

# Lezione R5

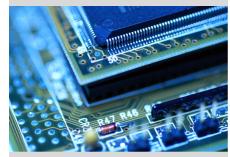
## Ottimalità di algoritmi priority-driven

Sistemi embedded e real-time

9 ottobre 2020

Ottimalità di  
algoritmi  
priority-driven

Marco Cesati



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

Marco Cesati

Dipartimento di Ingegneria Civile e Ingegneria Informatica  
Università degli Studi di Roma Tor Vergata

SERT'20

R5.1

### Di cosa parliamo in questa lezione?

In questa lezione si discute l'ottimalità o meno degli algoritmi di schedulazione priority-driven

Ottimalità di  
algoritmi  
priority-driven

Marco Cesati



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

- ➊ Il problema della validazione
- ➋ Il fattore di utilizzazione
- ➌ Il test di schedulabilità

SERT'20

R5.2

## Il problema della validazione

Ottimalità di algoritmi priority-driven

Marco Cesati

Gli algoritmi priority-driven in generale:

- sono semplici da implementare
- sono flessibili, non devo conoscere tutti i parametri.
- non richiedono necessariamente di conoscere esattamente il modello di carico
- è non banale dimostrare formalmente che i vincoli temporali dei job hard real-time saranno sempre rispettati, soprattutto se i parametri temporali non sono ben precisati

### Il problema della validazione

Dati un insieme di job, processori e risorse utilizzabili dai job, e l'algoritmo di schedulazione e accesso alle risorse, determinare se tutti i job rispetteranno i vincoli temporali

SERT'20

R5.3

## Validazione di algoritmi priority-driven

Ottimalità di algoritmi priority-driven

Marco Cesati

Per gli algoritmi priority-driven il problema della validazione è difficile da risolvere a causa delle *anomalie di schedulazione* (o *Richard's anomalies*, Graham 1976)

Sono comportamenti **temporali inattesi** che si verificano anche in sistemi semplici

Ad esempio, in un sistema con job **non interrompibili** il tempo di risposta dell'ultimo job che termina (*makespan*) può peggiorare se:

- Si aumenta il periodo (diminuisce la frequenza) di un job
- Si riduce il tempo di esecuzione di un job
- Si riducono le dipendenze tra i job
- Si aumenta la velocità del processore (Buttazzo 2006)

sono inattesi, non mi aspetto che aumentare la velocità di cpu peggiori il tempo di risposta dell'ultimo job.

*Perché le anomalie complicano il problema della validazione?*

Se i parametri dei job di un sistema possono variare, non si può validare il sistema esaminando solo il “caso peggiore”: è necessario esaminare tutte le combinazioni di parametri

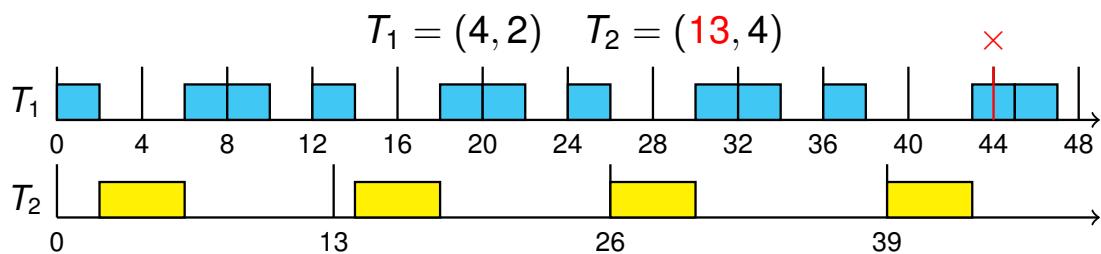
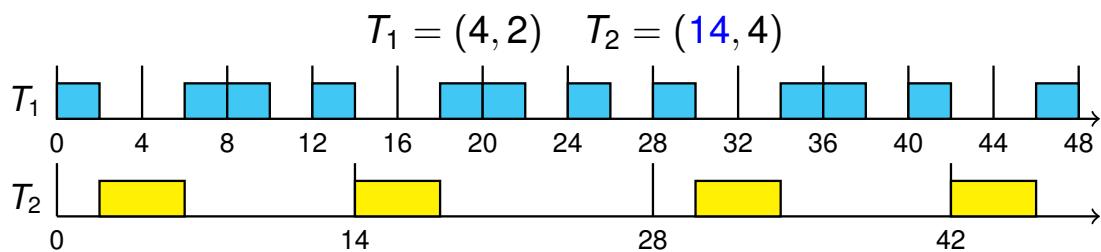
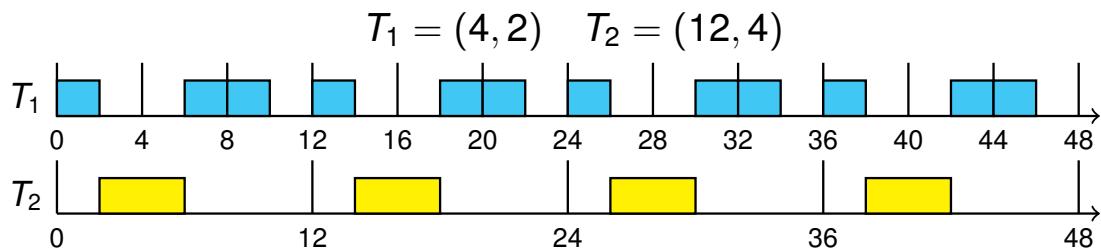
SERT'20

R5.4

## Esempio di anomalia di schedulazione (periodo)

(può accadere anche se interrompibili)

Due task **non** interrompibili schedulati con RM su 1 processore



Non esiste un caso peggiore, con  $T_2$  avente periodo 12 andava, con 14 anche, con 13 no.

Ottimalità di algoritmi  
priority-driven

Marco Cesati



Schema della lezione

Validazione

Fattore di utilizzazione

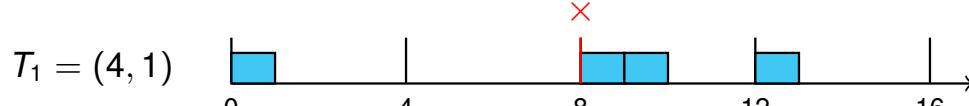
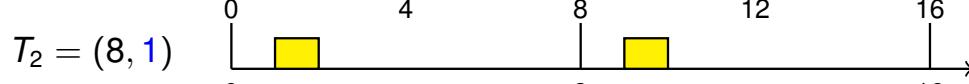
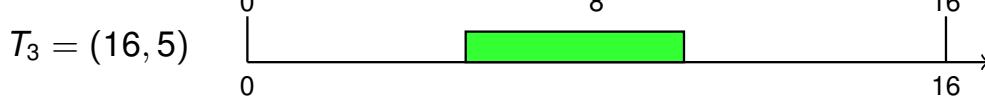
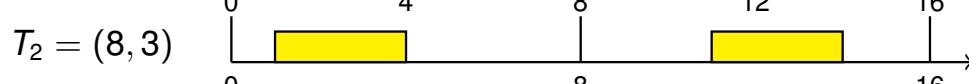
Test di schedulabilità

SERT'20

R5.5

## Esempio di anomalia di schedulazione (tempo d'esecuzione)

Tre task **non** interrompibili schedulati con RM su 1 processore



Ottimalità di algoritmi  
priority-driven

Marco Cesati



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

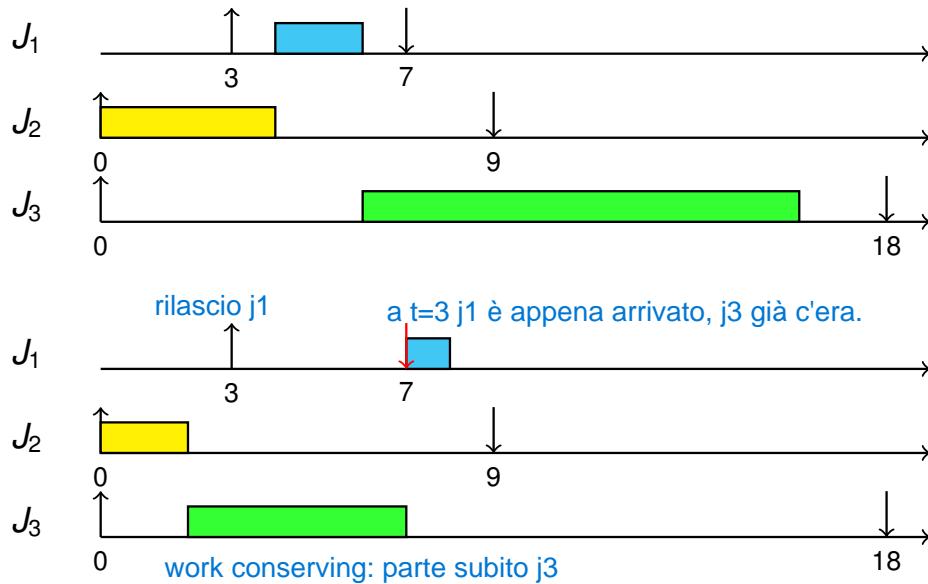
SERT'20

R5.6

Nell'ultimo gruppo vediamo che la problematica nasce da  $T_3$ , in quanto RM è work conservative.

## Esempio di anomalia di schedulazione (velocità processore)

Tre job **non** interrompibili schedulati con EDF su un processore



Una anomalia analoga si verifica con due job interrompibili che accedono ad una risorsa condivisa (Buttazzo 2006)

## Esecuzione predibile

Fissato un algoritmo, la schedulazione prodotta considerando i **tempi d'esecuzione massimi** (minimi) per tutti i job è detta **schedulazione massima (minima)**

L'esecuzione di un job è **predicibile** se è sempre entro i limiti temporali stabiliti dalle schedulazioni minima e massima

- siano  $\sigma^-$  e  $\epsilon^-$  gli **istanti di attivazione** e completamento di un job nella schedulazione minima
- siano  $\sigma^+$  e  $\epsilon^+$  gli istanti di attivazione e completamento dello stesso job nella schedulazione massima
- il job ha esecuzione **predicibile** se l'istante di attivazione è sempre in  $[\sigma^-, \sigma^+]$  e l'istante di completamento è sempre in  $[\epsilon^-, \epsilon^+]$

Fissato un algoritmo, un insieme di job è **predicibile** se lo è l'esecuzione di ciascuno dei suoi job quindi per ogni job nel task

## Predicibilità per gli algoritmi priority-driven

Ottimalità di  
algoritmi  
priority-driven

Marco Cesati

### Teorema (Ha & Liu, 1993)

Un insieme di job interrompibili, indipendenti, e con istanti di rilascio fissati schedulato su un processore da un algoritmo priority-driven è predicibile



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

*Qual è il vantaggio di lavorare con insiemi predicibili?*

Il processo di validazione è facile perché possiamo verificare solo il caso della **schedulazione massima**

*Come applicare il teorema a sistemi con più processori?*

Legando l'esecuzione di ciascun job ad un singolo processore (sistema **statico**)

Anche se ciò implica violare l'indipendenza, in quanto ci sarà sempre qualcosa in comune. Sono casi più delicati.

SERT'20

R5.9

## Fattore di utilizzazione

Ottimalità di  
algoritmi  
priority-driven

Marco Cesati

- Algoritmi come **FIFO** e **LIFO** non considerano l'urgenza dei job: nei sistemi real-time hanno prestazioni pessime
- Algoritmi **fixed-priority** con priorità associate alla importanza relativa dei task hanno prestazioni cattive
- Gli algoritmi migliori sono quelli che assegnano la priorità in base a parametri temporali **oggettivi**.



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

Come valutare le prestazioni degli algoritmi di schedulazione basati su parametri temporali?

Fissato un algoritmo di schedulazione **X**, il suo **fattore di utilizzazione** (o **schedulable utilization**) è un valore  $U_X \in [0, 1]$  tale che l'algoritmo può determinare una schedulazione fattibile per qualunque insieme di task periodici su un processore se l'utilizzazione totale dei task è minore o uguale ad  $U_X$

SEMPRE

Tanto maggiore è  $U_X$ , tanto migliore è l'algoritmo Il caso migliore è  $U_X = 1$

SERT'20

R5.10

## Confronto tra algoritmi di schedulazione (2)

Qual è il fattore di utilizzazione dell'algoritmo FIFO? **Zero!**

Esiste un insieme di due task con fattore di utilizzazione pari a  $\epsilon > 0$  piccolo a piacere che non è schedulabile con FIFO:

$$T_1 = (10, 5\epsilon), T_2 = (20/\epsilon, 10) \quad \text{Utilizzazione: } 5\epsilon/10 + 10\epsilon/20 = \epsilon \text{ piccolo, però se provo a mano manca le scadenze. Quindi } U_{\text{FIFO}} < \epsilon, \text{ quindi assumo 0.}$$

Qual è il fattore di utilizzazione dell'algoritmo EDF? **Uno!**

Ma non dovremmo dimostrarlo?! (Sappiamo solo che EDF è ottimale per job interrompibili ed indipendenti...)

L'algoritmo EDF è semplice e ottimale, perché dovremmo studiare/adottare/cercare altri algoritmi?

è poco predicibile, ad esempio con imprevisti.

- Non esiste un modo efficiente per determinare quali job saranno in ritardo in caso di sovraccarico o overrun in una schedulazione a priorità dinamica quale EDF
- Qual è la priorità EDF di un job in ritardo?
- Il comportamento di un algoritmo a priorità fissa è predicibile anche in caso di sovraccarico o overrun

Ottimalità di algoritmi priority-driven

Marco Cesati



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

SERT'20

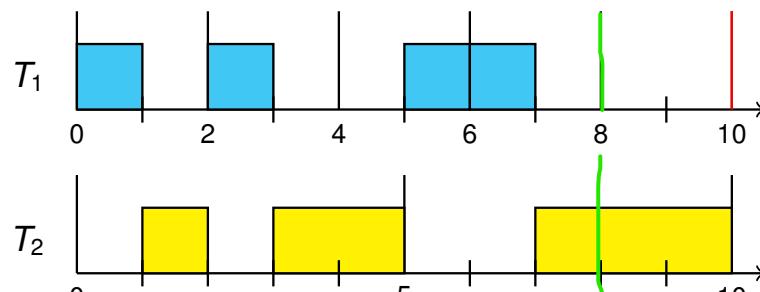
R5.11

## Comportamento di EDF con sovraccarico

$$T_1 = (2, 1) \\ T_2 = (5, 3) \\ (\mathbf{U} = 1.1)$$

$$1/2 + 3/5 = 1.1 > 1$$

Quindi manca una scadenza.



Ottimalità di algoritmi priority-driven

Marco Cesati



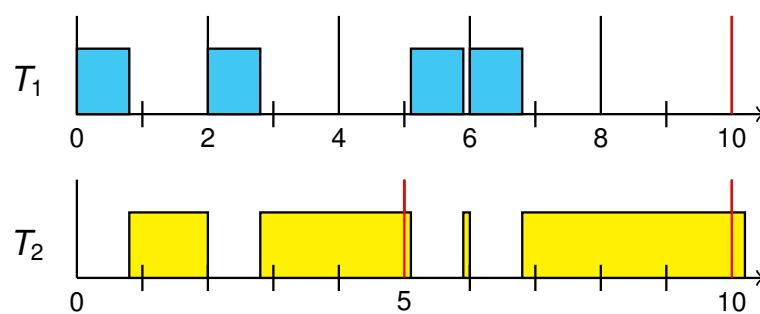
Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

$$T_1 = (2, 0.8) \\ T_2 = (5, 3.5) \\ (\mathbf{U} = 1.1)$$



in t = 5 T2 manca la propria scadenza, tra 8 e 10 ci ritroviamo nello stesso caso di sopra, dipende da chi faccio eseguire.

SERT'20

R5.12

# Fattore di utilizzazione di EDF

**Teorema (Liu & Layland, 1973)** domanda d'esame.

Un sistema  $\mathcal{T}$  di task indipendenti ed interrompibili con scadenze relative uguali ai rispettivi periodi e fattore di utilizzazione  $U_{\mathcal{T}}$  ha una schedulazione fattibile su un singolo processore se e solo se  $U_{\mathcal{T}} \leq 1$

Sotto tali condizioni, se utilizzazione dei task è  $\leq 100\%$ , esiste una schedulazione che rispetta le scadenze.

**Corollario** sappiamo che se esiste schedulazione che rispetta le scadenze, anche EDF mi permette di trovare una schedulazione che rispetta le scadenze.

L'algoritmo EDF ha fattore di utilizzazione  $U_{EDF} = 1$  per sistemi di task indipendenti, interrompibili e con scadenze relative uguali o maggiori dei rispettivi periodi

**Dim. del Teorema** (sketch):

- La parte “solo se” è banale
- Per la parte “se”: troviamo un algoritmo che produce una schedulazione fattibile di ogni sistema  $\mathcal{T}$  con  $U_{\mathcal{T}} \leq 1$
- Candidati? EDF!

Ottimalità di algoritmi priority-driven

Marco Cesati



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

SERT'20

R5.13

L'obiettivo non è tanto dimostrare che EDF ha utilizzazione 1, bensì che se esiste schedulazione, la trovo anche con EDF.

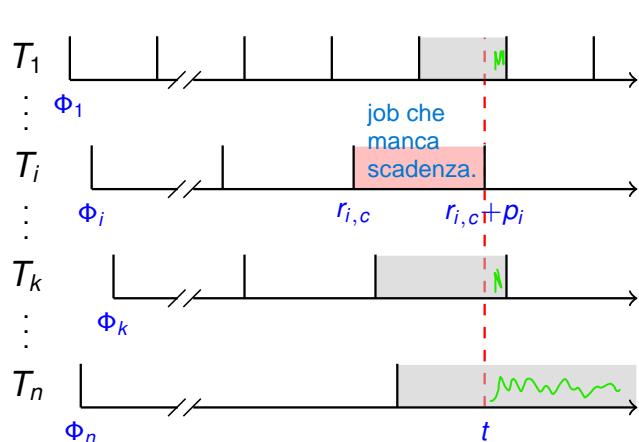
## Fattore di utilizzazione di EDF (2)

Da dimostrare: se EDF non trova una schedulazione fattibile, allora  $U_{\mathcal{T}} > 1$  allora esiste un job che non rispetta la scadenza.

Al tempo  $t$  il (primo) job  $J_{i,c}$  non completa entro la scadenza

Assumiamo che il processore non sia mai idle prima di  $t$

**1° caso:** i periodi che includono  $t$  iniziano sempre dopo  $r_{i,c}$



\_schedulando con EDF, le scadenze nei periodi grigi, venendo tutti dopo, hanno priorità inferiore a quello che ha mancato la scadenza. Devono ancora partire.

Tutti i job nei periodi che includono  $t$  non sono eseguiti prima di  $t$  perché hanno scadenze dopo  $J_{i,c}$

$$t - \Phi_i < \frac{(t - \Phi_i)e_i}{p_i} + \sum_{k \neq i} \left\lceil \frac{t - \Phi_k}{p_k} \right\rceil e_k \leq t \frac{e_i}{p_i} + \sum_{k \neq i} t \frac{e_k}{p_k} = t U_{\mathcal{T}} \Rightarrow U_{\mathcal{T}} > 1$$

tempo dall'inizio al mancamento della scadenza.

n° di periodi \* tempo esecuzione

t-0 < tutto il lavoro DA FARE

Ottimalità di algoritmi priority-driven

Marco Cesati



Schema della lezione

Validazione

Fattore di utilizzazione

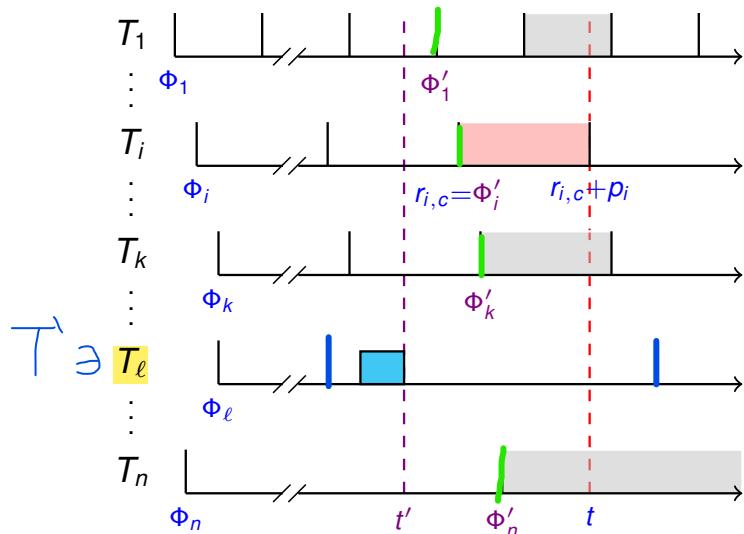
Test di schedulabilità

SERT'20

R5.14

## Fattore di utilizzazione di EDF (3)

**2° caso:** l'insieme dei task  $\mathcal{T}'$  in cui il periodo che include  $t$  inizia prima di  $r_{i,c}$  è non vuoto



Task in  $\mathcal{T}'$  possono essere eseguiti nel periodo che include  $t$  prima di  $r_{i,c}$

Sia  $t'$  l'ultimo istante di esecuzione dei task in  $\mathcal{T}'$  prima di  $t$

Per ogni task in  $\mathcal{T} \setminus \mathcal{T}'$ ,  $\Phi'_k$  è l'istante di rilascio del primo job in  $[t', t]$

$$\begin{aligned} t - t' &< \frac{(t - \Phi'_i) e_i}{p_i} + \sum_{\substack{T_k \in \mathcal{T} \setminus \mathcal{T}' \\ k \neq i}} \left\lfloor \frac{t - \Phi'_k}{p_k} \right\rfloor e_k \leq (t - t') \sum_{T_k \in \mathcal{T} \setminus \mathcal{T}'} \frac{e_k}{p_k} \\ &\leq (t - t') U_{\mathcal{T}} \Rightarrow U_{\mathcal{T}} > 1 \end{aligned}$$

non è sufficiente per fare tutto il lavoro.

Se EDF manca la scadenza, non riesce quindi a schedulare, è perchè la somma totale dell'utilizzazione dei task è maggiore di 1. Allora ho dimostrato anche il viceversa.

## Riassumendo...

Si assumano task indipendenti, interrompibili e schedulati su un singolo processore

Abbiamo stabilito che:

- Se un sistema di task ammette una schedulazione fattibile, l'algoritmo **EDF** ottiene una schedulazione fattibile per quel sistema di task
- Un sistema di task ammette una schedulazione fattibile se e solo se  $U_{\mathcal{T}} \leq 1$
- L'algoritmo **EDF** ha fattore di utilizzazione  $U_{\text{EDF}} = 1$  (almeno per scadenze uguali ai periodi) implicate.

Tra poco dimostreremo che l'algoritmo **EDF** ha fattore di utilizzazione  $U_{\text{EDF}} = 1$  anche quando le scadenze sono uguali o maggiori dei periodi



## Densità di un sistema di task

Il teorema appena dimostrato non è valido se qualche task ha scadenza relativa inferiore al periodo; ad esempio:

- $T_1 = (2, 0.9), T_2 = (5, 2.3) \Rightarrow U = 0.91 \Rightarrow$  schedulabile
- $T_1 = (2, 0.9), T_2 = (5, 2.3, 3) \Rightarrow$  non schedulabile ( $\Delta = 1.22$ )

Si definisce **densità** di un task  $(\Phi, p, e, D)$  il rapporto  $\frac{e}{\min(D, p)}$

### Teorema

Un sistema  $\mathcal{T}$  di task indipendenti ed interrompibili e densità  $\Delta_{\mathcal{T}}$  ha una schedulazione fattibile su un singolo processore se  $\Delta_{\mathcal{T}} \leq 1$

- È condizione sufficiente, non necessaria
- $T_1 = (2, 0.6, 1), T_2 = (5, 2.3) \Rightarrow \Delta = 1.06$  ma è schedulabile!

Quindi, se densità è  $\leq 1$  sono certo di schedularlo,  
se densità  $> 1$  dovrei comunque provare per vedere se è schedulabile.

Ottimalità di algoritmi priority-driven

Marco Cesati



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

SERT'20

R5.17

## Fattore di utilizzazione e densità

In un sistema  $\mathcal{T}$  di task interrompibili con fattore di utilizzazione  $U_{\mathcal{T}} = \sum_k e_k / p_k$  ed un singolo processore:

- A1) Se, per ogni task  $T_i$ ,  $D_i = p_i$ , allora esiste una schedulazione fattibile se e solo se  $U_{\mathcal{T}} \leq 1$
- A2) Se, per ogni task  $T_i$ ,  $D_i \geq p_i$ , allora esiste una schedulazione fattibile se e solo se  $U_{\mathcal{T}} \leq 1$
- A3) Il fattore di utilizzazione di EDF per task con  $D_i \geq p_i$  è  $U_{\text{EDF}} = 1$  (ossia EDF determina una schedulazione fattibile se  $U_{\mathcal{T}} \leq 1$ )
- A4) Se per qualche task  $T_i$ ,  $D_i < p_i$ , allora esiste una schedulazione fattibile se  $\Delta_{\mathcal{T}} = \sum_k e_k / \min(p_k, D_k) \leq 1$

Ottimalità di algoritmi priority-driven

Marco Cesati



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

Quando si verifica il caso  $U_{\mathcal{T}} > \Delta_{\mathcal{T}}$ ? **Mai!**

- Se  $\exists T_i$  tale che  $D_i < p_i$ , allora  $U_{\mathcal{T}} < \Delta_{\mathcal{T}}$  basta confrontare i denominatori
- Se  $\forall T_i$ ,  $D_i \geq p_i$ , allora  $U_{\mathcal{T}} = \Delta_{\mathcal{T}}$  perchè nella densità avrò al denominatore  $p(i)$ , ma allora coincide con la definizione della formula di  $U(\mathcal{T})$

SERT'20

R5.18

## Fattore di utilizzazione e scadenze oltre i periodi

Dimostriamo che: se per ogni task  $D_i \geq p_i$ , allora esiste una schedulazione fattibile solo se  $U_T \leq 1$

- Per un solo task (base dell'induzione):

$$e \leq D, \quad 2e \leq D + p \quad \dots \quad (k+1)e \leq D + kp \quad \dots$$

tempo esecuzione job < scadenza assoluta

$$\text{Quindi } \forall k \geq 1, \quad \frac{e}{p} < \frac{k+1}{k} \cdot \frac{e}{p} \leq 1 + \frac{D}{pk} \implies \frac{e}{p} \leq 1$$

- Sia vero per  $n-1$  task in fase, allora per  $T_n$  e ogni  $k$  intero:

$$(k+1)e_n \leq (D_n + kp_n) \cdot \left(1 - \sum_{i=1}^{n-1} \frac{e_i}{p_i}\right)$$

ho gli  $n-1$  task da eseguire, perchè  
il processore non è “tutto per me”,  
pari a 1- utilizzo altri job.

$$\text{tempo necessario} \leq \text{scadenza assoluta} * (\text{tempo che mi viene lasciato da altri task})$$
$$\forall k \geq 1, \quad \frac{e_n}{p_n} < \left(\frac{D_n}{kp_n} + 1\right) \cdot \left(1 - \sum_{i=1}^{n-1} \frac{e_i}{p_i}\right) \implies U_T \leq 1$$

- Per task non in fase: stessa idea applicata dall'istante della fase maggiore



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

SERT'20

R5.19

## Test di schedulabilità per EDF

Dato un sistema  $T$  completamente definito di task interrompibili, come stabilire se è schedulabile con EDF?

- Se  $\Delta_T \leq 1$  è schedulabile (T2 e C1)
- Se  $\Delta_T > 1$ : se  $D_i \geq p_i$  per ogni  $i$  non è schedulabile (C1)
- Se  $\Delta_T > 1$  e  $D_i < p_i$ : se i task sono in fase applichiamo EDF per un segmento lungo  $2H + \max p_i + \max D_i$ , ove  $H$  è l'iperperiodo (Baruah, Howell, Rosier 1993)  
posso solo simulare se ho tutte le info.

E se il sistema non è completamente determinato? Ad esempio, i tempi di esecuzione o gli istanti di rilascio possono variare



Schema della lezione

Validazione

Fattore di utilizzazione

Test di schedulabilità

Il sistema continua ad essere schedulabile anche quando:

- i tempi di esecuzione sono più corti dei tempi di esecuzione massimi (sistema predicibile)
- i task sono sporadici, ossia gli intervalli di rilascio dei job sono maggiori dei rispettivi periodi (dalla dimostrazione del teorema)

... ma se le fasi sono sconosciute non si può simulare!

SERT'20

R5.20

## Analisi di schedulabilità per EDF

Ottimalità di algoritmi priority-driven

Marco Cesati



Se alcuni task hanno scadenze relative inferiori ai periodi, la condizione di schedulabilità è sufficiente ma non necessaria

cioè quello che abbiamo visto prima, con le varie condizioni, è sufficiente ma in realtà non servirebbe. Basta applicare questo teorema:

### Teorema (Baruah & al., 1990)

necessaria e sufficiente

Un sistema  $\mathcal{T}$  con task periodici indipendenti, interrompibili, con  $U_{\mathcal{T}} < 1$ , è schedulabile con EDF se e solo se

parte un po' a "memoria"

$$\forall L > 0, \sum_{i=1}^n \left\lfloor \frac{L + p_i - D_i}{p_i} \right\rfloor \cdot e_i \leq L$$

La formula deve essere controllata soltanto per i valori  $L$  multipli dei periodi dei task entro l'iperperiodo e tali che

$$L \leq \max \left\{ D_1, \dots, D_n, \frac{\sum_{i=1}^n (p_i - D_i) \cdot (e_i/p_i)}{1 - U_{\mathcal{T}}} \right\}$$

SERT'20

R5.21

## Schedulabilità di algoritmi a priorità fissa

Ottimalità di algoritmi priority-driven

Marco Cesati



Gli algoritmi a priorità fissa sono in generale peggiori di quelli a priorità dinamica rispetto alla capacità di determinare schedulazioni fattibili

Ad esempio:  $T_1 = (2, 1)$  e  $T_2 = (5, 2.5)$ :

- $U = 1$ , quindi sono schedulabili (ad esempio con EDF)
- $J_{1,1}$  e  $J_{1,2}$  devono avere priorità maggiore di  $J_{2,1}$  ( $T_1 > T_2$ )
- $J_{2,1}$  deve avere priorità maggiore di  $J_{1,3}$  ( $T_2 > T_1$ )
- Se le priorità sono fisse, o  $T_1 > T_2$  oppure  $T_2 > T_1$

li vedo  
simulando  
lo schedule,  
ma vanno  
in contrasto  
tra loro.

Schema della lezione  
Validazione  
Fattore di utilizzazione  
Test di schedulabilità

Schema della lezione  
Validazione  
Fattore di utilizzazione  
Test di schedulabilità

Esistono classi di sistemi che ammettono un algoritmo a priorità fissa ottimale? Si! perché li cerco?  
se li trovo, sono più facili, a parte periodicità sono inutili.

Esempio: sistemi di task semplicemente periodici (o armonici)

### Task semplicemente periodici

Per ogni coppia di task  $T_i$  e  $T_k$  con  $p_i < p_k$ ,  
 $p_k$  è un multiplo intero di  $p_i$

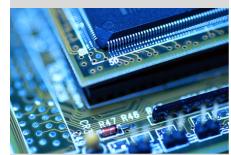
SERT'20

R5.22

## Ottimalità dell'algoritmo RM

Ottimalità di algoritmi priority-driven

Marco Cesati



### Teorema

Un sistema  $\mathcal{T}$  di task semplicemente periodici, interrompibili ed indipendenti le cui scadenze relative sono non inferiori ai rispettivi periodi ha una schedulazione RM fattibile su un singolo processore se e solo se  $U_{\mathcal{T}} \leq 1$

**Dim.** (sketch): supponiamo che tutti i task siano in fase, che le scadenze siano uguali ai periodi e che il processore non sia mai idle

Il task  $T_i$  manca la scadenza al tempo  $t$

Ogni task  $T_k$  con priorità maggiore di  $T_i$  ha periodo più piccolo di  $p_i$ , perciò  $t$  è un multiplo intero di tutti i  $p_k$

$$t < \sum_{k=1}^i \frac{e_k \cdot t}{p_k} = t \cdot \sum_{k=1}^i \frac{e_k}{p_k} \leq t \cdot U_{\mathcal{T}} \quad \Rightarrow \quad U_{\mathcal{T}} > 1$$

tempo da 0 a t  
esecuzione  
dei job prioritari  
fino ad "i"

SERT'20

R5.23

## Ottimalità dell'algoritmo DM

meglio di RM

Ottimalità di algoritmi priority-driven

Marco Cesati



### Teorema (Leung & Whitehead, 1982)

Se per un sistema di task periodici, indipendenti ed interrompibili che sono in fase ed hanno scadenze relative minori o uguali ai rispettivi periodi esiste un algoritmo a priorità fissa che produce una schedulazione fattibile, allora anche l'algoritmo DM produrrà una schedulazione fattibile

L'algoritmo DM coincide con RM se tutte le scadenze relative sono proporzionali ai rispettivi periodi

### Corollario

L'algoritmo RM è ottimale tra tutti gli algoritmi a priorità fissa qualora le scadenze relative dei task siano non superiori e proporzionali ai rispettivi periodi

*Perché il corollario non richiede che i task siano tutti in fase?*

Perché avere i task in fase è il caso peggiore possibile!