4.5. Statistisches ratenmonotones Scheduling SRMS

A. ATLAS, A. BESTAVROS: Statistical Rate Monotonic Scheduling: Algorithm, Analysis, and Evalution. TR 98-010, Boston Univ., May 1998.

http://www.cs.bu.edu/groups/realtime/SRMSworkbench

4.5.1. Ausgangspunkt

- **Motivation**

  Schwankende Bearbeitungszeiten

  Betriebsmittel-Auslastung

  Überlast

- **Grundidee**

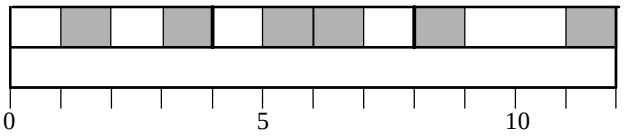
  Periodische Task muß nur zu bestimmtem Prozentsatz Deadline einhalten, gesteuert durch einen „Kredit“ über einen bestimmten Zeitraum („Superperiode“).

  Dazu muß Bearbeitungszeit am Beginn jeder Periode bekannt sein.

- **Hintergrund**

  Tasktransformation $t_i = (c_i, t_i) \mapsto \hat{t}_i = (\hat{c}_i, t_{i+1})$ mit $\hat{c}_i = \frac{t_{i+1}}{t_i} \cdot c_i$

  Beispiel. $t_3 = 4, \quad c_3 = 2, \quad t_4 = 12 \quad \Rightarrow \quad \hat{t}_3 = \frac{12}{4} \cdot 2 = 6$



  Es gilt: Ist $\{(c_1, t_1), ..., (c_i, t_i), (c_{i+1}, t_{i+1}), ...\}$ gemäß RMS einplanbar, so auch $\{(c_1, t_1), ..., (\hat{c}_i, t_{i+1}), (c_{i+1}, t_{i+1}), ...\}$.

4.5.2. SRMS-Modell

- **Taskbeschreibung**

$T = \{t_1, ..., t_n\}$	periodische Tasks
$t_i = (t_{ij})_{j = 1,2,...}$	t_{ij} : j-ter Job von Task t_i, $i = 1,...,n$
$t_i = (t_i, f_i, Q_i)$	t_i: Periode = Deadline
	f_i: Dichtefunktion der Bearbeitungszeit von t_{ij}
	Q_i: QoS-Parameter; Wkt., daß t_i Deadline erreicht
e_{ij}	Bearbeitungszeit von t_{ij}

  o.B.d.A. T geordnet gemäß t_i $(t_1 \le t_2 \le ...)$

- **Superperiode**

  von t_i: t_{i+1}

- ***T* einplanbar (feasible):**

  Alle $t_i \in T$ erreichen (ausgehandelten) QoS-Parameter.

4.5.3. Prinzipien und Vorgehen

- **Scheduling**

  feste Prioritäten, preemptiv

- **Admission control**

– ***lokal:*** zur Freigabezeit jedes Jobs.

  Grundsätze: kein Job überschreitet seine BM-Anforderung e_{ij}

  kein Job wird zugelassen, wenn Deadline-Einhaltung nicht garantiert werden kann

– ***global:*** Zulassung einer Task, Aushandeln des QoS-Parameters

- **Vorgehen**

– t_i besitzt „Konto“ b_i, zu Beginn jeder Superperiode aufgefüllt mit „Kredit“ a_i.

– Wird t_{ij} zugelassen, so wird Konto b_i mit BM-Bedarf e_{ij} belastet.

- **Systemstruktur**

  Basis-SRMS und Erweiterungen zur Leistungssteigerung

  (Zeitvererbung, 2. Chance)

4.5.4. Basic-SRMS für harmonische Tasks

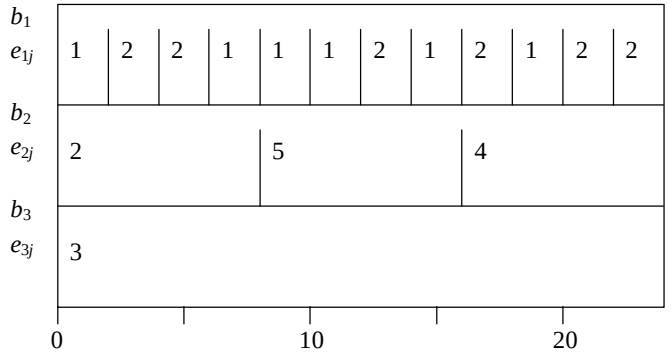
Harmonische Tasks (einfach-periodisch): $t_i < t_j \Rightarrow t_i \mid t_j$

- **Zulassung von t_{ij}**

$$\left(e_{ij} \le b_i\right) \wedge \left(e_{ij} \le t_i - \sum_{l=1}^{i-1} a_l \cdot \frac{t_i}{t_{l+1}}\right)$$

- **Beispiel.**

i	t_i	c_i	$\bar{c}_i$	u_i	$\bar{c}_{\text{supper}}$	a_i	$\tilde{u}_i$
1	2	1..2					
2	8	1..7					
3	24	1..11					



4.5.5. Basic-SRMS für beliebige Tasks

- **Überlappender Job**

  Bereitzeit und Deadline liegen in verschiedenen Superperioden

- **Lokale Zulassung**

  in Bereit-Superperiode: Job muß bis Ende dieser Periode beendet sein

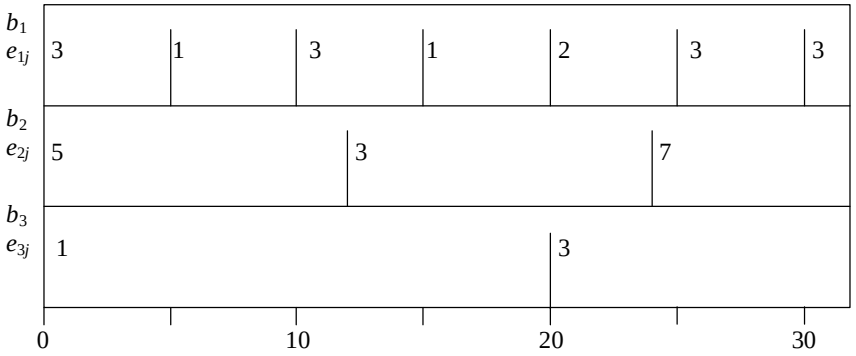
  in Deadline-Superperiode: Job muß bis Beginn dieser Periode verzögert werden

- **Zulassung von t_{ij}**

$$\left(e_{ij} \le b_i\right) \wedge \left(e_{ij} \le t_i - \sum_{l=1}^{i-1} \left(a_l \cdot \left\lfloor \frac{t_i}{t_{l+1}} \right\rfloor + \min\left(a_l, t_i - t_{l+1} \cdot \left\lfloor \frac{t_i}{t_{l+1}} \right\rfloor\right)\right)\right)$$

- **Beispiel.**

  t_i: 5; 12; 20 a_i: 4; 8; 4.



4.5.6. Globale Zulassung für harmonische Tasks

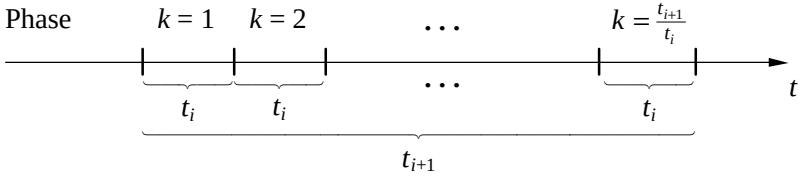
- ***T* einplanbar**

  Jedes $t_i \in T$ kann garantiert zu Beginn jeder Superperiode Kredit a_i erhalten.

  Kriterium: $\sum_{i=1}^{n} \frac{a_i}{t_{i+1}} \le 1$

- **Berechnung von $\tilde{Q}_i$:= QoS(a_i)**

– ***Taskphasen***



$$S_{ik} = \begin{cases} 1 & t_{ij} \text{ zu Beginn von Phase } k \text{ zugelassen} \\ 0 & t_{ij} \text{ zu Beginn von Phase } k \text{ abgewiesen} \end{cases}$$

– ***Berechnung von $\tilde{Q}_i$***

$\boldsymbol{P}(S_{i1} = 1) = \boldsymbol{P}(e_{ij} \le a_i)$

$\boldsymbol{P}(S_{i2} = 1) = \boldsymbol{P}(e_{i,j-1} \le a_i) \cdot \boldsymbol{P}(e_{i,j-1} + e_{ij} \le a_i) + \boldsymbol{P}(e_{i,j-1} > a_i) \cdot \boldsymbol{P}(e_{ij} \le a_i)$

...

$$\tilde{Q}_i = \frac{t_i}{t_{i+1}} \cdot \sum_{k=1}^{t_{i+1}/t_i} P(S_{ik} = 1)$$

- **Berechnung von a_i aus $\tilde{Q}_i$**

  $\tilde{Q}_i$ = QoS(a_i) monoton wachsend $\rightarrow$

  Intervallschachtelung: kleinstes a_i, so daß $\tilde{Q}_i \ge Q_i$

- **Beispiel**

i	P_i	E_i^{max}	$E(E_i)$	PDF	# Phases
1	5	2	1.5	uniform	2
2	10	3	2.0	uniform	3
3	30	13	7.0	uniform	3
4	90	4	2.5	uniform	∞

Table 1: Example Task System

Guarantee Calculations for Task 1

a_1	$P(S_{1,1}=1)$	$P(S_{1,2}=1)$	$QoS(\tau_1)$
2	1.0000	0.2500	0.6250
4	1.0000	1.0000	1.0000

Guarantee Calculations for Task 2

a_2	$P(S_{2,1}=1)$	$P(S_{2,2}=1)$	$P(S_{2,3}=1)$	$QoS(\tau_2)$
3	1.0000	0.3333	0.2345	0.5230
6	1.0000	1.0000	0.6296	0.8770
9	1.0000	1.0000	1.0000	1.0000

Guarantee Calculations for Task 3

a_3	$P(S_{3,1}=1)$	$P(S_{3,2}=1)$	$P(S_{3,3}=1)$	$QoS(\tau_3)$
21	1.0000	0.9110	0.5628	0.8250
24	1.0000	0.9820	0.7010	0.8944
27	1.0000	1.0000	0.8340	0.9448
30	1.0000	1.0000	0.9250	0.9750
33	1.0000	1.0000	0.9745	0.9915
36	1.0000	1.0000	0.9950	0.9980
39	1.0000	1.0000	1.0000	1.0000

Table 2: QoS Calculations for the Example Task System shown in Table 1

a_1	a_2	a_3	a_4	Utilization	$QoS(\tau_1)$	$QoS(\tau_2)$	$QoS(\tau_3)$	$QoS(\tau_4)$
4	9	24	3	1.0000	1.0000	1.0000	0.8944	0.7500
4	3	39	4	0.9778	1.0000	0.5230	1.0000	1.0000
2	9	39	4	1.0000	0.6250	1.0000	1.0000	1.0000
4	6	33	3	1.0000	1.0000	0.8770	0.9915	0.7500

Table 3: Example allowance assignments with corresponding achievable QoS and utilization

4.6. Imprecise Computations

J.-Y. CHUNG, J. W. S. LIU, K.-J. LIN: Scheduling Periodic Jobs That Allow Imprecise Results. IEEE Transact. on Computers, Vol. 39/9, 1990.

4.6.1. Ausgangspunkt

- **Motivation**

  „Akzeptable“ Qualität bei weichen Echtzeit-Anwendugen mit deterministischen Zeitschranken, insbesondere bei Überlast

- **Grundidee**

  Task aufteilen in Pflichtteil – Wahlteil.

  Pflichtteil muß stets Deadline einhalten, Wahlteil nicht.

  Quantifizierung durch eine Fehlerfunktion (Gütefunktion).

- **Taskklassen**

  N-Tasks: möglichst kleiner mittlerer Fehler

  C-Tasks: in bestimmten Abständen muß Deadline erreicht werden

4.6.2. Task-Modell

- **Allgemeine Voraussetzungen und Bezeichnungen**

$T = \{t_1,...,t_n\}$ periodische Tasks

$t_i = (t_{ij})_{j = 1,2,...}$ t_{ij} : j-ter Job von Task t_i, $i = 1,...,n$

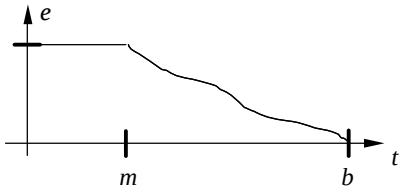
$t_i = (t_i, b_i)$ t_i: Periode
b_i: Ausführungszeit (konstant)

m_i: Mindestbearbeitungszeit für t_i

- **Fehlerfunktion und Einplanbarkeit**

e: für eine Task eingeplante Zeit

$e(e)$: Fehler bei $e < b$
monoton fallende Funktion



Exakter Ablaufplan: $e = b$

Ausführbarer Ablaufplan: $e \geq m$

- **Taskaufteilung**

$t_i = (t_i, b_i) \mapsto M_i = (t_i, m_i)$ Pflichtteil (mandatory part)
$O_i = (t_i, b_i - m_i)$ Wahlteil (optional part)

Gleiche Freigabezeit und Deadline wie t_i (Periodenanfang/-ende).

Pflichtteil muß stets Deadline erreichen, Wahlteil wird bei Periodenende abgebrochen.

4.6.3. Admission und Scheduling von N-Tasks

- **Admission**

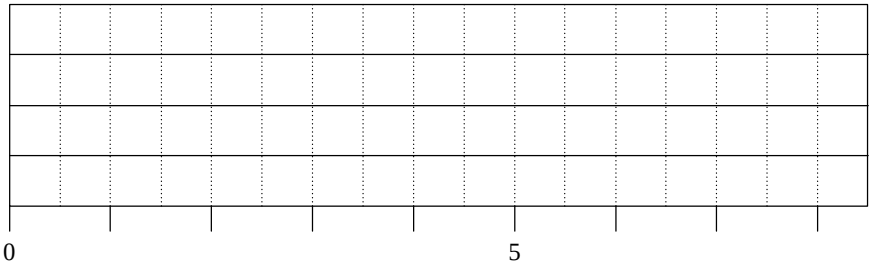
T wird zugelassen: $\sum_{i=1}^{n} \frac{m_i}{t_i} \leq n \cdot \left(\sqrt[n]{2} - 1\right)$

- **Scheduling von M-Jobs**

2 Prioritätsklassen: „M-Priorität > O-Priorität“ bzgl. ganz T
Scheduling von M-Jobs gemäß RMS.

Beispiel.

i	t_i	b_i	m_i	u_i	m_i/t_i	
1	2	1,0	0,5			
2	4	0,5	0,2			
3	5	0,5	0,1			
4	6	1,5	1,0			



- **Scheduling von O-Jobs**

LUF least utilization first $v_i = \frac{b_i - m_i}{t_i}$

Es gilt: Bei linearer Fehlerfunktion und gleichen Perioden führt LUF zu minimalem Fehler.

Aber: nicht optimal!

- **Dynamische Algorithmen für O-Jobs**

– least attained time first = shortest elapsed time first

– least slack time first slack = laxity

– earliest deadline

– best incremental

4.6.4. Admission und Scheduling von C-Tasks

- **Fehlerfunktion**

$$e(e) = \begin{cases} 1 & e \in [0, b) \\ 0 & e \geq b \end{cases}$$

- **Admission für $T = \{t_1,...,t_n\}$**

(1) $e_i \geq m_i$

(2) Für mindestens einen Job unter Q_i aufeinander folgenden Jobs von t_i ist $e_i \geq b_i$.

Q_i: Qualitätsparameter („cumulation rate“)

- **Einheitliche Perioden p und gleicher Qualitätsparameter Q**

Suche nach ausführbarem Ablaufplan äquivalent zu:

Ausführbaren Ablaufplan finden für die Wahlteile von T auf

Q Prozessoren mit gemeinsamer Deadline $p - \sum_{i=1}^{n} m_i$, ohne Entzug

$\rightarrow$ NP-vollständig!

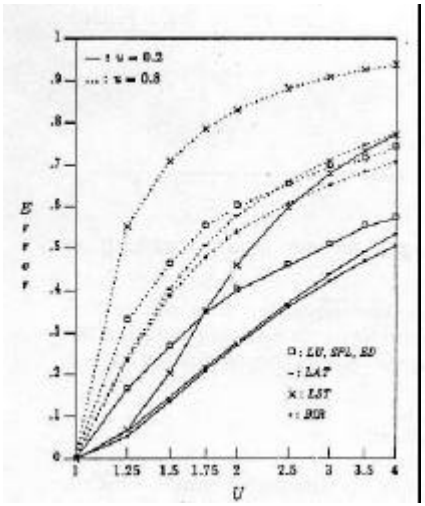
- **„Längenmonotoner Algorithmus“**

(1) Pflichtteile bearbeiten

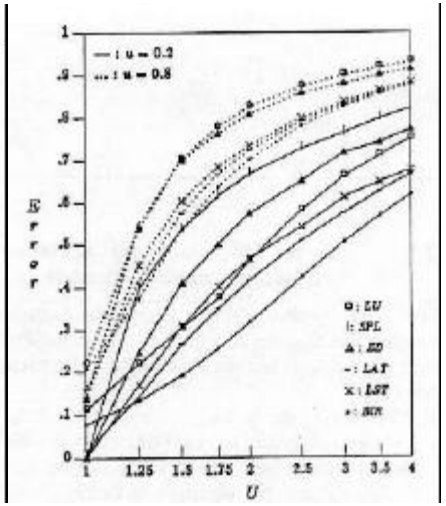
(2) Wahlteile fallend sortieren nach $b_i - m_i$

(3) Wahlteile in jeder Periode gemäß first-fit einplanen

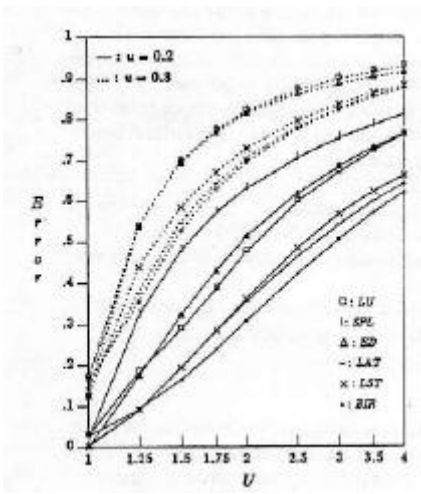
- **Simulationsergebnisse**



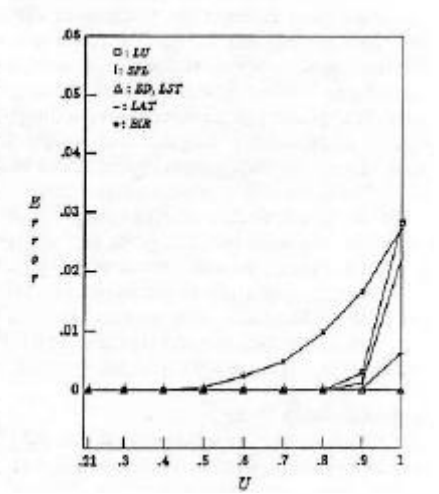
Identische Jobs



Gleichlange Perioden



Harmonische Perioden



Exakte Einplanbarkeit