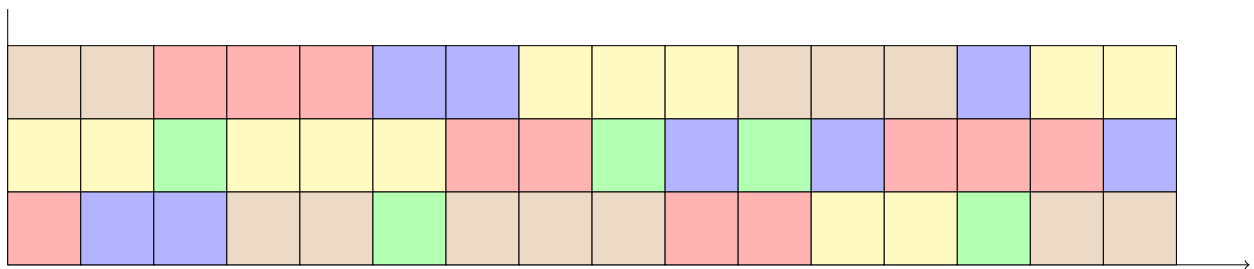
Vorlesung Echtzeitsysteme II

Thema 2.5: PFair-Scheduling

Robert Baumgartl

16. November 2023

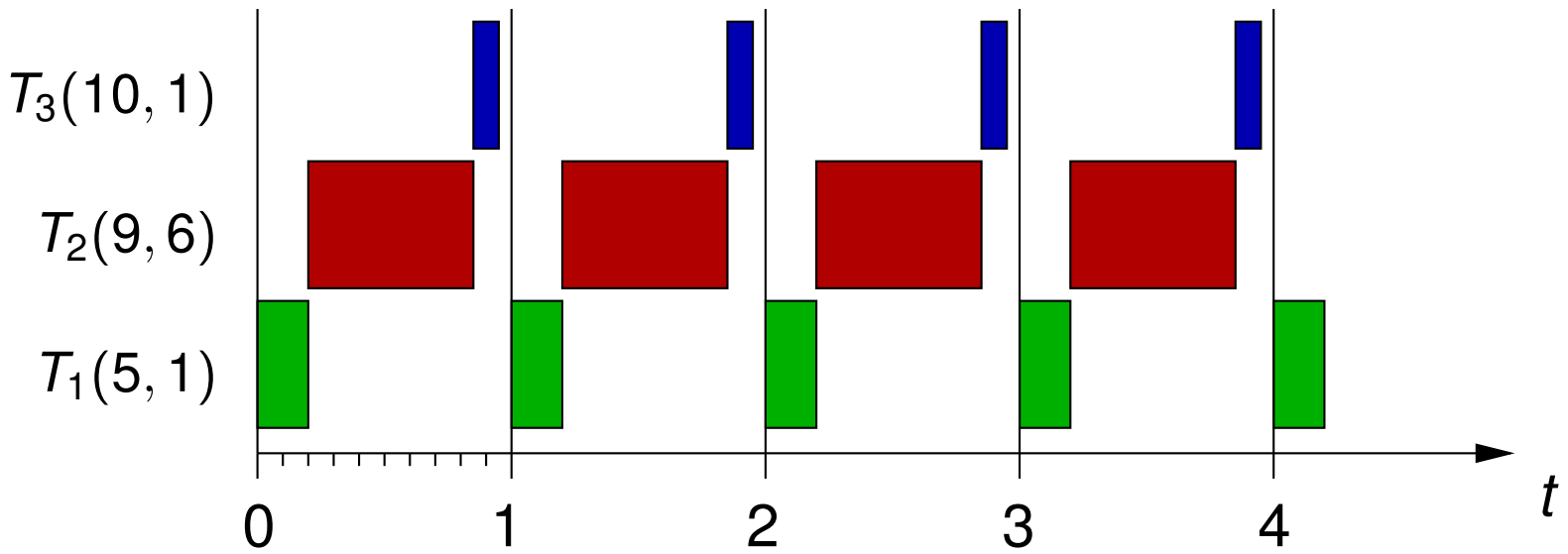


Grundüberlegung

- „**Proportionate Fairness**“: alle Tasks (bzw. deren Jobs) sollen *anteilig* gleich schnell voranschreiten
- d. h. , in einem beliebigen Zeitintervall $[t_1, t_2]$ soll jede Task $T_i(t_{p,i}, t_{e,i})$ genau $(t_2 - t_1)u_i$ Rechenzeit erhalten
- z. B. $t_2 - t_1 = 1 \rightsquigarrow$ jede Task bekommt in jeder Zeiteinheit genau $u_i = \frac{t_{e,i}}{t_{p,i}}$ Rechenzeit
- Taskmenge für $m = 1$ Prozessoren planbar, solange $u = \sum u_i \leq 1$ (Kontextwechsel vernachlässigt).
- Unter Vernachlässigung der Zeiten für Kontextwechsel und Taskmigration ist eine Taskmenge für m Prozessoren planbar, solange $u = \sum u_i \leq m$, $\max_i(u_i) \leq 1$ und Tasks beliebig teilbar[1].

[1] W. A. Horn. “Some Simple Scheduling Algorithms”. In: *Naval Research Logistics Quarterly* 21.1 (März 1974), S. 177–185. DOI: 10.1002/nav.3800210113.

Beispiel zur Proportionalen Fairness



- $u = \sum_i u_i = \frac{1}{5} + \frac{6}{9} + \frac{1}{10} = \frac{87}{90} < 1$
- $\rightsquigarrow$ auf $m = 1$ Prozessor planbar
- Priorität beliebig (!)
- Nachteil: extrem viele Kontextwechsel (in jedem diskreten Zeitintervall n)

Reduktion der Anzahl benötigter Kontextwechsel

Idee: Prozessorzeit nur ganzzahlig zuteilen!

Konsequenzen:

- Prozessorzeit wird nicht mehr möglichst feingranular, sondern „gequantelt“ zugeteilt
- dabei soll trotzdem proportional fair fortgeschritten werden
- zum Zeitpunkt t hat eine Task T_i entweder $\lfloor t \cdot u_i \rfloor$ oder $\lceil t \cdot u_i \rceil$ Zeiteinheiten erhalten
- Tasks können nicht mehr genau proportional fortschreiten $\rightarrow$ Parameter „lag“

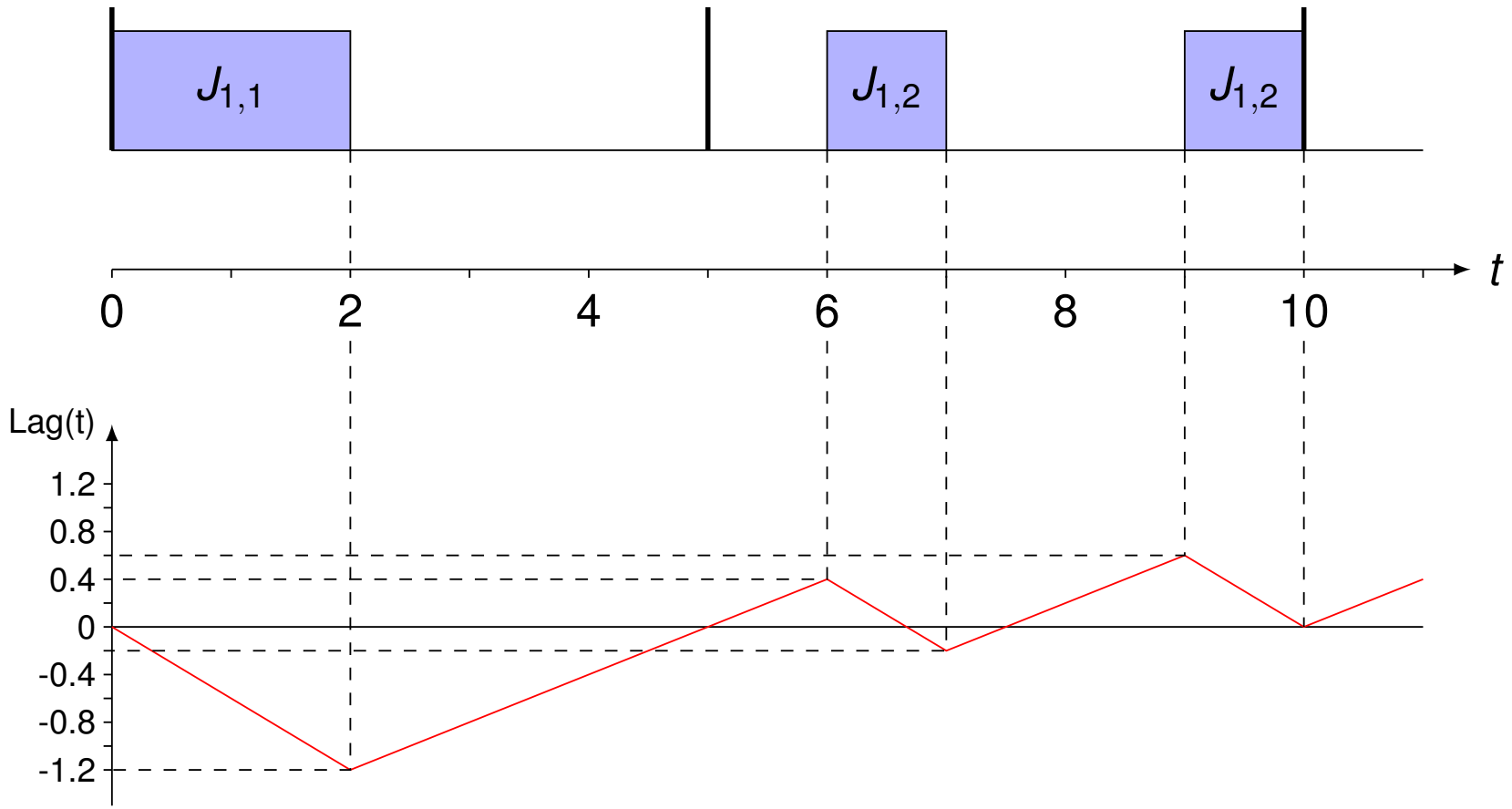
Lag („Rückstand“)

Def.

$$\operatorname{lag}(T_i, t) = u_i \cdot t - \sum_{k \in [0,t)} \operatorname{alloc}(T_i, k)$$

- Zeitanteil, den T_i zu t bekommen haben müsste: $u_i \cdot t$
- Zeitanteil, den T_i zu t tatsächlich bekommen hat: $\sum_{k \in [0,t)} \operatorname{alloc}(T_i, k)$
- Wird T_i für 1 Zeiteinheit ausgeführt, dann sinkt $\operatorname{lag}(T_i, t)$ um $1 - u_i$.
- Wird T_i für 1 Zeiteinheit *nicht* ausgeführt, dann steigt $\operatorname{lag}(T_i, t)$ um u_i.

Veranschaulichung von *Lag* für (5,2)



$$\text{Bsp: } \text{lag}(6) = u_i \cdot t - \sum_{k \in [0,6]} \text{alloc(T)} = \frac{2}{5} \cdot 6 - 2 = \frac{2}{5} = 0.4$$

Lag bei Pfair

Def. Ein Plan ist **proportional fair**, wenn gilt:

$$\forall t \in \mathbb{N}, T_i \in \mathcal{T} : -1 < lag(T_i, t) < 1.$$

- Ein solcher Plan existiert immer, sofern $\sum u \leq m$. (Baruah, 1996)[2]

[2] S. K. Baruah u. a. “Proportionate Progress: A Notion of Fairness in Resource Allocation”. In: *Algorithmica* 15.6 (1996), S. 600–625.

Charakteristischer String

Def. Der (potentiell unendliche) charakteristische String α einer Task T_i zu den diskreten Zeitpunkten t wird gebildet durch

$$\alpha(i,t) = \text{sign}\left(\frac{t_{e,i}}{t_{p,i}} \cdot (t+1) - \left\lfloor \frac{t_{e,i}}{t_{p,i}} \cdot t \right\rfloor - 1\right)$$

Beispiel: T(5,2)

$$\alpha(0) = \text{sign}\left(\frac{2}{5} \cdot 1 - \left\lfloor \frac{2}{5} \cdot 0 \right\rfloor - 1\right) = \text{sign}\left(\frac{2}{5} - 0 - 1\right) = -$$

$$\alpha(1) = \text{sign}\left(\frac{2}{5} \cdot 2 - \left\lfloor \frac{2}{5} \cdot 1 \right\rfloor - 1\right) = \text{sign}\left(\frac{4}{5} - 0 - 1\right) = -$$

$$\alpha(2) = \text{sign}\left(\frac{2}{5} \cdot 3 - \left\lfloor \frac{2}{5} \cdot 2 \right\rfloor - 1\right) = \text{sign}\left(\frac{6}{5} - 0 - 1\right) = +$$

$$\alpha(3) = \text{sign}\left(\frac{2}{5} \cdot 4 - \left\lfloor \frac{2}{5} \cdot 3 \right\rfloor - 1\right) = \text{sign}\left(\frac{8}{5} - 1 - 1\right) = -$$

$$\alpha(4) = \text{sign}\left(\frac{2}{5} \cdot 5 - \left\lfloor \frac{2}{5} \cdot 4 \right\rfloor - 1\right) = \text{sign}\,(2 - 1 - 1) = 0.$$

$\rightsquigarrow$ $\alpha =$„$--+-0--+-0--+-0$“ usw.

Charakteristischer Substring

Def. Der (stets endliche) charakteristische Substring $\alpha_i(t)$ einer Task T_i zum Zeitpunkt t ergibt sich aus

$$\alpha_i(t) \stackrel{def}{=} \alpha(i,t)\,\alpha(i,t+1)\,\ldots\,\alpha(i,t')$$
$$\text{mit } t' = \min k : k > t \wedge \alpha(i,k) = 0.$$

(„Ausschnitt aus α vom aktuellen Wert bis zur nächsten 0“)

Algorithmus PF

Grundidee: Immer so zuteilen, dass *lag* nicht zu groß (>1) und nicht zu klein (<-1) wird.

- Eine Task T_i ist zu t im Zustand *urgent*, wenn $\text{lag}(T_i, t) > 0$ (Task ist „hinterher“) und $\alpha(t, i) \neq -$.
- Eine Task T_i ist zu t im Zustand *tnegru*, wenn $\text{lag}(T_i, t) < 0$ (Task ist „voraus“) und $\alpha(t, i) \neq +$.
- Anderenfalls ist die Task im Zustand *contending* („wetteifernd“).

Algorithmus PF:

1. Plane alle Tasks im Zustand *urgent*.
2. Falls noch Prozessoren frei sind, plane Tasks im Zustand *contending*, wobei die entsprechenden Substrings $\alpha(t, i)$ lexikografisch miteinander verglichen werden und $+ > 0 > -$ gilt.

Beispiel

Taskmenge

Task	*Parameter*	*u*	α
T_1	(3,1)	0.33	$--0$
T_2	(4,2)	0.5	-0
T_3	(7,5)	0.714	$-++-++0$
T_4	(11,8)	0.720	$-++-+++-++0$
T_5	(462,335)	0.725	$-++-+++-++-++\ldots$

- $m = 3$ Prozessoren (oder Ressourcen)
- T_5 ist *idle*- oder Dummy-Task

$$(u_5 = m - \sum_{i=1}^{4} u_i = 3 - \frac{1051}{462} \approx 0.725)$$

Beispiel

Ableitung eines Plans gemäß PFair

	$Lag(i,t) \cdot t_{p,i}$					$\alpha(i,t)$					Urgent	Contending	Tnegru
t	T_1	T_2	T_3	T_4	T_5	T_1	T_2	T_3	T_4	T_5			
0	0	0	0	0	0	-	-	-	-	-	{}	$\mathbf{T_4} > \mathbf{T_5} > \mathbf{T_3} > T_2 > T_1$	{}
1	1	2	-2	-3	-127	-	0	+	+	+	$\{\mathbf{T_2}\}$	$\mathbf{T_4} > \mathbf{T_5} > T_3 > T_1$	{}
2	2	0	3	-6	-254	0	-	+	+	+	$\{\mathbf{T_1}, \mathbf{T_3}\}$	$\mathbf{T_2} > T_4 > T_5$	{}
3	0	-2	1	2	81	-	0	-	-	-	{}	$\mathbf{T_4} > \mathbf{T_5} > \mathbf{T_3} > T_1$	$\{T_2\}$
4	1	0	-1	-1	-46	-	-	+	+	+	{}	$\mathbf{T_4} > \mathbf{T_5} > \mathbf{T_3} > T_2 > T_1$	{}
5	2	2	-3	-4	-173	0	0	+	+	+	$\{\mathbf{T_1}, \mathbf{T_2}\}$	$\mathbf{T_4} > T_5 > T_3$	{}
6	0	0	2	-7	162	-	-	0	+	+	$\{\mathbf{T_3}, \mathbf{T_5}\}$	$\mathbf{T_2} > T_4 > T_1$	{}
7	1	-2	0	1	35	-	0	-	-	-	{}	$\mathbf{T_4} > \mathbf{T_5} > \mathbf{T_3} > T_1$	$\{T_2\}$
8	2	0	-2	-2	-92	0	-	+	+	+	$\{\mathbf{T_1}\}$	$\mathbf{T_4} > \mathbf{T_5} > T_3 > T_2$	{}
9	0	2	3	-5	-219	-	0	+	+	+	$\{\mathbf{T_2}, \mathbf{T_3}\}$	$\mathbf{T_4} > T_5 > T_1$	{}
10	1	0	1	-8	116	-	-	-	0	-	{}	$\mathbf{T_5} > \mathbf{T_3} > \mathbf{T_1} > T_2$	$\{T_4\}$
11	-1	2	-1	0	-11	0	0	+	-	+	$\{\mathbf{T_2}\}$	$\mathbf{T_4} > \mathbf{T_5} > T_3$	$\{T_1\}$
12	0	0	4	-3	-138	-	-	+	+	+	$\{\mathbf{T_3}\}$	$\mathbf{T_4} > \mathbf{T_5} > T_2 > T_1$	{}
13	1	2	2	-6	-265	-	0	0	+	+	$\{\mathbf{T_2}, \mathbf{T_3}\}$	$\mathbf{T_1} > T_4 > T_5$	{}
14	-1	0	0	2	70	0	-	-	-	-	{}	$\mathbf{T_4} > \mathbf{T_5} > \mathbf{T_3} > T_2$	$\{T_1\}$
15	0	2	-2	-1	-57	-	0	+	+	+	$\{\mathbf{T_2}\}$	$\mathbf{T_4} > \mathbf{T_5} > T_3 > T_1$	{}
⋮			⋮					⋮			⋮	⋮	⋮

Zugehöriger Plan

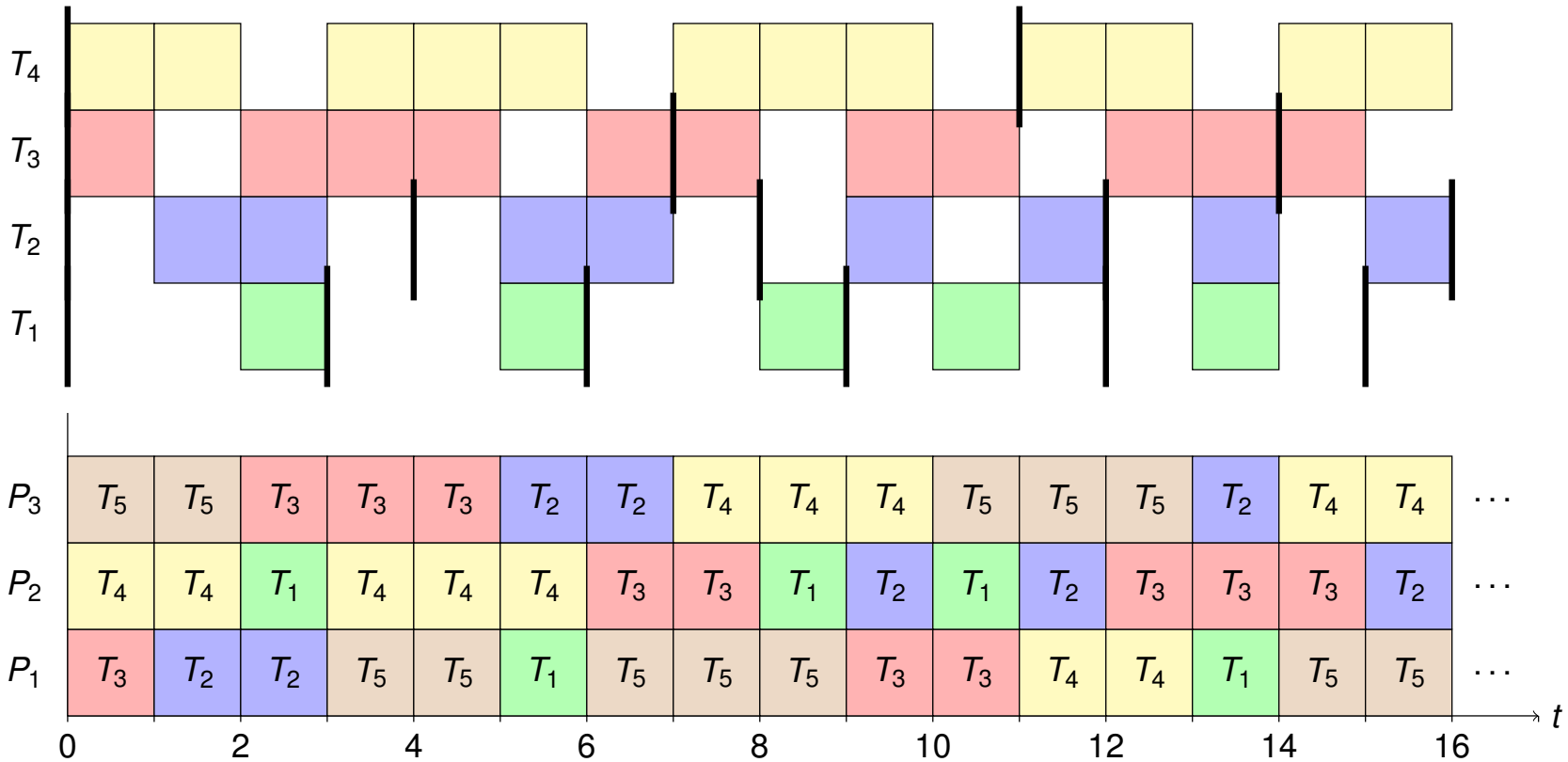


Abbildung: Mittels PF erzeugter Plan für $m = 3$ Prozessoren und $\mathcal{T} = \{T_1(3,1), T_2(4,2), T_3(7,5), T_4(11,8), T_5(335,462)\}$

Fazit

Vorteile von fairen Schedulingverfahren:

- gleichmäßiges Voranschreiten (Fairness)
- planbare Auslastung für m Prozessoren: $\sum_i u_i \leq m$. (kein Dhall's Effect, keine komplizierten Einplanungstests)

Nachteile:

- (zu) viele Kontextwechsel
- (zu) viele Prozessmigrationen
- algorithmischer Aufwand für PF ist hoch (polynomiell)

Weitere Entwicklung

Verbesserte Verfahren:

- PD: – effizienterer Vergleich
- PD^2: – noch effizienterer Vergleich
- ER-fair: – Einbeziehung sporadischer Tasks
- BFair – weniger Kontextwechsel, weniger Migration

Lesevorschläge

- S. K. Baruah u. a. *Proportionate Progress: A Notion of Fairness in Resource Allocation*. Techn. Ber. University of Texas at Austin, Department of Computer Science, Sep. 1994 (PF)
- S. K. Baruah u. a. “Proportionate Progress: A Notion of Fairness in Resource Allocation”. In: *Algorithmica* 15.6 (1996), S. 600–625 (PF)
- Sanjoy K. Baruah, Johannes E. Gehrke und C. Greg Plaxton. “Fast Scheduling of Periodic Tasks on Multiple Resources”. In: *Proceedings of the 9th International Parallel Processing Symposium*. Santa Barbara, CA, Apr. 1995, S. 280–288. DOI: 10.1109/IPPS.1995.395946 (PD)
- Sanjoy K. Baruah, Johannes E. Gehrke und C. Greg Plaxton. *Fast Scheduling of Periodic Tasks on Multiple Resouces*. Techn. Ber. TR-95-02. University of Texas at Austin, Department of Computer Science, Feb. 1995 (PD)
- James H. Anderson und Anand Srinivasan. “Early-Release Fair Scheduling”. In: *Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS’2000)*. Juni 2000. DOI: 10.1109/EMRTS.2000.853990 (ER-fair)
- Dakai Zhu u. a. “An optimal boundary fair scheduling algorithm for multiprocessor real-time systems”. In: *Journal of Parallel and Distributed Computing* 71.10 (Okt. 2011), S. 1411–1425. DOI: 10.1016/j.jpdc.2011.06.003 (BFair)