i \in \tau$$

The lower restriction results in non-work-conserving behavior, which causes the CPU to be idle even when there are tasks ready to execute [9]. *ERfair*, being a work-conserving variant of *Pfair*, necessarily disregards the lower bound.

$$\mathit{Lag}(T_i, t) \le +Q \quad \forall\, T_i \in \tau \tag{3}$$

This is the only modification of *ERfair* according to *Pfair*.

To adapt periodic task systems to *ERfair* scheduling, a task T_i of a periodic task set τ is split into a number of subtasks[4] T_i^k $(k = 1, ..., q;\ q \in \mathbb{N})$. Each T_i^k has the execution time Q. Therefore the number of subtasks q can be calculated from the task execution time $T_i.e$ by

$$q = \frac{T_i.e}{Q} \quad . \tag{4}$$

To fulfill (3), a subtask T_i^k has to be scheduled until the end of a time window $w(T_i^k)$. This starts with the pseudo[5]-release $r(T_i^k)$ and ends with the pseudo-deadline $d(T_i^k)$ ($\lfloor \chi \rfloor$ is the

[4]We denote T_i^k the k^{th} subtask of task T_i.

[5]The appendix *pseudo* is used to differ between task and subtask properties.

highest integer, smaller or equal to χ; $\lceil\chi\rceil$ is the smallest integer, higher or equal to χ).

$$r(T_i^k) = \left\lfloor \frac{k-1}{\mathrm{wt}(T_i)} \right\rfloor \tag{5}$$

$$d(T_i^k) = \left\lceil \frac{k}{\mathrm{wt}(T_i)} \right\rceil \tag{6}$$

As with *ERfair*, only the pseudo-deadline is used for subtask execution limitations. A subtask can be executed before the pseudo-activation, in contrast to *Pfair*. Due to the sequential dependency between subtasks of the same task, a subtask can not be executed before its antecessor has finished. In figure 1-a, the thin lines above the fluid schedule represent the earliest possible execution of a subtask.

The subtask window $w(T_i^k)$ is required for the calculation of further scheduling policies.We call the smallest time division where a complete subtask can be executed slot S. Depending on the weight of a task T_i a window $w(T_i^k) = \{S_1, S_2, ..., S_{\text{end}}\}$ has a number of slots. (We denote $|w(T^k)| = |\{S_1, S_2, ..., S_{\text{end}}\}|$ as the quantity of slots of a window.)

$$\left|w(T^k)\right| = \left\lceil \frac{k}{\mathrm{wt}(T)} \right\rceil - \left\lfloor \frac{k-1}{\mathrm{wt}(T)} \right\rfloor \tag{7}$$

Based on this concept, algorithms with different scheduling policies, having been proposed for *Pfair*, could also be applied to *ERfair*. For systems with more than two processors scheduling only based on Earliest-Pseudo-Deadline-First is not sufficient. Therefore tie-breaking rules have to be applied for optimality [6]. In this paper we focus on the algorithm *PD*^{2} [6] which is known to be the most efficient beside *PF* [5] and *PD* [10]. It uses only two additional tie-breaking rules and thus can be calculated efficiently during runtime.

Algorithm *ERfair-PD*^{2}

In the following part we provide a short introduction to the algorithm *ERfair-PD*^{2}. We denote *policy* as a criterion of prioritization. The calculation of the *PD*^{2} policies can be found in the attachment.

PD^2 schedules with *Earliest-Pseudo-Deadline-First* (pseudo-deadline $d(T_i^k)$ of the k^{th} subtask of the i^{th} task) and two additional tie-breaking rules. A tie-breaking rule is used, whenever a policy is not sufficient[6] for prioritization.

The first tie-breaking rule is called *overlapping-bit*. The overlapping-bit is calculated by $b(T_i^k)$ (formula 23). Informally, when the current subtask window overlaps with the window of the sequent subtasks, the overlapping bit is 1. *PD*^{2} prefers subtasks with overlapping bit equal to 1. For example, subtask T_1^2 in Figure 1-a has the overlapping bit set at $t = 4$, but not T_1^3 at $t = 5$.

The other tie-breaking rule is called *group-deadline*. The group-deadline is calculated by $D(T_i^k)$ (formula 24 and 25). The calculation of $D(T_i^k)$ is more complicated than the calculation of $b(T_i^k)$. Informally the group-deadline concerns the following scenario: A subtask of a task is not executed in the current slot, but will be executed in the next slot. Then the group-deadline is the time, at which one of the following subtasks has more than

[6]Not sufficient means that choosing the wrong task for execution, whenever both tasks have the same policy value, can produce *ERfair* non-conforming behavior.

one slot in its window left for scheduling, for the first time. *PD*^{2} prefers subtasks with a higher group-deadline.

ERfair-PD^2 is a global algorithm and performs each discrete time tick a schedule decision. Running tasks can be preempted by higher priority tasks.

As an example we examine three scheduling decisions. The example in figure 1-a illustrates the execution on a uniprocessor[7]. (The $>$ and $<$ operators are used to describe the scheduling prioritization: subtask T_X^a has to be preferred before subtask T_Y^b when $T_X^a > T_Y^b$; T_Y^b has to be preferred before subtask T_X^a when $T_X^a < T_Y^b$; otherwise the next policy has to be evaluated. If there is no further policy the selection is arbitrary.)
At timestamp 0 *ERfair-PD*^{2} prefers the subtask of task T_1 before the subtask of T_2, because $d(T_1^1) = 2 \;>\; d(T_2^1) = 3$.
At timestamp 3 *ERfair-PD*^{2} prefers the subtask of task T_2 before the subtask of T_1, because $d(T_1^3) = d(T_2^2) = 5$ and $b(T_1^3) = 0 < b(T_2^2) = 1$.
At timestamp 10, it is arbitrary if the subtask of task T_2 is preferred before the subtask of T_1 or otherwise, because $d(T_1^1) = d(T_2^2) = 12$, $b(T_1^1) = b(T_2^2) = 1$, and $D(T_1^1) = D(T_2^2) = 0$ (for details, see section A).

3. Multiple Time Base Task System

The Multiple Time Base (MTB) task system originates from the field of automotive powertrain applications, but concerns the general problem of many embedded systems, to have different tasks activation sources, e.g. Flexray or CAN. In a typical automotive powertrain system, two main sources of task activation exist. The first source is a periodic trigger, which activates tasks with different constant recurrences. The other source is the crank shaft of the engine, which activates tasks depending on the engine position. Analyzing such systems with a periodic task system model [1] is not possible, because the drifting behavior of the crank shaft activated tasks is not represented. Analyzing such systems with a sporadic task system model [11] will produce too pessimistic results, because the sporadic theorem assumes all tasks to be activated at the same time, which is not the case.

In MTB task systems, tasks refer to a time base of the system. All tasks concerning the same time base have a defined phasing in their activation compared to all other tasks referring to this time base.

A task set $\tau = \{T_i\}$ of tasks T_i, belonging to the MTB task system is defined in the following way:

Definition 3.1 *A task set τ consists of a number of tasks T_i.*

$$\tau = \{T_i\} \quad i = 1, ..., n; \; n \in \mathbb{N}$$

Definition 3.2 *A task T_i is defined by a tuple*

$$T_i = (p, o, e, d, b^v).$$

The elements of the tuple are the task properties: minimal task recurrence p, first task instance offset o, worst-case execution time e, deadline d, and a reference to a time base b^v.

[7] *ERfair-PD*^{2} scheduling on multiprocessors is analogous, with the difference that instead of one task, m tasks are selected for execution.

Definition 3.3 *A time base b^v is defined by the tuple*

$$b^v = (f, \varphi) \quad (v = 1, ..., w; \; w \in \mathbb{N}).$$

The time base properties are the frequency multiplier f and an angular phase shift φ. The variation of both properties defines the relation between the time base b^v and a unique global time.

Definition 3.4 *For the time base properties following restrictions exist.*

$$f \in \mathbb{R}^{\geq 1}$$

$$\varphi \in \mathbb{R}^{\geq 0}$$

As task recurrence p and task offset o of task T_i are related to the time base b^v, the task recurrence p' and task offset o' transformed to unique global time can be calculated by:

$$p'_i = p_i \cdot b^v.f + b^v.\varphi \tag{8}$$

$$o'_i = o_i \cdot b^v.f + b^v.\varphi \tag{9}$$

By definition, the frequency multiplier f cannot be smaller then 1, therefore p_i is the minimal recurrence and can be used for analysis purposes. Section 6 shows how this transformation is used to the detect worst-case response time.

A further extension of MTB task systems is a separation of the task T_i into task sections T_i^k.

Definition 3.5 *A task T_i is split into a number of task sections T_i^k $(k = 1, ..., q; \; q \in \mathbb{N})$. According to the task execution time $T_i.e$ and the task section execution time $T_i^k.e$ following relation exists.*

$$T_i.e = \sum_{k=1}^{q} T_i^k.e$$

Definition 3.6 *All task sections are sequentially dependent. Therefore, task section T_i^b can not be executed before task section T_i^a has finished its execution, if $a < b$ and $a, b \in \{1, ..., q\}$.*

4. Restrictions on MTB Task Systems for *P-ERfair*ness

In this part we examine the required restrictions on MTB task systems, for the application of *P-ERfair* scheduling, which will be introduced in the next section.

- **Sporadic task activation is allowed, but the minimal recurrence has to be a multiple of the time quantum**
  Tasks are allowed to be activated in continuous time, as long as the minimal distance between two subsequent activations is at least the minimal recurrence. The minimal recurrence has to be given in discrete time resolution Q.

  $$T_i.p = zQ \quad \text{, with } z \in \mathbb{N} \tag{10}$$

  As task activation can be sporadic, asynchronous task activation in continuous time is allowed.

- **Maximum task section execution time is restricted to discrete time resolution**
  The maximal task section execution time is limited to the discrete time resolution Q. (T_i represents the i^{th} task of a task set τ and T_i^k represents the k^{th} task section of task T_i.) This results in
$$\left\lceil \frac{T_i^k.e}{Q} \right\rceil = 1 \quad \forall\ T_i^k \in T_i \tag{11}$$

- **Explicit deadlines are allowed, but have to be a multiple of the time quantum**
  The deadline can be given explicitly, but has to be given in discrete time.
$$T_i.d = zQ \quad \text{, with } z \in \mathbb{N} \tag{12}$$

- **Restriction of task section quantity**
  The number of task sections multiplied with the discrete time resolution Q is not allowed to be higher than the minimum of recurrence and deadline. Therefore, the task section quantity is restricted in the following way.
$$\left|\left\{{T_i}^1, ..., {T_i}^q\right\}\right| Q \leq \min\{T_i.p, T_i.d\} \tag{13}$$

- **Restriction of maximal system utilization**
  In a system with m processing resources, where each resource has a capacity of 1, the maximal quantized system utilization U'_{sys}, calculated with the quantized task weight wt'(T_i), is restricted by
$$\text{wt'}(T_i) = \frac{\left|\left\{T_i^1, ..., T_i^k\right\}\right| Q}{\min\{T_i.p, T_i.d\}} \tag{14}$$
  and
$$U'_{sys} = \sum_{i=1}^{n} \text{wt'}(T_i) \leq m \tag{15}$$
  using the minimal recurrence $T_i.p$ and the deadline $T_i.d$ from all tasks T_i.

5. Partly Early Release Fair Scheduling

In the following section we introduce the concept of Partly Early Release Fair (*P-ERfair*) scheduling.

P-ERfair uses a similar task architecture as *ERfair*. A task is split in task sections (at *ERfair* called subtasks) and a task section has to be scheduled until the end of a window with a pseudo-deadline[8]. *P-ERfair* is based on MTB task systems with restrictions expressed in formulas 10 - 13, 15.

P-ERfair calculates the task weight wt'(T_i) by 14, the pseudo-release time by
$$r'(T_i^k) = \left\lfloor \frac{k-1}{\text{wt'}(T_i)} \right\rfloor \tag{16}$$
and the pseudo-deadline by
$$d'(T_i^k) = \left\lceil \frac{k}{\text{wt'}(T_i)} \right\rceil - 1 \quad . \tag{17}$$

[8]Therefore all algorithms implement *ERfair*, e.g. *PD*^{2}, theoretically could implement *P-ERfair*.

For the calculation of additional tie-breaking rules (e.g. at *PD*^{2}: overlapping-bit and group-deadline), the following replacement rules have to be applied ($\chi \to \chi'$ denotes symbol χ is replaced by χ'):

$$\text{wt}(T_i) \to \text{wt'}(T_i) \tag{18}$$

$$r(T_i^k) \to r'(T_i^k) \tag{19}$$

$$d(T_i^k) \to d'(T_i^k) \tag{20}$$

Algorithm *P-ERfair-PD*^{2}

In this section we present the *P-ERfair* scheduling algorithm *P-ERfair-PD*^{2}. *P-ERfair-PD*^{2} is based on PD^2 policies.

P-ERfair-PD^2 schedules task sets, like *ERfair-PD*^{2}, with Earliest-Pseudo-Deadline-First and with two additional tie-breaking rules: overlapping-bit and group-deadline. The calculation of these policies is analogous to *Pfair-PD*^{2}, with the difference in modifications through formula 18, 19, and 20.

P-ERfair-PD^2 is a cooperative algorithm, which means a task section cannot preempt another running task section. Another task section can just be allocated to the processing resource at executing task sections boundaries. The *P-ERfair-PD*^{2} scheduling routine is called, whenever a task is activated or a task section has finished.

As an example, we discuss a *P-ERfair-PD*^{2} schedule, which has slightly modified task properties compared to the *ERfair-PD*^{2} example in section 2. In Figure 1-b, both tasks T_1 and T_2 have task sections with an execution time $\leq Q$. Task T_1 is activated asynchronous (with a offset from zero). The execution again is on an uniprocessor[9]. At time stamp 0 only task T_2 is activated and therefore task section T_2^1 is executed. Between time stamp 10 and 11, after task section T_2^3 has finished, T^2 has finished execution and task T_1 is not activated. As there is no task ready for execution, the processor goes idle and the scheduler waits until next call occurs, which happens when task T_1 is activated.

6. Simulation-based Schedulability Estimation

This section explains why dynamic multiprocessor schedulability analysis extremely differs from uniprocessor schedulability analysis and provides an overview of the simulation based schedulability estimation methodology of MTB task sets, scheduled by *P-ERfair-PD*^{2}.

Classical theory of uniprocessors schedulability analysis, formed by Liu and Layland [1], defines following theorem according schedulability test.

Theorem 6.1 (see Liu and Layland [1]) *In a uniprocessor system, if a task set is schedulable under the assumption of the minimal recurrence, a simultaneous activation of all tasks[10], and a worst-case execution time for all tasks, then the task set is also schedulable at all other circumstances (namely higher recurrence and lower execution time).*

For multiprocessors, it is common knowledge that theorem 7.1 fails, when a dynamic scheduling approach is used [12]. Reasons for that are Richard's anomalies like the Graham theorem.

[9]Equal to *ERfair-PD*^{2}, *P-ERfair-PD*^{2} scheduling on multiprocessors is analogous to scheduling uniprocessors, with the difference that instead of one task, m tasks are selected for execution.

[10]For sporadic task systems only.

Theorem 6.2 (Graham Theorem, see [13]) *For the stated problem, changing priority list, increasing the number of processors, reducing execution times, or weakening the precedence constraints can increase the schedule length.*

Richard's anomaly [12] causes a breakdown of formal analysis (as stated in theorem 6.1). Consequently, new methods for schedulability estimation are required. Exact methods are extremely effort intensive e.g. the state enumeration with a brute-force approach [14]. Less work intensive approaches e.g methods using discrete event-based simulation for schedulability examination [15, 16] are only worst-case convergence mechanisms. Certainly, at the moment these methods are the only available techniques for schedulability calculation of continuous time models.

Simulation Approach

The discrete event-based simulation is based on a model of the hardware and software characteristics.
Comparable to the System-C based approach of Samii et al. [16], we vary simulation parameters to estimate the worst case response time. Because the execution time is constant at MTB task systems, only the activation relation of the tasks differs. At MTB task systems the task activation frequency differs because of the time base properties, described in section 3. We vary them within their valid range so that the simulation covers cases approximated to worst case scenarios and thus yields the largest relative response times for all tasks.

During the simulated time, a task T_i generates a number of jobs (i.e. instances) $T_{i,j}$ and its state changes occur through task activation, suspension, resumption, and termination. As a result of the simulation, a trace is generated containing a number of these state transitions of the system. König et al. [17] presented, how the real-time metrics like (worst-case) lateness $L(T_i)$ as introduced by Buttazzo in [7] can be used to analyze the trace of a simulation.

The lateness $l(T_{i,j})$ of the j^{th} job of task T_i is calculated by

$$l(T_{i,j}) = d_{i,j} - f_{i,j}$$

$l(T_{i,j})$ is equivalent to the time left until reaching the deadline. The lateness is negative when the finishing time $f_{i,j}$ is smaller than the absolute deadline $d_{i,j}$, i.e. if the calculation is finished in time. To determine the task-deadline compliance for a complete task set we identify the job which yields the largest lateness for each task. Additionally we normalize that lateness $l(T_{i,j})$ with the relative deadline D_i of the task T_i. We denote the maximum of that value of all tasks in a task set τ as maximal normed lateness *mNL* (τ).

$$mNL(\tau) = \max_{T_i \in \tau} \left(\frac{\max_{T_{i,j} \in T_i} \left(l(T_{i,j}) \right)}{D_i} \right) \tag{21}$$

To analyze *P-ERfair-PD*^{2}, we pseudo-randomly generate task sets τ_z and examine their maximal normed lateness *mNL*(τ_z). As the stochastic parameters for the pseudo-random generation are based on a stochastic task set description, we call this Monte-Carlo approach [15, 18].

Technical Experiment Setup

The Monte-Carlo approach combined with the simulative study of large numbers of task sets is quite calculation effort intensive. Therefore, we developed a *C++* based simulation environment using a cluster computing approach. The *Condor Cluster Computing* network [19] is used to execute multiple instances of our simulation. Since each task set can be simulated independently, huge speedups around a factor of 100≈150 are possible when using the PC pools of the Regensburg University of Applied Sciences.

7. Performance Examination of *P-ERfair-PD*^{2}

In order to examine the performance of *P-ERfair-PD*^{2} we pseudo-randomly generated 500000 task sets τ_z ($z = 1, ..., 500000$). All τ_z were simulated using the approach described in the previous section. Each task set was analysed on a system simulating a quad-core multiprocessor ($m = 4$) with a *P-ERfair-PD*^{2} scheduling algorithm.

The pseudo-random task set generation process is based on a stochastic task set description, similar to the manner it was introduced in [15].
The stochastic task set is deduced from automotive powertrain systems. We create the task sets $\{\tau_z\}$ in the following way.

First, the quantity of tasks n_z of task set τ_z is drawn from a discrete uniform distribution ($n_{\min} = 20, n_{\max} = 30$). Afterwards, for each task $T_i \in \tau$ the recurrence[11] is drawn from an equally distributed list of recurrences $\{2.5, 5.0, 7.5, 10.0, 20.0, 50.0\}$ and the utilization wt(T_i) is drawn from a Weibull distribution ($w_{\min} = 0.05, \overline{w} = 0.15, w_{\max} = 0.51, p_{\text{rest}@w_{\min}} = 10\%$). For all tasks, the deadline is equal to the recurrence. The quantum Q is set to 0.25. For the task section execution time, we randomly generate execution times (also from a Weibull distribution ($w_{\min} = 0.125, \overline{w} = 0.24, w_{\max} = 0.25, p_{\text{rest}@w_{\min}} = 10\%$) and assign the generated task sections T^k (with task section execution time $T^k.e$) to a task T_i as long as the condition

$$\sum_{k=1}^{q} T_i^k.e \le T_i.p \cdot \text{wt}(T_i)$$

is fulfilled.
As $T_i^k.e$ is drawn from a distribution with maximum equal to Q and MTB task set restriction 15 requires that quantized system utilization $U'_{sys} \le m$, the possibility to achieve $U'_{sys} = U_{sys}$ is very low. Therefore we generated 200000 task sets with $T_i^k.e = Q \forall i, k$ to achieve more task sets with $U'_{sys} = U_{sys}$.
The task offset o is drawn from a uniform distribution ($u_{\min} = 0.0; u_{\max} = 0.05$). Finally, each task is assigned to a time base $b.v$ according to a uniform distribution $\{1, 2, 3, 4\}$.

Figure 2 shows the result of all randomly generated and simulated task sets.The left experiment represents *Partly-Pfair-PD*^{2} and the right experiment represents *P-ERfair-PD*^{2}. [12]
One point represents the maximum normed lateness *mNL*(τ_z) (formula 21) of a task set τ_z. The line shows the quantity of the generated task sets as a function of system utilization $U_{sys}(\tau_z)$ calculated by formula 22.

$$U_{sys}(\tau_z) = \sum_{i=1}^{n_z} \frac{T_i.e}{T_i.p}. \tag{22}$$

[11] All further times are in unit *ms*.

[12] Total single CPU time for generation ~one year.

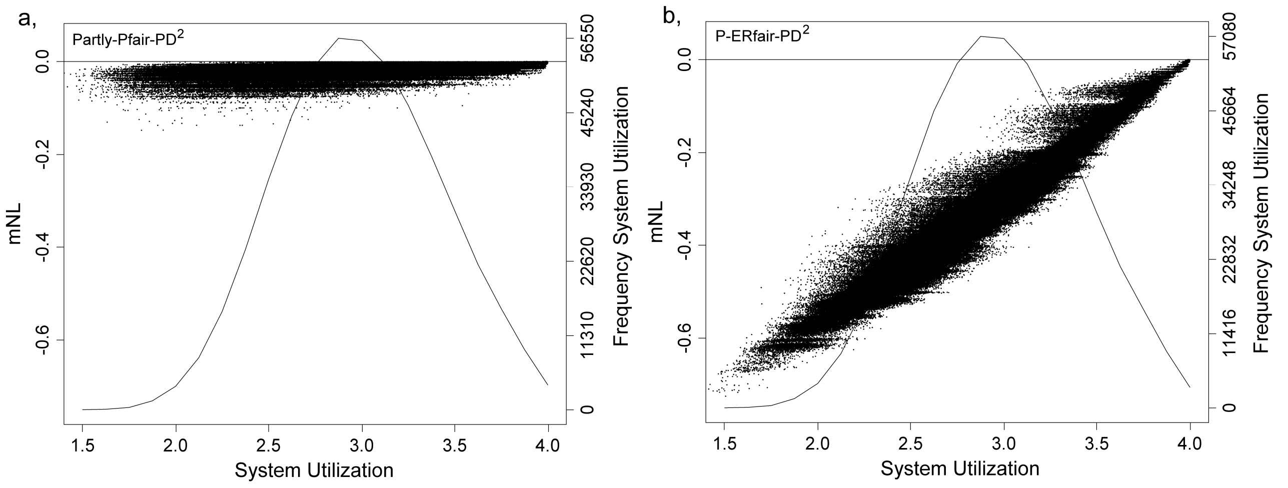

Figure 2: a, algorithm *Partly-Pfair-PD*^{2}; b, algorithm *P-ERfair-PD*^{2}: 500000 randomly generated and simulated task sets. The simulated hardware is a quadcore-processor ($m = 4$). Each point of *mNL* equates the worst relative task lateness of one task set. A negative *mNL* implies all task-deadlines of a task set are meet. The line represents the quantity of generated task sets as a function of system utilization.

At *Partly-Pfair-PD*^{2} one deadline violation of 0.005% occurs [4]. At *P-ERfair-PD*^{2} no deadline violation occurs for all generated and simulated task sets.

8. Reliability Considerations

In this section we analyze the reliability of *Partly-Pfair-PD*^{2} (*non-work-conserving behavior*) and *P-ERfair-PD*^{2} (*work-conserving behavior*) by violating MTB task system properties. Each task section is challenged with a random variation of the task section execution time via a Weibull distribution in the range of $T^k_{i,j}.e = [0.9 \cdot Q, ..., 2 \cdot Q] \ \ \forall i, j, k$, with $T^k_{i,j}.e \in \mathbb{R}$, and $T^k_{i,j}.\overline{e} = Q$.[13] Again, Q is set to 250 μs. In real systems, such task execution time variation could occur due to varying runtime of algorithms, preemptions caused by interrupt service routines, different control flows depending on the input, etc. In any case, such variations lead to a violation of the MTB task system restriction 11.

The simulation consists of 100000 generated task sets according to the definition in section 7, with adaptations to the task section execution time as described above. Figure 3 shows the schedulability examination results for both algorithms with a quad-core ($m = 4$) processor. Each box plot represents statistical estimators of *mNL* for a group of task sets, clustered by system utilization in 25 equal sized groups. At the box plot, the symbol represents the median, the box around the symbol the bootstrapped 99% confidence intervall of the median, and the whispers represent the upper and respectively the lower bootstrapped 99% confidence intervall of the inter-quantile distance (99%) boundaries.

Due to it its non-work-conserving behavior, the *Partly-Pfair-PD*^{2} algorithm leads to a *mNL* typically close to zero. While an unperturbed *Partly-Pfair-PD*^{2} system would hold the deadline up to high utilizations, the perturbed system shows the first deadline violation at a quite

[13]We denote $T^k_{i,j}$ the task section execution time of the k^{th} task section of the j^{th} job (i.e. instance) of task T_i.

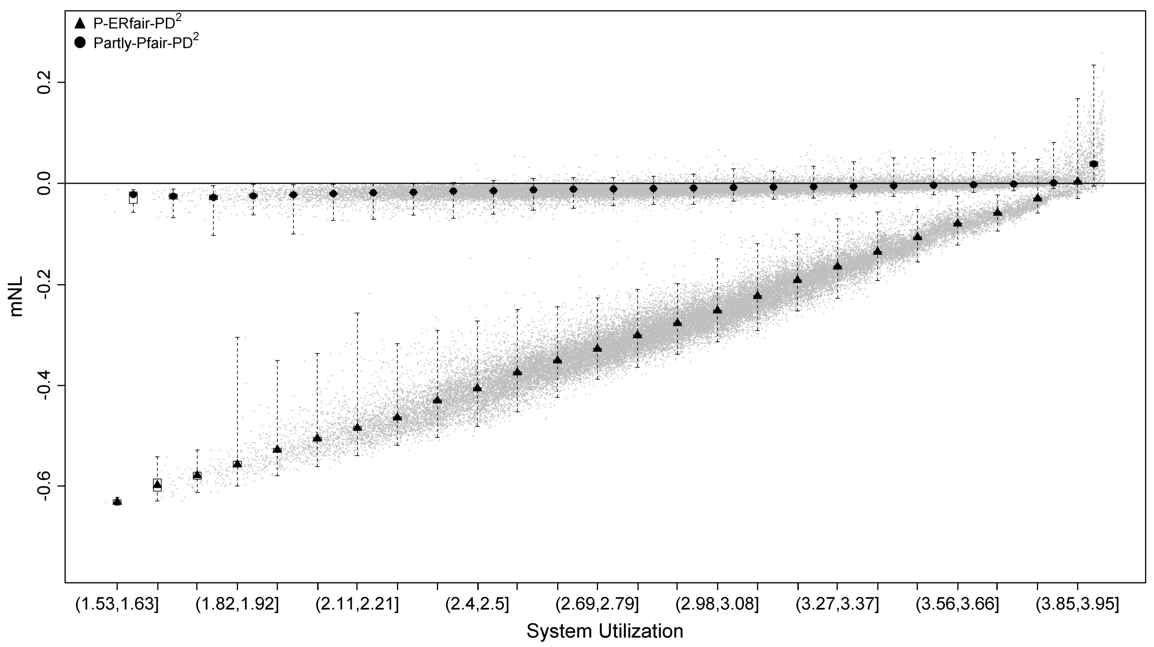

Figure 3: Stress test of *Partly-Pfair-PD*^{2} and *P-ERfair-PD*^{2} through random variation of the task section execution time via a Weibull distribution. The task section execution time can be twice as long as allowed.

low system utilization of 2.2 (which corresponds to 55%).
On the other hand, *P-ERfair-PD*^{2} is work-conserving, which means its tasks typically finish much earlier than their deadline, leading to a smaller *mNL* way below the *mNL*=0 limit (at least for smaller utilizations). Consequently, the perturbed system is able to keep all deadlines up to a rather high system utilization of 3.7 (which corresponds to 92.5%).
This shows that the *P-ERfair-PD*^{2} algorithm is able to handle variations of the task section execution time quite well, while *Partly-Pfair-PD*^{2} fails completely above 57.5% (i.e. upper box plot wisper is above *mNL* =0 limit).

9. Conclusion

Embedded systems, namely in the Automotive domain, demand for highly efficient hardware usage, also when the system uses multi-core architectures. Additionally, since tasks typically execute on external triggers and have variable task execution time, most algorithms that have been proven optimal under simplified conditions are not applicable in productive systems.

Driven by these circumstances, we introduced in this work the algorithm *P-ERfair-PD*^{2}, an work-conserving modification of the *Partly-Pfair-PD*^{2} [4] algorithm.

In a simulation based Monte Carlo analysis we showed that for the task sets, aligned to existing Automotive powertrain applications, the performance of this new *P-ERfair*ness based scheduling algorithm is very high (in examinations all deadlines are kept). By introducing perturbations into the system, i.e. adding violations to the defined rules for MTB task system, we showed that the algorithm is able to schedule the systems without deadline misses up to a very high utilization of 92.5%, whereas *Partly-Pfair-PD*^{2} completely fails above 57.5%. However, a quantitative analysis of the robustness of *P-ERfair* remains the subject of future work.

These results show that *P-ERfair* is an highly interesting scheduling algorithm for many embedded domains changing to multi-core architectures, especially for Automotive power-train applications.

Acknowledgment

This work is supported by BMBF grant DynaS^3 FKZ1752X07. More information at www.LaS3.de.

References

[1] C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” *Journal of the Association of Computing Machinery*, vol. 20, no. 1, January 1973.

[2] M. Deubzer, U. Margull, J. Mottok, M. Niemetz, and G. Wirrer, “Partitionierungs-Scheduling von Automotive Restricted Tasksystemen auf Multiprozessorplattformen,” *Proceedings of the Second Embedded Software Engineering Congress*, December 2009.

[3] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, “A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms,” *Handbook on Scheduling Algorithms, Methods, and Models*, 2004.

[4] M. Deubzer, U. Margull, J. Mottok, M. Niemetz, and G. Wirrer, “Partly Proportionate Fair Multiprocessor Scheduling of Heterogeneous Task Systems,” in *Proceedings of the 5th Embedded Real Time Software and Systems Conference*, May 2010.

[5] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, “Proportionate progress: a notion of fairness in resource allocation,” *Algorithmica*, vol. 15, pp. 600-625, 1996.

[6] J. Anderson and A. Srinivasan, “Mixed Pfair/ERfair Scheduling of Asynchronous Periodic Tasks,” *Proceedings of the 13th Euromicro Conference on Realtime Systems*, pp. 76-85, June 2001.

[7] G. C. Buttazzo, *Hard Real-Time Computing Systems.* Boston Dordrecht London: Kluwer Academic Publishers, 2002.

[8] J. Anderson and A. Srinivasan, “Early-Release Fair Scheduling,” in *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, 2000, pp. 35–43.

[9] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft Real-Time Systems: Predictability vs. Efficiency.* Springer Verlag, 2005.

[10] S. Baruah, J. Gehrke, and C. G. Plaxton, “Fast Scheduling of Periodic Tasks on Multiple Resources,” *Proceedings of the 9th International Parallel Processing Symposium*, pp. 280-288, April 1995.

[11] S. Ward and A. Mok, “Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment,” Ph.D. dissertation, Massachusetts Institute of Technology, 1983.

[12] C. Scheduling, “Implications of Classical Scheduling Results for Real-Time Systems,” *A practical approach to real-time systems: selected readings*, p. 301, 2000.

[13] R. Graham, “Bounds on the Performance of Scheduling Algorithms,” *Computer and Job-Shop Scheduling Theory*, 1976.

[14] T. Baker and M. Cirinei, “Brute-Force Determination of Multiprocessor Schedulability for Sets of Sporadic Hard-Deadline Tasks,” *10th International Conference on Principles of Distributed Systems*, pp. 62–75, 2007.

[15] M. Deubzer, F. Schiller, J. Mottok, M. Niemetz, and U. Margull, “Effizientes Multicore-Scheduling in Eingebetteten Systemen: Teil 2 - Ein simulationsbasierter Ansatz zum Vergleich von Scheduling-Algorithmen,” June 2010.

[16] S. Samii, S. Rafiliu, P. Eles, and Z. Peng, “A Simulation Methodology for Worst-Case Response Time Estimation of Distributed Real-Time Systems,” in *Proceedings of the 11th conference on Design, automation and test in Europe*. ACM New York, NY, USA, 2008, pp. 556-561.

[17] F. König, D. Boers, F. Slomka, U. Margull, M. Niemetz, and G. Wirrer, “Application Specific Performance Indicators for Quantitative Evaluation of the Timing Behavior for Embedded Real-Time Systems,” in *Proceedings of the 12th conference on Design, Automation and Test in Europe*, 2009.

[18] J. Axelsson, “A Method for Evaluating Uncertainties in the Early Development Phases of Embedded Real-Time Systems,” in *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings*, 2005, pp. 72–75.

[19] Ann Chervenak and Ewa Deelman and Miron Livny and Mei-Hui Su and Rob Schuler and Shishir Bharathi and Gaurang Mehta and Karan Vahi, “Data Placement for Scientific Applications in Distributed Environments,” in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, Austin, TX, September 2007.

[20] A. Srinivasan, “Efficient and Flexible Fair Scheduling of Real-time Tasks on Multiprocessors,” Ph.D. dissertation, University of North Carolina, 2003.

A. Appendix

A.1. Description of *PD*^{2} Policies

In the following part we describe the calculation of *PD*^{2} policies.
The $\succ$ and $\prec$ operators are used to describe the scheduling prioritization: subtask T_X^a has to be preferred before subtask T_Y^b when $T_X^a \succ T_Y^b$; T_Y^b has to be preferred before subtask T_X^a when $T_X^a \prec T_Y^b$; otherwise the next policy has to be evaluated. If there is no further policy the selection is arbitrary.

Policy 1 (Pseudo-Deadline):
$T_X^a \succ T_Y^b$ if $d(T_X^a) < d(T_Y^b)$; $T_X^a \prec T_Y^b$ if $d(T_X^a) > d(T_Y^b)$.
$d(T_i^k)$ equates the pseudo-deadline, calculated by formula 6.
Figure 1-a,b shows the windows for the subtasks of two task T_1 and T_2 with a weight of $\mathrm{wt}(T_1) = 6/10$ and $\mathrm{wt}(T_2) = 3/7$. The pseudo-deadlines of the subtasks $T_1^1, ..., T_1^6$ are $d(T_1^1) = 2$, $d(T_1^2) = 4$, $d(T_1^3) = 5$, $d(T_1^4) = 7$, $d(T_1^5) = 9$, and $d(T_1^6) = 10$.

Policy 2 (Overlapping-Bit):

$T_X^a \succ T_Y^b$ if $b(T_X^a) > b(T_Y^b)$; $T_X^a \prec T_Y^b$ if $b(T_X^a) < b(T_Y^b)$.
The *overlapping bit* function $b(T_i^k)$ denotes the overlapping of two successive subtask windows T_i^k and T_i^{k+1} of a Task T_i.

$$b(T_i^k) = \begin{cases} 1 & , if\ d(T_i^k) > r(T_i^{k+1}) \\ 0 & , if\ d(T_i^k) \leq r(T_i^{k+1}) \end{cases} \tag{23}$$

PD^2 prefers subtasks with overlapping windows, because delaying such a subtask means that the successive subtask has a smaller window to be scheduled. In figure 1-a,b the overlapping-bits of the subtasks $\{T_1^1, ..., T_1^6\}$ are $b(T_1^1) = 1,\ b(T_1^2) = 1,\ b(T_1^3) = 0,\ b(T_1^4) = 1,\ b(T_1^5) = 1,$ and $b(T_1^6) = 0$.

Policy 3 (Group-Deadline):
$T_X^a \succ T_Y^b$ if $D(T_X^a) > D(T_Y^b)$; $T_X^a \prec T_Y^b$ if $D(T_X^a) < D(T_Y^b)$.
The *group-deadline* function $D(T_i^k)$ concerns the effect of schedule decision of a subtask for its subsequent subtasks. Let $T_i^c, ..., T_i^d$ be a sequence of subtasks of a heavy task T_i (heavy means $wt(T_i) \geq 0.5$) in the way that $c < k \leq d$ with a window of subtask T_i^k either of length $|w(T_i^{k+1})| = 3$ (e.g. figure 1-a,b, subtask T_1^k, $k = 1$) or $|w(T_i^{k+1})| = 2 \wedge b(T_i^k) = 0$ (e.g. figure 1, subtask T_1^k, $k = 3$). Then scheduling subtask T_i^l $c < l \leq d$ in the last slot of its window $w(T_i^l)$ results in scheduling all subsequent subtasks in there last slot, excepted subtask T_i^{k+1} because there are 2 slots left. Therefore the sequence $T_i^c, ..., T_i^d$ can be seen as one subtask-unit where scheduling one subtask in the last slot results in scheduling each subsequent subtask in its last slot. Otherwise pseudo-deadlines are missed. The group-deadline is the last time slot of this subtask-unit at time $d(T_i^k) + 1$. Formally, the group-deadline $D(T_i^k)$ can be calculated for heavy tasks by (24) [20].

$$D(T_i^k) = \left\lceil \frac{\left\lceil \left\lceil \frac{k}{wt(T_i)} \right\rceil \cdot (1 - wt(T_i)) \right\rceil}{1 - wt(T_i)} \right\rceil \text{ if } wt(T_i) \geq 0.5 \tag{24}$$

Furthermore Srinivasan [20] proved for light tasks $(wt(T_i) < 0.5)$ the group-deadline is 0 (formula 25) $\forall\ T_i^k \in T_i$.

$$D(T_i^k) = 0 \text{ if } wt(T_i) < 0.5 \tag{25}$$

In figure 1 the group-deadlines of $T_1's$ subtasks are $D(T_i^1) = 3,\ D(T_i^2) = 5,\ D(T_i^3) = 5,\ D(T_i^4) = 8,\ D(T_i^5) = 10$, and $D(T_i^6) = 10$. At task T_2 each window has at least 2 slots left, when the precedent subtask is scheduled in the last slot.