SERT - 02/10/2020 - Modello di riferimento per i sistemi real-time - R03

Di cosa parliamo in questa lezione?

In questa lezione esaminiamo una classe di algoritmi di schedulazione dal comportamento deterministico e facili da validare

1. Algoritmi clock-driven
2. Cyclic executive
3. Suddivisione del tempo in frame
4. Gestione dei job aperiodici

Oggi parliamo di scheduler clock-driven. Sono fondamentalmente una classe di algoritmi di schedulazione molto semplici come formulazione, molto semplici da implementare.

Tipologie di algoritmi per la schedulazione real-time

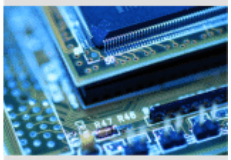
Esistono e vengano utilizzati un gran numero di differenti algoritmi per la schedulazione nei sistemi real-time

La maggior parte di essi possono essere ricondotti a tre grandi famiglie:

1. Algoritmi clock-driven
2. Algoritmi weighted round-robin
3. Algoritmi priority-driven

In questa lezione parliamo di algoritmi clock-driven

Schedulazione clock-driven
Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

In effetti esistono tanti tipi di algoritmi di schedulazione. Gli algoritmi di tipo **clock-driven** che vediamo oggi. Gli algoritmi **round-robin pesati**, che sono tipicamente utilizzati nei sistemi operativi general purpose ma che effettivamente sono una classe di algoritmi più grande. In generale, per i sistemi RT moderni, si tende oggi ad usare algoritmi clock-driven. In effetti oggi parliamo di algoritmi clock-driven.

Algoritmi clock-driven

Un algoritmo di schedulazione è detto essere *clock-driven* se le decisioni riguardanti i job da eseguire e gli intervalli di tempo in cui questi devono rimanere in esecuzione sono determinate in anticipo (off-line) e adottate in istanti di tempo predefiniti

Tipicamente, in un sistema schedulato con un algoritmo clock-driven:

- Gli istanti in cui lo scheduler interviene sono fissati una volta per tutti
- L'insieme dei task periodici, con tutti i loro parametri funzionali ed i vincoli temporali, sono conosciuti e costanti
- Una schedulazione opportuna può essere calcolata "off-line" dal progettista del sistema ed è seguita fedelmente dallo scheduler a "run-time"

Spesso i sistemi che adottano una schedulazione clock-driven utilizzano un componente hardware chiamato clock o timer in grado di generare interruzioni ad intervalli di tempo regolari

Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Come sono definiti? Le decision di un algoritmo di schedulazione *clock-driven* sono prese dallo scheduler soltanto limitatamente a controllare quali job effettivamente siano disponibili per essere eseguiti. In un algoritmo clock-driven è il progettista del sistema che decide quale, tra tutti i job che possono essere eseguiti, debbono effettivamente essere messi in esecuzione. E' il progettista del sistema, quando progetta il sistema, che decide la schedulazione. Lo scheduler, in realtà, non decide chi deve essere messo in esecuzione. Si limita a controllare che i job che effettivamente il progettista ha deciso di mettere in esecuzione, quei job siano presenti. Altrimenti lo scheduler predispone l'esecuzione di altri job, ma sempre tra quelli che il progettista ha detto che devono essere eseguiti.

Lo scheduler interviene, prende queste decisioni sui job che però ha deciso il progettista, ad intervalli di tempo ben definiti. Il tempo viene partizionato, viene suddiviso in tanti istanti, ed in quei istanti lo scheduler interverrà. Lo scheduler interviene in istanti prefissati. Non è uno scheduler che interviene quando succede qualcosa nel sistema. Lo scheduler non ha la libertà di prendere un altro job eseguibile, fuori dall'insieme definito, ed eseguirlo. Per poter realizzare un sistema *clock-driven*, il progettista deve conoscere l'insieme dei task che rappresentano il carico del sistema. Deve conoscere i parametri funzionali come l'istante di arrivo, il tempo di esecuzione, le mutue dipendenze. Tutto deve essere conosciuto dal progettista. E' lui che decide lui l'ordine con cui i job si avvicenderanno sul processore. E' veramente difficile costruire un sistema in cui qualche cosa non è conosciuto nel momento in cui lo progetto. Si tratta di una schedulazione *off-line*, costruita a priori, quando già tutto è noto sui task che caratterizzano il sistema.

Si tratta di un paragone denigratorio, però possiamo pensare al progettista del sistema come al compositore di un'opera orchestrale. Ed allo scheduler come a quel componente che prende la battitura, l'opera, e la esegue, la fa eseguire all'orchestra. Si tratta di un esempio denigratorio e totalmente falso dal punto di vista della logica,

ma come esempio può aiutare a capire che il direttore d'orchestra non ha la libertà di prendere scelte se il compositore non lo aveva previsto. Non può prendere iniziative.

Spesso, lo scheduler deve intervenire ad intervalli di tempo prefissati, che sono stabiliti in fase di progetto. Spesso per realizzare fisicamente questo sistema ci si appoggia a dei componenti hardware che sono dei chip che possono generare dei segnali hardware ad un certo istante di tempo, come i clock ed i timer. Il nome di questa classe di algoritmi deriva proprio da questo.

Perché sono utilizzati?

Quali sono i vantaggi più evidenti degli scheduler clock-driven?

L'algoritmo implementato dallo scheduler è molto semplice, quindi:

- lo scheduler è efficiente (ha un piccolo overhead)
- è facile validare il sistema nel caso di hard real-time

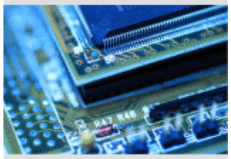
Qual è lo svantaggio più evidente degli scheduler clock-driven?

Lo scheduler è poco flessibile; ad esempio, è difficile gestire insiemi di task non periodici, oppure la creazione di nuovi task a run-time

In passato, la maggior parte dei sistemi embedded hard real-time erano basati su uno scheduler di tipo **clock-driven**

La tendenza generale oggi è quella di adottare quando possibile scheduler **priority-driven**

Schedulazione clock-driven
Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

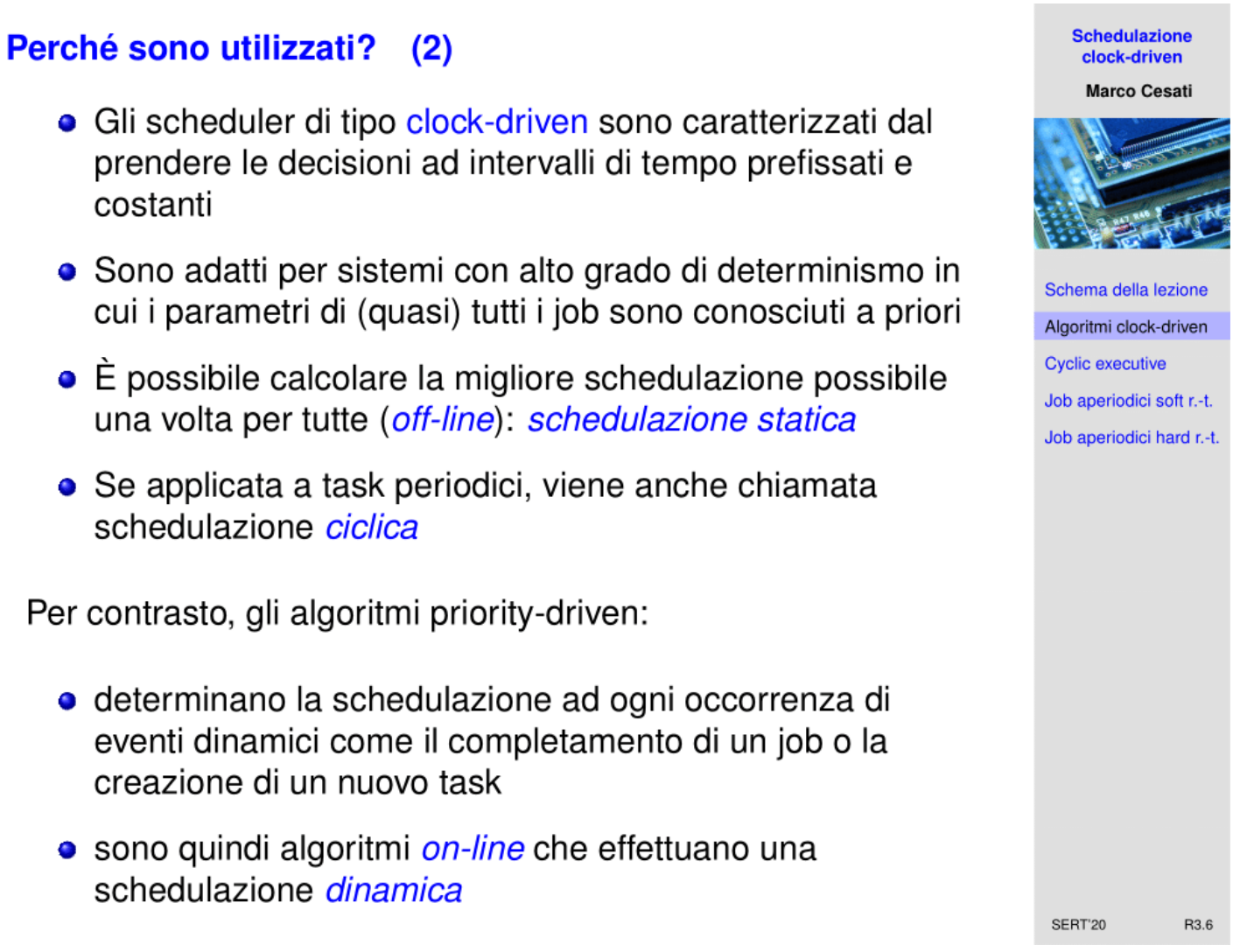
Quali sono i vantaggi più evidenti degli scheduler *clock-driven*? Non è difficile capire che il loro vantaggio principale è l'essere molto semplici. Fondamentalmente lo scheduler è così semplice che è piccolo, occupa poca memoria ed è efficiente. Non deve prendere decisioni o fare calcoli complicati, deve semplicemente *fare l'appello*. E' molto semplice e quindi poca memoria.

L'altro enorme vantaggio è che siccome la schedulazione è fatta a priori quando progetto il sistema, non c'è più questione sul rispettare o meno le scadenze. Se eseguo il job in quel preciso momento allora avrò il rispetto della scadenza. Se non è arrivato non può rispettare la scadenza. Da parte del progettista, convincere l'ente certificatore che il sistema rispetta le scadenze è facile. Non può succedere qualcosa di differente dal punto di vista del software che non faccia rispettare le scadenze. E' facile validare un sistema hard-RT.

D'altra parte però questi scheduler hanno degli svantaggi. Quello più evidente è la mancanza di flessibilità. Se io devo progettare un sistema, ma devo conoscere tutto il sistema, non posso aspettarmi che il tutto possa gestire dei task non previsti in fase di progetto. La differenza è tra costruire un sistema RT, specializzato, tale per cui devo definire in modo completo qualunque task che quel sistema dovrà portare avanti. Altrimenti non posso fare la schedulazione off-line. Oppure un sistema che si, è progettato per un certo ambito, ha certe caratteristiche, ma

ho una certa libertà nel definire i parametri fondamentali del carico che andrò ad eseguire. Il progettista ha progettato il sistema in modo tale che certi parametri possano essere decisi, entro certi limiti, quando il sistema verrà utilizzato. Questa seconda possibilità offre molta più flessibilità.

Nella maggior parte dei casi, in passato si progettavano sistemi hard-RT basati su scheduler clock-driven. Per tutti questi motivi. Ma ultimamente le cose sono cambiate. Ultimamente la tendenza è quella di progettare sistemi RT che siano **priority-driven**. Cioè in cui la schedulazione non è scelta dal progettista ma dall'algoritmo mentre il sistema è in funzione. Perché? Sono più flessibili. L'altro fattore che ha contribuito alla diffusione dei sistemi *priority-driven* è che oggi i sistemi embedded hanno potenze calcolo superiori rispetto al passato. Quindi il vantaggio dell'avere uno scheduler semplice si sta attenuando. Ultimamente i sistemi embedded acquisiscono sempre più potenza di calcolo. Oggi la tendenza è di usare sistemi *priority-driven*. Non è un caso che noi ai sistemi *clock-driven* dedichiamo una singola lezione.



Ricapitolando, gli scheduler di tipo *clock-driven* prendono le decisioni ad intervalli di tempo regolari prefissati, costanti. E le decisioni sono: "E' arrivato o no un job?", ma non quale job devo eseguire. Quindi sono adatti per sistemi con un grado di parallelismo abbastanza alto in cui i parametri di quasi tutti i job sono conosciuti a priori già dalla fase di progetto. E quindi, in fase di progetto, posso calcolare quale è la miglior schedulazione possibile una volta per tutte (**schedulazione statica**). Quando applico questi sistemi *clock-driven* al modello a task periodici, questo tipo di schedulazione viene chiamata **schedulazione ciclica**. Invece, gli algoritmi *priority-driven*, ad ogni invocazione dello scheduler ricalcolano quale è il miglior task da eseguire. Ed intervengono sugli eventi, non conosciuti dal progettista.

Modello a Task Periodici Ristretto

Per ragionare sugli algoritmi di schedulazione ciclica è utile fare riferimento ad una restrizione del modello a task periodici:

- Il numero n di task nel sistema è fissato
- I parametri di tutti i task periodici sono conosciuti a priori
- Ogni job può essere eseguito dal suo istante di rilascio (niente vincoli di precedenza o conflitti sulle risorse)
- Possono esistere job aperiodici con vincoli temporali soft e hard real-time

Notazioni per indicare i parametri di un task periodico T_i:

- (Φ_i, p_i, e_i, D_i): fase Φ_i, periodo p_i, tempo d'esecuzione e_i, scadenza relativa D_i
- (p_i, e_i, D_i): fase uguale a 0
- (p_i, e_i): fase 0, scadenza relativa uguale al periodo p_i

Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Come possiamo ragionare sui sistemi di schedulazione ciclica? E' utile fare riferimento ad una restrizione del modello a task periodici. Nel modello a task periodici abbiamo detto che ci sono tanti task. In realtà il numero di task può cambiare. Nel modello a task periodici ristretto abbiamo un numero n di task periodici fissati. Inoltre, i parametri di tutti i task periodici sono conosciuti a priori. Posso costruire un algoritmo di schedulazione a priorità, anche non conoscendo ad esempio il tempo di esecuzione di un job. Per validarlo lo devo in qualche modo conoscere, ma posso progettare un algoritmo che funziona a prescindere dai tempi di esecuzione dei job. Non devo necessariamente conoscerli in anticipo.

Inoltre, facciamo un'assunzione. Non ci sono vincoli di precedenza, non ci sono conflitti sulle risorse. Ogni job può essere eseguito dal suo istante di rilascio. Questo per studiare questi algoritmi. In effetti non è difficile tenere in considerazione questi vincoli sulle risorse quando progetto il sistema. Però, per fissarci le idee, pensiamo che non ce ne siano.

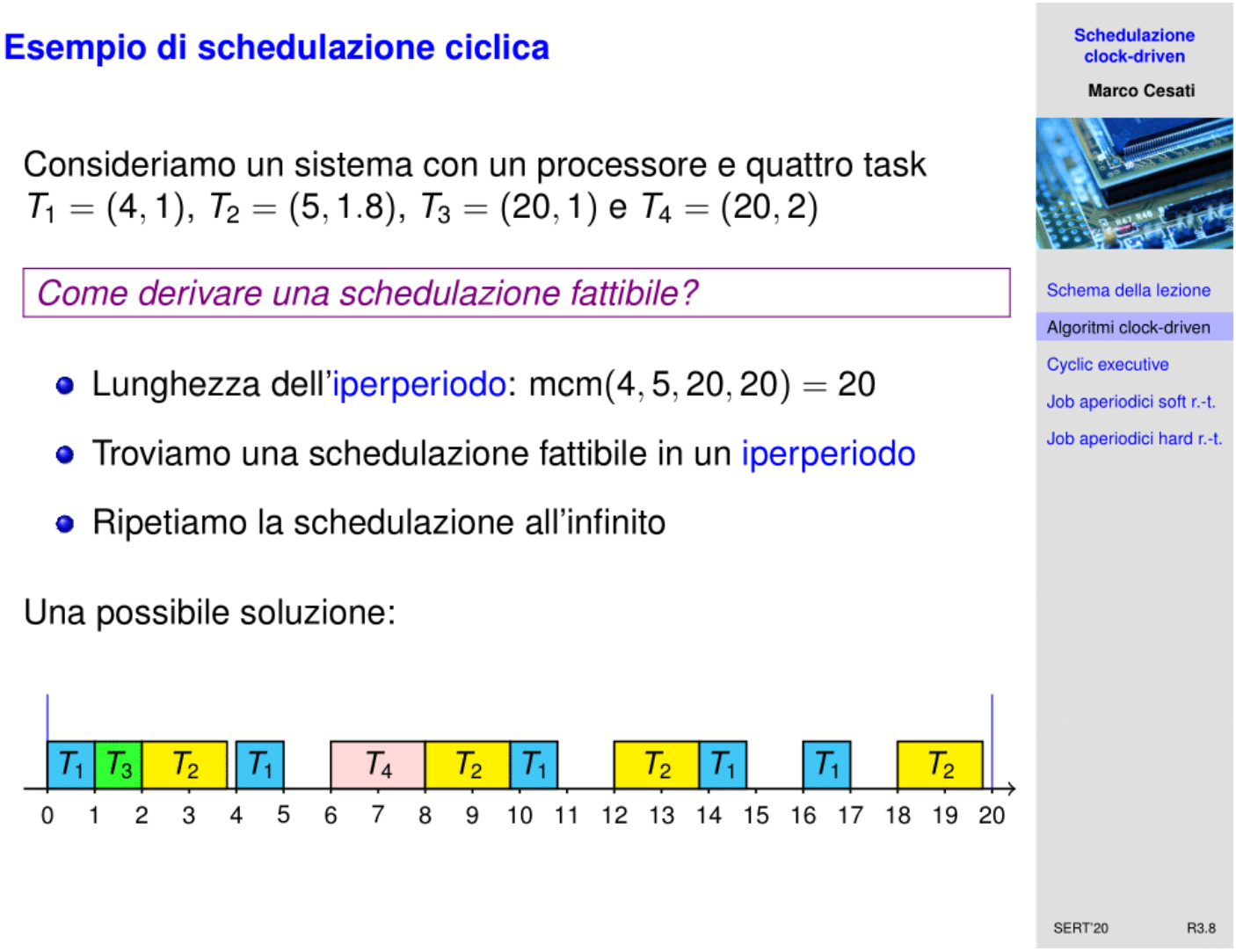
L'esistenza di job aperiodici, cioè job che possono arrivare in qualsiasi momento, può essere presa in considerazione. Vedremo come è possibile cercare di realizzare scheduler *clock-driven* in cui altri job, oltre a quelli periodici, possono arrivare quando vogliono. Saranno abbastanza semplici da gestire per i job soft-RT, in cui il rispetto delle scadenze è secondario. Saranno molto più complicati da gestire quando dovrò gestire hard-RT, in cui dovrò rispettare le scadenze.

Vediamo un po di introdurre qualche notazione per ragionare su questi sistemi. Quando parliamo di task periodico lo definiamo con parametri che sono:

- Fase
- Periodo p_i

- Tempo di esecuzione e_i
  - Il tempo massimo di esecuzione di tutti i job del task
- Scadenza relativa
  - Ciò che determina, per ogni rilascio di un job, la scadenza assoluta. La scadenza assoluta è istante di rilascio più scadenza relativa

Molte volte indico una terna di numeri. Se abbiamo la terna di numeri, indica che la fase è uguale a 0. Una coppia di numeri indica fase uguale a 0 e scadenza relativa uguale al periodo, cioè la scadenza esplicita.



Consideriamo un sistema con 4 task ed un solo processore. Questi task hanno:

- $T_1 = (4,\ 1)$
- $T_2 = (5,\ 1.8)$
- $T_3 = (20,\ 1)$
- $T_4 = (20,\ 2)$

Io sono il progettista e devo progettare l'algoritmo *clock-driven* di questo sistema di 4 task. Come si può fare? La prima cosa da fare è calcolare l'iperperiodo:

- **Iperperiodo:** $mcm(4,\ 5,\ 20,\ 20) = 20$

Cerco all'interno dell'iperperiodo una schedulazione fattibile, stando attendo però. In realtà devo anche tenere in considerazione le fasi dei task. Tutti questi task stanno in fase, tutti cominciano all'istante 0 i primi rilasci, quindi questo non è un problema.

Quando ho trovato una schedulazione fattibile, che rispetta le scadenze implicite, poi è facile. Prendo questa schedulazione che funziona in questo iperperiodo e la ripeto all'infinito. L'iperperiodo è il blocco che ripeto all'infinito.

Quindi, ad esempio, nell'immagine è presente una possibile soluzione. Quello che esce fuori è che in realtà, se vado a controllare, tutti i task rispettano le scadenze. Come facciamo ad implementare via software questa schedulazione?

Scheduler a tabella

Implementazione di uno scheduler clock-driven tramite tabella di voci $(t_k, T(t_k))$:

- t_k indica l'istante in cui una "decisione" è presa
- $T(t_k)$ indica il nome del job o task da eseguire oppure $\mathcal{I}$ se il processore è inutilizzato (*idle*)

La schedulazione precedente è rappresentata dalla tabella:

t_k	0	1	2	3.8	4	5	6	8	9.8
$T(t_k)$	T_1	T_3	T_2	$\mathcal{I}$	T_1	$\mathcal{I}$	T_4	T_2	T_1

10.8	12	13.8	14.8	16	17	18	19.8
$\mathcal{I}$	T_2	T_1	$\mathcal{I}$	T_1	$\mathcal{I}$	T_2	$\mathcal{I}$

- Job aperiodici soft real-time: schedulati negli intervalli "$\mathcal{I}$", ma interrotti se non completati entro il t_k successivo
- Job aperiodici hard real-time: non previsti (in questa versione)

Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Potrei costruire una tabella, in cui ho due voci. Una voce indica t_k, l'istante in cui una decisione è presa, e $T(t_k)$, che rappresenta il nome del job o del task che deve essere eseguito in quell'istante. Oppure I, che indica che il processore non viene utilizzato. La figura di prima in realtà la possiamo descrivere con questa tabella. Questa è esattamente la descrizione vettoriale della figura di prima.

Dopo di che, quando posso gestire i job aperiodici soft-RT? Ovviamente in tutti gli intervalli con I il processore non sta facendo nulla, e posso sfruttarlo per eseguire job aperiodici soft-RT. In realtà però, quando arrivo al t_k successivo, i job vengono interrotti se non completati. I job aperiodici devono essere interrompibili, sia quelli soft che hard RT. In questa versione però, non possiamo tenere conto dei job aperiodici per i job hard-RT.

Pseudo-codice per scheduler clock-driven

Procedura SCHEDULER:

Input: Tabella di schedulazione $(t_0, T(t_0)), \ldots (t_{N-1}, T(t_{N-1}))$

$i = 0$, $k = 0$
imposta il timer a t_k (tempo assoluto)
ripeti:
    accetta interruzioni di timer
    interrompi un eventuale job aperiodico
    job corrente $J = T(t_k)$
    $i = i + 1$, $k = i \bmod N$
    imposta il timer a $\lfloor i/N \rfloor H + t_k$ (tempo assoluto)
    se $J = \mathcal{I}$
        attiva il primo job nella coda aperiodica
    altrimenti
        attiva il job J
    sospendi l'esecuzione dello scheduler

Fine SCHEDULER

Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

A causa di questa tabella, questi sono anche chiamati scheduler a tabella. Fondamentalmente questo è lo pseudocodice. Ho come input la tabella di schedulazione, che sarebbe la *partitura orchestrale*, e successivamente abbiamo l'esecuzione dello scheduler. Abbiamo degli indici i (contatore che viene incrementato per ogni intervallo), mentre k è l'intervallo all'interno dell'iperperiodo. Abbiamo detto che lo scheduler deve intervenire ad intervalli di tempo prefissati. Possiamo farlo andando ad impostare il timer ad un determinato istante di tempo.

Schedulazioni cicliche strutturate

In generale è preferibile lavorare con schedulazioni che soddisfano certe proprietà strutturali, ad esempio:

- Attivazione dello scheduler ad intervalli regolari
- Distribuzione regolare degli intervalli "$\mathcal{I}$" (processore idle) nell'iperperiodo

Che vantaggi portano queste proprietà?

- Lo scheduler può essere attivato da un dispositivo hardware che genera interruzioni periodiche (ad es. il PIT, Programmable Interval Timer)
- I job aperiodici possono essere eseguiti in modo regolare in corrispondenza degli intervalli "$\mathcal{I}$"
- Possibilità di monitorare e/o forzare il rispetto dei vincoli temporali in caso di job che allungano la loro esecuzione

Una procedura che implementa un algoritmo di schedulazione ciclica "strutturata" è chiamata *cyclic executive*

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

In realtà quando si progettano questo tipo di scheduler conviene lavorare con scheduler progettati in maniera un po più sofisticata. Vorrei che le schedulazioni vadano a soddisfare certe proprietà, come che lo scheduler fosse attivato ad intervalli di tempo regolari. L'attività di programmare questi timer è costosa in termini di tempo. Programmare un timer costringe ad arrivare a programmare l'hardware. Sarebbe meglio se riuscissi a farlo una volta per tutte. E' più semplice fare in modo che il clock sia regolare, e quindi che lo scheduler sia attivato ad intervalli regolari.

Inoltre vorrei che, se possibile, questi intervalli I siano distribuiti in modo regolare all'interno dell'iperperiodo. Questa cosa posso farla se ho una certa regolarità del modo in cui vengono generati i clock.

Queste proprietà che vantaggi portano? Posso usare un dispositivo che è disponibile in tutti i sistemi hardware, il PIT (*timer interval programmabile*). I job aperiodici posso eseguirli in maniera regolare. Non devo aspettare quasi tutto un iperperiodo per arrivare al primo intervallo in cui il processore non fa nulla. Posso cercare di sparpagliare gli intervalli I in tutto l'iperperiodo, e fare in modo che i job aperiodici vadano avanti con una certa regolarità.

E poi, sopratutto, se progetto bene lo scheduler posso fare in modo che faccia un controllo. Oltre a capire quali job arrivano e quali no, lo scheduler assume un altro ruolo. Un ruolo di controllo sul fatto che i job schedulati precedentemente abbiano davvero rispettato la loro scadenza. Questo sembra un po un controsenso, in quanto se progettiamo bene il sistema allora il sistema deve rispettare le scadenze. Come dicevamo però, noi progettiamo il software di un sistema hard-RT, ma non tutto è sotto il controllo il controllo del software. Eventi hardware che per esempio rallentano l'esecuzione del processore, possono portare a mancato rispetto delle scadenze. Anche se a regola d'arte il software questo non lo faceva succedere. E' buona norma, è bene costruire anche il software in modo da tener conto che qualcosa possa andar male.

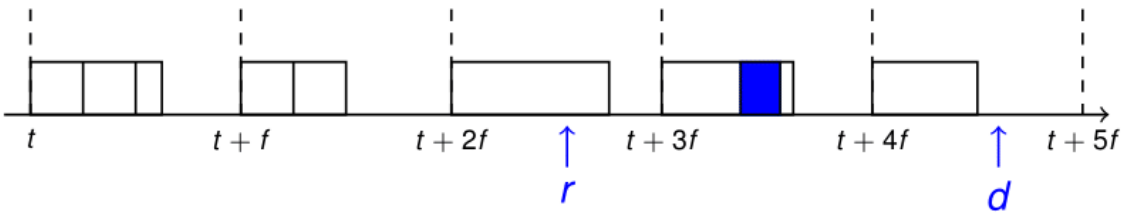
Quindi avere almeno la possibilità di rilevare il fatto che le scadenze siano mancate. Questo tipo di scheduler regolari, esecuzioni cicliche strutturate, permettono di fare questo lavoro. Una procedura che implementa gli algoritmi di schedulazioni cicliche strutturate è chiamato in inglese **cyclic executive**. L'esempio che abbiamo visto alla prima lezione sul flight controller dell'elicottero, quello che è un tipico esempio di **cyclic executive**. C'era un apparato che eseguiva diversi job in un ordine prefissato, ad intervallo di tempo regolari.

Frame

Gli istanti di tempo in cui uno scheduler ciclico strutturato prende decisioni partizionano la linea temporale in intervalli regolari chiamati *frame*

- La lunghezza f dei frame è prefissata
- In ogni frame è definita una lista di job da eseguire in sequenza (*blocco di schedulazione*)
- All'interno dei frame non si possono interrompere i job: se un job inizia in un frame, deve terminare entro lo stesso frame
- La fase di ogni task periodico è un multiplo intero non negativo della lunghezza del frame:

$$\forall i \in \{1, \ldots, n\}, \exists k \in \mathbb{N} : \Phi_i = k \times f$$



Schedulazione clock-driven

Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Gli instanti di tempo in cui lo **scheduler ciclico strutturato** prende decisioni, partizionano la linea temporale in intervalli regolari chiamati frame. Un problema critico è capire quanto è lungo il frame. La lunghezza del frame è fissata dal progettista. Quando devo far lungo questo frame? Consideriamo che dentro ad ogni frame devo definire la lista dei job che vanno eseguiti nel frame che sta partendo. Questa lista è chiamata **blocco di schedulazione**. Il compito dello scheduler all'inizio del frame è di dire: "Di tutti i job che il progettista ha definito dover essere eseguiti dentro questo frame che arriva, quali effettivamente sono stati rilasciati?". Questi che sono stati rilasciati vengono messi nel blocco di schedulazione del prossimo frame di questo frame. Quindi verranno eseguiti uno dopo l'altro senza che lo scheduler intervenga più dentro al frame. Finito il frame, lo scheduler si risveglia. Di questo blocco di schedulazione precedente, quali frame hanno effettivamente rispettato le scadenze e quali le hanno mancate? E poi va a vedere il prossimo frame. Chi dovrebbe essere eseguito? Costruisco un nuovo blocco di schedulazione e così via.

All'interno dei frame i job non si possono interrompere. Se un job inizia in un frame, deve terminare entro lo stesso frame. A meno che io non abbia un modo per suddividere il lavoro di un job in due sotto job che posso eseguire in due frame. Questo si può fare, ma di per se non si può interrompere un unico job. Se il progettista ha un insieme di job prefissati, che non può più suddividere, allora questi job devono essere eseguiti ciascuno dentro ad un frame. Non si può scavalcare il frame.

Un altro vincolo è che la fase di ogni task periodico deve essere un multiplo intero non negativo della lunghezza del frame. La fase è l'istante di rilascio del primo job di un task. Questo istante di rilascio deve cadere esattamente su un frame. In altre parole, per ogni task t_i, la fase del task i deve essere uguale esattamente ad un $k \cdot f$, dove k è un qualsiasi numero naturale, mentre f è la lunghezza del frame. Perché abbiamo questo vincolo? Lo vediamo in figura.

Abbiamo un frame che inizia al tempo t, un altro al tempo $t\ +\ f$ e così via. Ad ogni frame è presente un insieme di job che in qualche modo sono arrivati e che devono essere eseguiti. Il progettista assume che tutti i job che possono arrivare siano arrivati. Per il progettista questi sono esattamente tutti job che devono essere eseguiti all'interno del frame. Se qualche job non arriva, vuol dire che questo frame non avrà il corrispondente job e quindi non avrà più tempo libero. Il punto è che dentro ad ogni frame, il progettista ha piazzato un po di job. Considerando il job in blu, eseguito tra $t\ +\ 3f$ e $t\ +\ 4f$. Il punto è che se arriva all'istante r non può essere eseguito appena arriva, ma bisogna eseguirlo nel frame successivo a quello di arrivo. Se proprio arriva sulla scadenza di un frame e l'inizio del successivo, anche in quello successivo, ma non certamente se arriva in mezzo ad un frame come in questo caso.

Il job ha una scadenza, d. Questa scadenza, se cade in mezzo ad un frame, implica che il job non può essere eseguito in questo frame. Mettiamo che il job sia eseguito nel frame e termini entro la scadenza. Il problema è che chi deve fare il controllo che lui abbia terminato, cioè lo scheduler, e verrebbe invocato in un momento successivo alla scadenza. Quindi lo scheduler dice che il job è completato, ma non sa dire se è completato entro la scadenza oppure no. L'unico modo è se lo scheduler interviene prima della scadenza del job. Dunque ogni job deve essere completato nel frame che precede la sua scadenza altrimenti lo scheduler non può controllare.

Vincoli sulla dimensione dei frame

(1) Poiché non è possibile interrompere un job all'interno di un frame, il frame deve essere abbastanza lungo da garantire la completa esecuzione di ciascun job:

$$f \geq \max\{e_1, \ldots, e_n\}$$

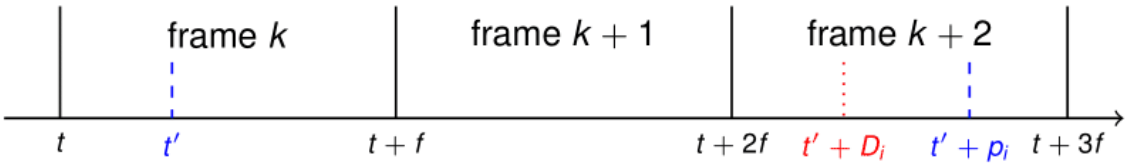
(2) La dimensione del frame deve dividere la lunghezza dell'iperperiodo H:

$$H/f - \lfloor H/f \rfloor = 0$$

Condizione sufficiente è che f divida il periodo p_i di almeno uno dei task.

(3) Il frame deve essere abbastanza piccolo così che tra l'istante di rilascio e la scadenza di ogni job ci sia sempre almeno un frame; condizione sufficiente:

$$2 \cdot f - \gcd(p_i, f) \leq D_i, \quad \forall i \in \{1, \ldots, n\}$$



Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

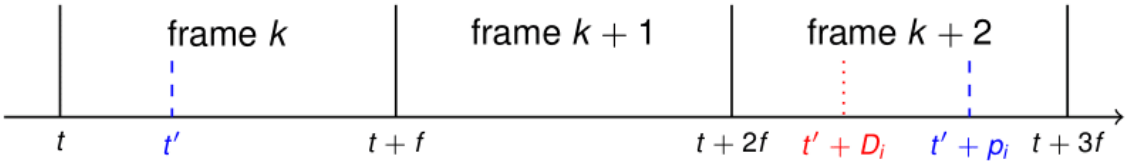
Questo pone alcuni vincoli sulle dimensioni del frame. Ad esempio, non è possibile interrompere un job all'interno di un frame, quindi il frame deve essere abbastanza lungo da garantire la completa esecuzione di ciascun job. Quindi il frame deve essere $\ge$ dei tempi di esecuzione di tutti i task.

Il secondo vincolo è che la divisione del frame deve dividere la lunghezza dell'iperperiodo. Perché? Altrimenti finisce l'iperperiodo ma non finisco un frame, e quindi l'iperperiodo successivo si ritroverebbe in mezzo ad un frame. Nulla di male, ma questo significherebbe che debbo considerare l'iperperiodo successivo come una schedulazione da fare diversa da quella dell'iperperiodo precedente. Non sono più nelle stesse condizioni dell'iperperiodo precedente. Se invece vale la condizione che il frame divide la lunghezza dell'iperperiodo, allora ovviamente quando finisce l'iperperiodo finisce anche un frame. Quando inizia un iperperiodo inizia un frame, e quindi tutti gli iperperiodi sono uguali. La condizioni sufficiente affinché questo sia vero è che il frame divida il periodo di almeno uno dei task. Se questo è vero, è vero anche che divide la lunghezza dell'iperperiodo.

La terza condizione è che, siccome deve esserci un intero frame tra rilasci e scadenze di un intero job, allora questo deve essere abbastanza piccolo. Frame troppo grandi si, sicuramente rispettano la condizione, ma non mi permettono di rispettare la condizione che lo scheduler deve controllare che tutti i job rispettino le scadenze. Se il frame fosse troppo lungo, tra il rilascio di un job e la sua scadenza non c'entrerebbe un intero frame, e quindi il discorso di prima: lo scheduler non potrebbe completare effettivamente che il job ha terminato entro la scadenza. In maniera non ovvia, una condizione sufficiente perché questo sia vero è che $2\cdot f - gcd (p_i, f) \le D_i$. Se questa condizione è vera per tutti i task, allora è vero che tra l'istante di rilascio e la scadenza di ogni job c'è sempre almeno un frame.

Piccola divagazione. In questa parte del corso parleremo di teoremi, parleremo di dimostrazioni. A che servono? Dimostrazioni servono a capire quello che c'è dietro. Fino a che vediamo un teorema che dice qualcosa ma non vediamo la dimostrazione, il teorema rimane qualcosa che si impara a memoria ma che non si capisce veramente il motivo. La dimostrazione, i dettagli, passaggi, possono essere scordati, ma l'idea di fondo rimane. L'idea di fondo sul perché un teorema è vero rimane, ed in qualche modo sono le idee di fondo del perché questa teoria della computazione RT è efficace e funzione. Dunque è essenziale da fare. Spesso e volentieri faremo dimostrazioni.

Vincoli sulla dimensione del frame (2)



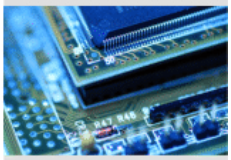
- Frame inizianti a t, $t+f$, $t+2f$, $t+3f$
- Rilasci a $t' \geq t$ e $t'+p_i$, scadenza $t'+D_i$
- $t' = \Phi_i + h' \cdot p_i = h \cdot f + h' \cdot p_i, \quad t = h'' \cdot f \quad (h, h', h'' \in \mathbb{N})$
- Se $g = \gcd(p_i, f)$: $t' - t = g \cdot \left(\frac{h \cdot f}{g} + \frac{h' \cdot p_i}{g} - \frac{h'' \cdot f}{g} \right) = g \cdot h'''$

Caso 1: $t' > t$

- $h''' \in \mathbb{N}, t' - t > 0 \quad \Rightarrow t' - t \geq g$
- $2f - g \leq D_i \Rightarrow \; 2f - (t' - t) \leq D_i \; \Rightarrow \; \boxed{t + 2f \leq t' + D_i}$

Caso 2: $t' = t$

- $2f - \gcd(p_i, f) \leq D_i \quad \Rightarrow \quad f \leq D_i \quad \Rightarrow \quad \boxed{t + f \leq t' + D_i}$

Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Vediamo perché quella condizione è sufficiente per dire che tra rilasci e la scadenza e la scadenza di un task c'è sempre un frame. Abbiamo questa situazione. Supponiamo di avere tre frame. Abbiamo un job che arriva all'istante t'. Il suo successivo rilascio è a $t'+p_i$. Ovviamente questi possono cadere in mezzo ai frame. Inoltre la scadenza assoluta di questo job è $t' + D_i$, ed anche questo cade in mezzo ad un frame. Quello che voglio dimostrare è che tra il rilascio t' e $t' + D_i$, c'è sempre un intero frame. In altri termini, questa distanza è sempre in qualche modo f.

Allora, che cosa possiamo dire? Cosa è t'? E' l'istante di rilascio di un job di quel task. Quindi il primo job di quel task è stato rilasciato all'istante ϕ_i. Questi sono task periodici, quindi tutti gli altri arrivano a multipli del periodo generato da questo ϕ_i. Questo t' quindi è $\phi_i + h' \cdot p_i$. Ma adesso abbiamo una condizione. La condizione era che la fase di ogni task periodico è un multiplo intero, non negativo, della lunghezza del frame: $\phi_i = k \cdot f$, dove k è un numero intero. Quindi questo ϕ_i è $h \cdot f$, dunque abbiamo che $t' = h \cdot f + h' \cdot p_i$, dove h ed h' sono numeri interi. Che cosa è t? E' dove c'è un frame. Iniziano a 0 e ne abbiamo uno ogni f. Dunque $t = h'' \cdot f$.

Supponiamo che, per brevità, $g = gcd(p_i, f)$, allora cosa possiamo dire di $t' - t$? Possiam dire che $t'-t = g (\frac {h \cdot f} g + \frac {h' \cdot è_i} g - \frac {h'' \cdot f} g) = g \cdot h''$. $\frac {h\cdot f} g$ è un numero intero in quanto g è il massimo comune divisore tra pi ed f. Tutti e tre sono numeri interi, quindi tutto è un numero intero. $t' - t$ è un numero intero moltiplicato per g. Il numero intero è ≥ 0, in quanto t' entra nel frame che inizia a t. A questo punto distinguiamo due casi.

1. Supponiamo che $t' > t$
    1. Questo vuol dire che $t'-t >0$, dunque $t' - t \ge g$

    2. $2f - g \le D_i$, $2f - (t'-t)\le D_i$, $t+2f \le t'+ D_i$. Dunque tra il rilascio e la scadenza c'è lo scheduler che interviene e dunque può controllare se il job ha finito entro la scadenza oppure no
2. Supponiamo $t' = t$
    1. Questa condizione significa che $2f - fcd(p_i, f) \le D_i$. In quanto il massimo comune divisore non può essere più grande di f, allora $f \le D_i$. Quindi $t+f \le t' + D_i$

Esempio di scelta della dimensione del frame

Consideriamo un sistema con quattro task

$T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$ e $T_4 = (20, 2)$

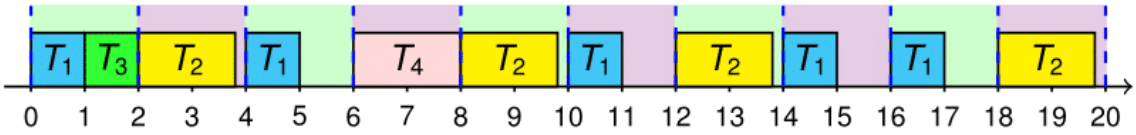
Come scegliere la dimensione del frame?

(1) $f \geq \max\{1, 1.8, 1, 2\} = 2$

(2) $H = 20$, quindi $f \in \{1, 2, 4, 5, 10, 20\}$

(3) $2f - \gcd(4, f) \leq 4$, $2f - \gcd(5, f) \leq 5$, $2f - \gcd(20, f) \leq 20$, quindi $f \leq 2$

Risultato: dimensione del frame $f = 2$



I gruppi di frame consecutivi (a cominciare dal primo frame) di lunghezza pari ad un iperperiodo sono detti *cicli maggiori*; ciascun frame è anche detto *ciclo minore*

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Facciamo un esempio. Prendiamo un sistema che ha quattro task.

- $T_1 = (4,1)$
- $T_2 = (5, 1.8)$
- $T_3 = (20, 1)$
- $T_4 = (20, 2)$

Siccome sono coppie, la fase è sempre 0, e la scadenza relativa è sempre uguale al periodo. Come facciamo a scegliere la dimensione del frame? Seguiamo le regolette che ci siamo dati.

1. Controllo che il frame abbiamo una dimensione maggiore dei tempi di esecuzione di ciascun job
    1. $f \ge max(1, 1.8, 1.2) = 2$
2. Quanto è lungo l'iperperiodo? $H=20$, dunque $f = (1, 2, 4, 5, 10, 20)$. Non può essere diverso, perché altrimenti quando finisce l'iperperiodo il frame non sarebbe finito ed invece no. f deve essere un divisore dell'iperperiodo
3. $2 \cdot f - gcd(4, f) \le 4$, $2 \cdot f - gcd(5, f) \le 5$, $2 \cdot f - gcd(20, f) \le 20$, quindi $f \le 2$
    1. Se vado a fare questi controlli, mi accorgo che il valore 2 va bene. Le condizioni sono sempre rispettate

Prendo il frame uguale a 2, dunque il mio iperperiodo lo partiziono in un certo numero di frame. In realtà nel *cyclic executive*, la lunghezza dell'iperperiodo è chiamato **ciclo maggiore**, è quello che si ripete all'infinito sempre uguale. Ciascun frame è chiamato **ciclo minore**. Dopo di che non ho finito in quanto devo *piazzare* dentro questo frame di questo ciclo maggiore i vari job. Questo lo deve fare il progettista. In realtà, se il sistema è complesso, ci sono tanti job ecc. ci sono dei tool che fanno questo in modo automatico. Noi, per fare l'esercizio, facciamo a mano.

Esempio di scelta della dimensione del frame (2)

Consideriamo un sistema con tre task

$T_1 = (4, 1),\ T_2 = (5, 2, 7),\ T_3 = (20, 5)$

Come scegliere la dimensione del frame?

(1) $f \geq \max\{1, 2, 5\} = 5$

(2) $H = 20$, quindi $f \in \{1, 2, 4, 5, 10, 20\}$

(3) $2f - \gcd(4, f) \leq 4, \quad 2f - \gcd(5, f) \leq 7,$
$2f - \gcd(20, f) \leq 20$, quindi $f \leq 4$

Risultato: non esiste una dimensione adatta!

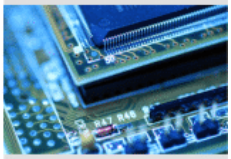
Come si può rimediare?

Il problema è dovuto al vincolo (1): possiamo spezzare uno o più job in modo da ridurre i tempi di esecuzione

Ovviamente, la possibilità di farlo realmente dipende dalla natura dei job troppo lunghi

Schedulazione clock-driven

Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Vediamo un altro sistema di task. Qui abbiamo tre task.

- $T_1 = (4,1)$
- $T_2 = (5,2,7)$
- $T_3 = (20, 5)$

Come scegliamo la dimensione del frame?

1. $f \ge max(1,2,5) = 5$
2. $H = 20$, quindi $f = (1, 2, 4, 5, 10, 20)$
3. $2f - gcd (4,f) \le 4$, $2f - gcd (7,f) \le 7$, $2f - gcd (20,f) \le 20$, quindi $f \le 4$

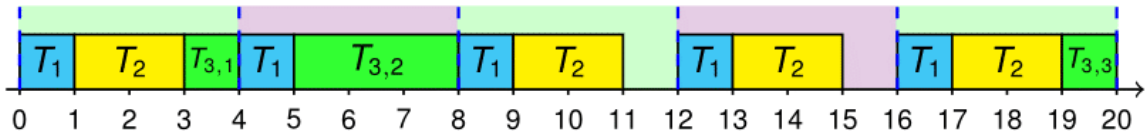
I valori di f che vanno bene solo solamente $1,\ 2,\ 4$. Qui però ho un problema. Perché in realtà la prima condizione pone $f \ge 5$. Dunque il risultato è che non esiste una dimensione del frame adatta. O non riesco a schedulare un intero job, se è troppo piccolo, ma se è troppo grande per schedulare almeno un intero job risulterebbe che non c'è sempre un frame tra un rilascio ed una scadenza.

Come si può rimediare? Ovviamente il problema è dovuto al primo vincolo. Se il progettista ha la possibilità di spezzare il job T_3 in due job più piccoli, allora riduce il tempo di esecuzione massimo e può ridisegnare il sistema con il primo vincolo rilassato.

Frammentazione dei job

Nell'esempio precedente suddividiamo ogni job di $T_3 = (20, 5)$ in tre frammenti: $T_{3,1} = (20, 1)$, $T_{3,2} = (20, 3)$, $T_{3,3} = (20, 1)$

È quindi possibile scegliere come dimensione del frame $f = 4$



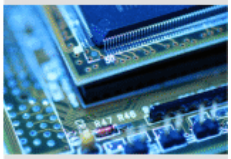
L'insieme di job frammentati è equivalente ai job di cinque task periodici $(4, 1)$, $(5, 2, 7)$, $(20, 1)$, $(20, 3)$, $(20, 1)$? ***No!***

Esistono vincoli di precedenza tra i frammenti!

In generale, per costruire una schedulazione ciclica dobbiamo:

- scegliere una dimensione del frame
- frammentare i job
- piazzare i frammenti nei frame (*blocchi di schedulazione*)

Ma le scelte non sono indipendenti tra loro!



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Per esempio possiamo dire di prendere il job T_3 e lo spezziamo in:

- $T_{3,1} = (20, 1)$
- $T_{3,2} = (20, 3)$
- $T_{3,3} = (20, 1)$

A questo punto posso rifare tutti i conti. **Attenzione però**. L'insieme di job frammentati, è equivalente ai job di cinque task periodici? In realtà non proprio. Questo nuovo insieme di task frammentati impone dei vincoli di precedenza tra i frammenti che non esistevano prima. Dal punto di vista delle precedenze è il progettista che fa la schedulazione che si deve assicurare che non eseguirò un $(3,1)$ prima di un $(3,2)$. Però non si può dire che i task siano equivalenti. Nel primo caso non ho vincoli di precedenza, mentre qui li ho.

In generale quando costruisco una schedulazione ciclica devo scegliere la dimensione dei frame, frammento i job e dopo piazzo i frammenti nel blocco di schedulazione. Le scelte non sono indipendenti tra loro! Come abbiamo visto, posso avere casi in cui devo frammentare e ricominciare.

Come gestire i task non armonici?

Consideriamo i task $T_1 = (3, 1)$, $T_2 = (7, 3)$ e $T_3 = (25, 3)$

L'iperperiodo è $H = 3 \cdot 7 \cdot 25 = 525$, l'unica dimensione ammissibile per il frame è $3 \Longrightarrow$ il ciclo maggiore ha 175 frame

Svantaggio principale: spreco di memoria per la tabella contenente i blocchi di schedulazione

Come gestire la situazione?

Abbassando i periodi di alcuni task nei requisiti di progetto!

Ad esempio: $T_1 = (3, 1)$, $T_2 = (6, 3)$, $T_3 = (24, 3) \Longrightarrow H = 24$ ed il ciclo maggiore ha 8 frame

Abbassare il periodo di un task è come alzare uno stipendio: nessuno si lamenterà, ma debbono esserci soldi per tutti...

Nell'esempio U aumenta di $3/6 - 3/7 + 3/24 - 3/25 = 7.6\%$

Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Vediamo qualche altro problema che si ha quando si progettano questo tipo di sistemi. Uno dei problemi è che i task potrebbero non essere armonici. Cosa è un **task armonico**? I task sono armonici quando i periodi sono uno multiplo dell'altro. Prendiamo questi tre task.

- $T_1=(3,1)$
- $T_2=(7,3)$
- $T_3=(25, 3)$

L'iperperiodo è $H = 3 \cdot 7 \cdot 25 = 525$. Il problema è che questi tre numeri sono tutti coprimi, quindi il minimo comune multiplo è 525. Quindi il ciclo maggiore dello scheduler ha 525 unità di tempo. Se vado a fare i conti come prima, l'unica dimensione ammissibile per i frame è 3. Questo vuol dire che il ciclo maggiore ha 175 frame. Questo è un problema in quanto questi scheduler sono implementati con le tabelle, ed è richiesto un elemento della tabella per ogni ciclo minore. Quando ho tabelle molto grandi, occupo memoria, e questi scheduler in realtà sono pensati per sistemi molto semplici, dove di memoria non ne abbiamo molto.

Quindi devo trovare una soluzione. Quale? La soluzione potrebbe essere di abbassare i periodi dei task nei requisiti di progetto. Cerchiamo di avere dei periodi "meno problematici".

- $T_1 = (3,1)$
- $T_2 = (6,3)$
- $T_3 = (24, 3)$

Da cui deriva che $H = 24$, con il ciclo maggiore che ha 8 frame ora. La tabella ora è molto più piccola. Quando abbasso il periodo di un task, alzo la frequenza. Questo vuol dire che se avevo un certo requisito che quell'attività

dovesse essere svolta abbastanza frequentemente, probabilmente andrà ugualmente bene farla più spesso. Il problema è che questo alza l'utilizzazione del sistema. Sto aumentando il carico di lavoro del processore. In questo caso particolare, il processore ha un carico di lavoro che aumenta del 7.6%.

Come gestire periodi non interi?

Se i periodi dei task non sono interi non è possibile applicare direttamente le formule per i vincoli sulla dimensione del frame

Ad esempio: $T_1 = (1.5, 0.5)$, $T_2 = (2.25, 0.25)$, $T_3 = (3, 0.75)$

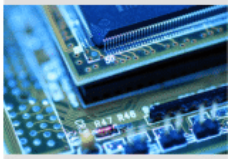
Come si può risolvere il problema?

Moltiplicando tutti i tempi per un fattore costante in modo da ottenere periodi interi

Ad es. moltiplicando per 4: $T'_1 = (6, 2)$, $T'_2 = (9, 1)$, $T'_3 = (12, 3)$

Risolvendo si ottiene ad es. $f' = 6$, ossia un frame di dimensione $f = 1.5$ per il sistema originale

Schedulazione clock-driven
Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Un'altra difficoltà che può arrivare che si può presentare quando lavoro con questi sistemi è quando ho delle specifiche in cui i parametri temporali dei task non sono numeri interi. Invece abbiamo visto che qui il ragionamento è fatto sul massimo comune divisore tra numeri interi, altrimenti non è neanche definito. Come posso fare? Ad esempio qui ho un sistema con tre task.

- $T_1 = (1.5, 0.5)$
- $T_2 = (2.25, 0.25)$
- $T_3 = (3, 0.75)$

Come risolvo il problema? Mi riconduco ad un problema intero. Abbiamo del tempo che è arbitrario. Dopodiché modelliamo il sistema e poi applicheremo questo sistemo al caso specifico e diremmo: *microsecondi*, *millisecondi* ecc. Noi lavoriamo con un'unità di tempo arbitraria. Possiamo moltiplicare tutti i tempi per un fattore costante in modo da ottenere periodi interi. Se moltiplico per 4 tutti questi numeri ottengo:

- $T'_1 = (6,2)$
- $T'_2 = (9, 1)$
- $T'_3 = 12,3)$

Ora posso riapplicare tutto quello ceh abbiamo fatto, ottenendo $f' = 6$.

Pseudo-codice per cyclic executive

Procedura CYCLIC_EXECUTIVE:

Input: Blocchi di schedulazione $L(0), L(1), \ldots, L(F-1)$;
Code di job aperiodici

$t = 0$, $k = 0$
ripeti:
accetta interruzione di clock al tempo $t \cdot f$ (assoluto)
blocco di schedulazione corrente $B = L(k)$
$t = t + 1$, $k = t \bmod F$
gestisci il caso di mancata conclusione dell'ultimo job
gestisci il caso di job (o frammento) in B non eseguibile
sveglia il server dei task periodici per eseguire B
sospendi l'esecuzione fino alla conclusione del server
ripeti finché la coda di job aperiodici è non vuota:
attiva il job in cima alla coda
sospendi l'esecuzione fino alla conclusione del job
rimuovi il job dalla coda
sospendi l'esecuzione (fino alla successiva interr. di clock)

Fine CYCLIC_EXECUTIVE

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Quale è lo pseudo codice per questo tipo di schedulazione ciclica strutturata? Lo scheduler fare questo fondamentalmente: abbiamo una serie di blocchi di schedulazione, che prendono il posto della tabella. Prima avevamo una tabella perché per ogni job da eseguire dovevo anche indicare quando dovevo eseguirlo. Qui lo scheduler interviene ad intervalli di tempo regolari. Però dico quali job andrò ad eseguire in un determinato ciclo minore del ciclo maggiore. Poi avrò anche come input delle code di job aperiodici, dove si accumulano tutte le attività extra che devo compiere.

t è un indice che viene incrementato ad ogni ciclo, e k è lo stesso numeretto però rapportato all'iperperiodo: è l'indice del ciclo all'interno dell'iperperiodo. In ogni iterazione ci prepariamo ad accettare l'interruzione di clock al prossimo frame, dico che B è il blocco di schedulazione corrente ed aggiorno gli indici. Controllo se i blocchi precedenti hanno terminato tutti l'esecuzione. Gestisco il caso in cui il nostro blocco abbia dei job che non sono stati rilasciati, compito dello scheduler. Sostanzialmente quindi dirà al componente che mette in esecuzione i job dentro ad un ciclo minore che un job non è eseguibile. Dopodiché risveglia il componente ed eseguirà i job periodici. In effetti, a questo punto sospende l'esecuzione fino a che il server dei task periodici conclude il lavoro. Che vuol dire? Questo scheduler effettua il lavoro in corrispondenza del frame, e quindi lancia l'esecuzione di tutti i job dentro al frame. Ma poi, quando questi job sono terminati, si risveglia ed adesso mette in esecuzione i job aperiodici. Fin quando la coda dei job aperiodici non è vuota, metto in esecuzione dei job aperiodici e continuo fino a che o il job è concluso, e quindi prendo il prossimo, oppure la coda si svuota. Quando la coda è vuota sospendo l'esecuzione fino alla prossima interruzione di clock.

Schedulazione di job aperiodici soft real-time

In un cyclic executive riveste molta importanza la gestione dei job aperiodici soft real-time:

- Sono eseguiti “in background” quando il processore non è occupato dai task periodici
- La loro esecuzione può essere ritardata
- Sono tipicamente attivati in conseguenza di eventi esterni
- Più rapido è il loro completamento, migliore appare la reattività del sistema rispetto ai segnali esterni

Minimizzare i tempi di risposta dei job aperiodici soft real-time è un obiettivo di progetto degli algoritmi di schedulazione

Come è possibile ottenere questo risultato se in ogni frame dobbiamo comunque gestire i task periodici?

Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Come si fanno a gestire i job aperiodici soft-RT? Abbiamo visto che c'è una coda e lo scheduler li mette in esecuzione quando il processore non ha più niente da fare. In realtà però nel cyclic esecutive, questi job RT devono essere gestiti in modo particolare. Perché sono eseguiti si in background, cioè quando il processore non è occupato dai task periodici, possono essere ritardati, però sono tipicamente attivati in conseguenza di eventi esterni. Questi eventi esterni spesso e volentieri sono eventi di operatori umani. Quindi una specifica di progetto vuole in generale che questi eventi esterni, questi job aperiodici, siano in qualche modo posti in esecuzione nel più breve tempo possibile. Non è vero per i task periodici hard-RT, l'unico criterio è il rispetto della scadenza, ma per i job aperiodici soft-RT un criterio di progetto, un'ottimizzazione da fare sul sistema, è minimizzare i loro tempi di risposta. E quindi in qualche modo migliorare la reattività del sistema verso i segnali esterni.

Sintetizzando, minimizzare i tempi di risposta dei job aperodici soft-RT è generalmente un obiettivo di progetto degli algoritmi di schedulazione. Questa cosa ovviamente come si può fare nell'ambito di uno schema cyclic executive in cui ai job aperiodici do il tempo di background, quello che rimane dall'esecuzione dei job periodici? Possiamo inventare qualche cosa per cercare di mettere in esecuzione i job aperiodici in modo che rispondono prima quando arrivano?

Slack stealing

Una tecnica per migliorare i tempi di risposta dei job aperiodici soft real-time chiamata *slack stealing* è stata proposta da Lehoczky e Ramos-Thuel nel 1992

Per ogni frame k, sia x_k l'ammontare di tempo già allocato, e sia $f - x_k$ lo slack (margine di tempo ancora disponibile)

In ogni frame, lo scheduler può eseguire i job aperiodici **prima** di quelli periodici se lo slack non è nullo

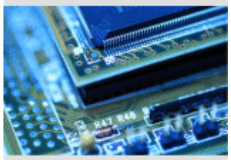
L'implementazione richiede:

- di calcolare la quantità di slack in ogni frame lasciata libera dai job periodici
- di tenere traccia della quantità di slack consumata dai job aperiodici durante la loro esecuzione

  si può utilizzare ad esempio un interval timer impostato all'inizio del frame con il valore dello slack disponibile

È possibile utilizzare lo slack stealing anche con algoritmi priority-driven, ma è molto più complicato

Schedulazione clock-driven
Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

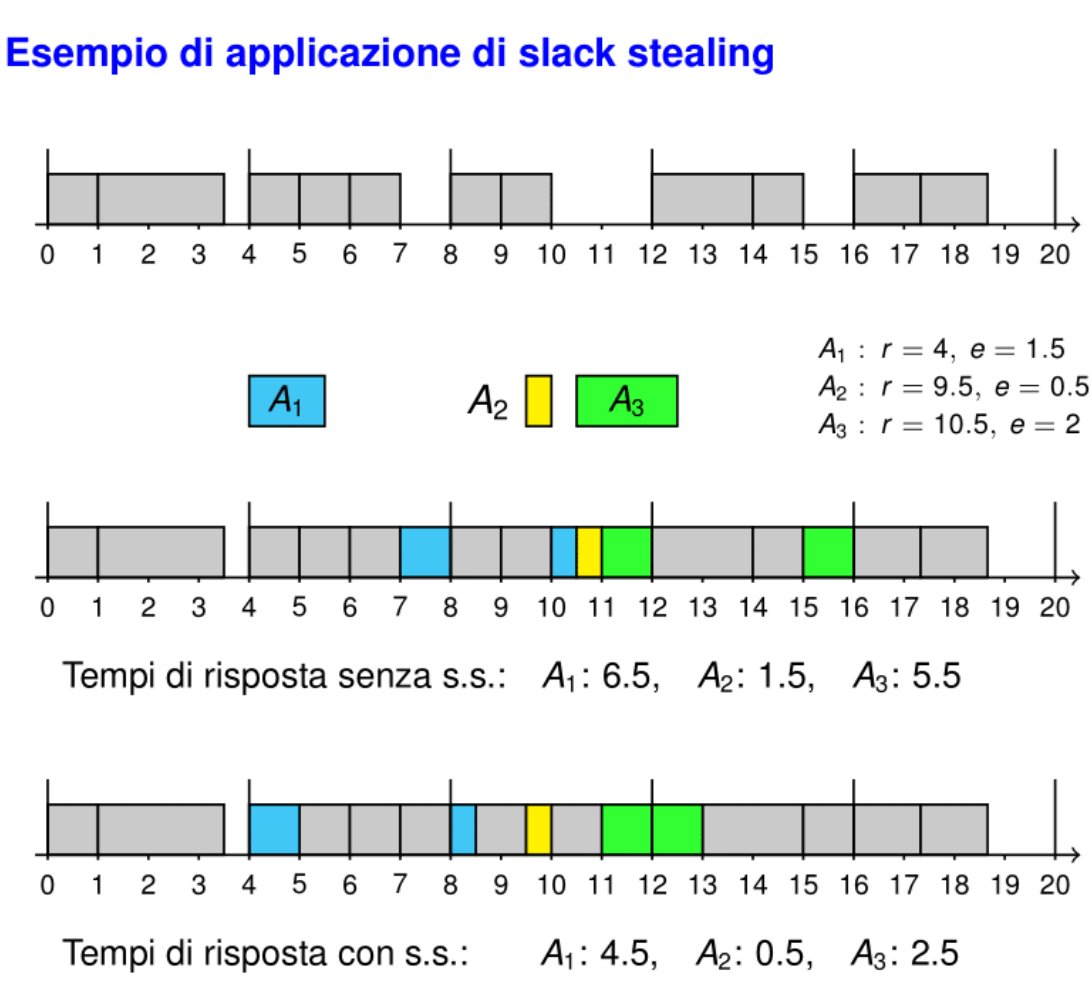
La risposta è Si, ed è una tecnica inventata nel '92 chiamata **slack steealing**. Che cosa è lo **slack**? E' il margine di tempo che rimane ad un certo job prima che la sua scadenza sia mancata. Chiamiamo, dentro ad un frame, x_k l'ammontare di tempo già allocato nel frame. Il progettista ha definito una schedulazione per cui in un certo frame ha detto che verranno eseguiti dei job. Questo tempo già allocato per i job è chiamato x_k. Chiamo la parte del frame in cui non c'è già un job allocato **slack**. E' il margine di tempo ancora disponibile all'interno del frame per fare altro. In ogni frame, possiamo dire che lo scheduler può eseguire job aperiodici *prima* di quelli periodici se lo slack non è nullo. Cioè se ci si può permettere di spendere un po di tempo senza che i job periodici finiscano oltre il frame e quindi potenzialmente oltre la scadenza.

Quindi l'implementazione:

- Calcola la quantità di slack in ogni frame lasciata libera dai job periodici
- Tengo traccia della quantità di slack consumata dai job aperiodici
  - Se io alloco un job aperiodico su un frame, in realtà sto riducendo lo slack a disposizione in quel frame. E devo fare in modo di interrompere l'esecuzione del job aperiodico non appena lo slack a disposizione si riduce a 0. A quel punto non c'è più tempo epr i job aperiodici, ma bisogna passare ad eseguire i job periodici

Un modo per fare questo è programmare un **interval timer**, un dispositivo hardware che dopo un tempo prefissato genera un interruzione hardware ceh forza l'esecuzione dei job periodici al posto di quelli aperiodici. Questo interval timer lo posso impostare al valore dello slack. Quando il timer scade vuol dire che lo slack si è ridotto a 0, quindi eseguo i job periodici. Questo tipo di approccio si può usare anche con algoritmi priority-

driven, ma è molto più complicato. In realtà i job aperiodici negli algoritmi priority-driven sono gestiti in modi diversi.



Facciamo un esempio. Abbiamo questo sistema schedulato su questo iperperiodo di dimensione 20 ed abbiamo diversi job periodici che occupano il frame del sistema. All'istante 4, arriva un job aperiodico. All'istante 9.5 arriva un altro job aperiodico. All'istante 10.5 arriva un altro job aperiodico.

Senza slack stealing, quali sono i tempi di risposta di questi job aperiodici? E' ovvio che A_1 deve finire all'istante 10.5, in quanto nel frame prima vengono eseguiti i job periodici e successivamente gli aperiodici.

- $A_1 = 6.5$
- $A_2 = 1.5$
- $A_3 = 5.5$

Che succede se uso invece lo **slack stealing**? Quando arriva il job aperiodico, lo scheduler lo pone prima dei job periodici.

- $A_1 = 4.5$
- $A_2 = 0.5$
- $A_3 = 2.5$

La tecnica di **slack stealing** è una tecnica valida per abbassare il tempo di risposta dei job aperiodic anche se i job periodici continuano a rispettare i limiti dei frame che il progettista ha imposto. Dove ogni job doveva concludersi dentro ciascun frame, continua a concludersi dentro ogni frame.

Schedulazione di job aperiodici hard real-time

I job aperiodici hard real-time vengono rilasciati ad istanti di tempo arbitrari, e se ne conoscono i parametri solo dopo il rilascio

A differenza di quelli soft real-time, i job aperiodici hard real-time non possono mancare la loro scadenza

Per ogni schedulazione ciclica è sempre possibile trovare un job aperiodico hard real-time che non può essere completato entro la scadenza a meno di ritardare uno o più job periodici

Anche i task periodici sono hard real-time: come schedulare insieme i job periodici e quelli aperiodici?

Per ogni nuovo job aperiodico hard real-time che viene rilasciato:

- Il cyclic executive invoca un *test di accettazione*
- se il test dimostra che il nuovo job può essere completato entro la scadenza senza danneggiare gli altri job hard real-time, viene accettato e schedulato
- altrimenti il job viene rifiutato

Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Se invece parliamo di job aperiodici hard-RT cioè con scadenze da rispettare, le cose sono molto più difficili. Perché se io accetto un job hard-RT, mi devo prendere l'impegno di rispettare la scadenza esattamente come le scadenze dei job periodici. In generale, questa cosa non è possibile. Nel senso che in generale può sempre arrivarmi, data una schedulazione, un job aperiodico hard-RT che pone un così alto carico sul sistema che in realtà o lui o qualcuno dei job periodici che stanno sotto, mancherebbe le scadenze. Quindi fondamentalmente che cosa significa? Devo trovare un compromesso. Come faccio a schedulare assieme job periodici ed aperiodici se tutti devono rispettare le scadenze e non posso fare previsioni sui parametri temporali dei job aperiodici? La risposta è semplice.

Devo implementare un test di accettazione. Cioè devo fare in modo che quando arriva un job aperiodico, il sistema non lo accetta a priori. Ma dice: fammi controllare se effettivamente sono in grado di soddisfare le tue esigenze oltre a quelle dei task periodici e oltre a quelle di tutti i task aperiodici hard-RT che ancora non ho completato pur avendolo accettati. Questo lavoro in generale è un lavoro complicato, sofisticato. Vedremo che l'algoritmo non sarà difficilissimo, ma comunque è un passo di complessità considerevole rispetto a quello a tabella che vedevamo prima. Se il test dimostra che il nuovo job può essere completato, lo accetto e lo inserisco in quelli da schedulare, di cui mi impegno a mantenere la scadenza. Altrimenti lo rifiuto. Se lo rifiuto, che cosa vuol dire? Che quel job non verrà eseguito. Questa cosa è tollerabile? Dipende dal contesto, dipende dalla situazione, dal sistema reale. Nel flight controller c'è una procedura per inserire un pilota automatico. Potrebbe essere che la procedura per l'inserimento del pilota automatico, evento sporadico, debba essere eseguito da questo job aperiodico che però ha delle scadenze hard-RT. Quello che può succedere è che nel manuale di volo dell'elicottero c'è scritto: *se vuoi inserire il pilota automatico, devi premere questo tasto, ma non puoi assumenere che il pilota automatico sia inserito fino a quando non si accende una certa spia di conferma.*

Quello che potrebbe succedere è che alla pressione del tasto il sistema prova a sottomettere questo job hard-RT, fino al momento in cui non viene accettato. A quel punto si da conferma all'utente che quel job è stato completato ed a quel punto la spia si accende. C'è un'impredicibilità nel sistema, dovuta al fatto che non sappiamo quando questo job verrà accettato. Quando accadrà, verrà portato a termine nei tempi giusti. Ma fin quando non viene accettato, il pilota deve sapere che è come se non avesse mai spinto il bottone. Di fatto dipende dalla situazione reale. Altrimenti bisogna rinunciare ad avere job aperiodici hard-RT, perché in assoluto non possono essere sempre accettati.

Schedulazione EDF per job aperiodici hard real-time

Un modo efficiente di gestire i job aperiodici hard real-time è quello di utilizzare l'algoritmo EDF: ha la precedenza il job con scadenza più vicina

La procedura cyclic executive utilizza due code di job aperiodici hard real-time ordinate secondo l'istante di scadenza:

- Una coda EDF per i job rilasciati e non ancora accettati
- Una coda EDF per i job accettati

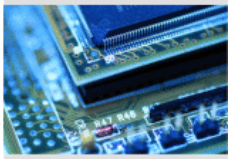
L'accettazione dei job e la schedulazione della loro esecuzione vengono sempre effettuate all'inizio di un frame

L'algoritmo EDF è ottimale, ossia sempre in grado di trovare una schedulazione fattibile per i job aperiodici hard real-time se questa esiste?

È ottimale solo nell'ambito dei vincoli imposti dai frame

Esempio: un job aperiodico potrebbe essere schedulabile solo se fosse eseguito non appena rilasciato in mezzo ad un frame

Schedulazione clock-driven
Marco Cesati



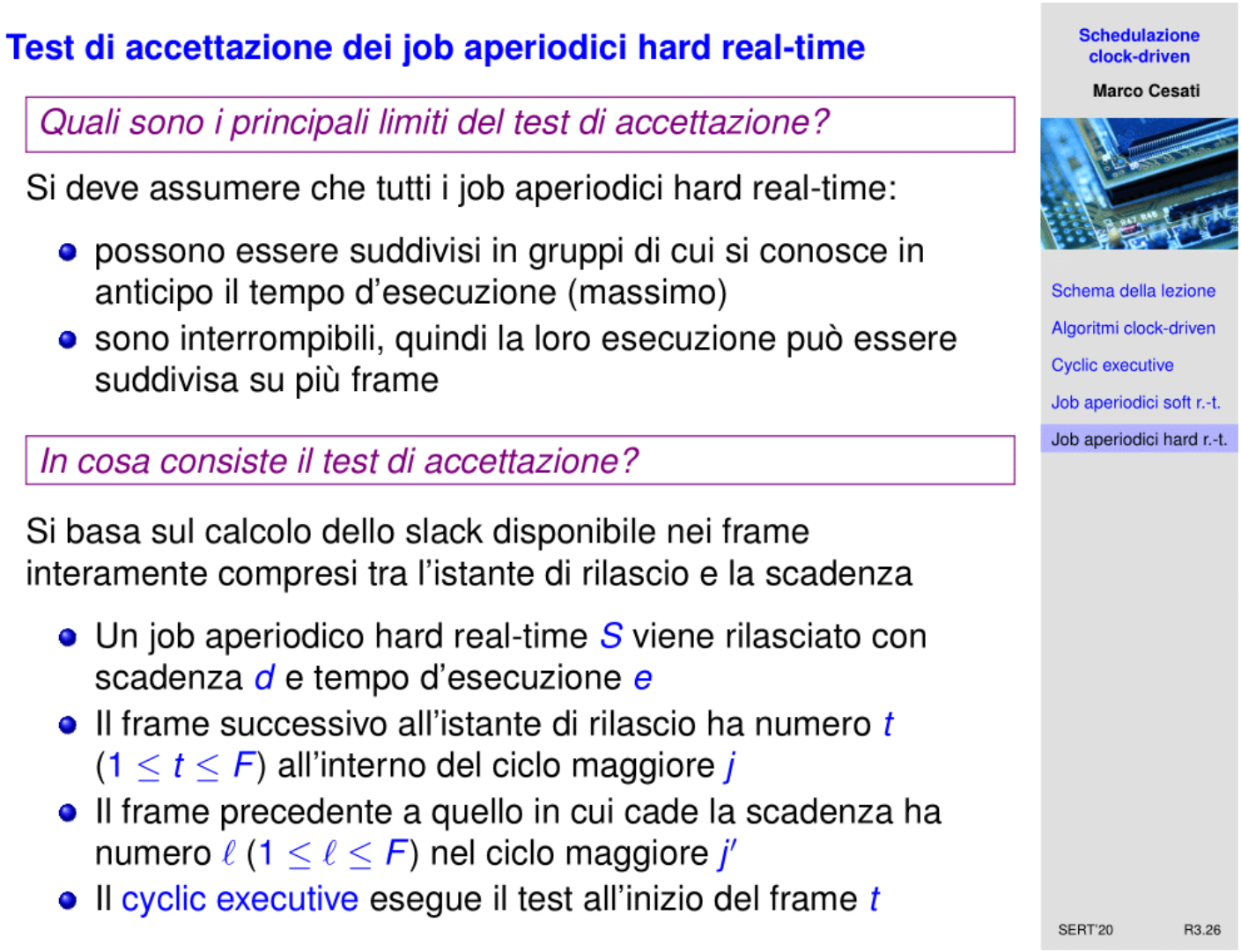
Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Come si fanno a gestire job aperiodici hard-RT? Alla fine c'è da considerare anche questo. Se arriva un job aperiodico hard-RT ed un altro è stato già definito, chi viene prima? In realtà, come vedremo, un metodo efficace per fare rispettare le scadenze di tutti è quello di utilizzare l'algoritmo **EDF**. Questo algoritmo lo vedremo più avanti quando parliamo di algoritmi priority-driven. La precedenza ce l'ha il job che la scadenza più vicina. Quindi fondamentalmente il job hard-RT che definisce una scadenza più vicina dell'altro passa in testa, ha la precedenza. Interrompe anche un job aperiodico che era già stato accettato. Questo ovviamente pone un problema perché quando io accetto un job devo anche controllare se, passando avanti agli altri, non fa mancare le scadenze agli altri. Anche questo deve essere controllato. Il cyclic executive ha fondamentalmente due code di job aperiodici, che sono ordinate secondo l'istante di scadenza.

- In una coda ci sono i job che sono stati rilasciati ma non ancora accettati
- In un'altra coda per i job accettati

Quando viene fatta l'accettazione dei job? La fa lo scheduler quando interviene. Tra le tante cose che fa, farà anche questo in più. Controllerà se la coda dei job rilasciati e non accettati è vuota o no, ed in caso fa i controlli

per accettare o rifiutare i job in arrivo. In corrispondenza di ogni frame viene fatto questo lavoro. In realtà, una domanda un po difficile a cui dare una risposta adesso, questo modo di portare avanti prima i job che hanno scadenze più vicine e poi gli altri, è in grado di soddisfare i vincoli dei job in maniera ottimale se però consideriamo il fatto che i job comunque non possono essere schedulati non appena arrivano. Nell'ambito dei vincoli posti dal cyclic executive, cioè che ci deve essere sempre uno scheduler che interviene all'inizio di un frame e che accetti o rifiuti i job, allora la politica di schedulazione **EDF** è ottimale. Non c'è un algoritmo migliore che possa far rispettare di più le scadenze. Se c'è un algoritmo che rispetta le scadenze, **EDF** allora rispetta le scadenze. Ma, potrebbe esserci il caso in cui un job aperiodico arriva, potrebbe essere schedulato rispettando le scadenze sue e di tutti gli altri, purché sia schedulato non appena arriva. Ma è arrivato in mezzo ad un frame, quindi debbo aspettare il confine di un frame per poterlo accettare. Questo già potrebbe essere sufficiente a non poterlo più accettare e rispettare le scadenze. Se guardo i vincoli imposti dal frame, l'algoritmo **EDF** è ottimale. In realtà però, di per se, non è l'algoritmo migliore, ma questo è un problema del cyclic executive e non dell'algoritmo in se.



Quali sono i limiti di questo test di accettazione? Devo essere in grado di classificare con precisione i parametri temporali dei job che arrivano. Mentre i task periodici devo essere in grado esattamente quando lo progetto il sistema, come sono fatti, in realtà devo farloa nche per i job aperiodici. Quando arrivano devo saper dire quanto tempo durerà qiascun job, quanle sarà la sua scadenza. Posso suddividerli in gruppi, ma di ogni gruppo devono conoscere almeno il tempo di esecuzione massimo. Devono essere interrompibili, perché non posso fare a meno di interromperli per eseguire i job periodici. La loro esecuzione può essere suddivisa su più frame.

Come è fatto il test di accettazione? Calcolo lo **slack** disponibile nei frame dopo i task periodici e dopo tutti i task aperiodici che ho già accettato. Vediamo di capirlo questo test di accettazione. Supponiamo di avere un job

aperiodic oche viene rilasciato in un certo tempo ed ha scadenza d e tempo di esecuzione e. Supponiamo che il frame successivo all'istante di rilascio abbia numero t, dove t è un indice dentro all'iperperiodo. Ho un certo iperperiodo chiamato $j-esimo$ ciclo maggiore, ma arriva esattamente nel $t-esimo$ frame del ciclo maggiore. La scadenza dove cade cade non importa. Quello che importa è dove sta il frame precedente, cioè il confine precedente tra i frame. Quindi supponiamo che il frame precedente a quello della scadenza abbia numero l dentro l'iperperiodo e che sia il ciclo maggiore j'. Quindi in realtà può essere anche una scadenza molto lontana, possono passare anche diversi iperperiodi prima della scadenza. Il cyclic executive esegue questo test all'inizio del frame t, cioè appena interviene lo scheduler e si accorge che c'è un nuovo job aperiodico hard-RT da accettare.

Test di accettazione dei job aperiodici hard real-time (2)

- La quantità di slack $\sigma(i, h)$ lasciata libera dai job periodici è precalcolata per ogni ciclo maggiore ($1 \leq i, h \leq F$)
- Su più cicli maggiori la quantità di slack totale è:

  $$\sigma(i + j\,F, h + j'\,F) = \sigma(i, F) + (j' - j - 1) \cdot \sigma(1, F) + \sigma(1, h)$$

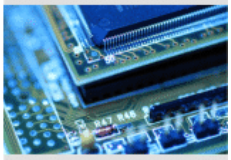
- Per ogni job aperiodico hard real-time S_k già accettato all'inizio del frame t si conosce la scadenza d_k, il lavoro ancora da svolgere $e_k - \xi_k$, lo slack rimanente σ_k
- La quantità totale di slack disponibile tra i frame t e ℓ:

  $$\sigma_c(t, \ell) = \sigma(t, \ell) - \sum_{d_k \leq d} (e_k - \xi_k)$$

  deve essere $\geq e$, altrimenti S viene rifiutato
- In caso di accettazione, per ogni job aperiodico hard real-time con scadenza oltre d lo slack rimanente deve essere diminuito di e: S viene rifiutato anche se $\sigma_k - e < 0$ per qualche k
- Se S è accettato, il suo slack è inizialmente

  $$\sigma = \sigma_c(t, \ell) - e$$

Schedulazione clock-driven
Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Come funziona? Calcolo la quantità di **slack** lasciata libera dai job periodici. Il progettista sa quanto processore è libero dentro ad ogni frame. E' un dato che so a prescindere per ogni ciclo maggiore, per ogni frame so quanto spazio libero ho li dentro. Quindi posso calcolare la quantità di slack totale tra il frame dopo il rilascio ed il frame prima della scadenza. Non è altro che la quantità totale di **slack** dentro tutti cicli interi tra rilascio e scadenza, più la quantità di slack nei frame che finiscono l'iperperiodo dopo il rilascio ed iniziano l'iperperiodo finale prima della scadenza. Dopodiché devo tener traccia dei job aperiodici che ho già accettato, perché anche questi mi sono impegnato a portarli a termini e quindi riducono lo slack a disposizione. Per ogni job aperiodico S_k già accettato, di questo si conosce la scadenza, che è d_k, ma sopratutto devo tenere traccia del lavoro ancora da svolgere, cioè di quanto ancora debbo svolgere per poterlo completare. Ed anche lo slack rimanente di questo job. Cioè di quanto margine ho a disposizione prima che questo job manchi la scadenza.

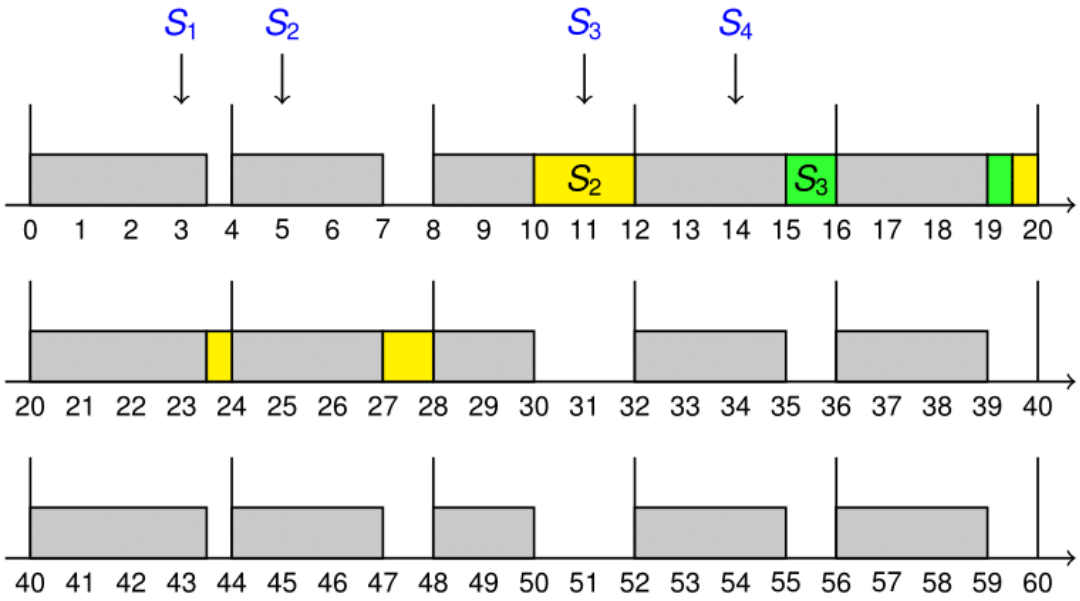
A questo punto posso calcolare la quantità totale di slack disponibile fra i frame t, successi al rilascio, e quello l, precedente alla scadenza. Questa quantità di **slack** disponibile che cosa è? E' la quantità totale di slack dato

dalla formula sulle slide, meno tutto il tempo che devo ancora svolgere per i job hard-RT che hanno una scadenza che precede quella di cui che voglio accettare. Questo perché tutti i job che hanno una scadenza che precede, secondo l'algoritmo **EDF**, hanno una priorità superiore e quindi sicuramente nello slack a disposizione ancora devo togliere questo tempo, perché questo se lo mangiano i job hard-RT che hanno la precedenza su quello che voglio accettare. Se questa quantità di slack è più piccola di e, cioè quanto mi costa eseguire il nuovo job, devo rifiutare perché non avrò tempo per rispettare la scadenza di questo job hard-RT che è arrivato. Ma non basta.

Perché adesso, se dovessi accettare il job, in realtà questo job accettato fa passare in secondo ordine, si mette davanti, a tutti i job già accettati che avevano una scadenza oltre la sua. E quindi questo vuol dire che devo rifiutare anche se qualcuno di questi job che hanno una scadenza oltre, ha uno slack inferiore ad e. Lo slack è il margine di sicurezza che ha quel job hard-RT per finire. Io fino adesso ho un margine di sicurezza di 1. Ma se mi arriva davanti qualcuno che mi ruba 1.2 del processore, devo rifiutare, perché vuol dire che se lo accetto io non avrò tempo per finire. In realtà il test di accettazione fa controlli su tutti i job. Se tutte le condizioni sono soddisfatte, accetterò il nuovo job, ridurrò gli slack degli altri con scadenza oltre, ma imposto anche lo slack del nuovo job: la quantità di slack disponibile meno il tempo di esecuzione del nuovo job.

Il cyclic executive, quando devo gestire i job aperiodici, controllo prima i job hard-RT e lo faccio anche prima dei task periodici. Quindi controllo che la coda dei job hard-RT non accettati sia non vuota. Faccio il test di accettazione se non è vuota e se lo accetto lo inserisco nella coda dei job accettati. Poi eseguo i task periodici (*non ho slack stealing*), dopodiché quando i job periodici hanno terminato, vado a controllare la coda dei job aperiodici hard-RT. Ovviamente quando la coda dei job hard-RT è vuota, allora posso passare a considerare i job soft-RT. Soltanto in quel caso, altrimenti i soft-RT li rimando in avanti. In questa impostazione non abbiamo lo slack stealing. In teoria possiamo aggiungerlo, ma ovviamente complica ancora di più l'algoritmo.

Esempio di schedulazione EDF di job aperiodico hard r.-t.



S_1	$r_1=3$	$d_1=17$	$e_1=4.5$	$\sigma_c(2,4)=4$	RIF
S_2	$r_2=5$	$d_2=29$	$e_2=4.0$	$\sigma_c(3,7)=5.5$	ACC $\sigma_2=1.5$
S_3	$r_3=11$	$d_3=22$	$e_3=1.5$	$\sigma_c(4,5)=2$ $\sigma_2 \geq e_3$	ACC $\sigma_2=0, \sigma_3=0.5$
S_4	$r_4=14$	$d_4=44$	$e_4=5.0$	$\sigma_c(5,11)=4.5$	RIF

Schedulazione clock-driven
Marco Cesati

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Vediamo un esempio. Supponiamo che arrivi all'istante 3 un job aperiodico hard-RT con scadenza 17, tempo di esecuzione 4.5. Quanto è lo slack nel sistema? Questo job lo posso cominciare ad eseguire dal secondo frame in poi. Non posso eseguirlo da quando arriva. Quanto è lo slack rimanente fra il frame successivo ed il frame precedente alla scadenza? Abbiamo i frame che iniziano a 4, 8 e 12. Quanto slack c'è? 4 unità di tempo libere a disposizione. Ma il tempo di esecuzione del job e 4.5, e quindi vuol dire che rifiuto il job in quanto non posso assicurare la scadenza.

Supponiamo che arrivi all'istante 5 un altro job aperiodico, con scadenza a 29 e tempo di esecuzione 4. Lo slack rimanente nel sistema, tra i frame 3 e 7, è di 5.5. Non ho altri job aperiodici hard-RT accettati. Il margine di sicurezza per questo job sarà di 1.5. Possono entrare altri job, che si inseriscono in mezzo, per un tempo massimo di 1.5. Oltre, anche lui mancherebbe la scadenza.

Al tempo 11 arriva un job che ha scadenza 22, tempo di esecuzione 1.5. Quanto è lo slack disponibile tra i frame 4 e 5? 2. Se lo accetto però, devo considerare che passo davanti al job già accettato, in quanto ho una priorità maggiore. Posso farlo in quanto il mio tempo di esecuzione è proprio uguale al margine di sicurezza del job accettato in precedenza. Il nuovo slack del job precedente sarà 0, mentre del job che ho appena accettato sarà 0.5.

Supponiamo arrivi un nuovo job all'istante 14 con scadenza 44 e tempo di esecuzione 5. Tra il quinto frame, al frame 11, che appartiene all'iperperiodo successivo, ci sono solamente 4.5 unità di prcessore disponibile. Perché? Alla quantità di tempo che lasciavano i task periodici, devo anche togliere il tempo preso per eseguire i task periodici di priorità superiore.

Gestione delle violazioni delle scadenze

Un job hard real-time completa sempre entro la sua scadenza a meno di malfunzionamenti dell'hardware, bug nel software, o difetti di progettazione del sistema real-time

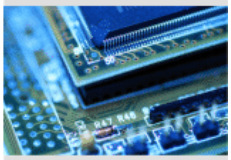
Uno scheduler progettato in modo robusto all'inizio del frame successivo alla scadenza controlla che il job sia effettivamente terminato

In caso contrario lo scheduler cerca di porre rimedio:

- Elimina completamente il job non terminato
- Interrompe il job ed alloca la parte di esecuzione restante come job aperiodico
- Continua l'esecuzione del job, allungando il frame contenente la scadenza e ritardando così tutti i frame successivi

L'azione di recupero più appropriata dipende ovviamente dalla natura del job e degli altri task del sistema

Schedulazione clock-driven
Marco Cesati



Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Cosa succede se un job hard-RT in qualche modo viola la scadenza? Questo è un problema. Queste cose non devono succedere, ma non succedono dal punto di vista del software. Hardware, o difetti di progettazione del sistema hard-RT possono portare a questo evento. Se progetto bene uno scheduler devo tenerne in considerazione. Cosa posso fare? Un cyclic executive potrebbe fare tre cose.

1. Eliminare completamente il job non terminato. Non lo reinserisco nei blocchi di schedulazione successivi. Questa è una possibilità, è difficile fare un cyclic executive in cui si possono inserire liberamente dei job in ritardo nei blocchi successivi. Questo è molto facile da implementare
2. Interrompere il job ed inserirlo nella coda dei job aperiodici. Il completamento verrà effettuato quando ci sarà tempo
3. Continuare l'esecuzione del job, allungando il frame contenente la scadenza e ritardando tutti i frame successivi.

Anche questa è una scelta che si può fare, ma la soluzione più appropriata dipende dai job e dalla natura del sistema reale. E' comunque un evento che dal punto di vista del software non deve accadere, non accade. Se accade è per cause che non dipendono dal progettista del software.

Vantaggi e svantaggi degli scheduler clock-driven

- Sono concettualmente semplici e facili da validare
- Non è necessario controllare l'accesso alle risorse condivise
- Non è necessario sincronizzare tra loro i job
- Scegliendo opportunamente la durata dei frame è possibile minimizzare l'overhead dello scheduler
- Possono essere ancora semplificati assumendo che gli eventi esterni si verificano in sincronia con i frame
- Gli istanti di rilascio dei job devono essere prefissati
- Tutte le possibili configurazioni del carico devono essere previste in anticipo
- Non efficienti per sistemi con molti job aperiodici

Schema della lezione
Algoritmi clock-driven
Cyclic executive
Job aperiodici soft r.-t.
Job aperiodici hard r.-t.

Tiriamo le somme di tutto il discorso. Questi scheduler **clock driven** sono semplici concettualmente e facili da validare. Non è necessario controllare l'accesso alle risorse condivise, perché il progettista fa tutto offline. Fa in modo che la schedulazione rispetti tutti i vincoli di precedenza e di accesso alle risorse condivise, quindi non ci sono meccanismi di sincronizzazione automatici. Se scelgo opportunamente la durata del frame, è possibile minimizzare l'overhead dello scheduler. Non mi conviene scegliere frame di durata piccola, in quanto mi conviene minimizzare l'overhead dello scheduler, rispettando sempre i vincoli. Se posso, scelgo un giusto compromesso che non appesantisce troppo il sistema e che mi garantisce una possibilità abbastanza frequente di tenere in considerazione i job aperiodici nel sistema. Questo tipo di sistemi si possono ancora semplificare se prevedo che tutti gli eventi esterni avvengano in sincronia con i frame.

Per contro, hanno dei gravi svantaggi. Devo conoscere gli istanti di rilascio di tutti i job. Devo conoscere i parametri temporali di tutti i job. Tutte le possibili configurazioni del carico, anche dei job aperiodici, le devo conoscere in anticipo. E' un sistema progettato per una singola applicazione. Se cambia l'applicazone, anche in aspetti marginali, devo rifare tutto lo scheduler. Non sono efficienti per sistemi che devono schedulare molti job aperiodici. Gli svantaggi sono così forti che oggi la maggior parte di coloro che progettano sistemi RT utilizzano scheduler di tipo differente, basati su algoritmi a priorità. Questi scheduler cyclic executive rimangono importanti in certe applicazioni di nicchia, erano molto importanti diversi anni fa. Oggi sono un po messi da parte, però è necessario conoscerli quando si parla di sistemi RT.