Sustainability e DP-fairness.

Paolo Parisen Toldin

Università di Bologna

23 july 2010

Il talk è basato sui seguenti due articoli:

- Sustainable Multiprocessor Scheduling of Sporadic Task Systems - T.P.Baker, S.K.Baruah
- DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling - G.Levin, S.Funk, C.Sadowsky, I.Pye, S.Brandt

È doveroso innanzitutto fare distinzione tra job e task.

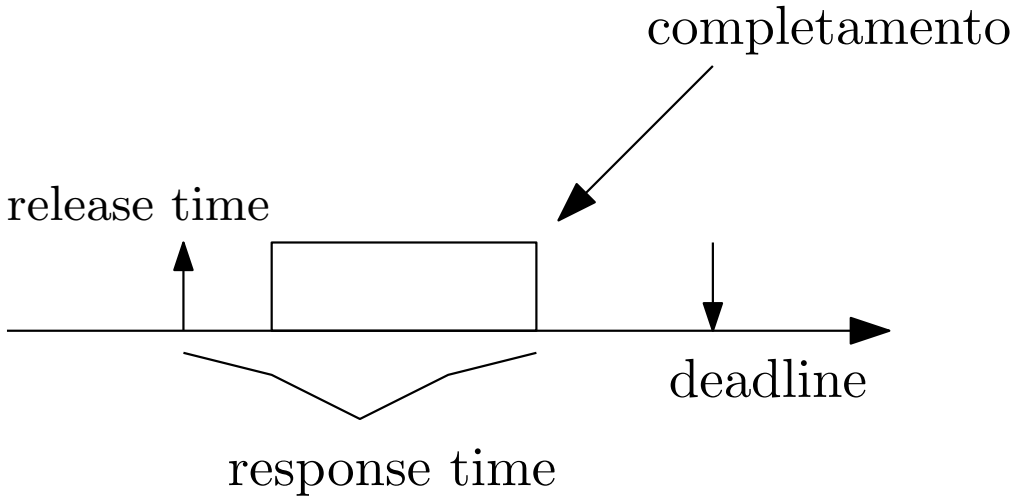
- Task: unità funzionale e architetturale; genera job per completare il suo lavoro.
- Job: è ciò che viene realmente eseguito; richiede delle risorse logiche e fisiche per eseguire.

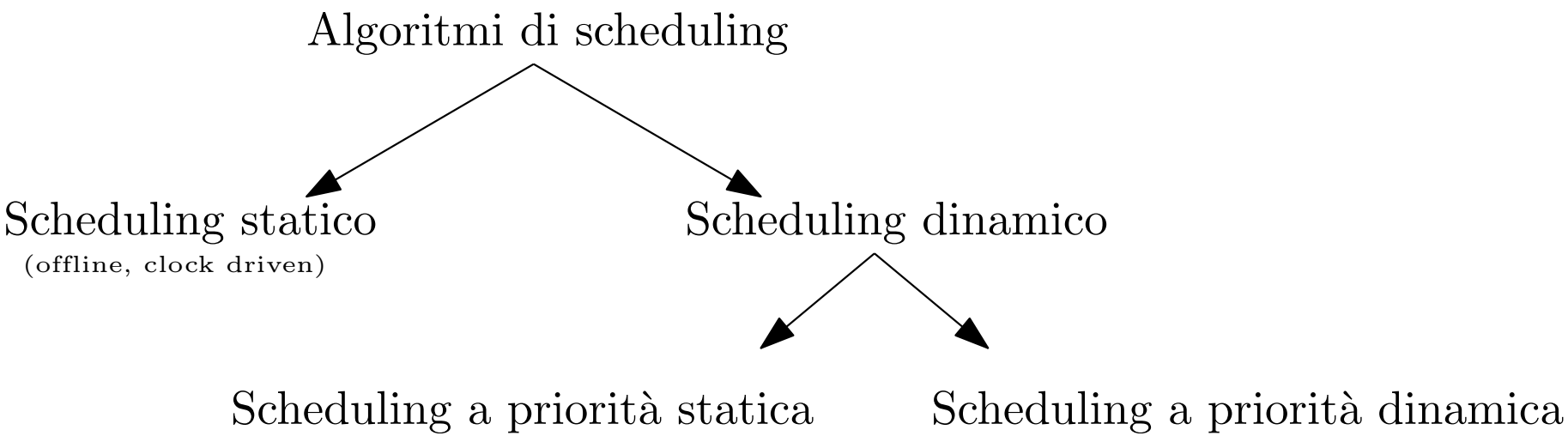
I task possono essere classificati nel seguente modo:

- Periodici. Diventano attivi ad intervalli regolari.
- Aperiodici. Sono ricorrenti ma in modo irregolare.
- Sporadici. Diventano attivi ad intervalli (limitatamente) variabili

I job possiedono le seguenti, principali, proprietà:

- Release time: l'istante di tempo in cui diventa pronto per esser eseguito.
- Deadline: l'istante di tempo entro il quale deve essere completato.
- Response time: la differenza tra il release time e il suo completamento.




Algoritmi di scheduling
Scheduling statico
(offline, clock driven)
Scheduling dinamico
Scheduling a priorità statica
Scheduling a priorità dinamica

Definition

Uno schedule per un insieme di task è valido se si avverano i seguenti presupposti:

- Ad ogni processore possiamo assegnare un solo job per istante di tempo.
- Ogni job può essere associato ad al più un processore per istante di tempo.
- Nessun job può essere considerato prima del suo tempo di rilascio.
- Il tempo dedicato al singolo job non può essere meno del suo BCET e più del suo WCET.
- Tutti i vincoli di precedenza tra i task sono soddisfatti.

Definition

Uno schedule è feasible se tutti i vincoli temporali sono soddisfatti.

Definition

Un insieme di job è schedulabile da un algoritmo di scheduling se quest'ultimo produce sempre uno schedule valido.

Definition

Un algoritmo di scheduling è ottimo se produce sempre uno schedule feasible quando esso esiste.

È sempre possibile trovare uno schedule feasible per un insieme di task in un sistema realtime?

dipende

È sempre possibile trovare un algoritmo di scheduling ottimale?.

È sempre possibile trovare uno schedule feasible per un insieme di task in un sistema realtime?

dipende

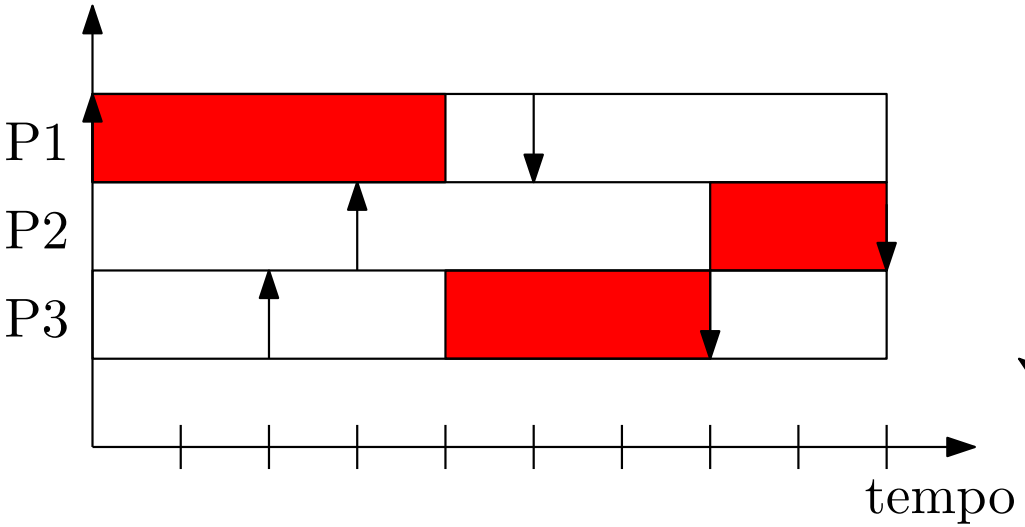
È sempre possibile trovare un algoritmo di scheduling ottimale?.

È sempre possibile trovare uno schedule feasible per un insieme di task in un sistema realtime?

dipende

È sempre possibile trovare un algoritmo di scheduling ottimale?.

Consideriamo il caso di avere un solo processore.
In letteratura sono noti molti algoritmi di scheduling, alcuni dei quali sono ottimi. Citiamo al riguardo EDF.



job indipendenti

monoprocessore

preemption

priorità dinamica

Earliest Deadline First

Cosa cambia se siamo in presenza di più processori?

Non possiamo più applicare le tecniche usate nel caso dei monocore. Ad esempio, EDF non è più ottimale!

Possiamo scegliere di cercare algoritmi di scheduling globali o partizionare.

Cosa cambia se siamo in presenza di più processori?
Non possiamo più applicare le tecniche usate nel caso dei monocore. Ad esempio, EDF non è più ottimale!

Possiamo scegliere di cercare algoritmi di scheduling globali o partizionare.

Cosa cambia se siamo in presenza di più processori?
Non possiamo più applicare le tecniche usate nel caso dei monocore. Ad esempio, EDF non è più ottimale!

Possiamo scegliere di cercare algoritmi di scheduling globali o partizionare.

Nel caso di multicore il tempo di calcolo è fondamentale. Devo usarlo tutto, altrimenti potrei avere problemi.
L'idea alla base del Priority Fair Scheduling è che, nel tempo, ogni job avanza un po', ogniuno secondo il proprio peso.

$$W_i = \frac{C_i}{T_i}$$

Ad ogni istante di tempo t, ogni task τ_i deve aver potuto eseguire per almeno $\lfloor W_i \times t \rfloor$ al più $\lceil W_i \times t \rceil$ istanti di tempo.

Dato un insieme di task schedulabile, cosa accade se modifichiamo in meglio alcuni parametri?

Molto spesso, i sistemi reali differiscono dai modelli.

Le assunzioni pessimistiche che si fanno, quanto sono davvero pessimistiche?

Definition (Sustainable scheduling policy)

Data una policy di scheduling A, sia τ un qualsiasi task sporadico A-schedulabile. Sia Υ l'insieme dei job generati dal task τ . A è sustainable se e solo se A soddisfa tutte le deadline anche nei casi in cui:

- i requisiti sul tempo di esecuzione vengono decrementati.
- le deadline vengono allungate.
- i job arrivano ad un istante di tempo più grande. (il successivo deve almeno arrivare dopo T_i istanti di tempo).

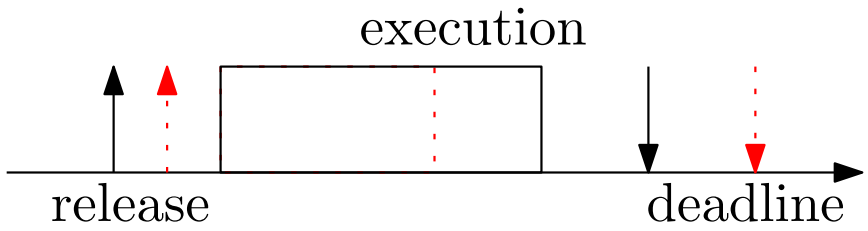
Definition (Sustainable schedulability test)

Consideriamo una policy di scheduling A e un test di schedulabilità F per sistemi con task periodici. Sia τ un qualsiasi task sporadico giudicato da F come A-schedulabile. Sia Υ l'insieme dei job generati dal task τ.
Si dice che F è un sustainable schedulability test se e solo se A soddisfa tutte le deadline cercando di schedulare tutti i job creati da Υ anche se alcuni parametri vengono cambiati, come:

- i requisiti sul tempo di esecuzione vengono decrementati.
- le deadline vengono allungate.
- i job arrivano ad un istante di tempo più grande. (il successivo deve almeno arrivare dopo T_i istanti di tempo).

Definition (Self-sustainability)

Un test di schedulabilità è self-sustainability se tutti i task con minor vincoli rispetto ad un task considerato schedulabile dal test sono anch'essi considerati schedulabili.



Theorem (combination of relaxations)

Se una policy di scheduling è sustainable sotto certi parametri in modo individuale, allora lo è anche in una qualsiasi combinazione degli stessi rilassamenti.

- La policy di scheduling di EDF non è influenzata dal WCET.
  - è sustainable per il decremento del tempo di esecuzione.
- EDF è sustainable rispetto ad un arrivo posticipato del job (sporadico).
- EDF non è sustainable per il posticipo della deadline

- La policy di scheduling di EDF non è influenzata dal WCET.
  - è sustainable per il decremento del tempo di esecuzione.
- EDF è sustainable rispetto ad un arrivo posticipato del job (sporadico).
- EDF non è sustainable per il posticipo della deadline

- La policy di scheduling di EDF non è influenzata dal WCET.
  - è sustainable per il decremento del tempo di esecuzione.
- EDF è sustainable rispetto ad un arrivo posticipato del job (sporadico).
- EDF non è sustainable per il posticipo della deadline

- La policy di scheduling di EDF non è influenzata dal WCET.
  - è sustainable per il decremento del tempo di esecuzione.
- EDF è sustainable rispetto ad un arrivo posticipato del job (sporadico).
- EDF non è sustainable per il posticipo della deadline

- FP-scheduling è sustainable rispetto al decremento del tempo di esecuzione.
- FP-scheduling è sustainable rispetto ad un arrivo posticipato del job (sporadico).
- Le FP-scheduling policy sono sustainable rispetto al posticipo della deadline.

- FP-scheduling è sustainable rispetto al decremento del tempo di esecuzione.
- FP-scheduling è sustainable rispetto ad un arrivo posticipato del job (sporadico).
- Le FP-scheduling policy sono sustainable rispetto al posticipo della deadline.

- FP-scheduling è sustainable rispetto al decremento del tempo di esecuzione.
- FP-scheduling è sustainable rispetto ad un arrivo posticipato del job (sporadico).
- Le FP-scheduling policy sono sustainable rispetto al posticipo della deadline.

Le soluzioni conosciute per uno scheduler globale sono varianti di PF-scheduler.

Theorem

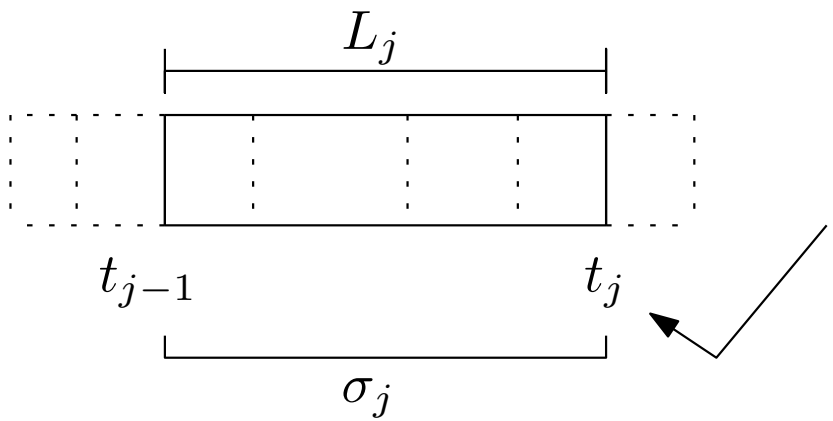
Non può esistere uno scheduler on-line ottimale per un insieme di job, con due o più deadline distinte, su un sistema multiprocessore ($m > 1$)

Ma il teorema non si applica per i task che hanno deadline uguali!

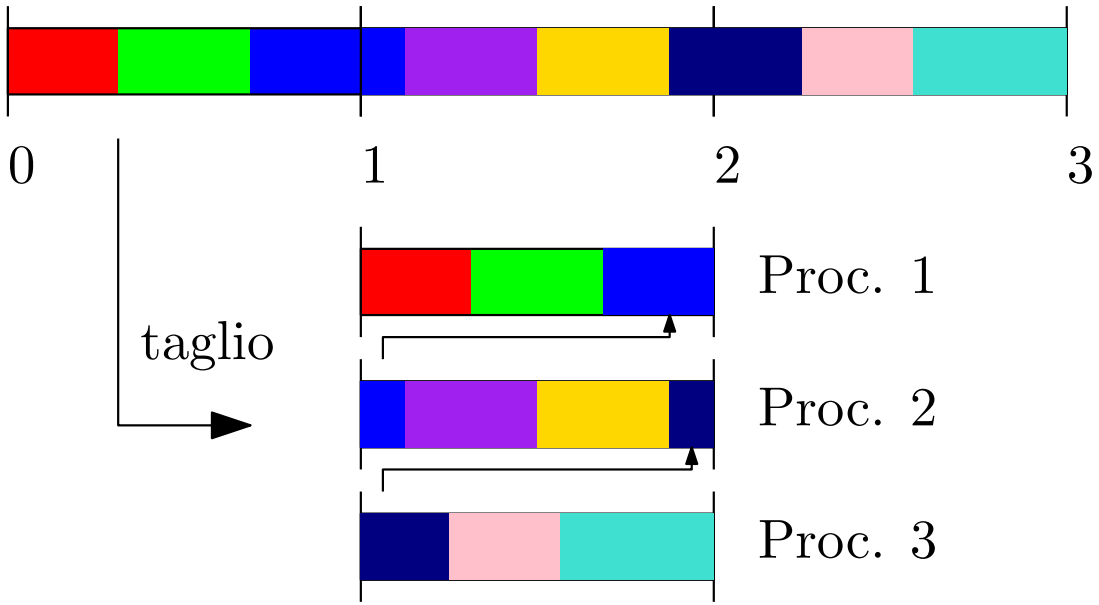
Definition (DP-Fairness)

Un algoritmo di scheduling è DP-fair se schedula i job in uno slice σ_j secondo le seguenti tre regole:

- regola 1: Esegue sempre un job con laxity pari a zero.
- regola 2: Non esegue mai un job che non ha più lavoro da fare.
- regola 3: Non alloca più di $(S(\tau) \times L_j) + F_j(t)$ unità di tempo inattivo in σ_j prima del tempo t.



Vediamo una applicazione: Algoritmo DP-Wrap.



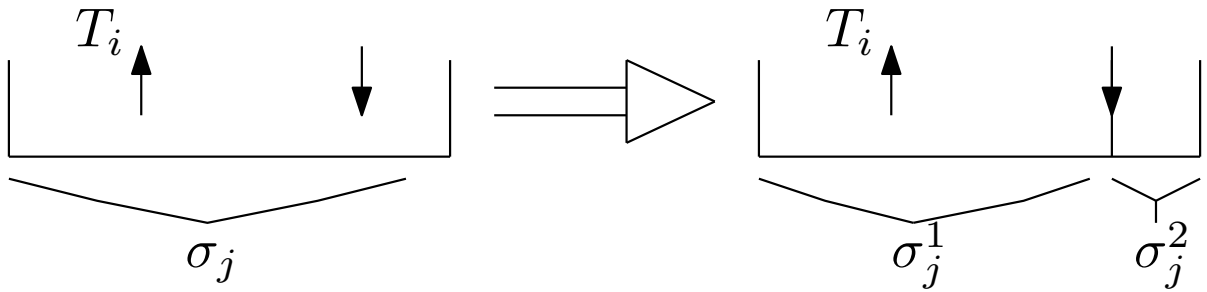
Per ogni task, creo un blocco di lunghezza δ_i e li metto in fila su una linea numerata (da 0). Divido dopo ogni unità.
La lunghezza totale sarà inferione a m. Abbiamo migrazione di job.
Ripeto per ogni slice di tempo.

Funziona per i task sporadici?

In generale, la soluzione non è più ottimale!

Supponiamo di essere nella slice σ_j

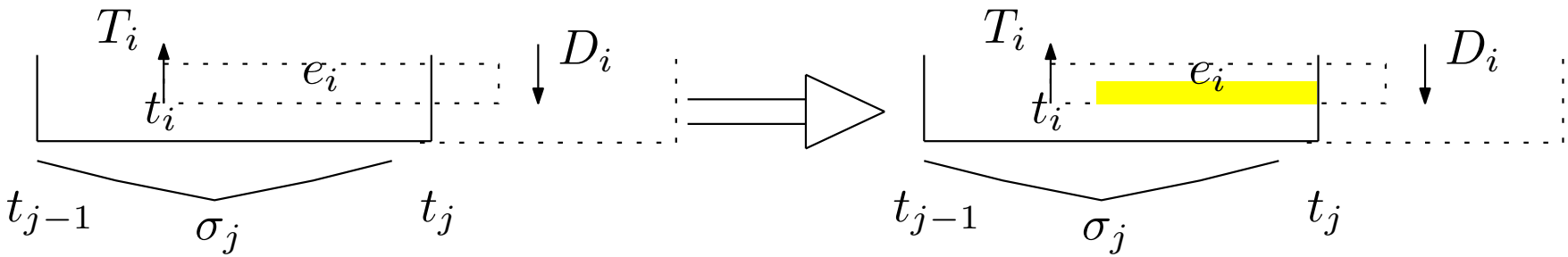
- regola 4: Inizializziamo $l_{i,t_{j-1}}$ a 0. All'inizio del tempo t' di ogni attività di T_i in σ_j che finisce in $t'' = \min\{a_{i,h} + D_i, t_i\}$, incremento $l_{i,t}$ di $\delta_i \times (t'' - t')$
- regola 5: Un task T_i arriva al tempo t'. Se la sua deadline cade dentro la slice σ_j, divido la slice σ_j in due sottoslice σ_j^1 e σ_j^2 tali che la slice σ_j^1 si concluda sulla deadline di T_i. In modo analogo suddivido tutte le capacità dei job in esecuzione.



Modifichiamo l'algoritmo DP-wrap nel seguente modo

Supponiamo che il task T_i generi un task al tempo t' dentro lo slice σ_j

- Se $t' + D_i \geq t_j$. Allochiamo $l_{i,t'} = \delta_i \times (t_j - t')$ e l'esecuzione di tale carico di lavoro viene posta alla fine dello schedule esistente.
- Se $t' + D_i < t_j$, dividiamo lo slice σ_j come spiegato nella regola 5. Il carico di lavoro di T_i viene assegnato completamente a σ_j^1. Il resto dei carichi di lavoro viene proporzionalmente suddiviso tra σ_j^1 e σ_j^2.



Theorem

Qualsiasi DP-fair scheduling è ottimale per un insieme di task sporadici con deadline vincolate, dove $\Delta(\tau) \leq m$ *e* $\delta_i \leq 1$.

Se ne deduce che:

- Se decrementiamo il tempo di esecuzione, questo non inciderà sulla schedulabilità, in quanto $\delta_i = \frac{e_i}{D_i}$ e $\Delta(\tau) = \sum_i \delta_i$.
- È sustainable rispetto ad un arrivo posticipato del job (sporadico)
- Cosa possiamo dire riguardo al posticipo della deadline?

Il teorema afferma che le deadline devo essere vincolate. Il vincolo dev'essere il seguente:

$$\forall i : D_i \leq p_i$$

e quindi

$$\delta_i = \frac{e_i}{D_i}$$

aumentando, dunque, il valore di D_i, i valori δ_i e $\Delta(\tau)$ decrescono.
Il teorema ci garantisce che l'algoritmo DP-Fair, in questo caso, rimane ottimale.

In caso di deadline arbitrarie, risolviamo (?) ponendo una deadline fittizia. quindi da

$$D_i > p_i$$

otteniamo

$$D'_i = p_i$$

Questa soluzione non aumenta δ_i. Se la deadline fittizia è raggiunta allora è raggiunta anche quella vera.
D'altro canto, queste nuove deadline potrebbero forzare la creazione di nuove slice non necessarie.