# 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 Electrical Engineering, 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 SMP multiprocessor scheduling algorithm up to now. The tributes for this are tight requirements to the task system. Drawbacks of the algorithm are restrictions to periodic tasks systems and quantized task execution time. Typical embedded real-time do not fulfill this restrictions. In this paper we address these restrictions and present an extended scheduling algorithm, based on Pfair scheduling, with weaker requirements. In a scheduling simulation we evaluate utilization bounds of the extended algorithm.

**Keywords:** Real-time systems, multiprocessors, dynamic scheduling, proportionate fairness, Pfair

## I. Introduction

Scheduling a taskset $\tau = \{T^1, ..., T^i\}$ under real-time constraints on multiple processing resources $P_n$, where each $P_n$ is a copy of the same processing resource $\forall\ n \in M$ for , has been widely studied in the last two decades.

One group are partitioning approaches which statically map tasks to processing resources 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 like bin-packing variants [2], [3] or optimization based partitioning [4]. Drawback of partitioning approaches is less maximal system utilization [5].

Dynamic scheduling approaches assign tasks during runtime to the processing resources, allowing to use the complete system capacity, as long as $\tau$ fulfills certain requirements [5]. The consequences are overhead for additional prioritization calculations at runtime and migration costs [6]. Proportionate-fair (PFAIR) Scheduling [7] is a class of algorithms known to be optimal. By restrictions to the task systems, listed below its possible to always fulfill all deadlines up to 100% system utilization.

The following restrictions apply to pfair ($T.x$ denotes the task property $x$ of a task $T$, and for explanation we concentrate on one task $T^i$ and call it $T$).

- Periodic task systems, meaning tasks have to be activated periodically, with a constant time $T.p$ between two successive activations and deadline $T.d$ equal $T.p$
- Time is quantized, meaning task execution time $T.e$ and task activation distance $T.p$ have to be multiples of a quantum $Q$
- Task execution time $T.e$ has to be constant over the complete time

In this paper we address these restrictions and introduce an algorithm which is based on a variant of Pfair scheduling called $PD^2$ [8] and extended to schedule task systems without the restrictions mentioned above. The remainder of this paper is organized as follows. In the second chapter we introduce Pfair Scheduling. Chapter 3 describes our extensions to $PD^2$. In Chapter 4 we describe our experimental simulation setup. In the subsequent chapter we used this simulation to evaluate our algorithm. Finally we give a conclusion of our work.

## II. Pfair Scheduling

Proportionate-fair scheduling was introduced by Baruah et. al [7] for real-time systems with periodic tasks. It implies that each task $T$ processing time is assigned with respect to its weight $\text{wt}(T) = T.e/T.p$ with the worst case execution time $T.e$ and the period $T.p$. Baruah et. al proved that Pfair scheduling is optimal for scheduling a task set $\tau$ in a multiprocessor system with $M$ processors iff $\sum_{T \in \tau} \text{wt}(T) \leq M$ [7]. The fluid schedule $\text{fluid}(T, t_1, t_2)$ represents the processing time, which has to be assigned to a task $T$ during a time interval between $t_1$ and $t_2$ in this ideal manner.

$$\text{fluid}(T, t_1, t_2) = \frac{T.e}{T.p}(t_2 - t_1) \tag{1}$$

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

$$\text{Lag}(T, t) = \text{fluid}(T, 0, t) - \text{received}(T, 0, t) \tag{2}$$

Published at ERTS 2010, Toulouse www.erts2010.org

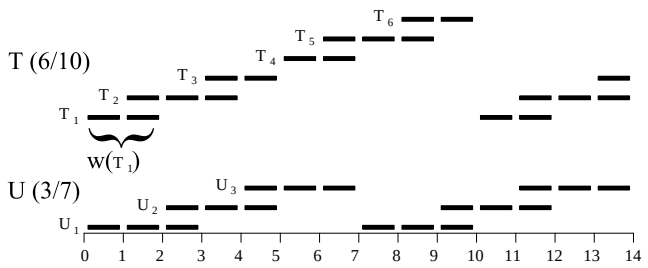


Fig. 1. Windows $w(X_k)$ of a heavy task $T$ with weight $6/10$ and light task $U$ with weight $3/7$. Subtasks $T_1, ..., T_6$ and $U_1, ..., U_3$ have to be scheduled in there window to guarantee pfairness.

Informally, if less processing time than the fluid schedule was allocated to a task it has a positive Lag and a negative Lag otherwise.
Pfair scheduling works with a quantum based model. Let $Q$ denote a quantum, then P-fairness implies that the maximal Lag is one quantum.

$$|\mathsf{Lag}(T,t)| \leq Q \quad \forall\, T,t \tag{3}$$

A task is split in a number of subtasks $T_1, ..., T_k$. We denote $T_k$ the $k^{\text{th}}$ subtask of task $T$. Each $T_k$ has the execution time of one quantum. This requires that the execution time of all subtasks $|\{T_1, ..., T_k\}| \times Q$ is equal to $T.e$. To fulfill (3), a subtask $T_k$ has to be scheduled in a time window $\mathsf{w}(T_k)$, starting with the pseudo-release $r(T_k)$ and the pseudo-deadline $d(T_k)$ ($\lfloor x \rfloor$ is the highest integer, smaller or equal to $x$; $\lceil x \rceil$ is the smallest integer, higher or equal to $x$).

$$r(T_k) = \left\lfloor \frac{k-1}{\mathsf{wt}(T)} \right\rfloor \tag{4}$$

$$d(T_k) = \left\lceil \frac{k}{\mathsf{wt}(T)} \right\rceil - 1 \tag{5}$$

We call the smallest time division, where a complete subtask can be executed a slot $t$. Depending on the weight of a task a window has a number of slots $t$, available for scheduling subtask $T_k$.

$$|\mathsf{w}(T_k)| = \left\lceil \frac{k}{\mathsf{wt}(T)} \right\rceil - \left\lfloor \frac{k-1}{\mathsf{wt}(T)} \right\rfloor \tag{6}$$

To realize the Pfair scheduling concept different scheduling policies has been published. For a system with two processors Anderson and Srinivasan proved that scheduling the subtasks after an Earliest-Pseudo-Deadline-First *EPDF* scheme is optimal [9] (They used pseudo to differ between the subtask and task deadline). For more general systems with more than two processors, tie-breaking rules have to be applied. In this paper we focus on the variant *PD*$^2$ [8] which is known to be the most efficient beside *PF* [7] and *PD* [10] because it uses only two additional tie-breaking rules, which can be calculated efficiently during runtime. In the following we define the PD$^2$ policies. The $\succ$ and $\prec$ operators are used to describe the scheduling prioritization: subtask $X_i$ has to be preferred before subtask $Y_j$ when $X_i \succ Y_j$, $Y_j$ has to be preferred before subtask $X_i$ when $X_i \prec Y_j$, and otherwise the next policy has to be evaluated. If there is no further policy the selection is arbitrary.

**Policy 1 (Pseudo-Deadline):**
$X_i \succ Y_j$ if $d(X_i) < d(Y_j)$; $X_i \prec Y_j$ if $d(X_i) > d(Y_j)$.
As above already mentioned, $d(T_k)$ is the pseudo-deadline and this rule is equal at each Pfair algorithm. As example shows figure 1 the windows for the subtasks of two task $T$ and $U$ with a weight of $6/10$ and $3/7$. The pseudo-deadlines of $T's$ subtasks are $\{d(T_1), ..., d(T_6)\} = \{2, 4, 5, 7, 9, 10\}$.

**Policy 2 (Overlapping-Bit):**
$X_i \succ Y_j$ if $b(X_i) > b(Y_j)$; $X_i \prec Y_j$ if $b(X_i) < b(Y_j)$.
The *overlapping bit* $b(T_k)$ denotes the overlapping of two successive subtask windows $T_k$ and $T_{k+1}$ of a Task $T$.

$$b(T_k) = \begin{cases} 1 & ,\ if\ d(T_k) > r(T_{k+1}) \\ 0 & ,\ if\ d(T_k) \leq r(T_{k+1}) \end{cases} \tag{7}$$

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 $T's$ subtasks are $\{b(T_1), ..., b(T_6)\} = \{1, 1, 0, 1, 1, 0\}$.

**Policy 3 (Group-Deadline):**
$X_i \succ Y_j$ if $D(X_i) > D(Y_j)$; $X_i \prec Y_j$ if $D(X_i) < D(Y_j)$.
The *group-deadline* concerns the effect of schedule decision of a subtask for its subsequent subtasks. Let $T_i, ..., T_k$ be a sequence of subtasks of a heavy task $T$ (heavy means $wt(T) \geq 0.5$) such that $i < j \leq k$ with a window of length $|w(T_j)| = 2$ and $T_{j+1}$ either of length $|w(T_j)| = 3$ or $|w(T_j)| = 2 \ \wedge\ b(T_{j+1}) = 0$. Then scheduling subtask $T_j$ in the last slot of its window $w_j$ results in scheduling all subsequent subtasks in there last slot, excepted subtask $T_{j+1}$ because there are 2 slots left. Therefore the sequence $T_i, ..., T_k$ 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 last time slot of this unit is $d(T_j)+1$ which is called group-deadline. Formally, the group-deadline $D(T_k)$ can be calculated for heavy tasks by (8) [11].

$$D(T_k) = \left\lceil \frac{\left\lceil \left\lceil \frac{k}{wt(T)} \right\rceil \times (1 - wt(T)) \right\rceil}{1 - wt(T)} \right\rceil \text{ if } wt(T) \geq 0.5 \tag{8}$$

Furthermore Srinivasan [11] proved for light tasks ($wt(T) < 0.5$) $D(T_k) = 0\ \forall\ T_k \in T$. In figure 1 the group-deadlines of $T's$ subtasks are $\{D(T_1), ..., D(T_6)\} = \{3, 5, 5, 8, 10, 10\}$. At task $U$ each window has at least 2 slots left, when the precedent subtask is scheduled in the last slot.

## III. Partly PFAIR Scheduling

In this chapter we present an extended algorithm, based on the $PD^2$ policies, called $PD^2_{ex}$. It allows heterogeneous tasks systems instead of periodic task systems, and relieves the restriction from quantized task execution time and periodicity. As long as we modify only task properties, used for the calculation of the original $PD^2$ policies, we denote $T.x \rightarrow T.x'$ the replacement of the task property $T.x$ by the property $T.x'$.

### A. Heterogeneous task systems

Periodic task systems are defined by a constant value for the time between two successive activations of a task. Heterogeneous task systems consist of task activation schemes where this value can change with the task instance. This e.g. can be a sensor triggered activation, dependent on physical values.
We use the following definition: In a heterogeneous task system, a task $T^i$ releases a number of jobs $T^{i,j}$. If we denote $T.x^{i,j}$ to be the property of the $j^{\text{th}}$ job of task $T^i$, then let $T.p^{i,j}$ denote the interarrival time of the job $T^{i,j}$ and $T^{i,j+1}$. For the calculation of the scheduling policies we use the minimal of the interarrival time:

$$T.p \rightarrow T.p' \ : \ T.p' = \min_{j \in T^i} \left( T.p^{i,j} \right) \tag{9}$$

### B. Variable subtask execution time

As already mentioned, Pfair scheduling works with a quantum based timing model, where after each quantum $Q$ a schedule decision is made for the next time slot. In order to minimize caching overhead and guaranteeing data consistency it is beneficial to provide execution time without interruption by other tasks. This implies a variable subtask execution time and scheduling decision after each finished subtask. We present here the cooperative variant, where the time left for a subtask is used to process the next schedule decision, when the subtask execution time is smaller than $Q$. The restrictions for the execution times $T_k.e^{i,j}$ of the $k^{\text{th}}$ subtask of the $j^{\text{th}}$ job of a task $T^i$ are the following. The subtask execution time of each subtask has to be smaller than one quantum $Q$.

$$T_k.e^{i,j} \leq Q \ \forall \ i,j,k \tag{10}$$

The count of subtasks of a task multiplied with $Q$ has to be smaller or equal than the interarrival time of the job $T.p^{i,j}$.

$$\left| \left\{ {T^{i,j}}_1, ..., {T^{i,j}}_k \right\} \right| Q \leq T.p^{i,j} \tag{11}$$

Because of the variation in the subtask execution time the following modification of the policies occurs:

$$T.e \rightarrow T.e' \ : \ T.e' = \left| \left\{ {T^{i,j}}_1, ..., {T^{i,j}}_k \right\} \right| Q \tag{12}$$

## IV. The Experimental Setup

This chapter describes a simulative approach for the investigation of the real time requirements. We used a discrete-event based simulation [12] with a model of the hardware and software characteristics. During the simulated time, a trace is generated with a number of state transitions of the system. We concentrate here on the state transitions of the tasks relevant for the evaluation of the real time properties.
Buttazzo [13] introduced several real time metrics and König et. al [14] presented how they can be applied on the trace of a simulation.
The lateness $l^{i,j}$ of the $j^{\text{th}}$ job of task $T^i$ is calculated by

$$l^{i,j} = d^{i,j} - f^{i,j} \tag{13}$$

which is equivalent to the time left until reaching the deadline. The lateness is negative when the finishing time $f^{i,j}$ is smaller the absolute deadline $d^{i,j}$. For a measure of deadline-compliance for a complete taskset, we determine for each task the worst job and norm this with the relative deadline $D_i$. We define the maximal value of all tasks in a taskset as maximal normed lateness $\mathsf{L}(\tau)$.

$$\mathsf{L}(\tau) = \max_{i \in \tau} \left( \frac{\max_{j \in T_i} \left( l^{i,j} \right)}{D_i} \right) \tag{14}$$

## V. Simulation Results

For the evaluation we compared $PD^2_{ex}$ to a partitioning approach with rate monotonic (RM) scheduling. The task to processor assignment was done using First-Fit Decreasing (FFD) and Worst-Fit Decreasing (WFD) heuristics [2], [3].
For a case study we analyzed an existing automotive powertrain heterogeneous taskset for a dualcore SMP processor. In a monte carlo analysis we generated randomly task sets close to the existing taskset with variable subtask execution times, to determine the utilization bound where the deadlines are missed. Thereby the subtask execution times differed within an interval of 100-300 $\mu$s. The other task properties like activation schemes, deadlines, priorities, and number of subtasks stayed constant. The deadlines and periods for the tasks of a taskset are in a range of 0.6-1000ms. The variance results in different system utilization

$$U_{sys} = \sum_{i \in \tau} \frac{T^i.e}{T^i.p} \tag{15}$$

Figure 2 shows the resulting L for the partitioning approaches FFD and WFD with RM scheduling and our extended Pfair algorithm. The resulting tasksets are clustered by their system utilization. For each algorithm and cluster statistical analysis are done. The symbol represents the median L, the box the 99% confidence interval for the median and the whispers the 99%

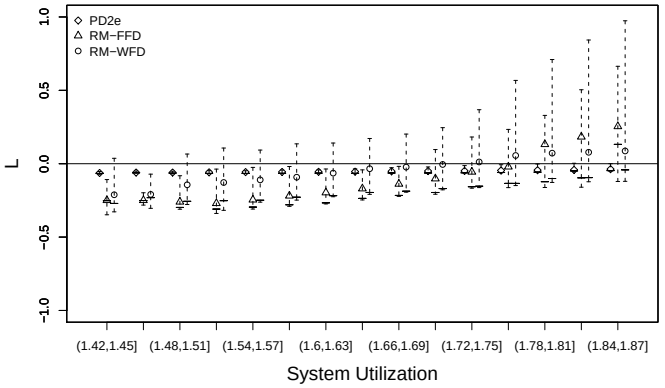


Fig. 2. Maximum Normed Lateness L of a randomly generated tasksets clustered by system utilization for a dualcore SMP processor.

confidence interval of the maximal and minimal L in this system utilization cluster.

It can be seen that WFD misses deadlines even at low utilizations. FFD allows for the generated tasksets a utilization of $\sim$1.7 until first deadline misses occur. Even though the L values in the low utilization area are worse than for the other algorithms, $PD^2_{ex}$ schedules nearly all tasksets over the complete utilization range $1.42 - 1.87$. The huge range of the whisper at the partitioning approach and the small at $PD^2_{ex}$ can be explained by the *non-work-conserving* processing (meaning the processor is set to idle instead of tasks are ready) of $PD^2_{ex}$. Therefore $PD^2_{ex}$ schedules the tasks in the manner that jitters are lower than at *work-conserving* approaches. At the partitioning approach, L strongly depends on the taskset, whereby at $PD^2_{ex}$, L is nearly taskset independent.

## VI. Conclusion

In this paper we propose an extension to the Pfair scheduling algorithm $PD^2$ which allows handling heterogeneous task systems with variable subtask execution times. The results of the systematic simulation approach shows that (partly) pfairness can achieve an immense benefit according maximal system utilization in comparison with partitioning approaches like bin-packing and uniprocessor scheduling algorithms like RM.

## 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] 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 (accepted for publishing)*, 2009.

[5] 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.

[6] G. Stamatescu, M. Deubzer, J. Mottok, and D. Popescu, “Migration overhead in multiprocessor scheduling,” *Proceedings of the Second Embedded Software Engineering Congress (accepted for publishing)*, 2009.

[7] 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.

[8] 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.

[9] 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.

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

[12] 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 conference on Design, automation and test in Europe*. ACM New York, NY, USA, 2008, pp. 556-561.

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

[14] 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 conference on Design, automation and test in Europe*, 2009.