Algoritmi e Strutture di Dati (3^a Ed.)
Real-Time Scheduling

Alan Bertossi, Alberto Montresor

In un problema per l'elaborazione in tempo reale ("real-time"), la correttezza del risultato non dipende soltanto dal suo valore, ma anche da quanto tempo è necessario per produrlo. In altri termini, ci sono delle costrizioni temporali che il risultato deve assolutamente rispettare, in modo da evitare conseguenze che possono essere catastrofiche (come lo scoppio di una centrale nucleare o la perdita del controllo di un aereo).

Una particolare classe di problemi real-time sono quelli di scheduling. Sia dato un insieme di processi periodici che devono essere eseguiti su un processore. Ciascun processo ricorre infinite volte, in accordo ad un tempo di interarrivo regolare e costante nel tempo (il suo periodo), e deve rispettare delle scadenze temporali (o *deadline*). Infatti, ciascuna occorrenza periodica del processo deve essere interamente eseguita prima della fine del suo periodo, cioè prima che si presenti la prossima occorrenza dello stesso processo periodico. Pertanto, si richiede di trovare un ordine in cui eseguire i processi (per infinite volte) in modo da rispettare le scadenze temporali di tutte le occorrenze periodiche dei processi. Gli algoritmi di scheduling che sono usati in pratica dai sistemi operativi real-time usano *prelazione* (*preemption*) dei processi e sono "guidati da priorità" (*priority-driven*). Le priorità sono assegnate una volta per tutte a ciascun processo in accordo ad un prefissato criterio. Ad ogni istante di tempo, viene eseguito il processo pronto per l'esecuzione che ha massima priorità, interrompendo se necessario un processo in esecuzione con priorità minore. Il processo interrotto sarà ripreso più tardi dal punto di interruzione, quando non ci sarà alcun processo pronto con priorità maggiore di esso. Pertanto, se un processo richiede C unità di tempo di computazione, la sua esecuzione può essere frazionata in più spezzoni di lunghezza qualsiasi, la somma dei quali deve ovviamente dare C. In sintesi, il problema REAL-TIME SCHEDULING può essere enunciato come segue:

REAL-TIME SCHEDULING. *Dato un insieme $\{p_1, \ldots, p_n\}$ di processi periodici, dove ciascun p_i è caratterizzato dal tempo di computazione C_i e dal periodo T_i (entrambi interi con $0 < C_i \leq T_i$), è possibile eseguire tutte le occorrenze periodiche dei processi per mezzo di un algoritmo con prelazione guidato da priorità in modo che ciascuna occorrenza periodica di ogni processo sia completamente eseguita prima della successiva occorrenza dello stesso processo, cioè entro la fine del suo periodo?*

In altri termini, la prima occorrenza del processo p_i dovrà essere eseguita nell'intervallo $[0, T_i]$, la seconda nell'intervallo $[T_i, 2T_i]$, la terza in $[2T_i, 3T_i]$, e così via. Il problema è NP-completo, ma può essere risolto in tempo pseudopolinomiale. Un algoritmo guidato da priorità che è molto usato in ambito industriale è l'algoritmo Rate-Monotonic (RM), ideato da Liu e Layland (1972), dove a ciascun processo è assegnata una priorità in accordo al suo tasso di richieste (cioè all'inverso del periodo). In altri termini, l'algoritmo RM assegna priorità più alta al processo con periodo più breve. Pertanto, assumiamo che i processi $p_1, \ldots, p_n$ siano stati ordinati in accordo alla priorità RM, cioè in modo che $T_1 \leq T_2 \leq \cdots \leq T_n$.

Esempio 1 (Rate-Monotonic)**.** Consideriamo due processi p_1 e p_2 con $C_1 = 1$, $T_1 = 3$, $C_2 = 3$, e $T_2 = 5$. Il processo p_1 ha priorità maggiore di p_2, e la prima richiesta di p_1 è eseguita nell'intervallo $[0, 1]$. Successivamente, la prima richiesta di p_2 è eseguita in $[1, 3]$. All'istante 3, giunge la seconda richiesta di p_1, l'esecuzione di p_2 è interrotta, e p_1 è eseguito in $[3, 4]$. All'istante 4 è ripresa l'esecuzione di p_2 interrotta precedentemente, ed è completata nell'intervallo $[4, 5]$, e così via. Lo schedule (infinito) risultante è periodico con periodo 15, il minimo comune multiplo di $T_1 = 3$ e $T_2 = 5$, ed è illustrato nella Fig. 1. □

È possibile dimostrare che un insieme di processi periodici può essere eseguito dall'algoritmo RM se è rispettata la scadenza della prima richiesta di ciascun processo a partire da un istante nel quale avvengono contemporaneamente tutte le richieste dei processi, poiché è questo il caso peggiore nel

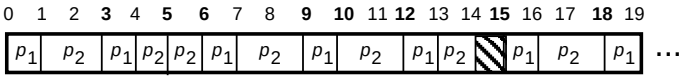

Figura 1: Algoritmo Rate-Monotonic. Esecuzione dei processi periodici p_1 e p_2 con $C_1 = 1$, $T_1 = 3$, $C_2 = 3$, e $T_2 = 5$.

quale il processore è "sovraccaricato" al massimo. Per esempio, un tale istante è l'istante 0 (o qualsiasi istante che sia un multiplo del minimo comune multiplo dei periodi dei processi). Come conseguenza, per verificare se un qualsiasi processo p_i potrà essere eseguito per un numero indefinito di volte, è sufficiente verificare che p_i possa essere eseguito una sola volta nel suo primo periodo $[0, T_i]$ insieme a tutti i processi che hanno priorità più alta di p_i, cioè insieme a $p_1, \ldots, p_{i-1}$. Grazie a questa proprietà, è facile dimostrare che tutte le occorrenze periodiche di p_i possono essere eseguite se e solo se:

$$\min_{0<t\leq T_i} \left\{ \sum_{1\leq k\leq i} C_k \left\lceil \frac{t}{T_k} \right\rceil \right\} \leq t$$

Intuitivamente, $\lceil t/T_k \rceil$ fornisce il numero di occorrenze di p_k nell'intervallo $[0, t]$, e quindi $C_k \lceil t/T_k \rceil$ rappresenta il tempo di computazione richiesto per eseguire tali occorrenze in $[0, t]$. Pertanto, p_i può essere completato in $[0, t]$ se e solo se il tempo di computazione richiesto dalle occorrenze dei processi p_1, $\ldots$, p_i non supera t. Se il minimo valore di t per il quale ciò accade è minore o uguale a T_i, allora p_i può essere eseguito una volta in $[0, T_i]$ insieme a tutti i processi che hanno priorità più alta di p_i. Inoltre, l'insieme $\{p_1, \ldots, p_n\}$ di tutti gli n processi può essere eseguito (un numero indefinito di volte) se e solo se:

$$\max_{1\leq i\leq n} \min_{0<t\leq T_i} \left\{ \sum_{1\leq k\leq i} C_k \left\lceil \frac{t}{T_k} \right\rceil \right\} \leq t.$$

La minimizzazione non è difficile da calcolare. Infatti la condizione su t deve essere verificata per un numero finito di volte. Consideriamo l'insieme $\{p_1, \ldots, p_i\}$ dei primi i processi (dove vale $T_1 \leq \cdots \leq T_i$). Il carico cumulativo $W_i(t)$ del processore richiesto per eseguire i processi in $\{p_1, \ldots, p_i\}$ durante $[0, t]$ è:

$$W_i(t) = \sum_{1\leq k\leq i} C_k \left\lceil \frac{t}{T_k} \right\rceil.$$

Consideriamo una sequenza di istanti $S_{i,0}$, $S_{i,1}$, $S_{i,2}$, ... tali che:

$$S_{i,0} = \sum_{1\leq k\leq i} C_k,$$
$$S_{i,l+1} = W_i(S_{i,l}).$$

Se per qualche ℓ si ha che $S_{i,\ell} = S_{i,\ell+1} \leq T_i$, allora il processo p_i può essere eseguito in $[0, T_i]$. Altrimenti, se $T_i < S_{i,\ell}$ per qualche ℓ, allora p_i non può essere eseguito in $[0, T_i]$. Si noti che $S_{i,\ell}$ è proprio uguale al minimo t, $0 < t \leq T_i$, per il quale $\sum_{1\leq k\leq i} C_k \lceil t/T_k \rceil \leq t$, come richiesto. Questo algoritmo di verifica è chiamato CTT (per "Completion Time Test") ed è ripetuto per $i = 1, 2, \ldots, n$. CTT è stato ideato da Joseph e Pandya (1986) e richiede tempo pseudopolinomiale, poiché la sua complessità dipende dai valori dei periodi dei processi e, in particolare, da quello del periodo maggiore T_n. Infatti,

per ogni i, CTT richiede $O(nT_i)$ tempo. Poiché CTT è ripetuto n volte e $T_1 \leq T_2 \leq \cdots \leq T_n$, in totale è richiesto $O(n^2 T_n)$ tempo.

Esempio 2 (CTT)**.** Consideriamo ancora i processi p_1 e p_2 dell'Esempio 1, e verifichiamo se p_2 è eseguibile. Si ottiene $S_{2,0} = 1 + 3 = 4$, $S_{2,1} = W_2(4) = 1\lceil 4/3 \rceil + 3\lceil 4/5 \rceil = 5$, e $S_{2,2} = W_2(5) = 1\lceil 5/3 \rceil + 3\lceil 5/5 \rceil = 5$. Poiché $S_{2,1} = S_{2,2} \leq T_2 = 5$, tutte le occorrenze periodiche di p_2 rispetteranno le loro scadenze temporali. □