



# Lezione R11

## Real-time su multiprocessore I

Sistemi embedded e real-time

30 ottobre 2020

Marco Cesati

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

SERT'20

R11.1

### Di cosa parliamo in questa lezione?

Da inizio corso, siamo partiti con un modello molto semplice, andando via via a dettagliarlo. Manca un ultimo aspetto: passare da microprocessore a multiprocessore.

In questa lezione si dà una visione introduttiva del problema della schedulazione real-time in sistemi multiprocessore

- ① Sistemi multiprocessore
- ② Effetto Dhall
- ③ Anomalie di schedulazione
- ④ Test e condizioni di schedulabilità
- ⑤ Schedulazione partizionata



Schema della lezione

Sistemi  
multiprocessori

Effetto Dhall

Anomalie di  
schedulazione

Schedulabilità

Schedulazione  
partizionata

SERT'20

R11.2

## Sistemi multiprocessore

Un sistema real-time è detto **multiprocessore** quando è dotato di due o più processori, ciascuno in grado di **eseguire job autonomamente**

I processori possono essere dello stesso tipo o di tipo diverso

Ad esempio si consideri un sistema costituito da

- Diversi microprocessori multi-core
- Diverse schede di rete
- Diverse schede PCI con controllori DMA

In generale modellando il sistema è necessario specificare:

- il numero  $\mu$  di **tipi di processore**
- il numero  $m_i$  di processori dell' $i$ -esimo tipo ( $1 \leq i \leq \mu$ )
- su quali tipi di processore può eseguire ciascun job

In questa lezione tutti i **processori sono dello stesso tipo**

Real-time su multiprocessore I

Marco Cesati



Schema della lezione

Sistemi multiprocessori

Effetto Dhall

Anomalie di schedulazione

Schedulabilità

Schedulazione partizionata

SERT'20

R11.3

## Real-time su sistemi multiprocessori

aggiungere un nuovo processore è una dimensione in più, ciò che vale con un processore non è riportabile su più processori con facilità.

*Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem.*

*The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors.*

C. L. Liu, 1969

Real-time su multiprocessore I

Marco Cesati



Schema della lezione

Sistemi multiprocessori

Effetto Dhall

Anomalie di schedulazione

Schedulabilità

Schedulazione partizionata

SERT'20

R11.4

## Sistemi statici

Real-time su multiprocessore I

Marco Cesati



Un sistema real-time è detto **statico** quando ciascun job è assegnato ad uno specifico processore

Esistono due varianti di sistemi **statici**:

- L'insieme dei job (o task) nel sistema è predeterminato, e l'assegnazione di ciascun job ad uno specifico processore è effettuata una volta per tutte nella fase di progetto del sistema
- L'insieme dei task nel sistema non è predeterminato, e l'assegnazione del task ad uno specifico processore è effettuata dal sistema operativo durante la fase di creazione del task (*scheduler partizionati*)

In entrambi i casi lo scheduler **non** decide su quale processore sarà eseguito un job appena rilasciato (è già stabilito)

SERT'20

R11.5

## Sistemi dinamici

Real-time su multiprocessore I

Marco Cesati



Un sistema real-time è detto **dinamico** quando lo scheduler può assegnare dinamicamente un job ad un qualunque processore disponibile

Esistono tre varianti di sistemi **dinamici**:

- Con job **non interrompibili**
- Con job **interrompibili e non migrabili**: anche se interrotto, il job deve riprendere l'esecuzione sullo stesso processore in cui era in esecuzione precedentemente
- Con job **interrompibili e migrabili**: una volta interrotto, il job può riprendere l'esecuzione su qualunque processore che possa eseguirlo

Un algoritmo di schedulazione per un sistema dinamico è detto **globale** perché stabilisce quale processore eseguirà ciascun job

SERT'20

R11.6



## Esempio di schedulazione in sistema statico

Real-time su multiprocessore I

Marco Cesati



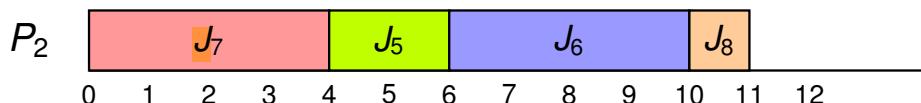
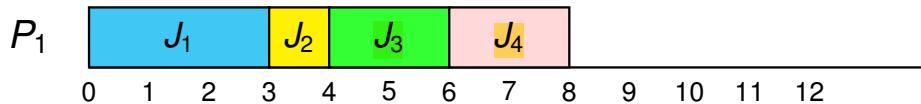
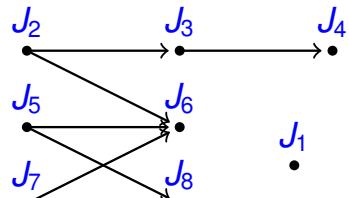
In un sistema **statico** esiste per ogni processore una lista di task o job con le relative priorità

vincoli sull'ordine, solo  $J_1$  indipendente, esistono vincoli anche tra processori diversi

Lista processore  $P_1$ :  $J_1, J_2, J_3, J_4$

Lista processore  $P_2$ :  $J_5, J_6, J_7, J_8$

$i$	1	2	3	4	5	6	7	8
$r_i$	0	0	0	0	4	0	0	0
$e_i$	3	1	2	2	2	4	4	1



SERT'20

R11.7

## Esempio di schedulazione in sistema dinamico

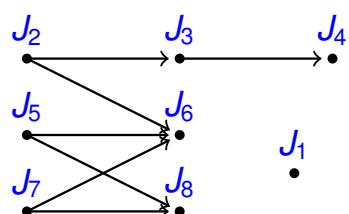
Real-time su multiprocessore I

Marco Cesati

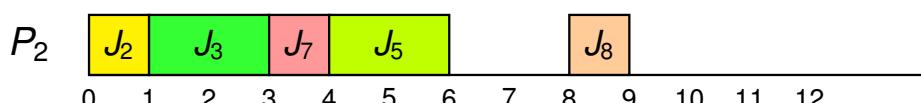
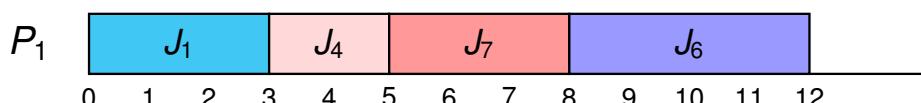


Lista:  $J_1, J_2, \dots, J_8$

$i$	1	2	3	4	5	6	7	8
$r_i$	0	0	0	0	4	0	0	0
$e_i$	3	1	2	2	2	4	4	1

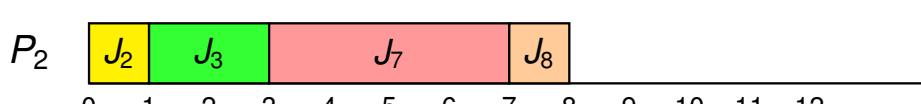
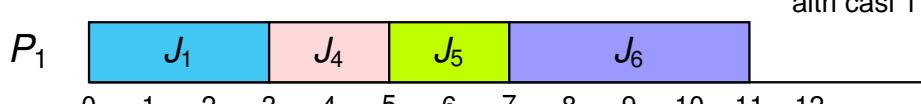


Job interrompibili e migrabili:



Job non interrompibili:

nel caso migrabili ho finito a 12, negli altri casi 11 (dove non erano migrabili)



SERT'20

R11.8

*Quali sono i vantaggi dei sistemi statici?*

- Si può analizzare la schedulabilità su ciascun processore utilizzando i risultati teorici validi per il caso uniprocessore (fondamentale per i sistemi hard real-time!)
- Un job che impiega più tempo di quanto previsto dal suo WCET (*overrun*) può ritardare l'esecuzione dei soli task associati al suo processore
- Poiché i job interrotti riprendono sempre l'esecuzione sullo stesso processore si evitano i costi dovuti alla migrazione del contesto ad un altro processore
- La coda di esecuzione (in cui i job rilasciati aspettano di essere attivati) è relativa al singolo processore ed è quindi più piccola



Schema della lezione

Sistemi multiprocessori

Effetto Dhall

Anomalie di schedulazione

Schedulabilità

Schedulazione partizionata

## Vantaggi dei sistemi dinamici

*Quali sono i vantaggi dei sistemi dinamici?*

- Hanno tipicamente meno cambi di contesto e interruzioni dei job, poiché un job è interrotto solo quando nessun processore è idle
- Se un job esegue per meno tempo di quanto previsto dal suo WCET, il tempo liberato sul processore può essere utilizzato potenzialmente da tutti i task nel sistema
- Se un job impiega più tempo di quanto previsto dal suo WCET (*overrun*), la probabilità che ciò comporti il mancato rispetto di una o più scadenze è minore
- Per ogni task del sistema che è creato a run-time, assegnazione e bilanciamento del carico sono “automatici” e determinati dall'algoritmo di schedulazione globale



Schema della lezione

Sistemi multiprocessori

Effetto Dhall

Anomalie di schedulazione

Schedulabilità

Schedulazione partizionata

Gli algoritmi di schedulazione **clock-driven** sono in generale utilizzabili senza problemi con i sistemi multiprocessore

Infatti la schedulazione effettiva è generata “off-line” e validata una volta per tutte [devo sapere già tutto, poco flessibile.](#)

Al contrario, non è immediato applicare gli algoritmi **priority-driven** ai sistemi multiprocessore

Diverse problematiche:

- Efficienza degli algoritmi (effetto Dhall)
- Predicibilità del sistema (anomalie di schedulazione)
- Test di schedulabilità (teoremi non più validi)

## Effetto Dhall

### Teorema (Dhall & Liu, 1978)

Per ogni numero di processori  $m \geq 2$ , esistono insiemi di task con utilizzazione **bassa** che non sono schedulabili con RM, DM o EDF

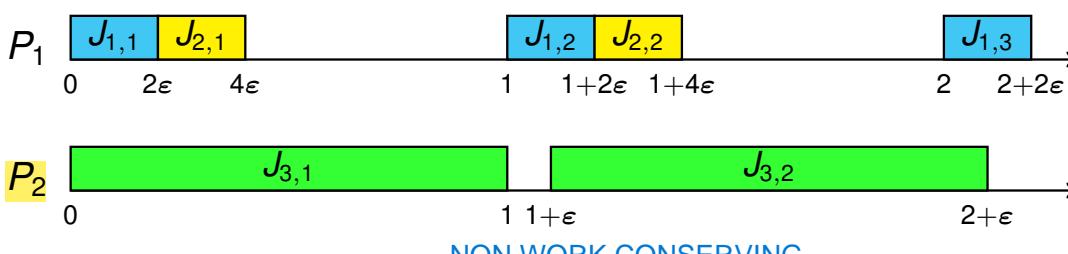
Consideriamo  $T_1 = (1, 2\varepsilon)$ ,  $T_2 = (1, 2\varepsilon)$ , ...,  $T_m = (1, 2\varepsilon)$ ,  $T_{m+1} = (1 + \varepsilon, 1)$

Utilizzazione globale:  $U_g = 2\varepsilon \cdot m + 1/(1 + \varepsilon) \rightarrow 1$  se  $\varepsilon \rightarrow 0$

$m$  può essere qualsiasi valore (anche un milione di task),

ma l'utilizzo sempre a 1 tende, quindi basta 1 proc (massimo massimo 2)

Schedulazione fattibile,  $m = 2$ :



non ho usato  
nè edf, nè rm,  
ne dm.

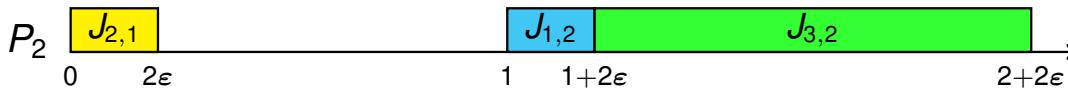
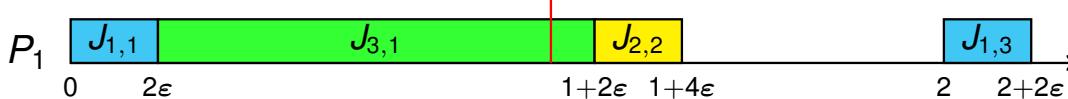
## Effetto Dhall (2)

Real-time su multiprocessore I

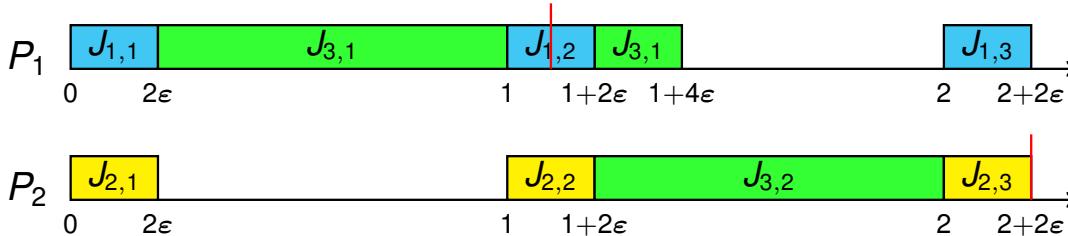
Marco Cesati



Schedulazione con EDF,  $m = 2$ :



Schedulazione con RM,  $m = 2$ :



Comportamenti analoghi all'effetto Dhall si verificano solo se almeno uno dei task ha una utilizzazione molto alta (Funk, Goossens & Baruah, 2001)

SERT'20 R11.13

## Anomalie di schedulazione

Real-time su multiprocessore I

Marco Cesati



Si definisce *anomalia di schedulazione* il comportamento di un algoritmo di schedulazione per cui, in presenza di variazioni apparentemente vantaggiose del carico del sistema, si ottiene un peggioramento delle prestazioni

Esempi di variazioni “vantaggiose”:

- Aumento del periodo di un task
- Diminuzione del tempo di esecuzione di un task
- Rimozione di vincoli di precedenza tra i job
- Aumento del numero di processori

Nei sistemi uniprocessore le anomalie di schedulazione possono verificarsi solo nel caso in cui job sono **non interrompibili** e/o **non indipendenti** (Mok, 2000)

SERT'20 R11.14

# Anomalie di schedulazione in sistemi multiprocessore

Assumiamo che tutti i job siano indipendenti:



Tipo di sistema	Anomalie ?
Statico, job non interrompibili	Si (1)
Statico, job interrompibili	No
Dinamico, job non interrompibili	Si (2)
Dinamico, job interrompibili ma non migrabili	Si (3)
Dinamico, job interrompibili e migrabili	Si (4)

- (1) Mok, 2000 (cfr. esempi in Lezione R5)
- (2) Graham, 1969
- (3) Ha & Liu, 1994
- (4) Andersson & Jonsson, 2000

*Perché le anomalie complicano il problema della validazione?*

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

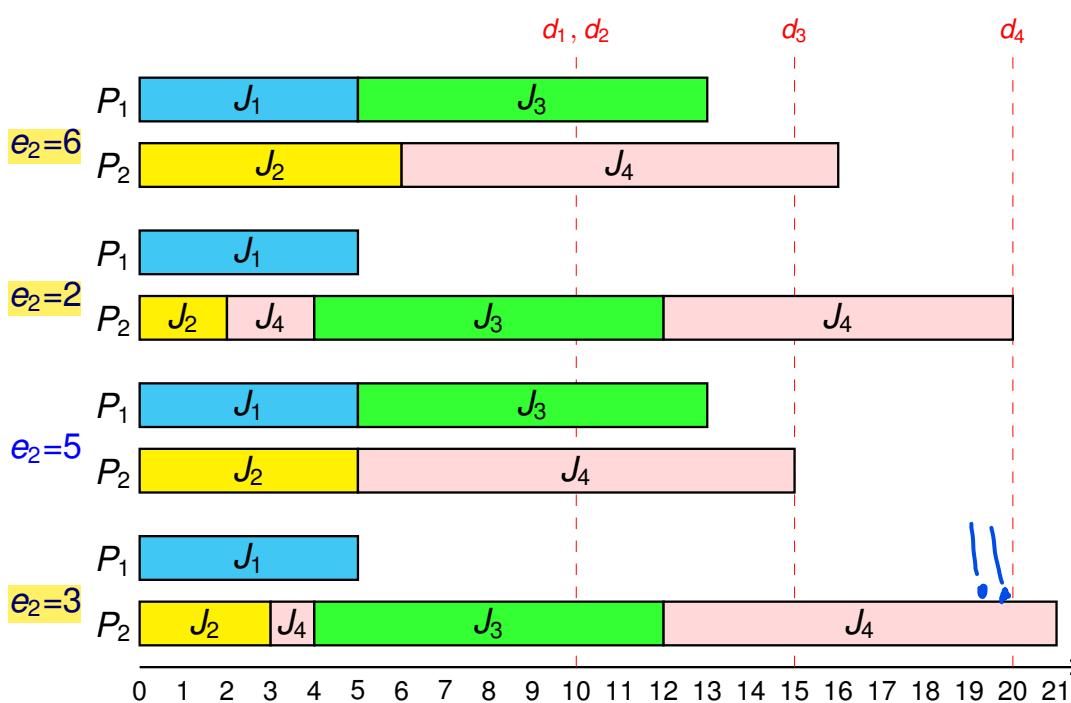
## Anomalie di schedulazione con job non migrabili

Job interrompibili ma non migrabili

$i$	1	2	3	4
$r_i$	0	0	4	0
$d_i$	10	10	15	20
$e_i$	5	[2, 6]	8	10

Indice minore  $\equiv$  priorità maggiore

$e_2$  varia da 2 a 6



Analiticamente,  
NON posso  
prevedere che:  
 $e = 2$  OK  
 $e = 3$  MANCA SCAD.  
 $e = 5$  OK  
 $e = 6$  OK

## Anomalie di schedulazione con job migrabili

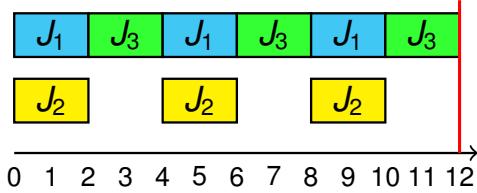
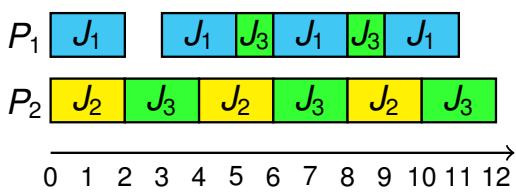
Real-time su multiprocessore I

Marco Cesati

Aumento del periodo di un task di priorità alta:

$$T_1 = (3, 2), T_2 = (4, 2), \\ T_3 = (12, 8)$$

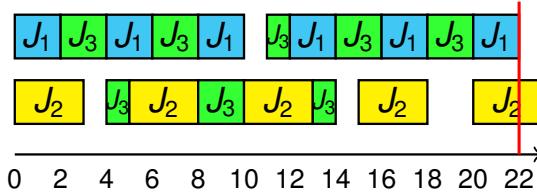
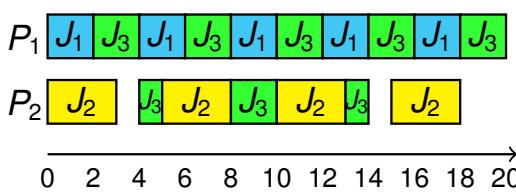
$$T_1 = (4, 2), T_2 = (4, 2), \\ T_3 = (12, 8)$$



Aumento del periodo di un task di priorità bassa:

$$T_1 = (4, 2), T_2 = (5, 3), \\ T_3 = (10, 7)$$

$$T_1 = (4, 2), T_2 = (5, 3), \\ T_3 = (11, 7)$$



J3 manca scadenza

SERT'20 R11.17

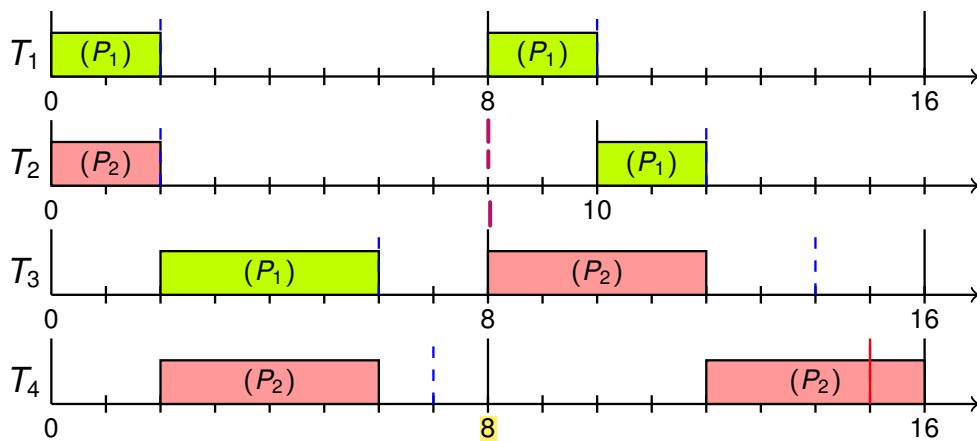
## Istanti critici in schedulazioni globali

non posso riadattare le idee monoprocessoressi a multiprocessoressi.

**Teorema (Lauzac, Melhem & Mosse, 1998)**

Utilizzando uno scheduler globale a priorità fissa a livello di task (es., DM), l'istante in cui un job di un task  $T_i$  è rilasciato contemporaneamente ai job di tutti i task di priorità superiore  $T_1, \dots, T_{i-1}$  **non** è necessariamente un istante critico di  $T_i$

$$T_1 = (8, 2, 2), T_2 = (10, 2, 2), T_3 = (8, 4, 6), T_4 = (8, 4, 7), m = 2$$

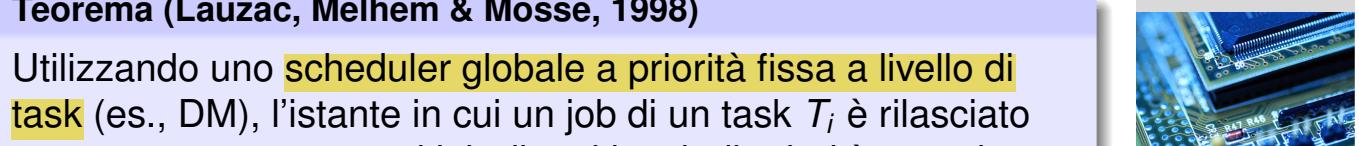


Il test di schedulabilità non funziona!

A PRIORI, NON POSSO SAPERE QUALE SIA IL CASO PEGGIORE.

Real-time su multiprocessore I

Marco Cesati



Schema della lezione

Sistemi multiprocessori

Effetto Dhall

Anomalie di schedulazione

Schedulabilità

Schedulazione partizionata

Il peggior caso per T4 non è quello di tutti i rilasci in fase, bensì quando ho questi tempi di rilascio (8,10,8,8).

SERT'20 R11.18

## Fattore di utilizzazione per multiprocessore

Se mi basassi su altra classe di risultati? Uso quelli legati alla schedulabilità.

Real-time su multiprocessore I  
Marco Cesati

### Teorema (Oh & Baker, 1998)

Dato un sistema di task periodici con scadenze uguali ai periodi e  $m$  processori, se  $X$  è un qualsiasi algoritmo di schedulazione partizionato con priorità fissa a livello di task:

$$U_X \leq \frac{m+1}{1 + 2^{1/(m+1)}}$$

Il teorema che stiamo per proporre estende quello di Oh e Baker, fornendo risultati migliori.

### Teorema (Andersson, Baruah & Jonsson, 2001)

Dato un sistema di task periodici con scadenze uguali ai periodi e  $m$  processori, sia  $X$

- un qualsiasi algoritmo di schedulazione partizionato, o
- un qualsiasi algoritmo di schedulazione globale con priorità fissa a livello di job;

allora per il fattore di utilizzazione di  $X$  si ha:  $U_X \leq \frac{m+1}{2}$

Non posso caricare il processore più di questo fattore.

SERT'20 R11.19

## Schedulazione a priorità fissa su multiprocessore

Real-time su multiprocessore I  
Marco Cesati

### Corollario (Andersson, Baruah & Jonsson, 2001)

Nessun algoritmo di schedulazione globale con priorità fissa a livello di job è ottimale su multiprocessore

Schedul. EDF di  $T_1 = (1, 1)$ ,  $T_2 = (2, 1)$ ,  $T_3 = (5, 5)$ ,  $m = 2$ :



Eppure una schedulazione fattibile non EDF esiste:



per ogni algoritmo a priorità fissa,  
posso presentare una soluzione  
avente SCHEDULAZIONE FATTIBILE,  
ma di cui l'algoritmo non riesce a schedulare.

SERT'20 R11.20

# Algoritmi ottimali per multiprocessore

Possono esistere algoritmi ottimali per multiprocessore? **Sì!**

LST, basato su slack.

- Alcuni algoritmi di schedulazione dinamica a livello di job hanno fattore di utilizzazione pari a  $m$
- Tuttavia, nessun algoritmo **on-line** (non “chiaroveggente”) è ottimale se gli istanti di rilascio dei job non sono esattamente prefissati (Fisher, 2007)

Una classe di algoritmi ottimali su multiprocessore è derivata dall'algoritmo **Pfair** (Baruah & al., 1996):

- basato sull'idea di schedulazione *fluida*: ogni task progredisce in modo proporzionale alla sua utilizzazione
- **tempo diviso in quanti**: allo scadere di ogni **quanto**, lo scheduler assegna i task ai processori in modo che per ogni task  $T_i$  il lavoro compiuto sia  $\lceil t \cdot e_i/p_i \rceil$  o  $\lfloor t \cdot e_i/p_i \rfloor$

Gli algoritmi dinamici a livello di job sono molto costosi in termini di overhead dello scheduler, quindi non sono adottati

Real-time su multiprocessore I

Marco Cesati



Schema della lezione

Sistemi multiprocessori

Effetto Dhall

Anomalie di schedulazione

Schedulabilità

Schedulazione partizionata

(tempo per cui sono arrivato fino a quel momento) \*  
(dimensione task)

SERT'20

R11.21

## Schedulazione partizionata

c'è algoritmo che associa task ai processori.

Nei sistemi real-time multiprocessore **statici** l'algoritmo di schedulazione è detto **partizionato**

Consiste di due componenti:

- **Allocazione dei task**: assegnazione di ciascun task ad uno specifico processore
  - questo problema è analogo a *bin packing* ed è **NP hard** (Garey & Johnson, 1979)
- **Problema di priorità**: schedulazione dei task su ciascun processore
  - questo è il problema della schedulazione su sistemi con un singolo processore

Real-time su multiprocessore I

Marco Cesati



Schema della lezione

Sistemi multiprocessori

Effetto Dhall

Anomalie di schedulazione

Schedulabilità

Schedulazione partizionata

SERT'20

R11.22



## Formulazione del problema

Dato un sistema di task periodici, partizionare i task in sottoinsiemi tali che ciascun sottoinsieme può essere schedulato in modo fattibile su un singolo processore utilizzando un determinato algoritmo di schedulazione

idea base: per ogni task do un processore, ma posso fare di meglio.

Un sistema di  $n$  task indipendenti è schedulabile con  $n$  processori (purché ciascun task abbia densità inferiore a uno)

Non è noto alcun algoritmo polinomiale che sia in grado di determinare, dato un sistema di  $n$  task indipendenti, il minimo numero  $m_0$  di processori che permetta di schedularlo

Gli algoritmi di allocazione dei task utilizzabili in pratica trovano soluzioni non ottimali:

- Non riescono ad associare i task ai processori in modo da sfruttarli nel miglior modo possibile sprecherò più processori del minimo teorico.
- Non riescono a determinare schedulazioni fattibili per ogni possibile insieme di task schedulabile

per alcuni insieme task, non trovo schedulazione fattibile anche se esiste.  
Non ottimale su schedulabilità.

SERT'20 R11.23

## Allocazione dei task (2)



### Come misurare la "bontà" di un algoritmo di allocazione?

Schema della lezione

Sistemi multiprocessori

Effetto Dhall

Anomalie di schedulazione

Schedulabilità

Schedulazione partizionata

Tre metriche principali:

valuta come uso le risorse. Difficile da calcolare, perchè devo sapere il minimo numero " $m_0$ ", che sappiamo non essere facile da trovare.

- **Rapporto di approssimazione:** è il massimo valore  $m/m_0$ , ove  $m$  è il numero di processori utilizzato dall'algoritmo di allocazione e  $m_0$  è il minimo numero teoricamente necessario, considerando ogni possibile sistema di task

l'idea è: ho allocato male, di quanto devo velocizzare i processori per riparare a questo errore?

- **Fattore di accelerazione:** quanto è necessario aumentare la velocità di esecuzione degli  $m_0$  processori per schedulare fattibilmente ogni possibile sistema di task con le assegnazioni determinate dall'algoritmo di allocazione  
se tale soglia =  $m$ , allora riesco ad allocare su tutti i processori il 100% dei task mantenendo tutte le scadenze

- **Fattore di utilizzazione:** il valore di soglia per cui tutti i sistemi di task con fattore di utilizzazione totale inferiore o uguale sono sempre schedulabili utilizzando l'algoritmo di allocazione dei task

SERT'20 R11.24

## Algoritmo RMFF

Real-time su multiprocessore I

Marco Cesati



Il più semplice algoritmo per l'allocazione dei task è **RMFF** (Rate Monotonic First Fit, Dhall & Liu, 1978):

- ① ordina i task per periodi non decrescenti:  $T_1, T_2, \dots$
- ② ordina arbitrariamente i processori:  $P_1, P_2, \dots$
- ③ cominciando da  $T_1$ , assegna ciascun task  $T_i$  al primo processore  $P_j$  tale che l'insieme dei task già assegnati a  $P_j$  insieme a  $T_i$  risulta ancora schedulabile tramite **RM**

- $U_{\text{RMFF}} = m \cdot (\sqrt{2} - 1)$  (Oh & Baker, 1998)
- Fattore di approssimazione: 2.23 (Oh & Son, 1993)

*RMFF può essere usato come un algoritmo on-line?* NO

L'ordinamento dei job richiede la conoscenza di tutti i periodi dei task da schedulare  $\Rightarrow$  usare **RMFF** on-line richiede di riallocare tutti i task quando ne viene creato uno nuovo

SERT'20 R11.25

## Algoritmo FFDU

Real-time su multiprocessore I

Marco Cesati



Un altro algoritmo per l'allocazione dei task è **FFDU** (First Fit Decreasing Utilization, Davari & Dhall, 1986):

- ① ordina i task per fattori di utilizzazione decrescenti:  $T_1, T_2, \dots$
- ② ordina arbitrariamente i processori:  $P_1, P_2, \dots$
- ③ cominciando da  $T_1$ , assegna ciascun task  $T_i$  al primo processore  $P_j$  tale che l'insieme dei task già assegnati a  $P_j$  insieme a  $T_i$  risulta ancora schedulabile tramite **RM**

- $U_{\text{FFDU}} = m \cdot (\sqrt{2} - 1)$  (Lopez & al., 2003)
- Fattore di approssimazione: 1.67 (Oh & Son, 1995)

Poiché richiede l'ordinamento dei task, **FFDU** è tipicamente utilizzato come algoritmo off-line

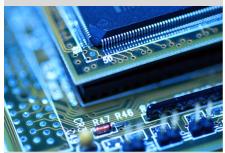
SERT'20 R11.26

## Algoritmo RM-FF

trattino = online, non ordinamento  
senza trattino = offline, ordinamento

Real-time su multiprocessore I

Marco Cesati



Una variante di RMFF è l'algoritmo **RM-FF** (Oh & Son, 1994) che sostanzialmente non effettua l'ordinamento dei task prima della allocazione:

- ➊ ordina arbitrariamente i processori:  $P_1, P_2, \dots$
- ➋ assegna ciascun task  $T_i$  al primo processore  $P_j$  tale che l'insieme dei task già assegnati a  $P_j$  insieme a  $T_i$  risulta ancora schedulabile tramite **RM**

- $U_{\text{RM-FF}} = m \cdot (\sqrt{2} - 1)$  (Oh & Baker, 1998)
- Fattore di approssimazione: 2.33 (Oh & Son, 1994)

A differenza di **RMFF**, **RM-FF** è facilmente utilizzabile come algoritmo on-line

SERT'20

R11.27

## Algoritmo EDF-FF

L'euristica "first fit" accoppiata all'algoritmo di schedulazione **EDF** dà luogo all'algoritmo di allocazione "on-line" **EDF-FF**:

- ➊ ordina arbitrariamente i processori:  $P_1, P_2, \dots$
- ➋ assegna ciascun task  $T_i$  al primo processore  $P_j$  tale che l'insieme dei task già assegnati a  $P_j$  insieme a  $T_i$  risulta ancora schedulabile tramite **EDF**

- $U_{\text{EDF-FF}} = \frac{\beta \cdot m + 1}{\beta + 1}$ ,  $\beta = \left\lfloor 1 / \max_k \frac{e_k}{p_k} \right\rfloor$  (Lopez & al., 2000)
- Fattore di approssimazione: 1.7 (Garey & Johnson, 1979)

**EDF-FF** è ottimale tra tutti gli algoritmi partizionati:

$$\beta = 1 \implies U_{\text{EDF-FF}} = (m+1)/2$$

$$\beta \rightarrow \infty \implies U_{\text{EDF-FF}} \rightarrow m$$

Real-time su multiprocessore I

Marco Cesati



Schema della lezione

Sistemi multiprocessori

Effetto Dhall

Anomalie di schedulazione

Schedulabilità

Schedulazione partizionata