Echtzeitbetriebssysteme

Oliver Jack

Ernst-Abbe-Hochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Sommersemester 2025

Ernst-Abbe-Hochschule Jena
University of Applied Sciences

Lerneinheit 7. Zeitgesteuertes Scheduling im Detail: Offline-Verfahren

Lerneinheit 7. Zeitgesteuertes Scheduling im Detail: Offline-Verfahren

Lernziele

- Kenntnis periodischer Taskmodelle
- Kenntnis von Offline-Planungsverfahren für periodische Prozesse

Lerneinheit 7. Zeitgesteuertes Scheduling im Detail: Offline-Verfahren

Task-Arten im periodischen Modell

Periodisch

periodische Aktivierung der Jobs einer Task, i. a. als Tupel $J_i(t_{\phi,i}, t_{p,i}, t_{e,i}, t_{d,i})$ notiert mit folgender Parameterbedeutung:

Parameter		Anmerkungen
Phase	t_ϕ	default: 0
Periode	t_p	
Ausführungszeit	t_e	(pro Aktivierung)
(relative) Deadline	t_d	default: t_p

Da meist $t_{\phi,i} = 0$ und $t_{d,i} = t_{p,i}$, wird deren Angabe häufig fortgelassen.

Task-Arten im periodischen Modell

aperiodisch

nicht-periodische Aktivierung der Jobs einer Task; die Zwischenankunftszeit zweier Jobs ist nicht nach unten beschränkt

sporadisch

nicht-periodische Aktivierung der Jobs einer Task; maximale Ankunftsrate der Jobs ist beschränkt

Auslastung

- Ermittlung für Task i:

$$u_i = \frac{t_{e,i}}{t_{p,i}}$$

- Gesamtauslastung u eines periodischen Tasksets mit n Tasks:

$$u = \sum_{i=1}^{n} u_i$$

- bei 1 Prozessor $u < 1$ nötig, sonst Überlast

Brauchbarkeit und Optimalität

Zur Wiederholung

Definition

Ein Plan (Schedule) einer Menge Jobs heißt brauchbar, wenn jeder Job aus dieser Menge vor seiner individuellen Deadline komplettiert ist.

Definition

Ein Schedulingalgorithmus heißt optimal, wenn der Algorithmus zu einer gegebenen Menge an Jobs einen brauchbaren Schedule erzeugt, sofern dieser existiert.

Das heißt, ein nicht-optimaler Algorithmus ist nicht immer in der Lage, einen brauchbaren Schedule zu erzeugen, obwohl dieser existiert.

Prinzip

- Grundidee: Ermittlung des Schedule Offline, zur Laufzeit des Systems wird der Schedule nur Position für Position abgearbeitet
- Periodisches Taskmodell nötig (sonst unendlicher Schedule, nicht speicherbar)
- Ermittlung des Schedule Offline, d. h. komplexe Algorithmen nutzbar (hier: Netzwerkflüsse)
- Konstante Anzahl periodischer Echtzeit-Tasks $T_i(t_{\phi,i}, t_{p,i}, t_{e,i}, t_{d,i})$, 1 Prozessor

Prinzip

- Schedule einer Menge aus n Tasks besteht aus aneinandergereihten Segmenten der Länge

$$H = \mathrm{kgV}(t_{p,i}) \; i = 1, 2, \ldots, n$$

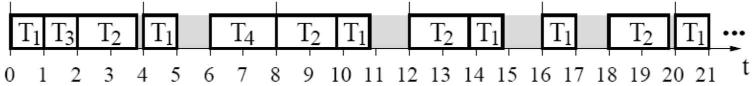
- H wird Hyperperiode genannt
- Tasks ohne Deadline konsumieren bei Bedarf verbleibende Zeit

Beispiel

- 4 periodische unabhängige Tasks $T_i(t_{p,i}, t_{e,i})$
  $T_1(4,1)$, $T_2(5,1.8)$, $T_3(20,1)$, $T_4(20,2)$.

$$u = \frac{1}{4} + \frac{1.8}{5} + \frac{1}{20} + \frac{2}{20} = 0.76, \;\; H = 20$$

- Brauchbarer Schedule



- Intervalle, in denen keine periodische Echtzeit-Task abgearbeitet wird (z.B. $[3.8, 4]$ oder $[5, 6]$) können nutzbar für aperiodische Jobs
- Aperiodische Jobs beim Eintreffen in Warteschlange (FIFO bzw. priorisiert) eingeordnet

Datenstruktur zur Beschreibung des Schedule

- Tabelle über eine Hyperperiode mit Einträgen

$$(t_k, T(t_k)) \; k = 0, 1, \ldots, n-1$$

t_k	Schedulingzeitpunkte
$T(t_k)$	assoziierte Tasks, bzw. "I" für idle

- Beispiel:

$$(0, T_1)(1, T_3)(2, T_2)(3.8, I)(4, T_1)(5, I)(6, T_4) \cdots (19.8, I)$$

Implementierung eines zeitgesteuerten Schedulers (Cyclic Executive)

Voraussetzungen: Schedule als Tabelle abgelegt; freier Timer

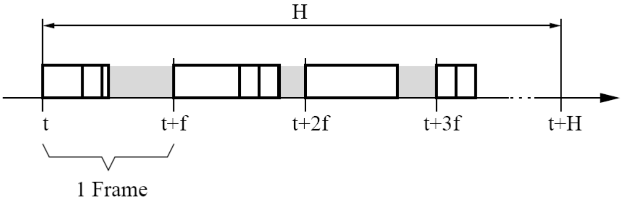
aktueller Tabelleneintrag $k = 0$
Timer konfigurieren, dass Interrupt zu t_0 ausgelöst wird
while(1) {
    Timerinterrupt bestätigen
    sichere Kontext des abgearbeiteten aperiodischen Jobs (falls nötig)
    aktuelle Task $T := T[k]$
    nächster Tabelleneintrag $k = ++kmod(N)$
    Timer auf nächsten Taskstart konfigurieren
    if $(T ==' I')$
        aperiodischen Job ausführen (falls vorhanden)
    else
        T ausführen
    sleep
}

ungünstig: Aktivierung des Schedulers nach jedem Job (hoher Overhead; besonders bei sehr kurzen Jobs)

Lerneinheit 7. Zeitgesteuertes Scheduling im Detail: Offline-Verfahren

Überlegungen

- günstig, die Hyperperiode (Länge H) in gleichlange Frames zu strukturieren
- Verringerung der Anzahl der Aktivierungen des Schedulers: nur noch am Anfang jedes Frames (timer-gesteuert)
- Vereinfachung: Timer besitzt konstante Periode (keine Reprogrammierung mehr nötig)
- innerhalb des Frames werden Jobs einfach nacheinander aufgerufen
- minimiert Anzahl vorzeitiger Beendigungen aperiodischer Tasks



Überlegungen zur Framegröße

- keine vorzeitige Beendigung (preemption) an Framegrenze, d. h. Framegröße f mindestens so groß wie größte Ausführungszeit eines Jobs

$$f \geq \max(t_{e,i}) \; \forall i : 1 \leq i \leq n \qquad (1)$$

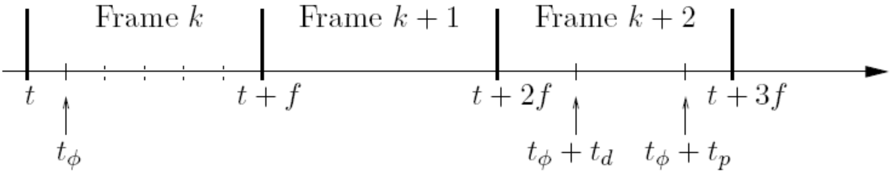
- Framegröße f ganzzahliger Teiler der Hyperperiode H, sonst Schedule u. U. sehr lang ($kgV(f, H)$).

$$f \mid H \qquad (2)$$

Überlegungen zur Framegröße (Forts.)

- Framegröße f so klein, dass zwischen t_ϕ und t_d jedes Jobs mindestens ein kompletter Frame liegt
- erleichtert Scheduler die Entscheidung, ob jeder Job seine Deadline einhält; Platzierung des Jobs in diesem Frame garantiert Einhaltung aller Zeitbedingungen, d. h. erleichtertes Scheduling.
- Ein Prozess werde in einem Frame k der Größe f zu t_ϕ bereit (mit $t \leq t_\phi \leq t + f$), besitze eine relative Deadline von t_d und eine (relative) Periode von t_p

Überlegungen zur Framegröße (Forts.)



Überlegungen zur Framegröße (Forts.)

- Damit wenigstens ein Frame zwischen t_ϕ und t_d liegt, muss gelten:

$$\begin{aligned} t + 2f &\leq t_\phi + t_d \\ 2f - (t_\phi - t) &\leq t_d \end{aligned}$$

- Wenn $t_\phi \neq t$, dann gilt für den Abstand von Grenze des Frames k und t_ϕ (also $t_\phi - t$):

$$2f - ggt(t_{p,i}, f) \leq t_{d,i} \; \forall i : 1 \leq i \leq n \qquad (3)$$

- Falls $t_\phi = t$, so muss die Framegröße f nur größer als die Deadline t_d sein. Dies gilt stets, wenn Gleichung 3 erfüllt ist.

Überlegungen zur Framegröße (Forts.)

- Die Gleichungen 1, 2 und 3 werden frame size constraints genannt.
- Ihre Einhaltung gestattet die Verwendung leistungsfähiger Schedulingverfahren und einer verbesserten Cyclic Executive.

Beispiel

- 3 periodische Tasks T_i mit

i	$t_{p,i}$	$t_{d,i}$	$t_{e,i}$
1	15	14	1
2	20	26	2
3	22	22	3

- $H = kgV(15, 20, 22) = 660$
  1. $f \geq 3$
  2. $f \in \{2, 3, 4, 5, 6, 10, 11, 12, 15, 20, 22, \ldots\}$
  3. $f \leq 6$
- $\Rightarrow f \in \{3, 4, 5, 6\}$

Eine bessere Cyclic Executive

Prinzip:

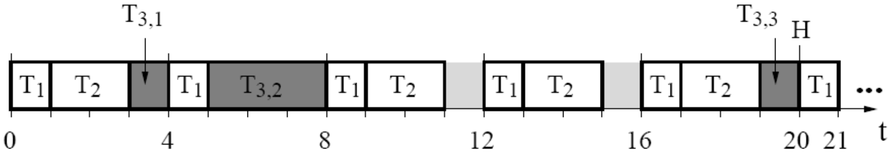
- zyklische Aktivierung an Framegrenze durch Timer
- Schedule besteht aus k Frames (1 Hyperperiode) mit entsprechenden Jobs (jeweils unterschiedliche Anzahl und Länge)
- Hintereinanderausführung aller Jobs des Frames ohne zwischenzeitliche Aktivierung des Schedulers
- Vorteil: geringere Aufrufruffrequenz der Cyclic Executive als in Variante 1 (Overhead geringer)
- etwaiges überlaufen eines Jobs an einer Framegrenze muss detektiert und behandelt werden

Lerneinheit 7. Zeitgesteuertes Scheduling im Detail: Offline-Verfahren

Beispiel

- $T_1(4,1)\,T_2(5,2,7)\,T_3(20,5)$
- $H = 20$, aber 1.) $f \geq 5$ und 3.) $f \leq 4$. Was tun?
- Idee: Partitionierung langer Tasks in Subtasks, um Kriterium 1 leichter erfüllen zu können.
- Nachteil: mehr Kontextwechsel nötig, da dekomponierte Task vorzeitige Beendigungen benötigt
- Dekomposition von $T_3(20,5)$ in $T_{3,1}(20,1)$, $T_{3,2}(20,3)$ und $T_{3,3}(20,1)$; damit:
  1. $f \geq 3$
  2. $f \in \{2,4,5,10,20\}$
  3. $f \leq 4$
- $\Rightarrow f = 4$

Struktur eines Schedule mit Jobslicing

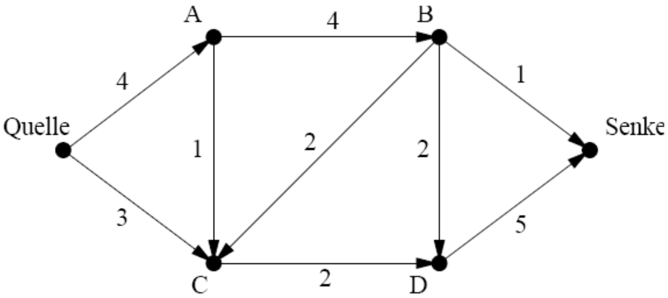


Lerneinheit 7. Zeitgesteuertes Scheduling im Detail: Offline-Verfahren

Flüsse in Netzwerken

- Eingabe ist Problembeschreibung in Form eines gerichteten Graphen $G = (V, E)$; V : Knotenmenge, E: Kantenmenge
- jeweils ein Knoten für Quelle Q und Senke S
- jede Kante $e \in E$ besitzt eine Kapazität $c(e)$ und einen aktuellen Fluss $\varphi(e)$

Netzwerk (noch ohne Fluss)



Flüsse in Netzwerken (Forts.)

- für alle Knoten außer Q und S muss die Summe der Zuflüsse gleich der Summe der Abflüsse sein ("Kirchhoff-Regel")
- Gesucht ist der maximale Durchfluss zwischen Quelle und Senke unter Beachtung der Kantenkapazität
- es existieren schnelle Lösungsalgorithmen mit polynomialem Aufwand
- (u.a.) für viele Scheduling-Probleme nutzbar

Lerneinheit 7. Zeitgesteuertes Scheduling im Detail: Offline-Verfahren

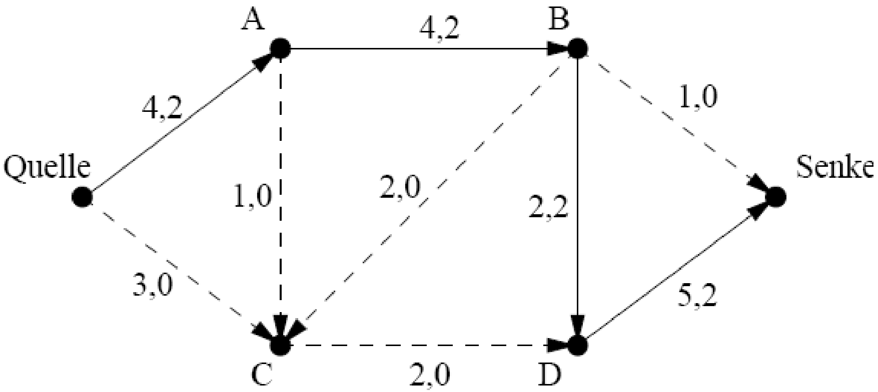
Algorithmus von Ford und Fulkerson

Benötigt zu einem Netzwerk $G = (V, E)$ und einem Fluss darin einen "Restgraphen" $RG = (V, E_\varphi)$ mit der gleichen Knotenmenge und den Kanten:

- $\forall e \in E$ mit $\varphi(e) < c(e)$ gibt es eine Kante e in E_φ mit $c_\varphi(e) = c(e) - \varphi(e)$
- $\forall e \in E$ mit $\varphi(e) > 0$ gibt es eine umgekehrte Kante e' in E_φ mit $\varphi(e') = \varphi(e)$

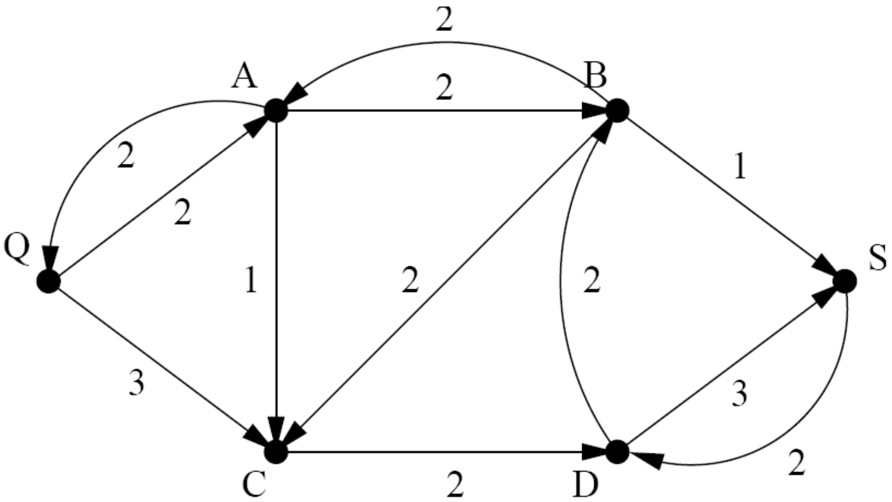
Beispiel

Fluss in Netzwerk, Kantenbewertung: $(c(e), \varphi(e))$



Beispiel (Forts.)

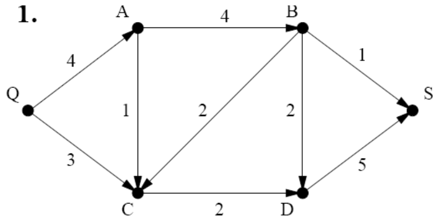
Zugehöriger Restgraph

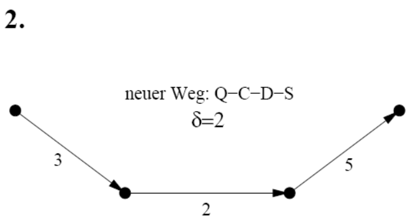


Algorithmus

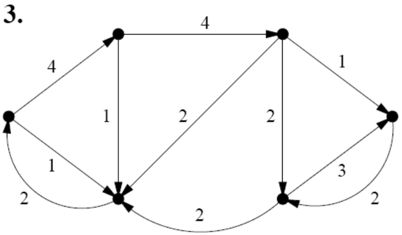
1. START, initialer Fluss $\varphi = 0$.
2. Markiere Q.
3. Markiere weitere Knoten nach folgender Regel: “Wenn x markiert ist, und es gibt im RG eine Kante $e = (x, y)$, dann markiere y, wenn dieser noch unmarkiert ist.”
4. S markiert? nein: Maximaler Fluss erreicht; END.
5. Erhöhung des aktuellen Flusses durch Adaption des RG:
   Kanten, die auf dem Weg von Q nach S liegen, seien e_i. Ermittle $\delta = \min(c_\varphi(e_i))$. Adaption des Flusses entlang des neuen Weges:
   - wenn $e_i \in E$, dann $\varphi(e_i) := \varphi(e_i) + \delta$
   - wenn $rev(e_i) \in E$, dann $\varphi(e_i) := \varphi(e_i) - \delta$
6. Aufhebung aller Markierungen, GOTO 2.

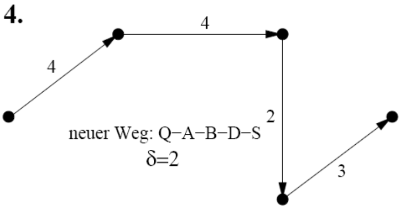
Beispiel



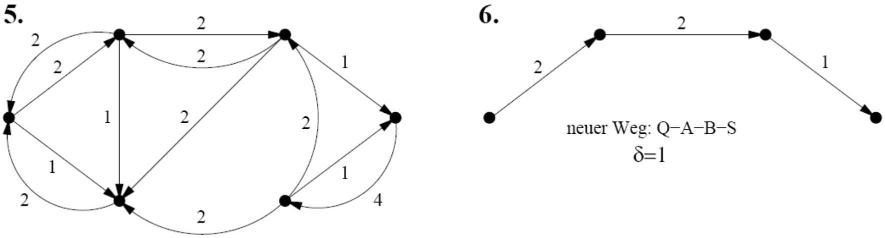


Beispiel (Forts.)

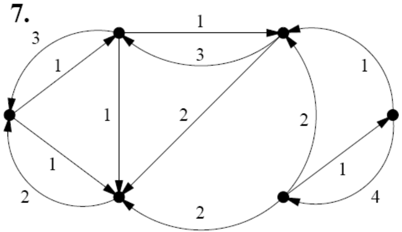




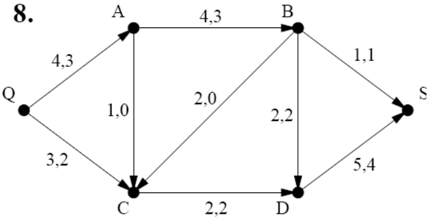
Beispiel (Forts.)



Beispiel (Forts.)



$\Rightarrow$



Kein Weg mehr, d. h. maximaler Fluss. $\varphi = 5$.

Lerneinheit 7. Zeitgesteuertes Scheduling im Detail: Offline-Verfahren

Scheduling-Problems als Fluss im Netzwerk

- Annahmen: Preemption immer möglich, Tasks unabhängig, alle möglichen Framegrößen f ermittelt
- Idee: beginnend bei Maximum alle f testen
- Für jeden Job und jedes Frame einen Knoten, zusätzlich Quelle und Senke
- Von der Quelle zu jedem Job J_i eine Kante mit Kapazität $t_{e,i}$
- Von jedem Frame zur Senke eine Kante mit Kapazität f (Framegröße)

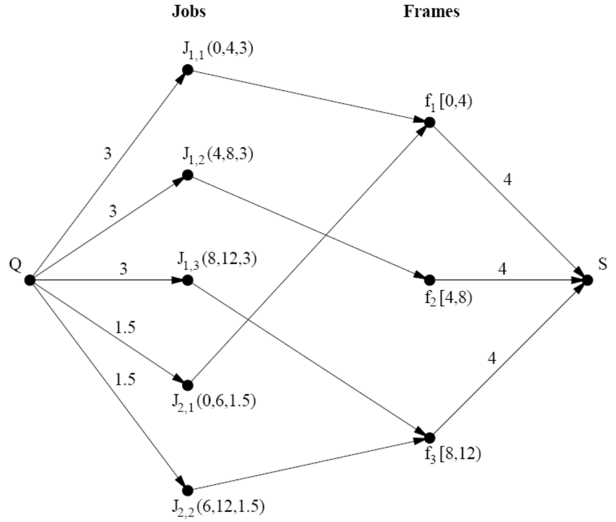
Scheduling-Problems als Fluss im Netzwerk (Forts.)

- Von Job zu Frame eine Kante gdw. Job in Frame abgearbeitet werden kann (Timingconstraints!), Kapazität = Maximum aus Ausgangs- und Zielknoten
  $\rightarrow$ Ein Job J_i kann nur in den Frames geplant werden, deren Startzeit größer oder gleich $t_{\phi,i}$ ist und die nicht später enden als $t_{d,i}$!
- Ermittlung des maximalen Flusses von Jobs zu Frames mittels geeignetem Algorithmus
- Ist der maximale Fluss durch den Graphen größer als $\sum_i t_{e,i}$, dann repräsentiert der Graph einen brauchbaren Schedule.

Beispiel

- 2 periodische Tasks: $T_1(4, 3)$ und $T_2(6, 1.5)$
- $H = 12$, $f = 4$, d. h. 3 Frames in Hyperperiode:
  $f_1[0, 4)$, $f_2[4, 8)$, $f_3[8, 12)$
- $T_1 \rightarrow J_{1,1}(0, 4, 3), J_{1,2}(4, 8, 3), J_{1,3}(8, 12, 3)$
- $T2 \rightarrow !J_{2,1}(0, 6, 1.5), J_{2,2}(6, 12, 1.5)$

Beispiel: zugehöriges Netzwerk



- Wie leicht ermittelbar, ist $\varphi = 11 < \sum_i t_{e,i} = 12$.
- Ein Schedule ist damit für $f = 4$ nicht möglich.

Verbesserung: Jobslicing

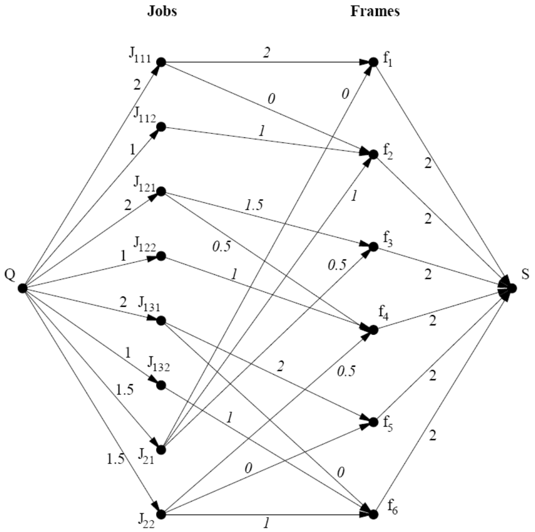
Verkleinerung des Frames auf $f = 2$; damit erhält man

- 6 Frames: $f_1[0,2)$, $f_2[2,4)$, $f_3[4,6)$, $f_4[6,8)$, $f_5[8,10)$, $f_6[10,12)$
- Dekomposition der Jobs von T_1:

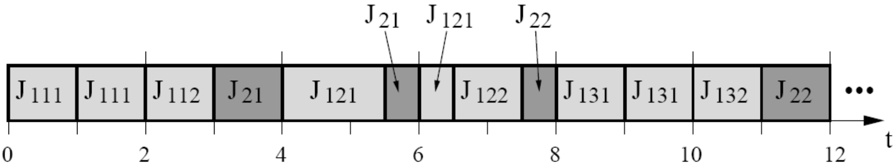
alt	$J_{1,1}(0,4,3)$	$J_{1,2}(4,8,3)$	$J_{1,3}(8,12,3)$
neu	$J_{111}(0,4,2)$	$J_{121}(4,8,2)$	$J_{131}(8,12,2)$
	$J_{112}(2,4,1)$	$J_{122}(6,8,1)$	$J_{132}(10,12,1)$

- $J_{2,1}$ und $J_{2,2}$ bleiben unverändert

Zugehöriges Netzwerk mit gültigem Fluss für Framegröße $f = 2$



Zugehöriger Schedule (optimal)



Bemerkungen

- Die Frage, ob Online- oder Offline-Scheduling geeigneter für Echtzeitsysteme ist, wird in der Fachwelt kontrovers diskutiert und ist nicht einfach zu beantworten.
- Für Online-Scheduling sprechen einerseits:
  - Offline Verfahren sind inhärent inflexibel (komplette Neuberechnung nötig bei Modifikationen an der Taskmenge).
  - Es existieren Online-Verfahren mit sehr geringem Schedulingoverhead (Rate-momtonic Scheduling, Deadline-monotonic Scheduling).
- Andererseits sind Online-Verfahren wie RMS und DMS nicht generell optimal, das bedeutet, es gibt Taskmengen, die per Offline-Verfahren planbar sind, per Online-Verfahren jedoch nicht geplant werden können.

Lerneinheit 7. Zeitgesteuertes Scheduling im Detail: Offline-Verfahren

Zusammenfassung

- Das periodische Taskmodell beinhaltet periodische, aperiodische und sporadische Tasks.
- Offline-Planungsverfahren können aufwendige Algorithmen zur Optimierung nutzen.
- Scheduling-Probleme für periodische Tasks können als Flussproblem in Netzwerken interpretiert werden und die einschlägigen Optimierungsverfahren können zur Lösung genutzt werden.