Scheduling real-time (parte 1)

Braione Pietro, revisione Domenico G. Sorrenti

Sistemi Embedded

Anno accademico 2019/20

Obiettivi

- vogliamo ragionare sulla capacità di un sistema di fare quello che desideriamo faccia
- Strategie di scheduling per sistemi real-time
- (gestione dell')accesso concorrente a risorse condivise

Terminologia: Tasks e jobs

- Un task fornisce una certa funzionalità ad un sistema
- Un task viene realizzato da uno o più jobs
- job è un termine generico che indica una attività da svolgersi, anche se non si tratta di una vera e propria esecuzione di istruzioni su una CPU, ad esempio la trasmissione di un pacchetto su una linea di comunicazione
- i job richiedono delle risorse per la loro esecuzione
- queste resources sono anche nominate come servers, active resources, processors al variare della letteratura e dell'area

Terminologia – release time

- Il release time r_i di un job *i* è l'istante temporale in cui un job è pronto per essere eseguito (data & control dependency conditions)
- Quando è noto con precisione di dice che si ha un fixed release time job
- Quando non è noto con precisione, ma si possono determinare lower&upper bounds per release time r_i : r_i^- e r_i^+
- L'intervallo $[r_i^-, r_i^+]$ è chiamato jitter del release time e si dice che si ha un jittered release time job
- Per alcuni jobs non si possono fare previsioni: sporadic / aperiodic

Ancora terminologia

- Il response time di un job è il tempo trascorso tra il release time e l'istante temporale in cui il job è completato
- La relative deadline di un job è il massimo response time ammissibile per quel job
- In termini assoluti ovvero non relativi al release time, la absolute deadline è l'istante temporale entro il quale un job deve essere eseguito
- La gran parte (non tutti) dei vincoli temporali possono essere specificati mediante release-time e deadline

Ancora terminologia - execution time

- L'execution time e_i di un job *i* è il tempo richiesto per la sua esecuzione in assenza di altri job e con la disponibilità di tutte le risorse di cui ha bisogno.
- Si tratta di un valore spesso difficile da valutare e quindi spesso resta piuttosto astratto (branch diversi con exec time diversi, dipendenza dai dati in ingresso, uso di cache & pipelining, etc.)
- Sul piano dei valori concretamente utili: si possono determinare (e poi usare) i lower&upper bounds dell'execution time: e_i^- ed e_i^+
- Usualmente ci servirà soprattutto l'upper bound (worst case execution time o WCET) e spesso capita di trovare e_i , ma si intende implicitamente proprio e_i^+
- In alcuni casi la differenza tra e_i^- ed e_i^+ è così grande che usare e_i^+ implica soluzioni troppo costose oppure che non ci siano soluzioni

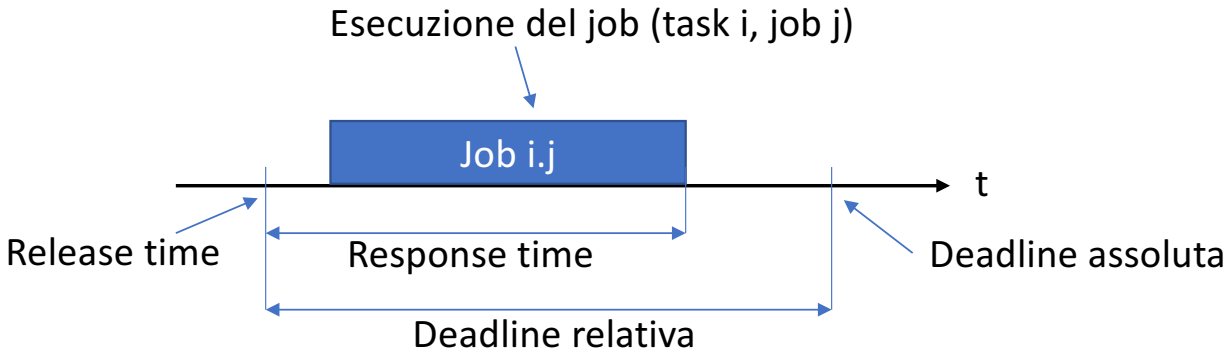
Vincoli temporali hard and soft real-time

- criticità funzionale: quanto critico è non rispettare la deadline?
  - fatal => hard, non-fatal => soft
- utilità di un late result: una qualche funzione di utilità come decresce al crescere del ritardo?
  - brusca => hard, graduale => soft
- deadline miss “deterministico” (never-ever) o probabilistico (meno di x%)?
  - deterministico => hard, probabilistico => soft
- alla fine: serve o non serve >>provare<< che il sistema non manca una o più deadline? => hard

Approcci tradizionali per provare soddisfacimento vincoli temporali hard

- approccio “tradizionale”: evitare qualunque alternativa hardware e software che introduca non-determinismo nel sistema => prova mediante simulazione del sistema su un esaustivo insieme di test
- approcci più recenti: prova online con validation test da superarsi per mettere in esecuzione un nuovo task/job hard real time, questo online test serve a verificare che il sistema sia in grado di soddisfare i vincoli temporali sia di questo nuovo job che di tutti gli altri

Esempio


Esecuzione del job (task i, job j)
Job i.j
t
Release time
Response time
Deadline assoluta
Deadline relativa

Modello a task periodici (esistono definizioni leggermente diverse)

- È il modello di carico deterministico più noto, per cui ci sono più sviluppi teorici e tools di supporto (ad esempio algoritmi di scheduling)
- Un task è periodico se i job di cui è composto sono rilasciati (released) in successione ad intervalli di tempo regolari
- I più importanti parametri di un task periodico:
  - La fase f, ossia il tempo di release del suo primo job
  - Il periodo P, ossia l'intervallo di tempo tra due release successive
  - Il tempo di esecuzione e, ossia la durata di esecuzione dei suoi jobs (supponendo o meno che tutti i suoi job abbiano stessa durata di esecuzione)
  - La deadline relativa D
- Modello poco accurato al crescere di jitter e variazioni di execution time
- Notazione per un task periodico T: T(f, P, e, D)

Job aperiodici e sporadici

- I job aperiodici e sporadici non eseguono periodicamente
  - Aperiodico: non ha una deadline (es. sensibilità ritorno radar)
  - Sporadico: ha una deadline (es. disabilitazione auto-pilota aereo)
- Parametri job sporadici:
  - Tempo di release *r*
  - Tempo di esecuzione *e*
  - Deadline relativa *D*
- Indichiamo un job sporadico come S(*r*, *e*, *D*)

Iperperiodo

- Supponiamo che un certo sistema abbia task periodici $T_1(f_1, P_1, e_1, D_1)$, …, $T_n(f_n, P_n, e_n, D_n)$
- L'iperperiodo del sistema è H = m.c.m. (P_1...P_n)
- Il massimo numero di job nell'iperperiodo è $N = \sum_{i=1}^{n} H/P_i$
- Esempio: ho 3 task con periodi 3, 4 e 10
  - H = m.c.m.(3, 4, 10) = 60
  - N = 60 / 3 + 60 / 4 + 60 / 10 = 41

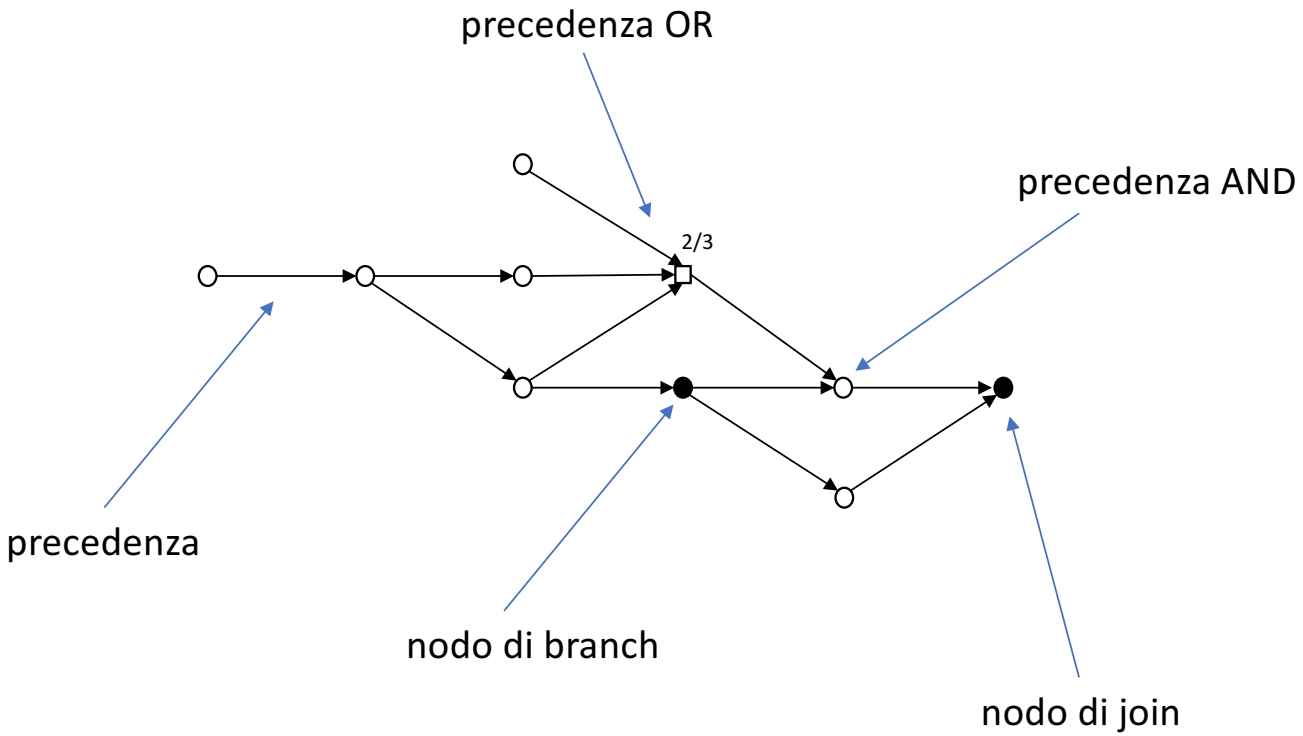
Utilizzazione

- Definiamo l'utilizzazione di un task periodico come $u = e / P$
- Intuitivamente è la frazione di periodo in cui il task tiene occupato il processore
- Per un insieme di n task T_i possiamo definire l'utilizzazione totale come $U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} e_i/P_i$
- Esempio: se i tre task di periodo 3, 4 e 10 hanno tempo di esecuzione 1, 1 e 3 abbiamo che:
  - L'utilizzazione dei tre task è 33%, 25% e 30%
  - L'utilizzazione totale è 88%

Altri vincoli: precedenza e dipendenza dati

- Oltre ai vincoli temporali in un sistema possono esistere vincoli di precedenza e di dipendenza dati tra i job
- I vincoli di precedenza stabiliscono che i job debbano essere eseguiti in un certo ordine (un certo job deve terminare l'esecuzione prima che inizi l'esecuzione di un altro)
- La dipendenza sui dati si stabilisce quando dei jobs accedono a dati condivisi
  - Può introdurre contesa sui lock e quindi, a seconda degli algoritmi di scheduling e di accesso alle risorse, vincoli su come i job vanno schedulati
  - Esempio: pipeline, un job consumer può proseguire l'esecuzione quando il job producer ha prodotto almeno un dato

Task graph


precedenza OR
precedenza AND
2/3
precedenza
nodo di branch
nodo di join