Analisi di schedulabilità di un task-set composto da task hard real-time e soft real-time

Francesco Scandiffio
francesco.scandiffio@edu.unifi.it

Sommario

Lo studio analitico di un task-set è un aspetto importante nella realizzazione di un sistema real-time perché permette di ottenere informazioni relative al comportamento dei task nel tempo. In questo elaborato viene condotta un'analisi sulla schedulabilità di un insieme di task hard real-time periodici e task soft real-time, i primi con tempi di esecuzione vincolati in un intervallo, i secondi con processi di arrivo e servizio stocastici con distribuzioni Esponenziali, HyperEsponenziali o HypoEsponenziali. Oltre ad una parte di modellazione teorica, vengono discussi i risultati ottenuti analizzando un task-set di esempio

I Introduzione

Il documento è strutturato come segue: nella Sezione II viene presentato il modello di task-set in esame, mentre nella Sezione III sono elencate le misure di interesse nell'analisi dei task insieme alla descrizione dei modelli matematici utilizzati. La Sezione IV contiene il modello delle classi del sistema e nella Sezione V vengono discussi i risultati di un esempio di analisi.

II Modello di task-set

Il modello in esame è un task-set composto da task hard real-time e soft real-time eseguiti su un sistema a singolo processore e regolati da due diverse politiche di scheduling. I primi sono task periodici con jtter, offset e tempi di servizio con distribuzione qualunque, purché limitata nel tempo in un intervallo $[t_{min}, t_{max}]$. La loro esecuzione può essere composta da più chunk, ognuno dei quali può richiedere l'utilizzo di una o più risorse, con eventuale presenza di meccanismi di sincronizzazione come semafori o mailbox nel caso in cui queste siano condivise tra più task. Una caratteristica molto importante dei task hard real-time è che assumiamo che questi siano schedulati secondo una qualche politica in grado di garantirne il rispetto delle deadlines, così da poter concentrare l'analisi sui task soft real-time che compongono il modello. I task soft real-time hanno tempi di arrivo e di servizio regolati da distribuzioni di probabilità che possono essere di tipo Esponenziale, HyperEsponenziale o HypoEsponenziale, con la conseguenza che l'arrivo ed il servizio di un job sono processi aleatori composti da una o più fasi. Ad ogni task soft real-time T_i è associata una

priorità $P_i \in \mathbb{N}$ su cui si basa un algoritmo di scheduling best-effort: ordinando i task soft real-time in base alla priorità, ovvero definendo l'insieme $T_s \coloneqq \{T_1, T_2, \ldots T_n : P_1 \leq P_2 \leq \ldots \leq P_n\}$ in cui a indice del task minore corrisponde un livello di priorità maggiore, possiamo affermare che il generico $T_i \in T_s$ ha accesso al processore solo nei momenti in cui questo non è utilizzato da alcun task a priorità più alta. A tal proposito è importante specificare che i task hard real-time, pur non avendo associato alcun valore P_i, vengono considerati sempre a priorità maggiore rispetto ai task soft real-time. Infine, ad ogni task del modello è associata una coda gestita con politica First In First Out e di capienza massima N_i all'interno della quale sono inseriti i job in attesa di essere serviti; se all'arrivo di un nuovo job la coda risulta già piena, il job in questione viene scartato, andando così ad aumentare il contatore dei denials del task di appartenenza.

II.I Rappresentazione mediante STPN

Una Stochastic Time Petri Net è un modello matematico che permette di rappresentare in modo efficace un sistema in cui i tempi di arrivo e di servizio dei task possono essere variabili aleatorie con distribuzione di probabilità arbitraria. Senza voler fornire una definizione esaustiva e formale di una STPN, ci limitiamo a descrivere come poter rappresentare il modello di task-set precedentemente introdotto tramite una rete di petri. Consideriamo ad esempio la rete in Figura 5: in alto

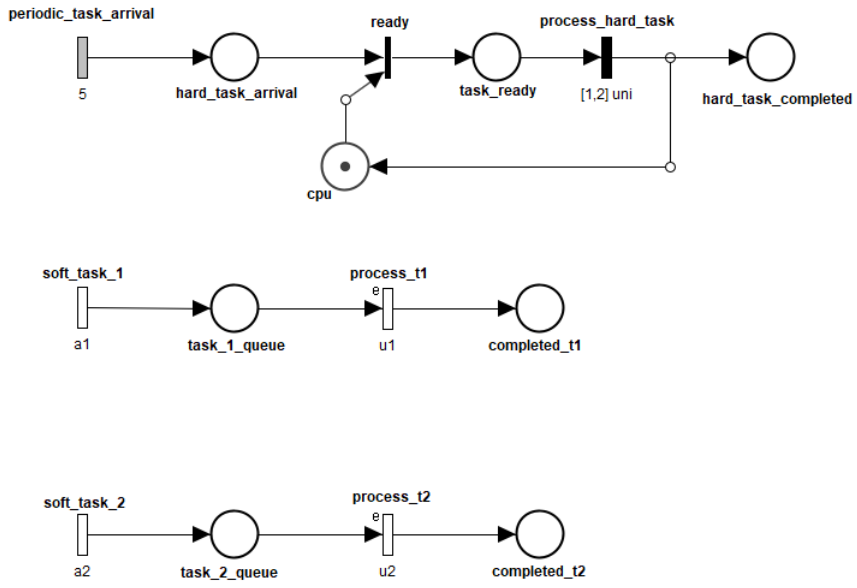

Figura 1: STPN composta da un task hard real-time (in alto) e due task soft real-time con processi di arrivo e servizio dei job regolati da distribuzioni esponenziali.

è presente un task hard real-time con tempi di esecuzione dei chunk con distribuzione uniforme; la periodicità del task è data dalle transizioni deterministiche, mentre la garanzia circa il rispetto delle deadline è ottenuta mantenendo la somma dei Latest Firing Time dei vari chunk inferiore al periodo del task. Nel caso in cui i task hard real-time fossero più d'uno sarà responsabilità dell'utente costruire la rete in modo tale da implementare un qualunque algoritmo di scheduling

che garantisca il rispetto delle deadline. In questo esempio l'unica risorsa necessaria per la computazione è l'accesso al processore, la cui disponibilità è identificata dalla presenza di un gettone nel posto `cpu`. Più in basso sono rappresentati due task soft real-time, entrambi con distribuzioni di arrivo e servizio esponenziali ma con rate diversi. L'algoritmo di scheduling best-effort basato sulle priorità viene implementato utilizzando delle enabling functions associate allo stato di `cpu`. La transizione di servizio del primo task soft real-time è attivata solo nel caso in cui $cpu == 1$, ovvero nessun task hard real-time sta utilizzando il processore. Volendo modellare il secondo task soft real-time con una priorità inferiore rispetto al precedente, la transizione `process_t2` ha come enabling function la valutazione dell'espressione $cpu == 1 \ and \ task_{1\,q}ueue == 0$ ovvero il processore non è in uso ai task hard real-time ed il task soft real-time a priorità più alta non ha alcun job in attesa di essere servito.

III Valutazione del comportamento dei task soft real-time

Assumiamo come condizione iniziale che i task hard-real time siano schedulabili e che tutte le loro deadline siano rispettate. Questo implica che non ci interessa conoscere quale algoritmo di scheduling sia effettivamente implementato, né conoscere l'ordine di esecuzione di questi task. Di conseguenza l'analisi del modello si concentra sullo studio della schedulabilià e del comportamento dei task soft real-time, in particolare delle misure discusse di seguito.

Availability del sistema

Lo scheduling best-effort basato sulle priorità permette ad un generico task $T_i \in T_s$ di poter utilizzare il processore solo negli istanti in cui questo non è utilizzato né dai task hard real-time né da un task $T_j \in T_s$ con $j < i$. Possiamo dunque definire $\alpha^i(t) \coloneqq P\{$ *il processore può essere usato dal task* $T_i \in T_s$ $\}$ che può essere riscritta in modo del tutto equivalente come $\alpha^i(t) \coloneqq P\{$ *il processore non è in uso a nessun task con priorità più alta di* T_i $\}$. Da quest'ultima formulazione segue immediatamente che $\alpha^1(t)$ è influenzata esclusivamente dalla computazione dei task hard real-time, motivo per cui possiamo affermare che $\alpha^1(t) = P\{$ *il processore non è in uso a nessun task hard real time al tempo t*$\}$: tale probabilità è ottenuta calcolando il reward transiente `cpu==1`. In generale, la probabilità che un task T_i utilizzi il processore al tempo t è una variabile aleatoria dipendente sia dalla $\alpha^i(t)$ che dalla probabilità che T_i abbia effettivamente bisogno di utilizzare il processore, ovvero dalla probabilità di avere almeno 1 job in coda in attesa di essere servito. Sebbene il numero di job in coda dei task hard real-time e quelli in coda per il task soft-real time a priorità più alta siano delle variabili aleatorie dipendenti, esse appaiono positivamente correlate: per questo motivo, indicando con $Q_0^1(t) \coloneqq P\{$*probabilità che il task* T_1 *non abbia nessun job in coda al tempo t*$\}$, il prodotto $\alpha^1(t) \cdot Q_0^1(t)$ fornisce una stima conservativa della probabilità che nessun job task hard real-time né alcun job del task T_1 sia in esecuzione al tempo t, cosa che corrisponde a $\alpha^2(t)$. Iterando il procedimento è possibile calcolare $\alpha^i(t) = \alpha^{i-1}(t) \cdot Q_0^{i-1}(t)$.

Probabilità di n job in coda

Un'altra misura di nostro interesse nello studio del task-set è l'analisi dello stato delle code dei task soft real-time. Come discusso nel paragrafo precedente, il numero di job in coda di un task incide

sul comportamento dei task a priorità inferiore, influenzandone in particolar modo l'availability. La probabilità che un task abbia la coda vuota è un caso particolare della generica probabilità $Q_n(t) \coloneqq P\{$ *probabilità di avere n job in coda al tempo t*$\}$ con $n \in [0, N]$.
Un task soft real-time con coda di dimensione N ed arrivi e servizi di Poisson di rispettivo tasso λ e μ implementa quello che nella teoria delle code è detto *Processo Birth-Death* ad $N + 1$ stati, ovvero un particolare tipo di processo Markoviano tempo-continuo in cui una transizione da uno stato ad un altro si manifesta solo in corrispondenza dell'arrivo o del servizio di un job.

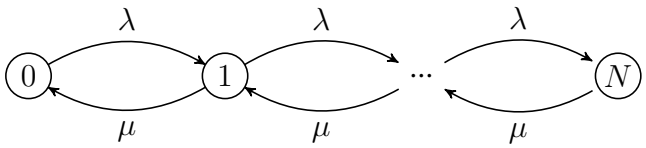

Figura 2: Catena di Markov di un processo Birth-Death in cui la coda ha capienza massima N

Considerando la rappresentazione in Figura 2 ed introducendo la notazione $R_{kj}(t) \coloneqq P\{Q(t) = j | Q(0) = k\}$, la probabilità $Q_n(t)$ può essere ottenuta risolvendo il seguente sistema di equazioni differenziali

$$\begin{cases} R'_{k,0} & = +R_{k,1}(t)\mu f_\mu(1)\alpha(t) \\ & \quad - R_{k,0}(t)\lambda f_\lambda(0) \\ R'_{k,j} & = +R_{k,j+1}(t)\alpha(t)\mu f_\mu(j+1) \\ & \quad + R_{k,j-1}(t)\lambda f_\lambda(j-1) \\ & \quad - R_{k,j}(t)(\lambda f_\lambda(j) + \mu f_\mu(j)\alpha(t)) \\ R'_{k,N}(t) & = +R_{k,N-1}(t)\lambda f_\lambda(N-1) \\ & \quad - R_{k,N}(t)\mu f_\mu(N)\alpha(t) \end{cases} \tag{1}$$

Le funzioni f_μ ed f_λ sono funzioni nella variabile del numero di job in coda e la loro effettiva rappresentazione dipende dal modello reale che si vuole modellare.

Come prima variazione rispetto al caso con sole Esponenziali consideriamo dei processi caratterizzati da distribuzioni HyperEsponenziali. La Probability Density Function di una variabile aleatoria X con distribuzione $HyperEXP(\lambda_1, \ldots, \lambda_k)$ è definita come la somma pesata delle pdf di k variabili aleatorie esponenziali (Eq.2), ognuna delle quali ha peso p_i e tasso λ_i.

$$f_X(x) = \sum_{i=1}^{k} f_{Y_i}(x) p_i \quad \text{con} \quad Y_i = Exp(\lambda_i) \tag{2}$$
$$p_i = \frac{\lambda_i}{\sum_{j=1}^{k} \lambda_j}$$

Dall'Equazione 2 segue che una $HyperEXP$ può essere rappresentata come un insieme di Esponenziali disposte in parallelo (Figura 3), dunque un passaggio da uno stato A ad uno stato B tramite HyperEsponenziale possiede un rate di transizione λ definito come in Equazione 3.

$$\lambda = \sum_{i=1}^{K} \lambda_i \cdot p_i \tag{3}$$

È importante evidenziare che per distribuzioni $HyperEXP$ il processo di transizione tra due stati si completa in un unico passaggio, proprio come avviene per le distribuzioni Esponenziali. Dal momento che il numero di stati non viene modificato, il sistema di equazioni differenziali precedente (1) rimane valido.

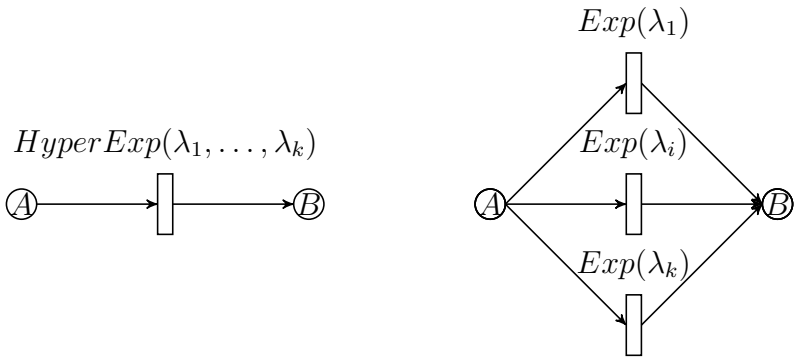

Figura 3: A sinistra: passaggio da un generico stato A verso uno stato B tramite distribuzione HyperEsponenziale. A destra: rappresentazione equivalente della distribuzione HyperEXP in forma di k Esponenziali parallele.

Come ulteriore estensione analizziamo un processo con distribuzione di tipo HypoEsponenziale. Per definizione, una variabile aleatoria con distribuzione $HypoExp(\lambda_1, \ldots, \lambda_k)$ è equivalente ad una somma di k variabili aleatorie Esponenziali, ognuna con un proprio tasso λ_i (Equazione 4).

$$X = \sum_{i=1}^{k} X_i \quad \text{con} \quad X_i = Exp(\lambda_i) \tag{4}$$

Dalla definizione 4 segue che una transizione di tipo $HypoExp(\lambda_1, \ldots, \lambda_k)$ tra due stati A e B può essere scomposta in una successione di k transizioni esponenziali, ottenendo la catena di Markov mostrata in Figura 4. Rappresentare un processo con distribuzione $HypoExp$ tramite le k esponenziali che lo compongono messe in serie vuol dire dover attraversare k archi, ognuno dei quali ha un proprio tasso λ_i. In alternativa al numero di archi risulta conveniente indicare che il numero di stati del processo passa da 2 a $k - 1$. Infatti una distribuzione HypoEsponenziale comporta un'inevitabile cambiamento nel numero degli stati della catena di Markov, con la conseguenza che il sistema di equazioni differenziali deve essere modificato. Distinguiamo dunque tra i due seguenti scenari:

1. l'arrivo di un job ha distribuzione HypoEsponenziale mentre il servizio ha distribuzione Esponenziale o HyperEsponenziale.

2. il servizio di un job ha distribuzione HypoEsponenziale, l'arrivo può avere una qualunque delle tre distribuzioni fino ad ora utilizzate.

Nel primo caso il sistema di equazioni differenziali da utilizzare è racchiuso in Equazione 5, dove $n \in [1, N-1]$ ed $h \in [1, H]$, con N dimensione della coda di arrivo, ed H è il numero di valori disponibili per i coefficienti λ relativi agli eventi di arrivo di un nuovo task.

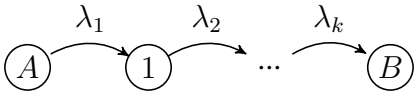

Figura 4: Catena di Markov di un processo con distribuzione HypoEsponenziale a k fasi rappresentato come una sequenza di k transizioni con distribuzione Esponenziale.

$$
\begin{cases}
Q'_{0,0,0}(t) & = +Q_{1,0,0}(t)\alpha(t-1)\mu f_m u(1) \\
& \quad - Q_{0,0,0}(t)\lambda_0 f_\lambda(0) \\
Q'_{0,h,0}(t) & = +Q_{1,h,0}(t)\alpha(t-1)\mu f_m u(h) \text{for h=1..H} \\
& \quad + Q_{0,h-1,0}(t)\lambda_{h-1} f_\lambda(h-1) \\
& \quad - Q_{0,h,0}(t)\lambda_h \\
Q'_{n,h,0}(t) & = +Q_{n,h-1,0}(t)\lambda_{h-1} \\
& \quad + Q_{n+1,h,0}(t)\alpha(t-1)\mu \\
& \quad - Q_{n,h,0}(t)(\lambda_h + \alpha(t-1)\mu) \text{for n=1..N-1 e for h=1...H} \\
Q'_{n,0,0}(t) & = +Q_{n-1,H,0}(t)\lambda_H \\
& \quad + Q_{n+1,0,0}(t)\alpha(t-1)\mu \\
& \quad - Q_{n,0,0}(t)(\lambda_0 + alpha(t-1)\mu) \text{for n=1..N-1} \\
Q'_{N,0,0}(t) & = +Q_{N-1,H,0}(t)\lambda_H \\
& \quad + Q_{N,H,0}(t)\lambda_H \\
& \quad - Q_{N,0,0}(t)(\lambda_0 + \alpha(t-1)\mu) \\
Q'_{N,h,0}(t) & = +Q_{N-1,h-1,0}(t)\lambda_{h-1} \\
& \quad - Q_{N,h,0}(t)(\lambda_h + \alpha(t-1)\mu) \text{for h=1..H}
\end{cases}
\tag{5}
$$

Probabilità di denial

Se all'arrivo di un nuovo job la coda risulta già piena, il job viene scartato andando ad incrementare il numero di denials del task di appartenenza. Definiamo dunque per un generico task soft real-time $p_{n,k}(t)$:=P{*il numero di job in coda al tempo t è n ed il numero di denials entro il tempo t è esattamente k*}. In modo analogo a quanto fatto per il calcolo di $Q_i(t)$, possiamo calcolare la probabilità $p_{n,k}(t)$ risolvendo un sistema di equazioni differenziali. Anche in questo contesto è necessario distinguere tra due scenari diversi, ognuno con il proprio sistema di ODE:

1. l'arrivo di un job ha distribuzione HypoEsponenziale mentre il servizio ha distribuzione Esponenziale o HyperEsponenziale.

2. il servizio di un job ha distribuzione HypoEsponenziale, l'arrivo può avere una qualunque delle tre distribuzioni fino ad ora utilizzate.

Infine siamo interessati al calcolo di $\eta_k(t)$:=P{*si verificano almeno k denials entro il tempo t*}. Questa quantità può essere calcolata utilizzando la formula

$$\eta_k(t) = 1 - \sum_{n=0}^{N} \sum_{h=0}^{k-1} p_{n,h}(t) \tag{6}$$

Probabilità di deadline miss

I task soft real-time tollerano, per definizione, il verificarsi di una o più deadline miss. Tuttavia, ponendosi l'obiettivo di voler studiare la schedulabilità del sistema, risulta indispensabile analizzare anche la probabilità di avere una o più deadline miss dei task soft real-time nel corso del tempo. L'obiettivo che ci poniamo è quello di calcolare la probabilità $P_{t_0} \coloneqq \{$*il job arrivato a* t_0 *manca la deadline*$\}$. Per fare ciò consideriamo innanzitutto un task soft real-time conforme al modello in esame con deadline relativa D e supponiamo che all'istante t_0 siano già presenti dei job in coda in attesa di essere serviti e che il k-esimo sia appena arrivato: la probabilità $\theta_{k,t_0} \coloneqq P\{$*il k-esimo job completa dopo* $t_0 + D\}$ può essere riscritta utilizzando il suo evento complementare, ossia $\theta_{k,t_0} = 1 - \varphi_{k,t_0}$ dove $\varphi_{k,t_0} \coloneqq \{k$ *job completano entro* $t_0 + D\}$. Osserviamo che φ_{k,t_0} può anche essere letta come la probabilità che nell'intervallo $[t_0, t_0 + D]$ sia possibile computare per un tempo sufficiente tale da poter servire i k job in coda, ovvero $\varphi_{k,t_0} = P\{$ *tempo di esecuzione dei* k *job* $\leq T_{t_0}\}$ dove T_{t_0} :=*tempo medio per cui i job possono computare nell'intervallo* $[t_0, t_0 + D]$. Poiché l'unica risorsa di cui hanno bisogno i task soft real-time è il processore, risulta immediato che il tempo medio di computazione in un intervallo è dato dall'integrale del tempo di utilizzo della cpu su tale intervallo, come mostrato dall'Equazione 7.

$$T_{t_0} = \int_{t_0}^{t_0+D} \alpha(\tau) d\tau \tag{7}$$

Modellando solo la parte realtiva al servizio di un job tramite STPN, un'analisi transiente sulla posizione assorbente finale fornisce, per costruzione, la Cumulative Distribution Function della variabile aleatoria X_i :=*tempo di esecuzione di 1 job*. A partire dalla CDF_{X_i}, la pdf può essere ottenuta tramite un'operazione di derivazione numerica, utilizzando ad esempio il metodo della differenza finita centrata (Eq.8).

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{8}$$

Nota la pdf di X_i, chiamiamola per semplicità di notazione pdf_1, vogliamo calcolare la pdf del tempo di esecuzione di k job, ovvero la pdf della variabile aleatoria $Z = X_1 + \cdots + X_k$. In generale la probability density function di una somma di variabili aleatorie è equivalente alla convoluzione delle pdf delle singole variabili che compongono la somma [**?**]; da quanto appena scritto segue che la pdf_k è il risultato di una serie di convoluzioni del tipo

$$pdf_i = pdf_{i-1} * pdf_1 \tag{9}$$

Integrando la pdf_k si ottiene la $F_k =$ *Cdf del tempo di esecuzione di* k *job* che insieme a T_0 ci permette di scrivere $\varphi_{k,t_0} = F_k(T_{t_0})$. Infine, ricordando che il task possiede una coda di capacità

massima N e che con i sistemi di equazioni differenziali è possibile conoscere la probabilità $Q_j(t)$ di avere j job in coda al tempo t, la probabilità obiettivo P_{t_0} si calcola risolvendo l'equazione

$$P_{t_0} = \sum_{k=1}^{k=N} \varphi_{k,t_0} \cdot Q_{k-1}(t) \tag{10}$$

Valutazione del comportamento steady-state

Come già detto, la $\alpha^i(t)$ è una variabile aleatoria dipendente dalle $\alpha^j(t)$ con $j < i$. Per questo motivo è corretto affermare che in generale la distribuzione $Q_k(t)$ non raggiunge mai uno steady-state. Tuttavia, dal momento che l'availability relativa al task soft real-time a priorità più alta dipende solo dall'esecuzione dei task hard real-time, i quali sono periodici e rispettano le deadline, possiamo affermare che la loro influenza sull'acquisizione del processore da parte del soft task T_1 viene esercitata in modo periodico. Come conseguenza otteniamo che il numero di job in coda per il task a priorità più alta raggiunge uno steady-state all'inizio di ogni iperperiodo. La distribuzione in questione può essere calcolata risolvendo $\underline{\alpha}(m+1) = \underline{\underline{P}}\,\underline{\alpha}(m) \forall m \in \{0, 1, \ldots, n\}$ dove $\underline{\underline{P}} \coloneqq [P_{k,j}]$ con $k, j \in \{0, 1, \ldots, K\}$ and $P_{k,j} \coloneqq P\{Q(T) = j \mid Q(0) = k\} = R_{k,j}(T)$

IV Modello delle classi

La modellazione dei task è stata realizzata attraverso una gerarchia di classi nel package `task`. La classe `RealTimeTask` rappresenta la base comune a tutti i task real-time e contiene un identificativo univoco assegnato in modo atomico, dunque predisposto per utilizzo multithread. Gli altri attributi di questa classe base sono le distribuzioni di probabilità che modellano gli eventi di arrivo e servizio di un task. Da questa classe derivano `HardRealTimeTask` e `SoftRealTimeTask`, che aggiungono rispettivamente i seguenti parametri: periodo, jitter, offset e deadline per i task hard real-time; priorità per i task soft real-time. Queste variabili sono necessarie per poter modellare i task così come definiti in Sezione II. La dimensione massima della coda ed il numero massimo di denials tollerati da un task sono stati un parametro del sistema da analizzare, motivo per cui non è stata inserita tra gli attributi del task.

Il package `Function` contiene le classi dedicate all'implementazione delle diverse distribuzioni di probabilità precedentemente descritte. La classe `FunctionBuilder` implementa una factory unificata che restituisce l'istanza di una specifica distribuzione a partire da parametri statistici di input, quali il coefficiente di variazione e il valore medio di una sequenza di eventi osservati. In questo modo risulta possibile modificare rapidamente i parametri, evitando all'utente di dover selezionare manualmente le distribuzioni e facilitando l'individuazione del risultato che meglio modella una certa serie di dati osservati. Passando alla parte di analisi del modello, le classi `QueueEDSolver` e `DenialEDSolver` implementano la risoluzione dei sistemi di equazioni differenziali che descrivono l'evoluzione temporale delle probabilità associate al numero di job in coda e al numero di richieste rifiutate. In presenza di distribuzioni composte, la soluzione viene ricondotta ad un insieme di processi esponenziali elementari: questa scomposizione permette di trattare in maniera uniforme distribuzioni con più fasi o più tassi. La classe `DistributionExtender` è quella depudata all'estrazione automatica dei tassi λ_i di distri-

buzioni esponenziali a partire dalla distribuzione in ingresso. In particolare, distinguiamo tra le seguenti casistiche:

- Input distribuzione esponenziale semplice: restituisce l'unico parametro λ della distribuzione in ingresso.
- Input distribuzione iper-esponenziale: calcola una combinazione pesata dei tassi delle varie fasi, restituendo un unico λ equivalente.
- Input distribuzione di Erlang o HypoEsponenziale; restituisce un array contenente i tassi di tutte le fasi.

Poiché le probabilità di interesse per l'analisi sono definite in funzione del tempo, ed in particolare dato che la CDF è calcolata tramite l'analisi del transiente, ho optato per un approccio di risoluzione numerico. Ad esempio, la derivata in un generico punto t è calcolata utilizzando il metodo della differenza finita centrale (Equazione 8), che garantisce una maggiore precisione rispetto alla differenza in avanti o indietro. Questa formula non può ovviamente essere utilizzata per il primo e ultimo punto di una sequenza, per i quali utilizzo rispettivamente la derivata in avanti (Equazione 11) e indietro (Equazione 12).

$$f'(x) = \frac{f(x+h) - f(x)}{h} \tag{11}$$

$$f'(x) = \frac{f(x) - f(x-h)}{h} \tag{12}$$

Per automaizzare la valutazione dei tasks, ho implementato la classe `TaskSetAnalyzer`, all'interno del quale ho integrato le diverse analisi matematiche discusse nella sezione precedente. In particolare, questa classe ha due responsabilità principali:

1. Gestione del task-set
Il `TaskSetAnalyzer` riceve in input un insieme di task soft real-time, ciascuno associato a parametri di gestione della coda, distribuzioni di arrivo e servizio e massimo numero di denials. I task vengono ordinati per priorità, così che l'analisi rispetti la logica del best-effor scheduling. Osserviamo che il task-set non contiene al suo interno oggetti di tipo `HardRealTimeTask`, ma piuttosto utilizza come input la distribuzione di probabilità che la cpu sia libera da un qualunque task hard real-time. Questa scelta permette di simulare con maggiori libertà una distribuzione qualunque, senza necessità di doverla ricostruire tramite trial-and-error a partire da combinazioni di hard tasks.

2. Collegamento ai solver
Per ogni task, vengono utilizzati un `QueueEDSolver` ed un `DenialEDSolver` per valutare i processi di Birth-Death della coda di attesa ed il numero di job scartati. Poiché il task-set può essere composto da più processi soft real-time ordinati in base alle rispettive priorità, il `TaskSetAnalyzer` coordina i solver per utilizzare ad ogni step la probabilità di *cpu free* risultante al termine dello step precedente. Questo comportamento implementa dunque il meccanismo di assegnazione della cpu descritto nella sezione di Availability del sistema, riportato nuovamente in Equazione 13.

$$\alpha^i(t) = \alpha^{i-1}(t) \cdot Q_0^{i-1}(t) \tag{13}$$

V Esperimenti

In questa sezione riporto i risultati di test simulativi condotti su un sistema composto da un task hard real-time periodico ed un numero variabile di task soft real-time. L'obiettivo è di analizzare il comportamento del numero di job in coda e del numero di denial al variare dei parametri relativi a rate di arrivo e servizio dei job soft real-time. Le metriche di utilizzo della cpu da parte del task hard real-time sono state generate prendendo a riferimento la rete di petri in figura **??**. Osserviamo che si tratta di un task periodico con tempi di servizio generati secondo una distribuzione uniforme.

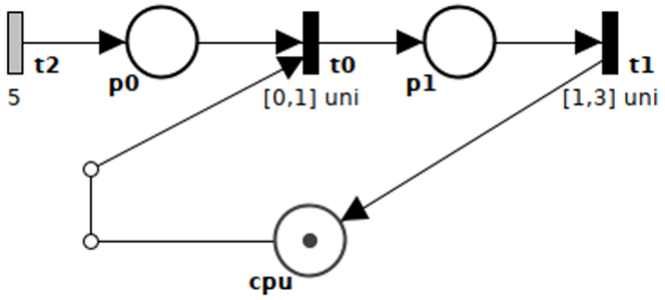

Figura 5: Petri Net di riferimento per il calcolo della distribuzione di probabilità di utilizzo della cpu da parte di un task hard real-time periodico.

Il primo esperimento analizza il comportamento per un singolo task soft real-time che può accedere alla cpu solo quando questa non è utilizzata dal task hard real-time. I parametri della sperimentazione sono riportati in Tabella 1. I risultati sono riportati in Figura 6. Osserviamo che il sistema raggiunge lo steady state già entro il terzo periodo, che il numero medio di job in coda si assesta a circa 1.4.

Arrivo soft task	Exp(2.1)
Servizio soft task	Exp(3.5)
Max Queue size	10
Step campionamento	0.001

Tabella 1: Valori di riferimento per l'esperimento con un task soft real time

Consideriamo nuovamente la stessa struttura dell'esperimento precedente, variando però i parametri dell'analisi. In particolare, assegnamo la stessa distribuzione per gli eventi di arrivo e di servizio del task soft real-time, fissati in $Exp(1)$. In questo scenario, analizziamo la probabilità di N job in coda e di almeno K denial in funzione del tempo. La Figura 7 mostra la dinamica di evoluzione della distribuzione di probabilità associata al numero di job in coda n nel sistema, con $N \in [0; 3]$. La condizione iniziale del sistema prevede una coda vuota, come visibile dalla probabilità unitaria assegnata allo step zero per il grafico associato a *prob queue 0*. Tuttavia, si osserva un rapido decadimento che lascia spazio ad una crescita della probabilità di 3 job in coda, che tende a prevalere nella simulazione.

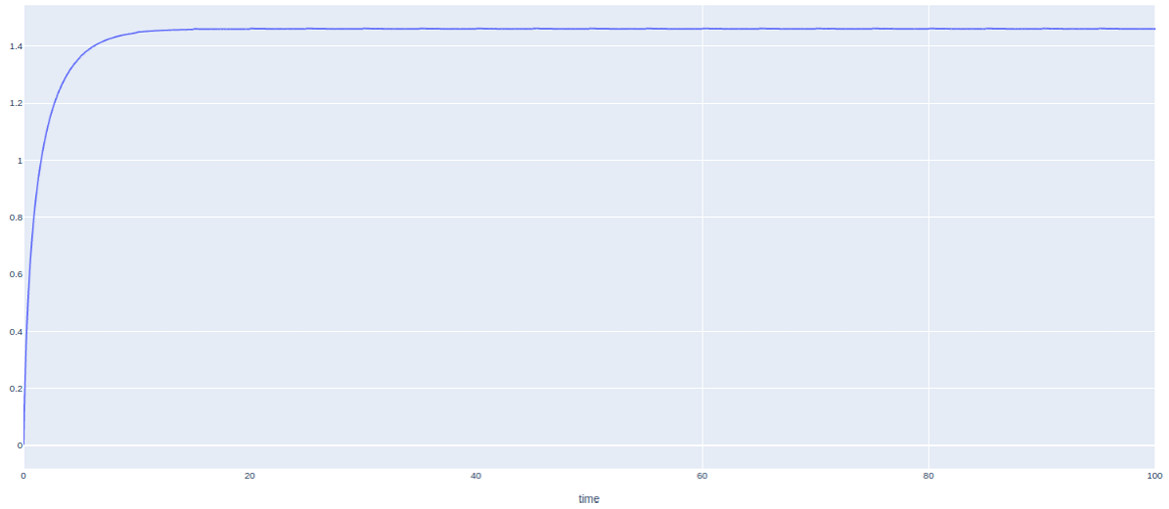

Figura 6: Esperimento 1, andamento del numero di job in attesa per il task soft real-time

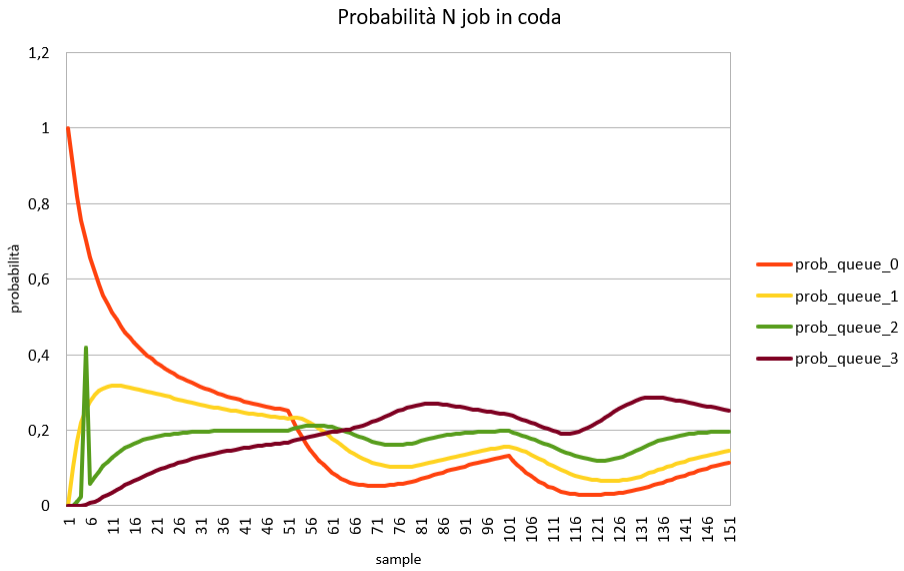

Figura 7: Esperimento 2, probabilità di job in coda al variare di n valore della coda, da 0 a 3

La Figura 8 mostra invece l'analisi sulle probabilità che si siano verificati almeno K denials, con K variabile tra 1 e 4. Le curve evidenziano un andamento crescente, coerente con la natura cumulativa della metrica considerata. La probabilità di osservare almeno un denial cresce rapidamente, tendendo a valori prossimi allo 0.9 nei campioni finali della simulazione.

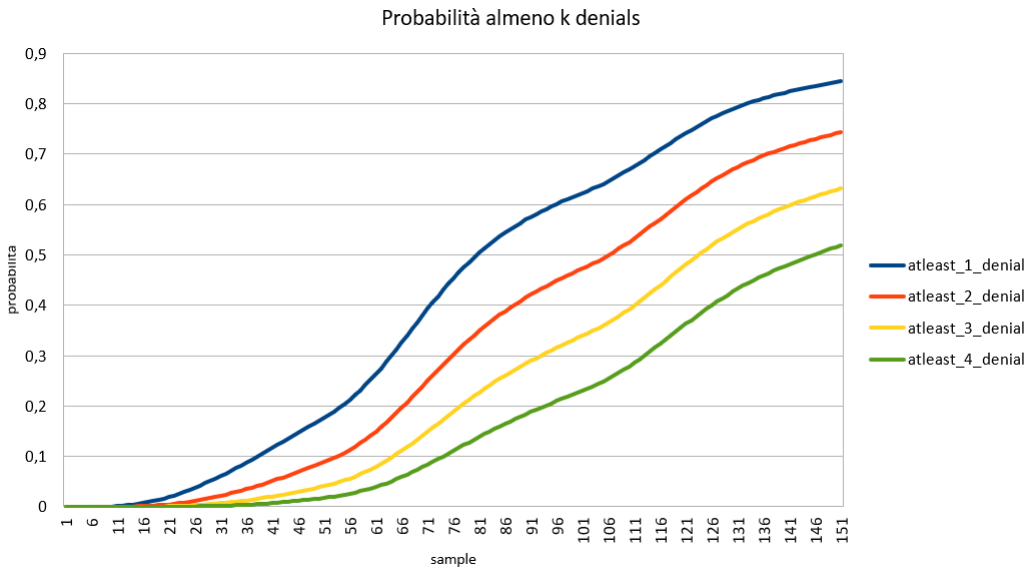

Figura 8: Esperimento 2, probabilità di almeno K denial, con K da 1 a 4

Consideriamo adesso un task soft real-time con distribuzione di servizio modellato secondo una erlang (Tabella 2). Dai risultati in Figura 2 osserviamo una crescita nettamente maggiore del numero di job in coda, che raggiunge circa 8 in soli 20 secondi, equivalenti a 4 periodi del task hard real-time. Anche in questo caso però, il numero di job in coda si stabilizza, non superando il limite massimo imposto a 10.

Arrivo soft task	Exp(1.5)
Servizio soft task	Erlang(shape = 3, rate = 5) + Exp(4)
Max Queue size	10
Step campionamento	0.001

Tabella 2: Valori di riferimento per il secondo esperimento con un task soft real time

Introduciamo ora un secondo task soft real-time con priorità inferiore rispetto al primo, con parametri riassunti in Tabella 3. Il task 1, che beneficia di priorità superiore, mostra un comporta-

Arrivo Task 1	Exp(1.5)
Servizio Task 1	HyperExp(2, 5)
Max Queue size	50
Arrivo Task 2	Erlang(2, 1.5) + Exp(1.5)
Servizio Task 2	Erlang(2, 4) + Exp(2)
Max Queue size	50

Tabella 3: Valori di riferimento per due task soft real time

mento altamente reattivo rispetto all'utilizzo periodico della cpu da parte del task hard real time. La coda raggiunge rapidamente uno steady sate, e oscilla stabilmente intorno ad un valore medio

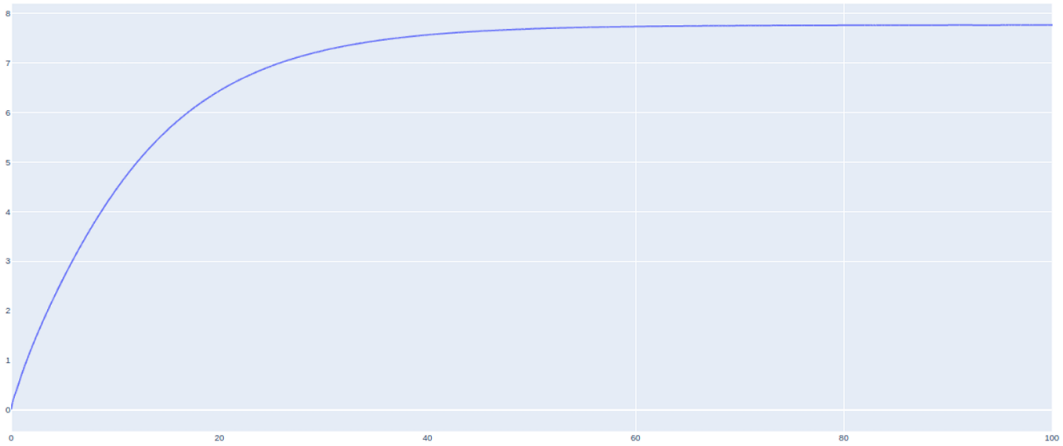

Figura 9: Esperimento 3, andamento del numero di job in attesa per il task soft real-time

di circa 0.6 job in coda. Le leggere fluttuazioni periodiche mostrano chiaramente l'influenza del task hard real-time sulle possibilità di esecuzione dei job del task soft real time in esame.

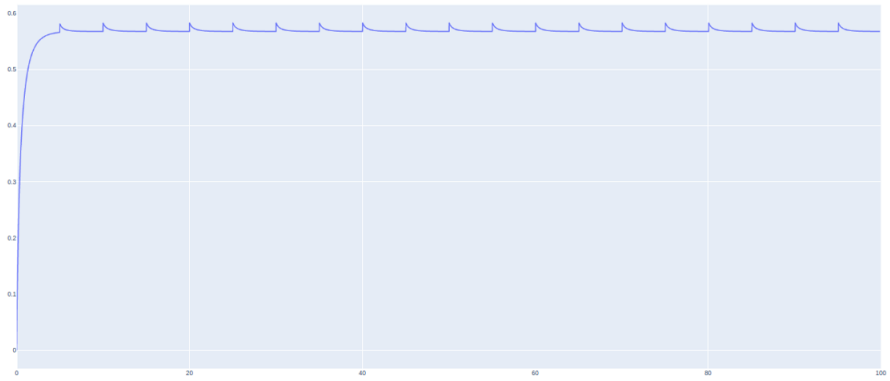

Figura 10: Esperimento 4, andamento del numero di job in coda per il task soft real-time a priorità maggiore.

Osservando invece la Figura 11, il task a priorità inferiore mostra un comportamento di accumulo del numero di job in coda, andando però a stabilizzarsi in modo graduale a circa 1.1 jobs. Osserviamo, a differenza del precedente grafico, la totale assenza di comportamenti periodici. Questo vuol dire che già dal secondo task non è più visibile una diretta correlazione con l'utilizzo del task hard real time. Questo risultato è in linea con le premesse teoriche, perché la distribuzione di probabilità dell'utilizzo della cpu in ingresso per il task 2 dipende dall'uso combinato del task hard real-time e dal task soft real-time a priorità maggiore. Questo significa anche che, maggiore è la necessità di utilizzo delle risorse da parte dell'altro task soft, minore sarà la correlazione direttamente visibile tra i carichi di lavoro del task hard real-time e quelli del task soft real-time a priorità più bassa.

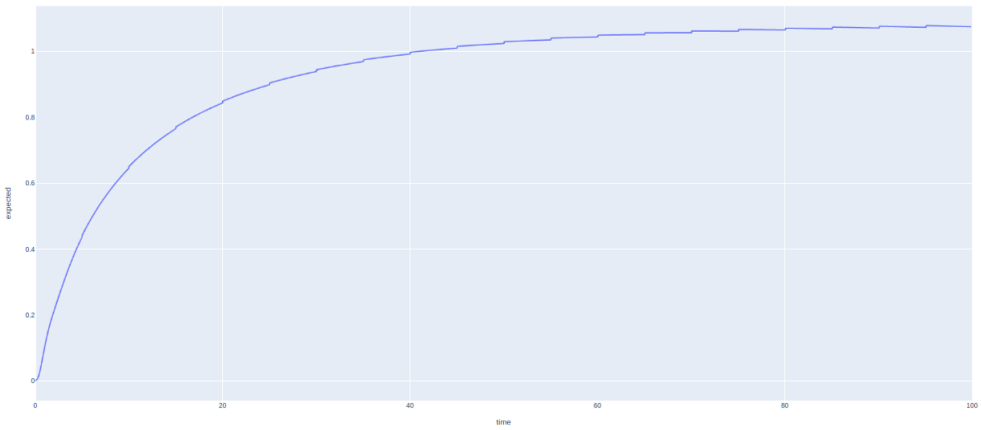

Figura 11: Esperimento 4, andamento del numero di job in coda per il task soft real-time a priorità minore.