Partly Proportionate Fair Multiprocessor Scheduling of Heterogeneous Task Systems

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: Proportionate fair (*Pfair*) scheduling, which allows task migration at runtime and assigns each task processing time with regard to its weight, is one of the most efficient group of SMP multiprocessor scheduling algorithms known up to now. Drawbacks are tight requirements to the task system, namely the restriction to periodic task systems with synchronized task activation, quantized task execution time, and implicit task deadline. Most likely, a typical embedded real-time system does not fulfill these requirements.
In this paper we address violations of these requirements. For heterogeneous task systems, we define the multiple time base (MTB) task system, which is a less pessimistic model than sporadic task systems and is used for automotive systems. We apply the concept of *Pfair* scheduling to MTB task systems, called partly proportionate fair (*Partly-Pfair*) scheduling. The restrictions on MTB task systems required for *Partly-Pfair*ness are weaker than restrictions on periodic task systems required for *Pfair*ness.
In a simulation based study we examined the performance of *Partly-Pfair-PD*^{2} and found it capable to schedule feasible MTB task sets causing a load of up to 100% of the system capacity.

Keywords: Real-time systems, multiprocessors, dynamic scheduling, proportionate fairness, *Pfair*, *PD*^{2}

I. 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 $M = \{P_x\}$ $(x = 1, ..., m;\ m \in \mathbb{N})$ 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], [3], applied before runtime. Drawback of the partitioning approach is a low maximal system utilization [4].
The other group consists of algorithms using the dynamic scheduling approach, which allocates tasks during runtime. Proportionate fair (*Pfair*) Scheduling [5] is a mechanism of dynamic scheduling, known to be optimal[2] for algorithms like PF [5] or *PD*^{2} [7]. Using tight restrictions on task systems, it is theoretically possible to fulfill all deadlines up to 100% system utilization.
These restrictions are:

- a periodic task activation in discrete time[3]
- a synchronous task activation
- a constant task execution time in discrete time
- an implicit deadline in discrete time

Embedded Systems, particular automotive systems, typically violate these conditions.

Several extensions to *Pfair* scheduling have been proposed over the past few years. The problem of task activation in continuous time was discussed by [8], the problem of sporadic task activation was examined by [9], and [10] shows a solution for a continuous execution time model [11] and sporadic task activation. [12] considers practical properties and analyzes overhead through scheduling execution times and migration. Missing examinations are cooperative scheduling and variable task execution times.

In this paper we present a task system closely resembling embedded systems, like automotive systems, and apply proportionate fairness scheduling. The resulting model is called partly proportionate fairness (*Partly-Pfair*).

The remainder of this paper is organized as follows. In the second chapter we give a brief review to *Pfair* Scheduling and *PD*^{2} policies, as introduced by [5], [7].

[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]Optimality, defined by Buttazzo [6]: *[...] 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.*

[3]In contrast to Baruah et. al [5] we use *discrete time* in terms of quanted time.

Chapter 3 describes the multiple time base (MTB) task system and chapter 4 defines the required restrictions on MTB task systems to apply *Partly-Pfair*ness. In Chapter 5 we introduce the concept of partly proportionate fair scheduling (*Partly-Pfair*) and a new scheduling algorithm *Partly-Pfair-PD*^{2} which is based on *Partly-Pfair* and *PD*^{2}. In Chapter 6 we give a brief introduction to the simulation-based scheduling analysis. In Chapter 7 we show a performance examination of *Partly-Pfair-PD*^{2}. After a consideration of practical benefits of *Partly-Pfair-PD*^{2} in chapter 8, we give a conclusion of our contribution to dynamic multiprocessor scheduling.

II. Proportionate Fair Scheduling

Proportionate-fair (*Pfair*) scheduling 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 $\mathsf{wt}(T_i)$,

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

with the worst case execution time[4] e and the activation period p. Anderson et. al [7] 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.

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

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 $\mathsf{wt}(T_i)$. Figure 1 shows an example with two tasks T_1 and T_2. The fluid schedule graph of both tasks is shown as a thick line.

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

The 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 an approximation is necessary to apply it.

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

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

Pfair scheduling is an adjustment of fluid scheduling theory to physical processors and a discrete time

[4] $T_i.\chi$ denotes the task property χ of task T_i.

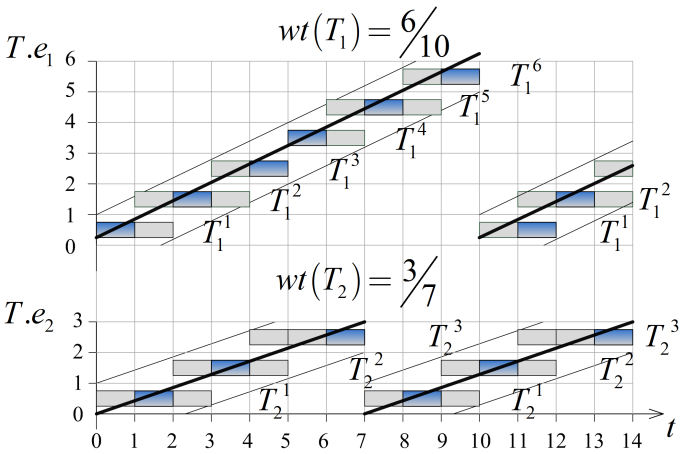

Fig. 1. *Pfair* Scheduling with *Pfair-PD*^{2}: Windows $w(T_i^k)$ of a heavy task T_1 with weight $\mathsf{wt}(T_1) = (6/10)$ and light task T_2 with weight $\mathsf{wt}(T_2) = (3/7)$. Subtasks $T_1^1, ..., T_1^6$ and $T_2^1, ..., T_2^3$ (black) have to be scheduled in their window to guarantee *Pfair*ness. (Example shows subtask execution on a uniprocessor.)

model. The resolution of the discrete time is called *quantum*. Let Q denote a quantum, then *Pfair* implies that the maximal *Lag* is $+Q$ and the minimal *Lag* is $-Q$.

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

In figure 1, the thin lines represent the minimal and maximal allowed *Lag* of one quantum and the fluid schedule.

To adapt periodic task systems to *Pfair* scheduling, a task T_i of a periodic task set τ is split in a number of subtasks[5] T_i^k $(k = 1, ..., q;\ q \in \mathbb{N})$. Each T_i^k has the execution time of one quantum. Therefore the number of subtasks q can be calculated from the task execution time $T_i.e$ by formula 4.

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

To fulfill (3), a subtask T_i^k has to be scheduled in a time window $w(T_i^k)$, starting with the pseudo[6]-release $r(T_i^k)$ and ending with the pseudo-deadline $d(T_i^k)$ ($\lfloor\chi\rfloor$ is the highest integer, smaller or equal to χ; $\lceil\chi\rceil$ is the smallest integer, higher or equal to χ).

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

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

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 $w(T_i^k) = \{S_1, S_2, ..., S_{\mathsf{end}}\}$ has a number of slots, available for scheduling subtask

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

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

T_i^k. (We denote $\left|w(T^k)\right| = |\{S_1, S_2, ..., S_{\text{end}}\}|$ as the quantity of slots of a window.)

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

Various *Pfair* algorithms with different scheduling policies have been published. Anderson and Srinivasan proved for a system with two processors that scheduling the subtasks according to an Earliest-Pseudo-Deadline-First scheme *EPDF* is optimal [13]. For more general systems with more than two processors, tie-breaking rules have to be applied for optimality. In this paper we focus on the algorithm *PD*^{2} [7] which is known to be the most efficient beside *PF* [5] and *PD* [14]. It uses only two additional tie-breaking rules and thus can be calculated efficiently during runtime.

Algorithm PD^2

In the following part we provide a short introduction to the algorithm *PD*^{2}. We denote *policy* as a criteria of prioritization. The calculation of the *PD*^{2} policies are attached to the appendix.

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[7] for prioritization.

The first tie-breaking rule is called *overlapping-bit*. The overlapping-bit is calculated by $b(T_i^k)$ with 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.

The other tie-breaking rule is called *group-deadline*. The group-deadline is calculated by $D(T_i^k)$ with 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 one slot in its window left for scheduling, for the first time. *PD*^{2} prefers subtasks with a higher group-deadline.

Pfair-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 example we examine three scheduling decisions. The example in figure 1 illustrates the execution on a uniprocessor[8]. (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.)
At timestamp 0 *Pfair-PD*^{2} prefers the subtask of task T_1 before the subtask of T_2, because $d(T_1^1) = 2 \succ d(T_2^1) = 3$.
At timestamp 3 *Pfair-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 $b(T_1^3) = 0 \prec b(T_2^2) = 1$.
At timestamp 10, it is arbitrary if the subtask of task T_2 is preferred before the subtask of T_1 or otherwise, because $d(T_1^1) = d(T_2^2) = 5$, $b(T_1^1) = b(T_2^2) = 1$, and $D(T_1^1) = D(T_2^2) = 0$.

III. 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 [15] 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 (possibly variable) 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 4.1: A task set τ consists of a number of tasks T_i.

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

Definition 4.2: 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

[7] Not sufficient means that choosing the wrong task for execution, whenever both tasks have an equal police value, can produce *Pfair* non-conform behavior.

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

a time base b^v.

Definition 4.3: 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 4.4: 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 VI shows how this transformation is used to the detect worst-case response time.

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

Definition 4.5: 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 $T_i^k.e$ the relation

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

exists.

Definition 4.6: 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 $a, b \in \{1, ..., q\}$.

IV. Restrictions on MTB Task Systems for *Partly-Pfair*ness

In this part we examine the required restrictions on MTB task systems, for the application of *Partly-Pfair* 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 deadline is allowed, but has 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

  $$\text{wt'}(T_i) = \frac{\left|\{T_i^1, ..., T_i^k\}\right| \ Q}{\min\{T_i.p, T_i.d\}} \tag{14}$$

  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.

V. Partly Proportionate Fair Scheduling

In the following section we introduce the concept of *Partly-Pfair* scheduling.

Partly-Pfair uses a similar task architecture as *Pfair*. A task is split in task sections (at *Pfair* called subtasks) and a task section has to be scheduled in a window with a pseudo-release time and a pseudo-deadline[9].

[9]Therefore all algorithms implement *Pfair*, e.g. *PD*^{2}, theoretically could implement *Partly-Pfair*.

Partly-Pfair is based on MTB task systems with restrictions of formula 10 - 13, 15.

Partly-Pfair calculates the task weight wt'(T_i) by formula 14,
the pseudo-release time by formula 16,

$$r'(T_i^k) = \left\lfloor \frac{k-1}{\text{wt'}(T_i)} \right\rfloor \tag{16}$$

and the pseudo-deadline by formula 17.

$$d'(T_i^k) = \left\lceil \frac{k}{\text{wt'}(T_i)} \right\rceil - 1 \tag{17}$$

For the calculation of additional tie-breaking rules (e.g. at *PD*^{2}: overlapping-bit (formula 23) and group-deadline (formula 24 & 25)), 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 Partly-Pfair-PD^2

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

Partly-Pfair-PD^2 schedules task sets, like *Pfair-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.

Partly-Pfair-PD^2 is a cooperative algorithm, which means that 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 *Partly-Pfair-PD*^{2} scheduling routine is called, whenever:

- a task is activated,
- a task section has finished, or
- there is a free processing resource and a task section is pseudo-activated. (This causes non-work-conserving behavior of *Partly-Pfair-PD*^{2}.)

As example, now we discuss a *Partly-Pfair-PD*^{2} schedule with slightly modified task properties, compared with the task properties of the *Pfair-PD*^{2} example. Shown in figure 2, both tasks T_1 and T_2 have task sections with an execution time $\leq Q$. Task T_1 is asynchronous

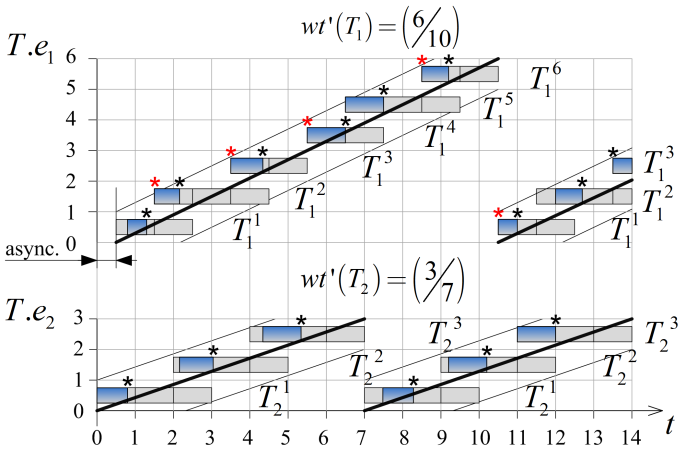

Fig. 2. *Partly-Pfair* Scheduling with *Partly-Pfair-PD*^{2}: Windows $w(T_i^k)$ of a heavy task wt'$(T_1) = \frac{6}{10}$ and light task with wt'$(T_2) = \frac{3}{7}$. Task sections $T_1^1, ..., T_1^6$ and $T_2^1, ..., T_2^3$. A star signals a *Partly-Pfair-PD*^{2} scheduler call. (Example shows subtask execution on a uniprocessor.)

activated. The execution is also on an uniprocessor[10].
At timestamp 0 only task section T_2^1 is pseudo-activated and therefore is executed.
Between timestamp 0 and 1, after task section T_2^1 has finished, T_1^1 is activated and task section T_2^2 hasn't reached its pseudo-activation. Therefore task section T_1^1 is executed.
A scheduler call occurs each time when a task section has finished. When there is no pseudo-activated task section available for execution (e.g. at time 3), an additional scheduler call is set at the next pseudo-activation time.

VI. Simulation-based Schedulability Estimation

This section provides overview of the schedulability estimation methodology of MTB task sets, scheduled by *Partly-Pfair-PD*^{2}.

Simulation Approach

It is common knowledge that schedulability analysis of dynamically allocated tasks with dynamic priorities in multiprocessor systems is infeasible with current general analysis techniques. Problems which occur only at multiprocessors e.g. Richard's anomaly [16], cause formal analysis to breakdown.
As the task system of *Partly-Pfair-PD*^{2} is much more complex than the task system of *Pfair-PD*^{2} (continuous time, sporadic activation, variable task (section) execution time), we use a discrete event-based simulation method for the performance examination [17].

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

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 [18], 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. 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 [19] presented, how the real-time metrics like e.g. lateness $l(T_i)$ as introduced by Buttazzo in [6] 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 $l_{i,j}$ 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).

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

To analysis *Partly-Pfair-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 [20], [21].

Technical Experiment Setup

As the Monte-Carlo approach combined with the simulative study of the task sets is quite calculation effort intensive, we developed a *C++* based simulation environment, which is distributed via a cluster computing approach. The *Condor Cluster Computing* network [22] allows us to execute multiple instances of our simulation and thus simulation of different task sets in parallel.

VII. Performance Examination of *Partly-Pfair-PD*^{2}

In order to examine the performance of *Partly-Pfair-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 analyzed on a quad-core SMP multiprocessor and *Partly-Pfair-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 [20].
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

$$\sum_{k=0}^{q} T_i^k.e \leq T_i.p \ \cdot \ \text{wt}(T_i)$$

is fulfilled. 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 3 shows the result of all randomly generated and simulated task sets.[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

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

For the generated and simulated task sets, the only deadline violation occurs at a system utilization of 3.995. The *mNL* value of this task set is +0.00005, which equates a deadline violation of 0.005 %. The *mNL* value results from a task with 50 ms recurrence.

[11] All further times are in ms.

[12] Total CPU time for generation ∼one year. (But, with the condor cluster network and the computer pools of our university we have a speedup of 150.)

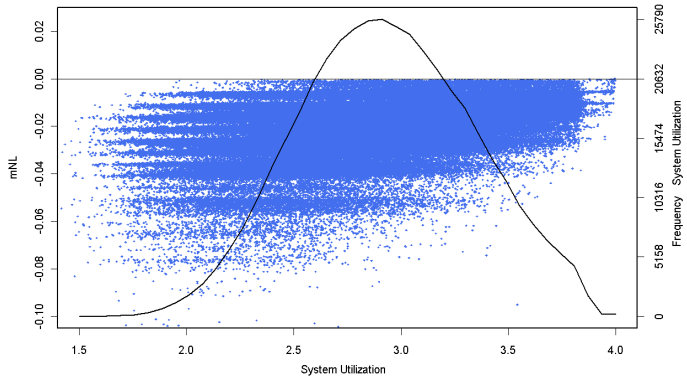

Fig. 3. 500000 randomly generated and simulated task sets. The simulated hardware is a quadcore-processor ($m = 4$) with the algorithm *Partly-Pfair-PD*^{2}. Each point of *mNL* equates the worst relative task lateness of one task set. A negative *mNL* implies all task-deadlines of a task set are meet. The line represents the quantity of generated task sets as a function of system utilization.

VIII. Application Benefits and Practical Consideration of *Partly-Pfair-PD*^{2} for Embedded (Automotive) Systems

As discussed, many dynamic multiprocessor scheduling algorithms that have been proven by formal methods to be optimal, can not be applied in embedded systems due to their restrictive properties. This section gives requirements of real systems, which are fulfilled by *Partly-Pfair-PD*^{2} but also indicates some areas where still additional extensions of the algorithm are needed.

Fulfilled Requirements

- As it was shown in Section V, *Partly-Pfair-PD*^{2} is a cooperative scheduling algorithm. This is not a hard requirement in automotive systems, but has some functional and efficiency benefits, e.g. a lower overhead for stack migration at a defined points of preemption.
- The restrictions of the minimal recurrence and deadline to discrete time is manageable. Otherwise both have to be rounded to the next lower multiple of the discrete time resolution, or the time resolution has to be increased by using a smaller quantum.
- The restriction to the maximum task section execution time is weak because similar requirements are used to manage cooperative behavior in existing systems. However, the maximal execution time of task sections has to be followed exactly.

Further Work

The algorithm discussed here does not consider scheduler nor migration overhead. Scheduler calls can occur quite often, so possible improvements should reduce the number of scheduler calls. Depending on the features of the underlying system hardware, migration is more or less costly. It is therefore desirable to minimize the number of migrations necessary for scheduling.

IX. Conclusion

The restrictions of the *Pfair* approach are often too restrictive for multiprocessor control devices that have to execute tasks on external triggers, have variable task execution time, and use cooperative task suspension for efficiency reasons. These conditions make the application of algorithms like *Pfair-PD*^{2} impossible for such systems.

In this paper we introduced *Partly-Pfair-PD*^{2}, a dynamic multiprocessor scheduling algorithm based on *Pfair-PD*^{2}. It supports scheduling of multiple time base (MTB) task systems which are a more realistic description of many embedded systems, like for example automotive powertrain systems or systems with Flexray and CAN bus systems.

With a simulation-based schedulability examination methodology, we showed for randomly generated task sets that *Partly-Pfair-PD*^{2} allows to use multiprocessors in a highly efficient way, especially for applications with complex timing requirements.

Acknowledgment

This work is supported by BMBF grant DynaS^3 (”Dynamische SW-Architekturen in Steuergeräten in Fahrzeugsystemen unter Berücksichtigung von Anforderungen zur Funktionalen Sicherheit”) 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] J. M. Lopez, J. L. Diaz, and D. F. Garcia, “Utilization bounds for edf scheduling on real-time multiprocessor systems,” *Real-Time Systems*, vol. 28, no. 1, pp. 39-68, 2004.

[3] N. W. Fisher, “The multiprocessor real-time scheduling of general task systems,” Ph.D. dissertation, University of North Carolina, 2007.

[4] 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.

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

[7] 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.

[8] U. Devi and J. Anderson, “Desynchronized pfair scheduling on multiprocessors,” in *19th IEEE International Parallel and Distributed Processing Symposium, 2005. Proceedings*, 2005, pp. 85b–85b.

[9] A. Srinivasan, “Efficient and flexible fair scheduling of real-time tasks on multiprocessors,” Ph.D. dissertation, Citeseer, 2003.

[10] S. Funk and V. Nadadur, “LRE-TL: An Optimal Multiprocessor Algorithm for Sporadic Task Sets,” in *proceedings Real-Time Networks and Systems conference*, 2009, pp. 159–168.

[11] H. Cho, B. Ravindran, E. Jensen, and D. ETRI, “An optimal real-time scheduling algorithm for multiprocessors,” in *27th IEEE International Real-Time Systems Symposium, 2006. RTSS’06*, 2006, pp. 101–110.

[12] S. Funk, V. Nelis, J. Goossens, D. Milojevic, and G. Nelissen, "On the Design of an Optimal Multiprocessor Real-Time Scheduling Algorithm under Practical Considerations (Extended Version)," January 2010.

[13] J. Anderson and A. Srinivasana, "Pfair scheduling: beyond periodic task systems," *Proceedings of the Seventh International Conference on Real-time Computing Systems and Applications*, pp. 297-306, December 2000.

[14] 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.

[15] 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.

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

[17] M. Deubzer, F. Schiller, J. Mottok, M. Niemetz, and U. Margull, "Effizientes Multicore-Scheduling in Eingebetteten Systemen: Teil 1 - Algorithmen für zuverlässige Echtzeitsysteme," *atp Automatisierungstechnische Praxis*, June 2010.

[18] 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.

[19] 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.

[20] 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," *atp Automatisierungstechnische Praxis*, June 2010.

[21] 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.

[22] 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.

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

X. Appendix

A. 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 shows the windows for the subtasks of two task T_1 and T_2 with a weight of wt$(T_1) = 6/10$ and 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.

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

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 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 $\left|w(T_i^{k+1})\right| = 3$ (e.g. figure 1, subtask T_1^k, $k = 1$) or $\left|w(T_i^{k+1})\right| = 2 \ \wedge \ b(T_i^k) = 0$ (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) [23].

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

Furthermore Srinivasan [23] proved for light tasks ($wt(T_i) < 0.5$) the group-deadline is 0 (formula 25) $\forall\ T_i^k \in T_i$.

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

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.