Lezione R5

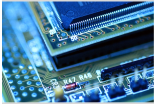
Ottimalità di algoritmi priority-driven

Sistemi embedded e real-time

6 novembre 2012

Marco Cesati

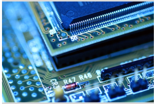
Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università degli Studi di Roma Tor Vergata



Di cosa parliamo in questa lezione?

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

1. Ottimalità di EDF
2. Il problema della validazione
3. Il fattore di utilizzazione
4. Il test di schedulabilità

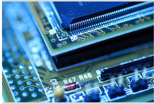


Ottimalità dell'algoritmo EDF

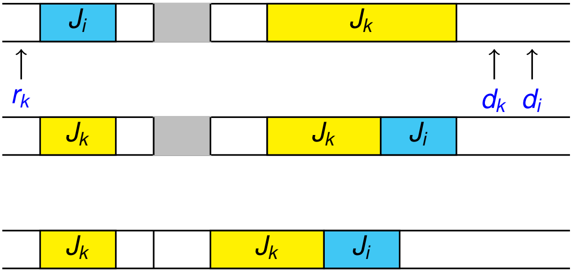
L'algoritmo EDF è ottimale nel senso che riesce sempre a trovare una schedulazione fattibile di un insieme di job interrompibili e indipendenti su un singolo processore, ovviamente a condizione che tale schedulazione esista

Sketch della dimostrazione:

- Ogni schedulazione fattibile di un insieme di job arbitrari può essere trasformata in una schedulazione prodotta dall'algoritmo EDF
- Sia J_i schedulato prima di J_k con $d_i > d_k$
- Se r_k è oltre l'intervallo in cui è schedulato J_i, i due job rispettano l'algoritmo EDF; assumiamo che r_k è prima dell'intervallo in cui è schedulato J_i
- Scambiamo tra loro J_i e J_k (se necessario utilizzando l'interrompibilità dei job per tenere conto di lunghezze diverse degli intervalli di tempo)
- Ora J_i e J_k rispettano le priorità EDF



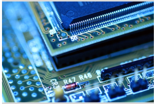
Ottimalità dell'algoritmo EDF (2)

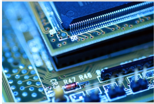


Scambiare sistematicamente di posto tutti i job che non rispettano le priorità EDF è sufficiente a trasformare la schedulazione in quella prodotta dall'algoritmo EDF? ***No!***

Potrebbero rimanere intervalli di tempo in cui il processore è inutilizzato pur essendoci job pronti per l'esecuzione ma schedulati dopo

È sempre possibile anticipare l'esecuzione di uno o più job in modo da eliminare questi casi



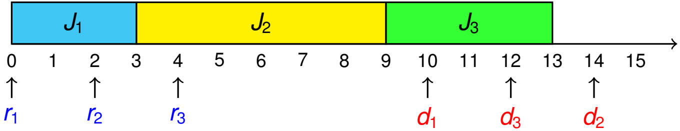


Algoritmi priority-driven e job non interrompibili

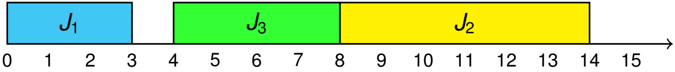
Gli algoritmi priority-driven (ossia work-conserving), così come LRT, **non** sono ottimali se i job sono non interrompibili

Scheduliamo J_1, J_2 e J_3 non interromp.:

i	1	2	3
r_i	0	2	4
d_i	10	14	12
e_i	3	6	4



Eppure una schedulazione fattibile non priority-driven esiste:



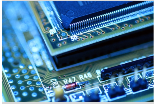
Il problema della validazione

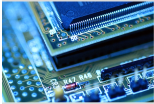
Gli algoritmi priority-driven in generale:

- sono semplici da implementare
- sono flessibili
- non richiedono necessariamente di conoscere esattamente il modello di carico
- è difficile 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





Validazione di algoritmi priority-driven

Per gli algoritmi priority-driven il problema della validazione è difficile da risolvere a causa delle *anomalie di schedulazione*, ossia di comportamenti temporali inattesi

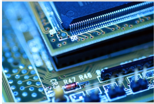
Anomalie di schedulazione si verificano anche in sistemi semplici

Ad esempio, in un sistema con job non interrompibili con lo stesso istante di rilascio, 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

Perché le anomalie complicano il problema della validazione?

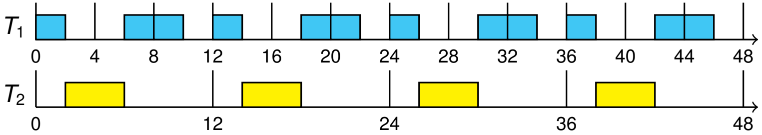
Se i parametri dei job di un sistema possono variare, non si può validare il sistema esaminando solo il "caso peggiore": è necessario esaminare tutte le combinazioni di parametri



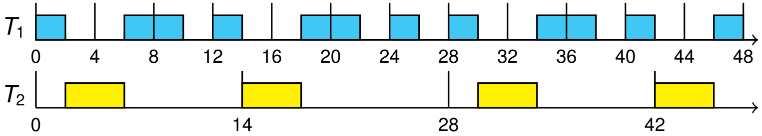
Esempio di anomalia di schedulazione (periodo)

Due job **non** interrompibili schedulati con RM su un processore

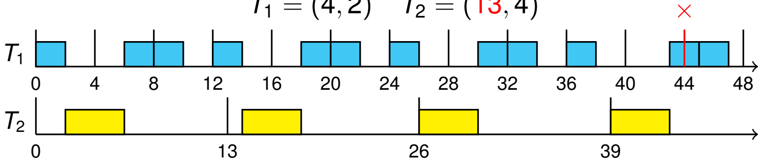
$$T_1 = (4, 2) \quad T_2 = (12, 4)$$

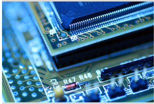


$$T_1 = (4, 2) \quad T_2 = (14, 4)$$



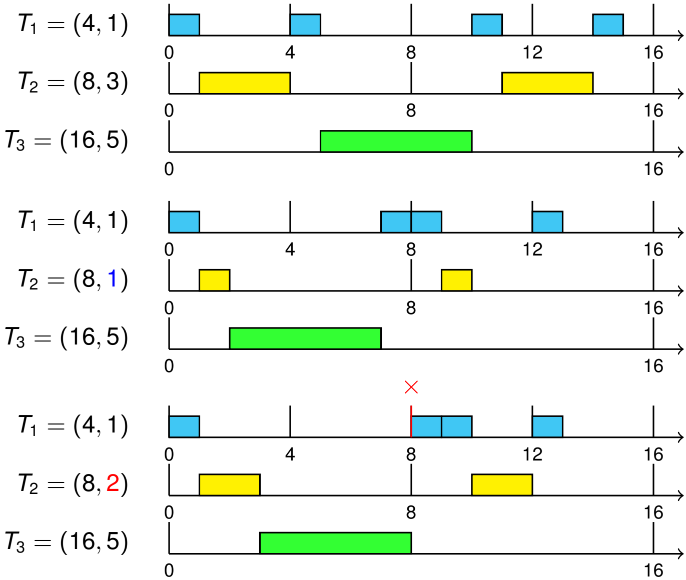
$$T_1 = (4, 2) \quad T_2 = (13, 4)$$





Esempio di anomalia di schedulazione (tempo d'esecuzione)

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



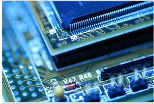
Esecuzione predicibile

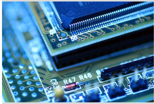
Fissato un algoritmo, la schedulazione prodotta considerando i tempi d'esecuzione massimi (minimi) per tutti i job è detta *schedulazione massima (minima)*

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

- siano σ^- e ϵ^- gli istanti di attivazione e completamento di un job nella schedulazione minima
- siano σ^+ e ϵ^+ 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





Predicibilità per gli algoritmi priority-driven

Teorema

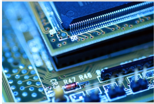
Un insieme di job interrompibili, indipendenti, e con istanti di rilascio fissati schedulato su un processore da un algoritmo priority-driven è predicibile

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)



Fattore di utilizzazione

- 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

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

Tanto maggiore è U_X, tanto migliore è l'algoritmo

Confronto tra algoritmi di schedulazione (2)

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

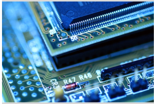
Esiste un insieme di due task con fattore di utilizzazione pari a $\varepsilon > 0$ piccolo a piacere che non è schedulabile con FIFO:
$T_1 = (10, 5\,\varepsilon)$, $T_2 = (20/\varepsilon, 10)$

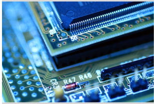
Qual è il fattore di utilizzazione dell'algoritmo EDF? ***Uno!***

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?

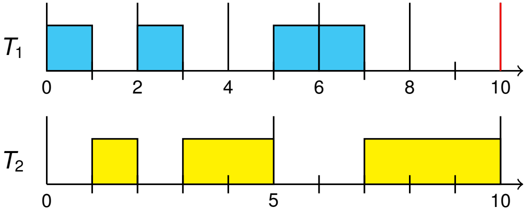
- Non esiste un modo efficiente per determinare quali job saranno in ritardo in caso di sovraccarico o overrun in una schedulazione a priorità dinamica quale EDF
- Qual è la priorità EDF di un job in ritardo?
- Il comportamento di un algoritmo a priorità fissa è predicibile anche in caso di sovraccarico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



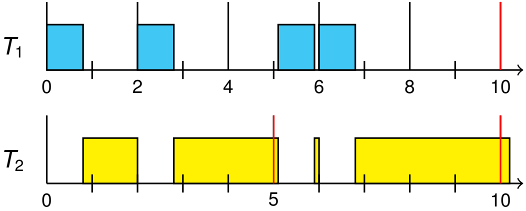


Comportamento di EDF con sovraccarico

$T_1 = (2, 1)$
$T_2 = (5, 3)$
$(U = 1.1)$



$T_1 = (2, 0.8)$
$T_2 = (5, 3.5)$
$(U = 1.1)$



Fattore di utilizzazione di EDF

Teorema

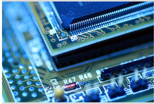
Un sistema $\mathcal{T}$ di task indipendenti ed interrompibili con scadenze relative uguali ai rispettivi periodi e fattore di utilizzazione $U_{\mathcal{T}}$ ha una schedulazione fattibile su un singolo processore se e solo se $U_{\mathcal{T}} \leq 1$

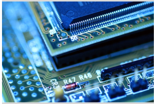
Corollario

L'algoritmo EDF ha fattore di utilizzazione $U_{\text{EDF}} = 1$ 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!





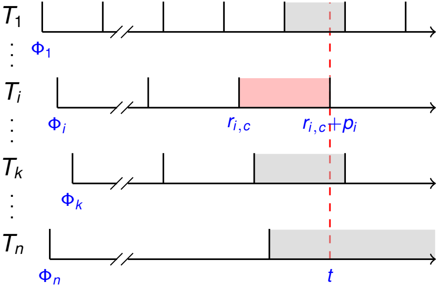
Fattore di utilizzazione di EDF (2)

Da dimostrare: se EDF non trova una schedulazione fattibile, allora $U_{\mathcal{T}} > 1$

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

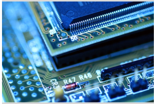
Assumiamo che il processore non sia mai idle prima di t

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



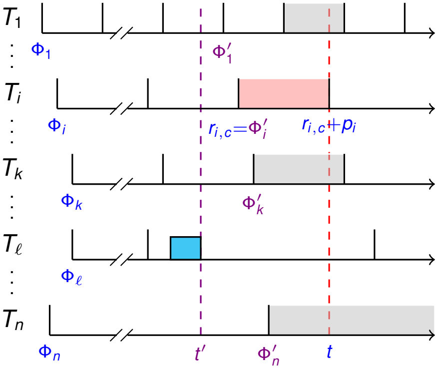
Tutti i job nei periodi che includono t non sono eseguiti prima di t perché hanno scadenze dopo $J_{i,c}$

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



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

$$t - t' < \frac{(t - \Phi'_i)\, e_i}{p_i} + \sum_{\substack{T_k \in \mathcal{T} \setminus \mathcal{T}' \\ k \neq i}} \left\lfloor \frac{t - \Phi'_k}{p_k} \right\rfloor e_k \leq (t - t') \sum_{T_k \in \mathcal{T} \setminus \mathcal{T}'} \frac{e_k}{p_k}$$

$$\leq (t - t')\, U_{\mathcal{T}} \Rightarrow \boxed{U_{\mathcal{T}} > 1}$$

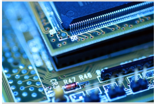
Riassumendo...

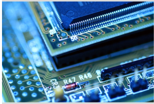
Si assumano task indipendenti, interrompibili e schedulati su un singolo processore

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)

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 (Φ, p, e, D) il rapporto $\dfrac{e}{\min(D, p)}$

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!

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:

T1) Se, per ogni task T_i, $D_i = p_i$, allora esiste una schedulazione fattibile se e solo se $U_{\mathcal{T}} \leq 1$

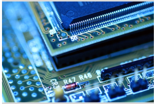
C1) Se, per ogni task T_i, $D_i \geq p_i$, allora esiste una schedulazione fattibile se e solo se $U_{\mathcal{T}} \leq 1$

C2) 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 $U_{\mathcal{T}} \leq 1$)

T2) Se per qualche task T_i, $D_i < p_i$, allora esiste una schedulazione fattibile se $\Delta_{\mathcal{T}} = \sum_k e_k/\min(p_k, D_k) \leq 1$

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

- Se $\exists\, T_i$ tale che $D_i < p_i$, allora $U_{\mathcal{T}} < \Delta_{\mathcal{T}}$
- Se $\forall\, T_i$, $D_i \geq p_i$, allora $U_{\mathcal{T}} = \Delta_{\mathcal{T}}$



Fattore di utilizzazione e scadenze oltre i periodi

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

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

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

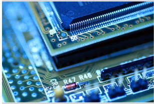
$$\text{Quindi } \forall\, k \geq 1, \quad \frac{e}{p} < \frac{k+1}{k} \cdot \frac{e}{p} \leq 1 + \frac{D}{pk} \quad \Longrightarrow \quad \frac{e}{p} \leq 1$$

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

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

$$\forall\, k \geq 1, \quad \frac{e_n}{p_n} < \left(\frac{D_n}{k\, p_n} + 1\right) \cdot \left(1 - \sum_{i=i}^{n-1} \frac{e_i}{p_i}\right) \quad \Longrightarrow \quad \boxed{U_T \leq 1}$$

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



Test di schedulabilità per EDF

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

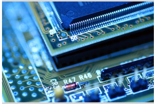
- Se $\Delta_{\mathcal{T}} \leq 1$ è schedulabile (T2 e C1)
- Altrimenti: se $D_i \geq p_i$ per ogni i non è schedulabile (C1)
- Altrimenti: se i task sono in fase applichiamo EDF per un segmento lungo $2H + \max p_i + \max D_i$, ove H è l'iperperiodo (Baruah, Howell, Rosier 1993)

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

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!



Schedulabilità di algoritmi a priorità fissa

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 $T_2 = (5, 2.5)$:

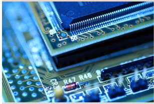
- $U = 1$, quindi sono schedulabili (ad esempio con EDF)
- $J_{1,1}$ e $J_{1,2}$ devono avere priorità maggiore di $J_{2,1}$ ($T_1 > T_2$)
- $J_{2,1}$ deve avere priorità maggiore di $J_{1,3}$ ($T_2 > T_1$)
- Se le priorità sono fisse, o $T_1 > T_2$ oppure $T_2 > T_1$

Esistono classi di sistemi che ammettono un algoritmo a priorità fissa ottimale? ***Sì!***

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



Ottimalità dell'algoritmo RM

Teorema

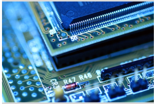
Un sistema $\mathcal{T}$ di task semplicemente periodici, interrompibili ed indipendenti le cui scadenze relative sono non inferiori ai rispettivi periodi ha una schedulazione RM fattibile su un singolo processore se e solo se $U_{\mathcal{T}} \leq 1$

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

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



Ottimalità dell'algoritmo DM

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!