UNIVERSITÀ DEGLI STUDI DI PARMA
FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Elettronica

ANALISI E VALUTAZIONE DI ALGORITMI DI SCHEDULING IN TEMPO REALE CON CARATTERISTICHE DI ADATTAMENTO

Relatore:
Chiar.mo Prof. Stefano Caselli

Correlatore:
Dott. Ing. Monica Reggiani
Chiar.mo Prof. Francesco Zanichelli

Tesi di laurea di:
Cristian Marastoni

ANNO ACCADEMICO 2002-2003

A mia madre,
Rosanna

Durante lo svolgimento di questa tesi una vita intera è trascorsa dinnanzi ai miei occhi... Ricordo ancora quell'ombra stanca e china su quei libri, alla conquista di una meta importante... i volti, le parole, l'amore, l'odio, la passione, la paura, il vuoto ... quante ne sono passate in questi anni. Come dimenticare... Questo è uno di quei momenti in cui odio.. odio essere me stesso: rischio sempre di tralasciare qualcuno..(eheheh).

Ringrazio innanzitutto il prof. Stefano Caselli per la sua pazienza, la sua competenza, le sue insistite provocazioni e i suoi rimproveri, senza i quali non sarei mai stato in grado di concludere questo lavoro di tesi.

Ringrazio l'ing. Monica Reggiani, per l'infinito supporto tecnico e soprattutto morale che ha sempre saputo infondermi.

Un saluto particolare ai miei amici Jacopo, Luca, Germano, Lory, Fiorda, "compagni di merende" al mitico mulino.... Franko, Pazzya, Bezzo, Lo Stordito, La Faina, che in questi mesi hanno sopportato i miei momenti di crisi, quando tutto andava storto. (Vi amo ragazzi....)

Infine grazie.. grazie destino, per avermi fatto conoscere una persona meravigliosa... Emanuela.

Indice

Capitolo 1

Introduzione

L'obbiettivo di questa tesi è applicare i paradigmi dell'apprendimento con rinforzo al controllo e all'adattamento di sistemi di tipo soft real-time. Lo studio è ristretto al caso di insiemi di processi periodici indipendenti, schedulati tramite l'algoritmo Rate Monotonic. L'idea oggetto di studio è innovativa e presenta diverse possibilità applicative ad ambienti altamente dinamici quali robotica e multimedia.

1.1 Sistemi in tempo reale

Per molti anni i sistemi di elaborazione in tempo reale sono stati oggetto di uso in poche applicazioni di nicchia. Si tratta di applicazioni spesso critiche, come ad esempio sistemi militari, controllo di sistemi di volo, controllo di impianti industriali, missioni spaziali e di applicazioni costose, la cui progettazione richiedeva un elevato investimento di energie e risorse. Sviluppati principalmente all'interno di comunità di esperti, i sistemi di elaborazione in tempo reale hanno continuato ad evolvere la propria struttura, presentando al contempo costi più ridotti e tecniche progettuali sempre più affinate. Dalle elaborate tecniche di programmazione ad-hoc si è passati ad algoritmi ottimi, che permettono al progettista del sistema real-time di stabilire a priori, sulla carta, quali devono essere le caratteristiche del sistema e quali i vincoli da rispettare. I fondamentali contributi dati da Liu e Layland, Sha, Lehoczky e moltissimi altri ricercatori hanno esteso i principi della programmazione real-time a condizioni di lavoro sempre più realistiche

[LSD89] [LL73] [LRL90] [BSL89]. Oggi, i sistemi di elaborazione in tempo reale vengono utilizzati sempre più frequentemente: nuove esigenze, quindi, nate dallo sviluppo del multimedia, del commercio elettronico, della robotica, dei sistemi embedded, hanno richiesto l'impiego di sistemi ad alta predicibilità e che consentano di fornire le prestazioni desiderate utilizzando un vero e proprio formalismo di programmazione e non soltanto empirici criteri.

Questa estensione dell'utilizzo dei sistemi in tempo reale ha portato con sè diverse problematiche, legate principalmente alla gestione dei sovraccarichi e all'ottimizzazione dell'uso delle risorse, già affrontate dalla comunità real-time ma questa volta applicate ad un contesto diverso. Quando le risorse a disposizione sono limitate, le tecniche di programmazione cosiddette "hard real-time" non sono più adatte, in quanto richiedono di sovra-dimensionare il sistema. Inoltre, mentre nei sistemi critici le condizioni di sovraccarico e di deadline miss erano eventi eccezionali, nei nuovi contesti applicativi esse diventano condizioni normali e in parte note in sede di progetto. Prende vita, quindi, il concetto di elaborazione *best-effort* anche all'interno della comunità real-time: nascono i primi algoritmi espressamente soft real-time, il cui obbiettivo(a questo punto) non è solo quello di garantire la correttezza, in termini di deadline, della parte critica ma, in particolare, quello di garantire anche le necessarie prestazioni, misurate secondo opportuni indicatori.

1.2 Il paradigma dei sistemi soft real-time

Quando la criticità del sistema viene meno, cioè il mancato rispetto di tutti i vincoli temporali non comporta disastri e non pregiudica la correttezza dell'elaborazione, quando la completa predicibilità del sistema e la caratterizzazione precisa dei processi non è possibile, il paradigma di progettazione dei sistemi di elaborazione in tempo reale con tecniche classiche non è più il metodo più efficiente. Le motivazioni sono diverse:

- Il dimensionamento del sistema basato sul caso peggiore è troppo dispendioso in termini di risorse computazionali. Per insiemi di processi con elevata varianza dei tempi di esecuzione, criteri di progetto basati sul worst-case-execution-time sono del tutto inadeguati.

- In un sistema non critico eventuali deadline-miss sono tollerate, se non previste, in sede di progetto.
- Il carico di lavoro del sistema è altamente variabile. In ambito soft real-time condizioni di sovraccarico, che si traducono nell'incapacità di elaborare tutti i task assegnati, sono realizzabili con elevata probabilità. Occorrono sistemi per la gestione dei sovraccarichi che consentano da un lato di massimizzare l'uso delle risorse, dall'altro, di garantire il livello minimo di prestazioni in termini di rispetto dei vincoli temporali.
- Il comportamento del sistema non è sempre predicibile. Un'applicazione real-time non critica di solito fa uso di sistemi di calcolo di tipo general-purpose in cui meccanismi come DMA, cache, interrupt, meccanismi di IO, algoritmi per il riordino ed esecuzione delle istruzione fuori ordine non consentono di prevedere il comportamento del sistema in modo deterministico.
- In presenza di risorse non sufficienti tutti gli algoritmi di schedulazione ottima, quale ad esempio Rate Monotonic e Earliest Deadline First, non possono più garantire il rispetto delle deadline. La capacità di schedulare i processi nel rispetto delle deadline decade in modo veloce, non permettendo più il rispetto dei vincoli temporali su diversi task.

Le due definizioni che seguono riassumono i concetti chiave che si delle considerazioni precedenti.

Definizione 1 *Un sistema di elaborazione soft real-time è un sistema caratterizzato da vincoli temporali non stringenti, il cui obbiettivo è massimizzare l'uso delle risorse a propria disposizione, e fornire sufficienti livelli di prestazioni secondo metriche che possono variare da applicazione ad applicazione.*

Definizione 2 *Un sistema real-time adattativo è un sistema in grado di reagire prontamente a cambiamenti delle condizioni operative e di carico e di modificare la politica di gestione delle risorse dinamicamente, allo scopo di garantire il miglior utilizzo del sistema.*

È da sottolineare che le due definizioni precedenti fanno riferimento ad aspetti diversi. La natura adattativa di un sistema real-time prescinde dal rispetto o meno di tutte le deadline. In questo senso un sistema può essere adattativo ed essere di tipo hard. Allo scopo di rispettare le deadline il meccanismo adattativo provvede a regolare l'accesso alla risorsa CPU in modo differente. All'opposto, un sistema soft real-time non necessita sempre di essere adattativo. Le prestazioni medie del sistema possono eventualmente essere determinate a tempo di progetto mediante opportune metodologie messe a disposizione della teoria. Può anche essere adattativo, quindi non rispettare le deadline di tutti i task, tuttavia cercare di conseguire il maggior numero di successi mediante qualche meccanismo di gestione del carico. Quindi le due definizioni fanno riferimento ad attributi distinti delle applicazioni.

All'interno di questa tesi l'ambito applicativo che considereremo sarà quello dei *sistemi soft real-time di tipo adattativo*.

1.3 Apprendimento e schedulazione di processi in tempo reale

In ciascun momento, quando un processo diventa attivo, quando un processo termina o in un qualsiasi istante in cui occorra, lo schedulatore deve prendere una *decisione*. La decisione riguarda ad esempio quale sarà il prossimo task da eseguire, come organizzare la coda dei processi pronti, etc. Il fatto che questa scelta sia poi insita nella politica di schedulazione è una mera conseguenza del fatto che **è stata definita la politica stessa**.

Una politica in generale, non solo all'interno del contesto schedulazione, non è altro che un insieme di regole/azioni che occorre seguire per ottenere un certo risultato. Da questo punto di vista, quindi, definendo il generico algoritmo di scheduling A non facciamo altro che *risolvere un problema decisionale*: quale processo p al tempo t devo attivare affinchè tutti i processi possano terminare entro le loro deadline.

Che la schedulazione di attività, anche real-time, sia un processo di decisione è ben noto e lo dimostrano i diversi approcci con il quale è stato affrontato da vari ricercatori. Non sono infatti novità l'applicazione di tecniche di intelligenza artificiale quale metodi di apprendimento con rinforzo [ZD95] [AMS97], logica fuzzy [MCM96], reti neurali [ZCBO91], algoritmi genetici, quali metodi di risoluzione e/o ottimizzazione per questa classe di problemi. È bene notare però come tutti i casi citati siano problemi di schedulazione di natura *statica*, cioè problemi in cui occorre trovare una politica di scheduling ad-hoc che verrà poi utilizzata così com'è per quello specifico problema.

Dal punto di vista dei sistemi soft real-time, in particolare per quanto riguarda l'adattamento del carico, l'utilizzo di tecniche di intelligenza artificiale è di rara applicazione. Per quanto noto, l'apprendimento con rinforzo non si trova applicato a nessun caso. Metodi basati su insiemi di regole fuzzy sono stati realizzati indipendentemente da diversi ricercatori [TC94] [L.P97] [LTY94] [WRJ96] [MCM96]. Metodi adattativi per sistemi distribuiti basati su RT-CORBA e fuzzy logic sono stati indagati in [DVJB02].

L'obbiettivo che ci si propone in questa tesi è però differente da quello proposto negli articoli citati. Nel contesto dei sistemi di elaborazione soft real-time adattativi a cui si rivolge la tesi, infatti, la politica di scheduling è già nota a priori ed è rappresentata dall'algoritmo Rate Monotonic. Il meccanismo di apprendimento indagato si concentra nella ricerca di una politica di gestione del carico del sistema mediante adattamento delle frequenze di lavoro dei vari processi. Stiamo cioè cercando di apprendere *come* e *quando* adattare il sistema.

In un certo senso, si tratta ancora una volta di applicare tecniche di intelligenza artificiale alla regolazione di un sistema dinamico.

1.4 Motivazioni e contributi

In letteratura la varietà delle tecniche per il controllo di sistemi real-time di tipo soft è amplissima. Tra i contributi fondamentali di ultima introduzione dati alla

comunità real-time è sicuramente da citare il lavoro svolto da Stankovic [SCST99]. Il suo approccio per assicurare l'opportuna della qualità di servizio di un sistema real-time mediante controllo in retroazione ha dato l'input a molti altri ricercatori, per esempio [JPr00] [CLTS99] [Lu01].
Anche dal punto di vista di questa tesi, il lavoro di Stankovic costituisce un aggancio di fondamentale importanza. Il meccanismo dell'apprendimento, quello con rinforzo ancor più in particolare, può essere visto come un sorta di controllo in retroazione di tipo "senziente". Il meccanismo di controllo *non necessita del modello del sistema* allo scopo di formulare una politica di governo; al contrario, la politica stessa viene *appresa e maturata con l'esperienza*.

Anche non considerando l'eventuale similitudine con il controllo in retroazione, esistono altri motivi per prendere in esame un meccanismo adattativo basato su apprendimento. Qualunque sia l'approccio che si desidera seguire, per poter rendere reattivo il sistema real-time ad eventuali condizioni di sovraccarico, malfunzionamento o comunque ad eventi esterni non modellabili, occorre sempre rispondere a due domande:

1. Quale tipo di intervento occorre fare?

2. Quando si deve intervenire sul sistema ?

Per poter realizzare una politica adattativa completa entrambe le questioni devono essere chiarite. Un'ulteriore motivazione che ha portato allo studio di tecniche per l'apprendimento quale sistema di controllo adattativo per i sistemi in tempo reale risiede proprio nella capacità di questi ultimi di poter rispondere indifferentemente alla prima,alla seconda, oppure ad entrambe le domande.
In linea di principio, non conoscendo il sistema e non conoscendo il meccanismo di schedulazione ma solo quali sono i processi che si possono controllare, è possibile in *modo del tutto automatico* decidere sia quando operare un adattamento che la modalità con cui effettuarlo.

Ad esempio, in un precedente lavoro, Beccari et al. [Be99] hanno realizzato diversi algoritmi per la modulazione delle periodicità di processi periodici in condizioni di sovraccarico. Possiamo utilizzare questo risultato per rispondere alla seconda domanda e sfruttare l'apprendimento per *decidere* quando intervenire e

che tipo di modulazione effettuare.

Tale soluzione, apparentemente flessibile non è esente da problemi. Il processo di apprendimento è un processo lento, dispendioso anche in termini computazionali. Questo influirà immediatamente sulla scelta dell'algoritmo da utilizzare (reinforcement learning) e sui vincoli da imporre allo scopo di rendere il problema trattabile.

Il lavoro svolto in questa tesi non si propone quindi di dare al progettista real-time un nuovo strumento di immediato utilizzo, ma di aprire una nuova direzione verso cui volgere l'attenzione.

1.5 Organizzazione della tesi

Il capitolo 2 tratta la problematica della schedulazione dei processi periodici di tipo real-time. Verrà dapprima analizzato l'algoritmo Rate Monotonic e successivamente saranno brevemente elencati alcuni approcci soft-real time di interesse.

Nel capitolo 3 verrà introdotto il paradigma dell'apprendimento con rinforzo riferito in particolar modo al caso dell'ottimizzazione del rinforzo medio e alla risoluzione di problemi di tipo semi-Markov.

Nel capitolo 4 verrà analizzato il progetto del controllo mediante apprendimento con rinforzo. Saranno analizzati parte dei problemi riguardanti lo stato del sistema e descritte le scelte progettuali.

Nel capitolo 5 descriveremo brevemente la realizzazione di un simulatore per algoritmi di schedulazione di tipo preemptive real-time. Il simulatore sarà utilizzato per svolgere le attività di sperimentazione descritte nel capitolo 6.

La tesi si conclude con il capitolo 7 in cui varranno tratte le conclusioni sul lavoro svolto e presentate le possibili estensioni.

Infine in appendice sono presentati alcuni dettagli riguardanti il generatore dei processi utilizzato per creare i vari problemi di schedulazione sottoposti al simulatore.

Capitolo 2

Schedulazione di processi in tempo reale

2.1 Definizione di schedulazione

Dato un insieme di processi $J = \{J_1, \cdots, J_n\}$, una *schedulazione* è un assegnamento dei task al processore. Formalmente, una schedulazione è una funzione $\sigma : \Re^+ \rightarrow \mathcal{N}$ tale che $\forall\, t \in \Re^+, \exists\, t_1, t_2$ tali che $t \in [t_1, t_2)$ ed inoltre $\forall\, t' \in [t_1, t_2)$ si abbia $\sigma(t) = \sigma(t')$. In altri termini, $\sigma(t)$ è una funzione costante a tratti che indica se all'istante t il processore è libero, $\sigma(t) = 0$, oppure se il task J_k è in esecuzione, $\sigma(t) = k$.
Valgono inoltre le seguenti definizioni:

- L'istante in cui il $\sigma(t)$ cambia di valore viene detto *cambio di contesto* e corrisponde all'assegnazione al processore di un nuovo task oppure alla condizione di CPU libera.
- Ogni intervallo t_i, t_{i+1} in cui $\sigma(t)$ è costante è detto *time slice*.
- Una schedulazione si dice di tipo *preemptive* se il processo in esecuzione può essere sospeso in qualsiasi istante, allo scopo di assegnare il processore ad un altro task secondo una strategia prefissata. Una schedulazione preemptive si compone quindi di una sequenza di time slice, ciascuna costituente una porzione dell'intero tempo di esecuzione di un processo.

- Una schedulazione è *fattibile* se esiste un assegnamento dei task al processore che consente a tutti i task di essere completati rispettando i vincoli temporali imposti.
- Un insieme $\mathcal{J}$ si dice *schedulabile* se per esso esiste una schedulazione fattibile.

2.2 Vincoli temporali sui processi

Per definizione di real time, i processi devono rispettare dei vincoli temporali per garantire la validità dell'elaborazione. Un tipico vincolo temporale è rappresentato dalla *deadline*, il limite massimo entro cui il processo deve terminare la propria esecuzione.
A seconda delle conseguenze provocate dal mancato rispetto della propria deadline, i processi real-time vengono divisi in due categorie:

Hard- un processo real-time dice di tipo hard se la violazione della deadline comporta gravi conseguenze sul sistema.

Soft- un processo real-time si dice soft se il mancato rispetto della propria deadline non compromette il funzionamento del sistema pur essendo un evento indesiderato che causa una riduzione delle prestazioni.

Un sistema in cui esistono scadenze di task real-time di tipo hard vengono rispettate è detto **hard real-time**. È bene notare che un sistema hard real-time può contenere anche elaborazione di tipo soft, delle quali tuttavia non è costretto a rispettare i vincoli.
Il generico processo real-time, oltre che dalla deadline, è caratterizzato da:

- **tempo di arrivo** a_i: tempo in cui il processo diventa pronto per l'esecuzione;
- **tempo di esecuzione** C_i: tempo necessario al processore per eseguire completamente il processo *senza interruzioni*;
- **deadline** d_i: tempo massimo entro cui l'esecuzione del task deve terminare;
- **offset** o_i: tempo in cui il processo va in esecuzione la prima volta;

- **tempo di terminazione** f_i: tempo in cui il processo termina la sua esecuzione.

Un'altra caratteristica temporale di un processo real-time concerne la regolarità delle sue attivazioni. Si possono distinguere in questo senso due classi di processi: processi *periodici* e processi *aperiodici*. I processi aperiodici per i quali è previsto un intervallo minimo dei tempi di arrivo sono anche detti sporadici. o sporadici.

Il tempo della prima attivazione della prima istanza di un task periodico si chiama *fase*. Se ϕ indica la fase del task, la sua k-esima attivazione avverrà al tempo $t = \phi + (k-1)T$, dove T indica il *periodo* del processo. Per contro, i processi sporadici sono una sequenza di attività identiche, ciascuna delle quali caratterizzata da un proprio tempo di esecuzione e di arrivo.

2.3 Schedulazione di processi periodici: l'algoritmo Rate Monotonic

Allo scopo di prevedere il comportamento di un insieme di processi periodici, di solito vengono fatte delle ipotesi semplificative che consentano di ricavare risultati analitici sulle proprietà degli algoritmi di schedulazione per questa classe di processi.
La teoria che segue è stato originariamente sviluppata da Liu e Layland [LL73] ed estesa da innumerevoli ricercatori. Si assumano le seguenti ipotesi:

I1. Tutte le istanze di un processo periodico sono attivate ad una frequenza costante. L'intervallo T tra due attivazione è il periodo del task.

I2. Il tempo di esecuzione di processo è costante nel tempo e non varia da istanza ad istanza, C_i.

I3. La deadline di ciascun processo coincide con la fine del periodo corrente e pertanto coincide anche con la successiva richiesta di attivazione.

I4. Tutti i task periodici sono *indipendenti*. Questo significa che non si hanno ulteriori vincoli di precedenza tra i processi derivanti da accesso (in mo-

do mutuamente esclusivo) a risorse condivise, se non quello derivante dal processore stesso.

L'insieme delle ipotesi I1,I2,I3,I4 permette di caratterizzare il processo mediante due soli parametri: C_i e T_i.

2.3.1 Fattore di utilizzazione

Per un insieme di processi periodici Γ, si definisce *fattore di utilizzazione* $\mathcal{U}$ del processore la frazione di tempo utilizzata dalla CPU per eseguire l'insieme dei processi. Vale la relazione

$$\mathcal{U} = \sum_{i=0}^{n} \frac{C_i}{T_i}$$

dove $\frac{C_i}{T_i}$ rappresenta la frazione di tempo utilizzata dal processore per eseguire il task τ_i. Il fattore di utilizzazione fornisce dunque una misura dell'occupazione della CPU. Sebbene si possa aumentare tale utilizzo variando la frequenza oppure il tempo di esecuzione dei task, esiste un limite superiore $U_{ub}(\Gamma, A)$ oltre il quale l'insieme dei processi non è più schedulabile. Questo limite dipende sia dalle caratteristiche dell'insieme di processi che dall'algoritmo di schedulazione utilizzato.

Quando $U = U_{ub}$ il processore si dice *pienamente utilizzato*. In queste condizioni un incremento del tempo di esecuzione di un processo implica la non schedulabilità.
Si definisce *limite superiore minimo* dell'utilizzazione il minimo tra i fattori di utilizzazione su tutti gli insiemi di processi che utilizzano pienamente il processore

$$U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A)$$

Questa definizione è importante in quante permette di stabilire in modo semplice quando un insieme di processi è schedulabile oppure no. Vale infatti il teorema:

Teorema 2.1 *Un insieme di processi periodici Γ è schedulabile con un algoritmo $\mathcal{A}$ se risulta*

$$U < U_{lub}(A)$$

Inoltre, se il fattore di utilizzo di un insieme di task è superiore ad uno, la schedulazione non è possibile con nessun algoritmo. Sussiste infatti il seguente risultato

Teorema 2.2 *Un insieme di processi periodici* Γ *è non schedulabile se* $U > 1$

È importante sottolineare come entrambi i teoremi 2.1, 2.2 forniscano solamente condizioni sufficienti ma non necessarie per quanto concerne la schedulabilità di un dato insieme di processi. Va fatto inoltre notare come, nel caso $U_{lub} \equiv 1$, la condizione fornita dai teoremi 2.1 e 2.2 diventi condizione necessaria e sufficiente.

2.3.2 L'algoritmo Rate Monotonic

L'algoritmo rate monotonic (RM) è una semplice regola che assegna le priorità di esecuzione ai vari task in dipendenza al loro tasso di attivazione. Più precisamente, la priorità ai vari processi viene assegnata in modo monotono decrescente sulla base del periodo di attivazione: a frequenze di attivazione più grandi corrisponde una priorità più alta.
Essendo il periodo dei task fissato, l'algoritmo rate monotonic è di tipo *statico*, cioè le priorità sono assegnate prima della esecuzione e non cambiano nel tempo. Inoltre, RM è intrinsecamente preemptive: il processo corrente può subire una preemption ad opera di una nuova attivazione di task a priorità più alta.

Liu e Layland [LL73] hanno dimostrato che questo tipo di assegnamento delle priorità è *ottimo*, nel senso che dato un qualunque insieme di processi Γ che non può essere schedulato mediante RM, allora l'insieme non è schedulabile con nessun altra regola di assegnazione a priorità fissa.
Sempre Liu e Layland hanno inoltre derivato un primo limite minimo del fattore di utilizzazione per l'algoritmo rate monotonic per un generico insieme di n processi.

$$U_{lub} = n\left(2^{\frac{1}{n}} - 1\right)$$

Il limite decresce con n come mostrato in tabella 2.1 e converge al valore $U_{lub}^{\infty} = ln2 \simeq 0.69$ per n molto grandi.

È importante notare che la condizione $U < U_{lub}$ è soltanto sufficiente e non

n	U_{lub}
1	1.00
2	0.828
3	0.780
4	0.757
5	0.743

Tabella 2.1: Valori di U_{lub} proposto da Liu e Layland in funzione di n.

necessaria. Questo significa che qualora un insieme di processi abbia un fattore di utilizzazione superiore al limite di Layland e inferiore ad uno, nulla può essere detto riguardo alla schedulabilità dell'intero insieme.

Una grossa mole di lavoro è stata fatta allo scopo di migliorare il limite di schedulabilità e rilassare i vincoli imposti dalle ipotesi I1, I2, I3, I4. Lehoczky, Sha e Ding [LSD89] hanno condotto uno studio statistico su insiemi di processi generati in modo casuale stabilendo che l'utilizzazione limite è approssimativamente 0.88 e diventa unitaria qualora sussistano relazioni armoniche tra i processi.
Test per la schedulabilità esatta, quindi condizione necessaria e sufficiente, sono stati derivati indipendentemente da diversi ricercatori. In particolare, usando l'algoritmo proposto da Audsley [ABR+93], un insieme di processi è schedulabile se e solo se il tempo di risposta di ciascun processo nel caso peggiore è minore o uguale alla propria deadline. Il tempo di risposta peggiore, R_i si può calcolare utilizzando la formula iterativa

$$\begin{cases} R_i^{(0)} = & C_i \\ R_i^{(k)} = & C_i + \sum_{j:D_j<D_i} \left\lceil \dfrac{R_j^{(k-1)}}{T_j} \right\rceil \end{cases}$$

Il limite di questo algoritmo risiede nella sua complessit à che è pseudo-polinomiale. Altro approccio interessante è quello proposto da Bini [EG01], l'Hyperbolic Bound (HB), che migliora il limite di Liu e Layland soprattutto nei casi con $n \gg 1$.

Secondo questo metodo un insieme di processi è schedulabile se vale la relazione

$$\prod_{i=1}^{n}(U_i + 1) \leq 2$$

Un test più raffinato, infine, è stato proposto da Buttazzo e Bini in un loro recente articolo [EG02].

Oltre alla succitate tecniche, atte a migliorare il test di schedulabilità, sono state proposte diverse estensioni all'algoritmo originario. Lehozcky e Sha [LRL90] hanno esteso l'analisi RM in presenza di risorse condivise introducendo i protocolli di accesso alle risorse, quali ad esempio Priority Inheritance Protocol (PIP) e Priority Ceiling Protocol (PCP). Sprunt, Lehozcky e Sha [BSL89] hanno introdotto il concetto di *deferrable server* e *sporadic server* per provvedere alla gestione di processi sporadici utilizzando sempre l'analisi RMA.

Le estensioni sin qui presentate sono solo una piccola parte degli arricchimenti apportati all'analisi RM. Questo a conferma dell'interesse della comunità real-time verso questo strumento quale mezzo di progettazione di sistemi real-time critici ad alte prestazioni e garanzie. Le motivazioni principali sono da ricercare in diversi aspetti, tipici di tutti gli algoritmi di scheduling di tipo statico:

- Semplicità di realizzazione
- Basso overhead
- Possibilità di integrazione su sistemi operativi di tipo general-purpose preesistenti (vedi Linux,SunOS, Windows NT)
- *Predicibilità dei fallimenti*

Nonostante l'esistenza di algoritmi di schedulazione di tipo *dinamico*, quali EDF, che consentono di ottenere $U_{lub} \equiv 1$, gli innegabili pregi della predicibilità e della semplicità dell'algoritmo Rate Monotonic costituiscono un motivo vaido per continuare ad applicare questo tipo di analisi ai sistemi in tempo reale.

2.4 Algoritmi di scheduling in condizioni di sovraccarico

Quando un sistema in tempo reale si trova in condizioni di sovraccarico, cioè l'utilizzazione complessiva del processore risulta tale per cui l'insieme dei processi non può essere correttamente schedulato, non tutti i task completano entro la propria deadline. Purtroppo in queste condizioni non esiste una politica ottima e la schedulazione deve avvenire mediante algoritmi di tipo best-effort.

2.4.1 Schedulazione con meccanismi di accettazione. Approcci "value-based"

Un primo approccio al problema del sovraccarico è stato proposto da Stankovic e Ramamrithan [JR91] con l'algoritmo Spring, in cui le richieste di attivazione dei vari processi venivano vagliate da un meccanismo di accettazione che decideva, in base alla capacità del sistema di garantire l'esecuzione corretta del task, se eseguire o meno il processo. Il meccanismo di accettazione è stato poi utilizzato da moltissimi altri ricercatori ed è diventato un vero e proprio paradigma di progetto per sistemi in tempo reale in condizioni di risorse insufficienti.

Un passo successivo importante, sempre in questa direzione, è stato fatto da Locke in [C.D82]. Nel suo lavoro, Locke associa a ciascun task una funzione V il cui valore dipende dal tempo di terminazione del processo. V rappresenta l'importanza del singolo task nei confronti degli altri processi nell'insieme. Il meccanismo di scheduling di tipo best-effort proposto adotta una politica di assegnazione/reiezione dei task disegnata ad-hoc, basata sulla densita del valore dei task, intesa come rapporto tra valore del task e tempo di esecuzione. Il sistema calcola la probabilità che un sovraccarico avvenga basandosi sullo slack-time residuo. Quando la probabilità dell'evento supera una certa soglia vengono rimossi alcuni processi in ordine crescente di densità di valore.
Oltre che per l'approccio, il contributo di Locke è interessante anche dal punto di vista filosofico in quanto mette bene in evidenza lo stretto legame che sussiste tra processi di decisione best-effort e meccanismi adattativi. Qualunque sia la politica

di schedulazione, accettazione/reiezione, qualunque sia il meccanismo applicato, occorre sempre prendere una decisione su come gestire i vari processi cercando di eseguire la migliore azione disponibile, basandosi sulle sole informazioni acquisite.

Meccanismi "value-based" sono stati successivamente proposti da diversi ricercatori. Un lavoro recente nel caso di sistemi soft real-time critici che utilizza l'algoritmo Rate Monotonic è stato presentato da Richardson [PS99]. Una visione generale di questa classe di algoritmi si trova in [BPB+00]. Di particolare rilevanza è il risultato attenuto da Baruah [G.B97]. Un algoritmo di scheduling di tipo best-effort ha un *fattore competitivo* ϕ_A $0 \leq \phi_A \leq 1$ se e solo se è in grado di accumulare in condizioni di sovraccarico un valore complessivo V_A pari a ϕ_A volte il valore che si otterrebbe da un algoritmo ottimo in condizioni nominali. Baruah ha inoltre stabilito, sotto certe ipotesi, il limite massimo del fattore competitivo raggiungibile da parte di qualunque algoritmo di schedulazione. Shasha e Koren [G.B97] hanno dimostrato che l'algoritmo D^{over} è ottimo, nel senso che che consente di ottenere il più alto fattore competitivo possibile. È da notare come l'ottimalità in questo senso non significa ottenere le migliori prestazioni in tutte le condizioni di carico nè l'ottimalità in senso stretto.

2.4.2 Meccanismi adattativi basati su modulazione delle frequenza

Oltre che mediante meccanismi di accettazione, la schedulazione di un insieme di processi può essere ottenuta variando alcuni dei parametri del processo allo scopo di ridurre il carico complessivo. Variabili di interesse possono essere la deadline, il tempo di esecuzione e il periodo. Mok e Kuo [KA91] hanno elaborato un meccanismo di riduzione graduale del carico mediante un'opportuna variazione del periodo dei vari task. L'algoritmo assume che tutti i task abbiano la stessa importanza e si propone di diminuire il numero delle frequenze fondamentali in modo da incrementare le capacità di schedulazione degli algoritmi a priorità statica quali Rate Monotonic. Buttazzo et al.[BLA98] hanno proposto l'elastic task model per algoritmi di sheduling sia a priorità statica che dinamica. Ciascun task è viene caratterizzato da cinque parametri: tempo di esecuzione , periodo nominale, perio-

do massimo T_{max}, periodo mimino T_{min} e da un coefficiente di elasticità $e \geq 0$. Un processo può modificare il proprio periodo nell'intervallo $[T_{min}, T_{max}]$. In risposta a questo cambiamento tutti i processi subiscono una variazione della propria frequenza di lavoro in modo da mantenere il sistema schedulabile. In analogia con un sistema elastico, l'utilizzazione da parte del processo è vista come la lunghezza di una molla, che presenta quindi vincoli di lunghezza, mentre il coefficiente e rappresenta la difficoltà di modificare il periodo del processo.
Lee et al. in [CRM96] e Beccari et al. in [Be99] hanno presentato diverse politiche di assegnazione dei periodi in condizioni di sovraccarico. In particolare Beccari ha realizzato diverse politiche di modulazione del periodo adatte per l'algoritmo Rate Monotonic.

2.4.3 Schedulazione statistica

Un approccio differente per la fattibilità e la gestione delle incertezze nelle schedulazione si può ottenere utilizzando lo statistical-scheduling (schedulazione-statistica). In questa classe di algoritmi le garanzie, in termini di completamento dei task entro la propria deadline e di ammissione dei task, non vengono specificate in modo rigido, bensì, in modo probabilistico.

Tia et al. in [TSDS$^+$95] forniscono un'analisi sulla probabilità che un processo rispetti la propria deadline. Viene proposta una estensione probabilistica della Time Demand Analysis denominata Probabilistic Time Demand Analysis (PDTA). Dalla conoscenza della distribuzione dei tempi di esecuzione de vari processi è possibile calcolare il limite inferiore della probabilità di rispetto della deadline per ciascun task. Il metodo fornito tuttavia non è più attendibile quando l'utilizzazione è prossima ad uno.
SRMS, Statistical Rate Monotonic Scheduling, proposto da Atlas et al [AA98] è una estensione diretta dell'algoritmo Rate Monotonic. Si compone di due parti: un controllo di ammissione dei task e uno schedulatore di tipo statico preemptive. Lo scopo dell'algoritmo di scheduling proposto è quello di elaborare processi con tempi di esecuzione estremamente variabili in modo che in "media" il rispetto delle deadline sia assicurato. Il sistema di ammissione rilascia un processo solo se la sua deadline è garantita. L'analisi effettuata in [AA98] si riferisce al caso di processi

armonici.
Gardner in [M.K99] propone uno studio statistico interessante. Introduce il concetto di Stochastic Time Demand Analysis che estende in un certo senso il precedente lavoro in [TSDS$^+$95]: anche in questo caso, infatti, viene calcolato il limite inferiore della probabilità che la deadline del processo venga rispettata. L'analisi viene poi arricchita mediante l'estensione al caso multiprocessore e in presenza di mutua esclusione.

Approcci simili come spirito, legati in particolar modo ai sistemi embedded, sono stati presentati da Zhou,Hu e Sha [ZHS95], [XTS01].

2.4.4 Approcci basati su retroazione

Nell'ambito dei controlli automatici, nello specifico nel campo del controllo dei sistemi in retroazione, metodologie di progetto collaudate che consentono di progettare sistemi di controllo opportunamente dimensionati sono presenti ormai da anni. Un algoritmo di scheduling in tempo reale di tipo adattativo può essere visto come un sistema di controllo. Infatti, se consideriamo il sottosistema in tempo reale, costituito dai processi da schedulare e dalla risorsa CPU, appare evidente come già il solo meccanismo di accettazione/reiezione possa essere visto come un sistema di controllo. Analizzando infatti opportuni segnali di ingresso (utilizzazione del processore, slack-time,etc.) elabora un segnale di controllo (task da accettare/rigettare) che va a modificare il comportamento del sistema.
Un primo passo in questa direzione è stato fatto da Stankovic [SCST99] presentando l'algoritmo di scheduling Feedback Control EDF (FC-EDF). Un controllore di tipo PID regola il deadline miss-ratio di un insieme di processi soft real-time con tempo di esecuzione variabile, variando l'utilizzazione dei vari task. Viene assunto che il singolo processo possa variare il proprio livello di servizio (in termini di tempo di esecuzione) passando attraverso diverse versioni dello stesso algoritmo. È inoltre presente un sistema di accettazione in grado di gestire variazioni troppo grandi da parte del carico di lavoro.
Lu et al. in [CLTS99] hanno esteso l'approccio proposto da Stankovic, aggiungendo un ulteriore controllo PID sull'utilizzo del processore. Combinando i due segnali di controllo, $FC - EDF^2$ fornisce ottimi risultati. Lu et al. in [Lu01]

hanno ulteriormente esteso il framework suddividendo il controllo in più moduli funzionali e aggiungendo un sistema per la gestione della qualità di servizio.
Eker et al. [JPr00] hanno proposto uno schedulatore in retroazione legato al caso dei controlli quadratici-lineari (LQ). Il problema della distribuzione delle risorse viene modellato come problema di ottimizzazione ricorsiva sulla derivata della funzione costo associata al controllo LQ. Visto l'elevato costo della risoluzione completa, viene presentata una soluzione approssimata mediante approssimazione quadratica della funzione costo.
Cervin [A.C03] riprende questi concetti applicandoli al Control Server, un modello di nuova concezione per il controllo di sistemi in tempo reale.

Capitolo 3

Apprendimento con rinforzo e processi decisionali di Markov

L'apprendimento con rinforzo (Reinforcement Learning) si propone di risolvere il problema dell'evoluzione del comportamento di un agente in grado di percepire ed agire all'interno di un ambiente. Alcuni esempi possono essere far apprendere ad un robot a navigare all'interno di una struttura, ottimizzare la produzione di un industria manifatturiera, imparare a giocare a giochi da tavolo, trovare la miglior soluzione per la gestione di un gruppo di ascensori, etc.
L'apprendimento viene portato termine assegnando premi e punizioni alle azioni fatte dall'agente, entrambi dipendenti della risposta ottenuta dall'ambiente durante l'interazione da parte dell'agente.

Nei paragrafi successivi verranno descritti due importanti modelli: i processi decisionali di Markov e i processi decisionali di tipo semi Markov. Entrambi i modelli vengono utilizzati nella rappresentazione matematica del processo di apprendimento con rinforzo.

3.1 Processi decisionali di Markov

Un processo decisionale di Markov è un processo stocastico caratterizzato da cinque elementi: epoca di decisione, stati, azioni, probabilità di transizione e rinforzo.

Inoltre è presente un agente (colui che prende le decisioni) che controlla il percorso del processo stocastico. Ad un certo punto del percorso e ad un certo istante t, l'agente interviene e prende una decisione che influenzerà l'evoluzione futura del processo. Questi istanti vengono detti *epoche di decisione*, mentre le decisioni prese assumono la connotazione di *azioni*.
Ad ogni epoca decisionale il sistema si ritrova in un certo stato s. Come risultato dell'azione fatta dall'agente, il sistema si troverà quindi in nuovo stato s' con una probabilità $P_{ss'}$ detta *probabilità di transizione*. L'agente, a seguito delle sue azioni, riceve da parte dell'ambiente un segnale di rinforzo r che potrà essere positivo o negativo a seconda del tipo di azione effettuata e dalla avvenuta transizione di stato. Occorre subito notare come un processo di Markov sia un processo di tipo *discreto* in cui sia i cambiamenti del tempo che le transizioni di stato avvengono solamente in corrispondenza alla epoca di decisione.
L'insieme

$$\mathbf{X} = \{X_n : n \in \mathcal{N}, X_n \in \mathcal{S}\}$$

rappresenta la catena di Markov sottostante il generico MDP, dove X_n indica lo stato del sistema alla $n-esima$ epoca di decisione e $\mathcal{S}$ rappresenta lo spazio degli stati. In ciascuna epoca n, l'azione presa è $A_n = a \in A_i$ ove A_i rappresenta l'insieme di tutte le possibili azioni nello stato i e $\cup A_i = \mathcal{A}$. Ad ogni azione a è associata una *matrice di transizione* $P(a)$ della catena $\mathbf{X}$, in cui ciascun elemento $P_{ij(a)}$ rappresenta la probabilità di muoversi dallo stato i allo stato j a seguito dell'azione a. Viene definita inoltre una funzione rinforzo $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$, dove $r(i, a)$ é il rinforzo previsto per aver scelto di eseguire l'azione a nello stato i.

Definizione 3 *Un processo stocastico in cui l'evoluzione è schematizzabile mediante una catena di Markov si dice processo di decisione di Markov se la seguente condizione è soddisfatta*

$$\begin{aligned} &\forall n \in \mathcal{N}, j \in \mathcal{S} \\ P\{X_{n+1} = j | X_0, \ldots, X_n\} \\ = P\{X_{n+1} = j | X_n\} \end{aligned} \tag{3.1}$$

L'equazione 3.1 costituisce la condizione di Markov. Un processo di Markov è quindi un processo stocastico senza memoria, in cui lo stato successivo dipende solo dallo stato presente e non dalla storia pregressa.

La risoluzione di un MDP è rappresentata da una *politica deterministica e stazionaria* $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ che massimizza un qualche criterio di ottimalità. All'interno di questa tesi verrà utilizzato un criterio di ottimalità basato sul *modello a rinforzo medio*, scelta non casuale ma supportata da opportune considerazioni che verranno esposte nei capitoli successivi.

3.1.1 Tecniche per la risoluzione di MDP:l'algoritmo Value Iteration

Definizione 4 *Si definisce* guadagno *di un MDP ottenuto a partire da uno stato i seguendo la politica π il valore medio del rinforzo definito come*

$$g^{pi} = \lim_{n\rightarrow\infty} \frac{1}{n} E_i^{\pi}\{\sum_{i=0}^{n} r(X_i, A_i)\}$$

L'equazione di ottimalità di Bellman nel caso di MDP basati sul criterio del valor medio del rinforzo è definita come

$$V^*(s) = \max_{a\in A_i}\{r(i,a) - g^{\pi} + \sum_{j\in\mathcal{S}} P_{ij}(a)V^*(j)\}$$

Maggiori dettagli al riguardo possono essere trovati in [S.M96].
L'equazione di Bellman consente di esprime matematicamente il concetto di politica ottima π^* come $\pi^* = \arg\max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S}$.
Qualunque algoritmo per la risoluzione di MDP, quindi, deve risolvere l'equazione di ottimalità di Bellman. Un possibile approccio è quello di trovare la funzione V^{π} ottima mediante un semplice algoritmo iterativo detto *value iteration*(figura 3.1). Si può dimostrare che value-iteration converge alla soluzione ottima V^* cercata.

1. Setta il valore di V(s) in modo arbitrario.
2. per tutti gli stati $s \in \mathcal{S}$ calcola
$$V_{n+1}(s) = max_a\{r(i,a) + \sum_{j \in \mathcal{S}} P_{ij}(a)V_n\}$$
3. se $\max_{j \in \mathcal{S}}\{V_{n+1}(j) - V_n j\} - \min_{j \in \mathcal{S}}\{V_{n+1}(j) - V_n j\} \leq \epsilon$ allora vai al punto 4 altrimenti $n = n + 1$ e ritorna al punto 2
4. per tutti gli stati $i \in \mathcal{S}$ scegli l'azione ottima a in modo che $d_\epsilon(i)$ sia
$$d_\epsilon \in \arg\max_{a \in \mathcal{A}}\{r(i,a) + \sum_{j \in \mathcal{S}} P_{ij}(a)V_n\}$$
quindi fermati.

Figura 3.1: Schematizzazione dell'algoritmo value iteration.

3.2 Processi decisionali di tipo semi-Markov

Nei processi di decisione Markov, lo stato del sistema può variare solamente a seguito di una azione intrapresa dall'agente; tale transizione quindi avviene solamente in corrispondenza di ciascuna epoca di decisione. Esistono problemi decisionali di varia natura in cui la modellazione mediante catene di Markov non è sufficiente a descrivere la struttura probabilistica del processo stesso. Un esempio potrebbe essere quello di un sistema di produzione all'interno del quale è previsto che con una certa probabilità si verifichi un guasto. Questo problema non può essere modellato come MDP, ma può essere descritto utilizzando processi di decisione di tipo semi-Markov.

La caratteristica fondamentale che differenzia un SMDP da un MDP è il concetto di tempo. Mentre in un MDP il tempo evolve in modo discreto, tra un epoca di decisione e l'altra, in un SMDP il tempo è continuo. Le decisioni tuttavia sono permesse solo in istanti discreti, che in questo contesto assumono la connotazione di *eventi*. Tra le epoche di decisione, lo stato del sistema può variare continua-

mente, contrariamente a quanto succede nei MDP dove la transizione è dovuta alle azioni.

Consideriamo una catena di Markov $\mathbf{X}$ e la sequenza di valori in $\Re^+$ tale che $0 = T_0 < T_1 < T_2 < \dots$.
Il processo stocastico $(\mathbf{X}, \mathbf{T}) = \{X_n, T_n : n \in \mathcal{N}\}$ è detto processo rinnovo di Markov (MRP) con spazio degli stati $\mathcal{S}$ quando è soddisfatta la seguente condizione

$$\begin{aligned}&\forall n \in \mathcal{N}, j \in \mathcal{S}, t \in \Re^+\\&P\{X_{n+1} = j, T_{n+1} - T_n \leq t | X_0, \dots, X_n, T_0, \dots, T_n\}\\&= P\{X_{n+1} = j, T_{n+1} - T_n \leq t | X_n\}\end{aligned}$$

L'equazione 3.2 rappresenta l'estensione della condizione di Markov al caso continuo. In un processo di rinnovo di Markov, quindi, l'evoluzione dello stato $X_n \rightarrow X_{n+1}$ e il tempo di transizione $T_{n+1} - T_n$ dipendono solo dallo stato corrente X_n, indipendentemente dalle precedenti evoluzioni del sistema. Il processo è, quindi, senza memoria.

Si consideri ora il generico processo di rinnovo di Markov $(\mathbf{X}, \mathbf{T}) = \{X_n, T_n\}$. Definiamo un nuovo processo stocastico $\mathbf{Y} = \{Y_t : t \in \Re^+\}$ in modo che $Y_t = X_n$ se $T_n < t < T_{n+1}$. Il processo $\mathbf{Y}$ è detto *processo di tipo semi-Markov*. La caratteristica principale di Y_t è quella di essere un processo di tipo continuo, le cui transizioni di stato sono governate dal sottostante processo di rinnovamento di Markov. Tra due istanti temporali t_1 e t_2, quindi, possono avvenire diverse transizioni di stato.

Se il comportamento stocastico di un problema decisionale è caratterizzato dal processo $(\mathbf{X}, \mathbf{T})$, il problema viene chiamato *processo di decisionale di tipo semi-Markov*. In particolare, il processo $\mathbf{Y}$ viene detto *processo naturale* mentre il processo di rinnovamento di Markov ristretto alle sole epoche di decisione $\{\hat{X_m}, \hat{T_m}\}$ rappresenta il *processo decisionale*.

3.2.1 Struttura della funzione rinforzo e risoluzione di SMDP

Il rinforzo accumulato dall'agente nel caso SMDP può essere diviso in due parti. Un primo rinforzo $k(s, a)$ viene assegnato in seguito all'azione $a \in \mathcal{A}$. Un secondo contributo invece viene accumulato tra le due epoche di decisione al tasso $c(Y_t, s, a)$

per tutti gli $Y_t \in S$ durante la transizione di stato $s \to s'$.

$$r(x,a) = k(x,a) + E_x^a\{\int_0^\tau c(Y_t,x,a)dt\}$$

dove τ rappresenta il tempo di transizione tra le epoche di decisione $\hat{T_m}, \hat{T_{m+1}}$ e E_x^a il valore atteso.

Il guadagno $\rho^\pi(i)$ a partire dallo stato i a utilizzando la politica π per un processo di decisione di semi-Markov si definisce come

$$\lim_{N\to\infty} \frac{E_x^\pi \left\{\sum_{m=0}^N r(x,a)\right\}}{E_x^\pi \left\{\sum_{m=0}^N \tau_m(x,a)\right\}}$$

dove τ rappresenta il tempo di transizione tra le epoche di decisione. Utilizzando l'espressione per il rinforzo nel caso SMDP si ottiene la seguente espressione

$$\lim_{N\to\infty} \frac{E_x^\pi \left\{\sum_{m=0}^N \left[k(x,a) + \int_0^{\tau_N} c(Y_t,X_m,A_m)dt\right]\right\}}{E_x^\pi \left\{\sum_{m=0}^N (\hat{T_{m+1}} - \hat{T_m})\right\}}$$

Alla stregua di quanto visto per i processi di Markov, possiamo ora definire l'equazione ottima di Bellman per i processi decisionali di tipo semi-Markov con criterio di rinforzo medio come

$$V^* = \max_{a\in A_i}\{r(i,a) - \rho^\pi \tau(i,a) + \sum_{j\in\mathcal{S}} P_{ij}(a)R^*(j)\}$$

dove $\tau(i,a)$ rappresenta il tempo di soggiorno medio tra lo stato i ad opera dell'azione a, definita come $\tau(x,a) = E_x^\pi\{\hat{T_{m+1}} - \hat{T_m}\}$.
L'algoritmo value iteration può essere esteso direttamente al caso SMDP (figura 3.2). L'unica differenza rispetto al caso precedente riguarda il punto 2 dell'algoritmo, in cui nel calcolo del valore dello stato V viene sottratta una costante arbitraria $v^n(k^*)$ per normalizzare il risultato. Maggiori dettagli sono forniti in [DGMM99].

1. Setta il valore di V(s) in modo arbitrario e scegli un qualsiasi stato $k^* \in \mathcal{S}$. Setta inoltre $n = 0$.

2. per tutti gli stati $s \in \mathcal{S}$ calcola

$$V_{n+1}(s) = max_a\{r(i,a) - v^n(k^*) + \sum_{j \in \mathcal{S}} P_{ij}(a)V_n\}$$

3. se $\max_{j \in \mathcal{S}}\{V_{n+1}(j) - V_n j\} - \min_{j \in \mathcal{S}}\{V_{n+1}(j) - V_n j\} \leq \epsilon$ allora vai al punto 4 altrimenti $n = n + 1$ e ritorna al punto 2

4. per tutti gli stati $i \in \mathcal{S}$ scegli l'azione ottima a in modo che $d_\epsilon(i)$ sia

$$d_\epsilon \in \arg\max_{a \in \mathcal{A}}\{r(i,a) + \sum_{j \in \mathcal{S}} P_{ij}(a)V_n\}$$

quindi fermati.

Figura 3.2: Schematizzazione dell'algoritmo value iteration nel caso SMDP.

3.3 Apprendimento con rinforzo

Il paradigma dell'apprendimento con rinforzo consiste nel far apprendere ad un agente esperto [1]il comportamento ottimo da seguire all'interno di un sistema dinamico allo scopo di raggiungere determinati obbiettivi. L'agente decide quale è la migliore azione da intraprendere, in risposta alla propria percezione dello stato dell'ambiente, procedendo per tentativi. Come conseguenza delle proprie azioni, l'agente riceve dal sistema premi e punizioni sotto forma di valore numerico, a conferma che le operazioni da lui eseguite siano più o meno corrette sulla base di una qualche critica esterna assegnata.
Ad ogni azione che viene applicata ad esso, il sistema risponde compiendo una o più transizioni di stato, fino alla successiva epoca di decisione. Durante il tempo intercorso tra le due epoche, l'agente è in grado di percepire lo stato del sistema e di ricevere eventuali segnali di rinforzo.

[1]nell'ambito dell'Intelligenza Artificiale viene usato spesso anche il termine *senziente*.

Utilizzando queste informazioni ed un opportuno algoritmo, l'agente aggiorna le proprie conoscenze e sceglierà la successiva azione di conseguenza.

La successione di queste operazioni costituisce il singolo passo dell'intero processo di apprendimento. Il modello dell'apprendimento è rappresentato in figura 3.3.

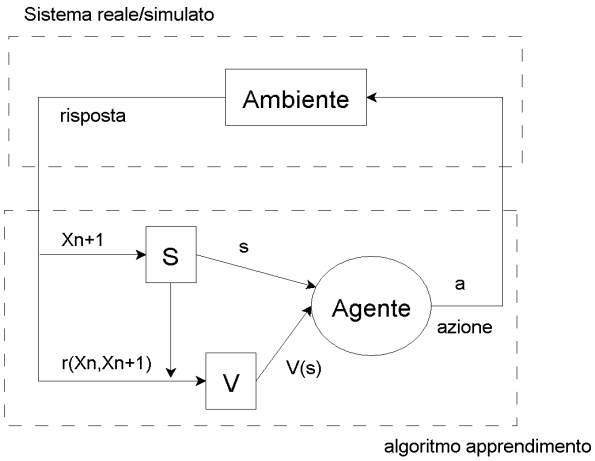

Figura 3.3: Modello dell'apprendimento con rinforzo

All'n-esimo passo d'interazione, basandosi sulla conoscenza dello stato x_n e sul valore dei rinforzi $V(x_n) = \{V(x_n, k) : k \in \mathcal{A}\}$ per tutte le azioni disponibili nello stato x_n, l'agente sceglie di compiere l'azione a tale per cui $V^*(i) = \max_k V(i, k)$. Il sistema evolve in modo stocastico in risposta alla coppia (x_n, a), generando in uscita alla successiva epoca di decisione un nuovo stato x_{n+1} e un rinforzo $r(x_n, x_{n+1})$. La nuova coppia $x_{n+1}, r(x_n, x_{n+1})$ permette all'agente di aggiornare il valore associato all'azione k in presenza dello stato x.

Il valore di ciascuna azione a viene mantenuto utilizzando una tabella $V(x, a)$ che contiene per tutte le possibile coppie (x, a) il valore del rinforzo accumulato. Questo approccio è valido fintanto che la dimensione dello spazio degli stati e delle azioni è trattabile. Quando il numero degli stati è eccessivamente grande, al limite un continuo, tale rappresentazione non è praticamente realizzabile, pertanto devono essere sfruttate tecniche di rappresentazione alternativa quali reti neurali,

alberi di decisione e altri sistemi per l'approssimazione di funzione. La trattazione di questi argomenti esula dagli scopi di in questa tesi.

3.3.1 Algoritmi di apprendimento con rinforzo di tipo model-free

Il problema dell'apprendimento con rinforzo è modellabile mediante processi di tipo Markov. Gli algoritmi di risoluzione presentati nelle sezioni precedenti si basano sulla conoscenza del modello probabilistico sottostante il processo allo scopo di poter utilizzare la probabilità di transizione $P(i, j)$ all'interno del procedimento iterativo. Non sempre però il modello può essere disponibile, soprattutto per sistemi complessi.
Un punto di forza dell'apprendimento con rinforzo risiede nella capacità di quest'ultimo di poter essere applicato anche in assenza di questo modello. Questa categoria di algoritmi prende il nome di apprendimento con rinforzo in assenza di modello, brevemente *model-free*.
Questa tipologia di algoritmi è stata utilizzata da moltissimi ricercatori nei più disparati ambiti: dal controllo di un gruppo di ascensori (Crites e Barto [RA96]), all' allocazione dinamica di canali in un sistema di telecomunicazione cellulare (Singh e Bertzekas [SD97]), dal classico controllo del pendolo inverso (Sutton [R.S95]), all'apprendimento di tecniche difensive utilizzate dai robot calciatori nella Robocup (Citare).
Il problema da affrontare è quello di come assegnare il valore ad un'azione che ha ripercussioni lontane nel futuro. Non è possibile attendere all'infinito per assicurarsi dell'effettivo rinforzo ed assegnare il credito. L'apprendimento con rinforzo assegna il rinforzo alla coppia stato-azione basandosi sul rinforzo immediato ottenuto a seguito dell'azione svolta e sul valore delle azioni nello stato successivo.

Sutton [RA98] ha proposto un approccio basato sulla *differenza temporale* del rinforzo chiamato $TD(\lambda)$, in cui, al variare di lambda, si ottengono tre diverse classi di algoritmi. Nel caso $lambda = 0$, si ha l'algoritmo $TD(0)$, in cui il rinforzo ottenuto va a modificare il solo valore della coppia (s, a) corrispondente al precedente passo dell'algoritmo. All'opposto, quando $0 < \lambda \leq 1$,il rinforzo ottenuto va

ad influire sul valore finale dell'azione a in tutti gli stati s recentemente visitati dall'agente. Caso particolare infine per $\lambda = 1$ in cui il valore di a viene aggiornato in tutto lo spazio $\mathcal{S}$. La regola di aggiornamento del valore della coppia stato-azione, a partire dall'equazione di Bellman nel caso MDP, con criterio basato sul rinforzo medio e $\lambda = 0$, è data da [2]

$$V_{new}(s,a) = (1-\alpha)V_{new}(s,a) + \alpha[r(s,s\prime,a) - \rho + \max_a V_{old}(s,a)]$$

dove α è il tasso di apprendimento e il termine tra parentesi quadre identifica il rinforzo stimato ottenuto a seguito dell'azione a nello stato s. L'algoritmo SMART [SNDA97], discusso successivamente, è simile come spirito all'approccio $TD(0)$ e, in un certo senso, ne rappresenta l'estensione al caso SMDP.

3.3.2 L'Algoritmo S.M.A.R.T

Si consideri l'equazione di Bellman nel caso SMDP:

$$V^* = \max_{a \in A_i}\{r(i,a) - \rho^\pi \tau(i,a) + \sum_{j \in \mathcal{S}} P_{ij}(a)R^*(j)\}$$

La somma pesata dei rinforzi ottenuti dall'agente seguendo la politica non stazionaria π può essere quindi scritta come

$$V^\pi(s,a) = r(i,a) - \rho^\pi \tau(i,a) + \sum_{s' \in \mathcal{S}} P_{ss'}(a)R^\pi(x)$$

e la politica ottima π^* diventa $\pi^*(s) = \arg\max_a V(s,a)$
Siccome $V(s,a)$ fa uso esplicito delle azioni, è possibile utilizzare una regola di aggiornamento basata sulla differenza temporale. Il valore della coppia (s,a) ottenuta alla n-esimo passo decisionale subisce alla (n+1)-esima epoca, in corrispondenza di un nuovo stato s', la seguente variazione

$$V_{new}(s,a) = (1-\alpha_n)V(s,a) + \alpha_n(r(s,s',a) - \rho_n^\pi \tau(s,s',a) + \max_a V_{old}(s',a)$$

dove $r(s,s',a)$ è il rinforzo accumulato, $\tau(s,s',a)$ rappresenta il tempo di transizione tra le due epoche s ed s', α_n è il tasso di apprendimento alla n-esima epoca

[2] l'equazione rappresenta il caso particolare $\lambda = 0$ dell'approssimazione stocastica di Robbins-Monro **??**

decisionale e ρ_n è il guadagno dell'SMDP, che viene stimato come rapporto tra il credito accumulato e il tempo totale di simulazione

$$\rho_n = \frac{\sum_{k=0}^{n} r(s_k, s_{k+1}, a_k)}{\sum_{k=0}^{n} \tau(_k, s_{k+1}, a_k)}$$

I particolari dell'algoritmo sono illustrati in figura 3.4.

1. La simulazione parte con epoca di decisione $n = 0$. Si inizializza $V_n(x, a) = \mathbf{0}$. Lo stato iniziale è x, arbitrario. Il rinforzo totale c_n e il tempo t_n sono pari a 0.
2. While $(n < MAXSTEP)$ do
   (a) Con probabilità $(1-\epsilon_n)$ scegli l'azione a che massimizza $V_n(s, a)$ altrimenti scegli un azione a caso.
   (b) Esegui l'azione a. Sia z il nuovo stato in corrispondenza della nuova epoca di decisione , τ il tempo di transizione tra le epoche e r_{imm} il rinforzo accumulato.
   (c) Modifica $V_n(x, a)$ utilizzando l'espressione
   $$V_{n+1}(x, a) = (1 - \alpha)V_n(x, a) + \alpha(r_{imm} - \rho_n\tau + \max_a V_n(z, a))$$
   (d) Nel caso una azione non casuale venga fatta nel passo 2(a)
   - $c_n = c_n + r_{imm}$
   - $t_n = t_n + \tau$
   - $\rho_n = \frac{c_n}{t_n}$

   (e) Setta lo stato corrente $x = z$ e $n = n + 1$. Decresci il valore di α e ϵ_n

Figura 3.4: L'algoritmo SMART

Per una piccola percentuale del tempo, dettata dalla variabile ϵ_n, viene scelta un'azione diversa da quella stimata come ottima (cioè quella con il rinforzo più alto). Questa pratica in letteratura viene chiamata *esplorazione* e gioca un ruolo

fondamentale nell'assicurare che tutti gli stati della catena di Markov vengano visitati e che tutte le azioni potenzialmente valide vengano provate.
Senza l'opportuno grado di esplorazione non si concede all'agente di sperimentare a sufficienza le possibili alternative; è quindi una pratica necessaria almeno in una prima fase dell'apprendimento. Mano a mano che l'agente compie esperienze le sue conoscenze, in termini di valore delle azioni, vanno consolidandosi; pur restando utile anche in questa fase, l'esplorazione dovrebbe essere mitigata a favore di uno sfruttamento della conoscenza acquisita (*exploitation*).
Per questo, il tasso di esplorazione ϵ_n viene fatto decrescere lentamente a 0, secondo la funzione

$$\Theta(n) = \frac{\Theta_0}{1 + \frac{n^2}{\Theta_l + n}}$$

dove Θ_0 e Θ_l sono due costanti libere definite dal progettista. Questa è soltanto una delle possibili scelte per la funzione Θ: l'unica regola da rispettare è che la successione degli ϵ_n sia decrescente.

Oltre all'approccio $\epsilon-greedy$ qui descritto, un altra strategia di esplorazione/sfruttamento molto utilizzata è quella dell' *esplorazione di Boltzmann*. In questo caso, la probabilità di scegliere un azione è data da

$$P(x, a) = \frac{e^{V(x,a)/T}}{\sum_{a'} e^{V(x,a')/T}}$$

Il parametro *temperatura* T regola l'esplorazione: valori grandi di T la favoriscono, mentre per valori piccoli della temperatura l'agente tende a sfruttare le conoscenze già acquisite. Anche in questo caso, occorre regolare il decadimento di T mediante un opportuna funzione lentamente decrescente. L'approccio di Boltzmann funziona ottimamente quando i valori delle azioni sono ben distinti.
In termini di convergenza dell'algoritmo verso la politica ottima, è condizione necessaria che tutti gli stati vengano visitati infinite volte. Inoltre, la serie degli α_n deve soddisfare a due criteri:

$$\sum_{n=0}^{\infty} \alpha_n = \infty \quad \sum_{n=0}^{\infty} [\alpha_n]^2 < \infty$$

Per questo motivo, al passo 5 dell'algoritmo, il tasso di apprendimento α_n viene fatto decrementare lentamente secondo una funzione monotona decrescente. Una

possibile scelta anche per α è la funzione $\Theta(n)$ indicata in precedenza. Un'obiezione che si può fare a questa formula è quella decrementare il tasso di apprendimento anche per quelle coppie stato-azione poco visitate. Una possibile soluzione è di utilizzare per α_n l'espressione $\alpha_n = \frac{1}{1 + visite(x, a)}$, dove $visite(x, a)$ indica il numero di volte che la coppia (x, a) è stata visitata. In questo caso però gli stati visitati frequentemente decadono troppo velocemente. Un'idea è di sfruttare entrambi i vantaggi delle due formulazioni utilizzando una funzione composta $\Theta(visit(x, a))$. Questo tipo di soluzione è possibile solo nel caso in cui il numero degli stati sia finito e di dimensioni ridotte. Qualora venga utilizzata per V una rappresentazione mediante reti neurali oppure altri meccanismi per l'approssimazione di funzione, occorre regolare la riduzione di α_n utilizzando una successione in stile Θ.

Capitolo 4

Schedulazione adattativa basata su apprendimento con rinforzo

Per poter applicare il paradigma dell'apprendimento con rinforzo con successo, è necessario, almeno in linea di principio, che il problema in esame possa essere modellato come processo di decisione di tipo Markov oppure di tipo semi Markov. Qualora la condizione di Markov questa condizione venga meno, le tecniche di apprendimento con rinforzo posso essere applicate comunque (casi quasi-Markoviani), tuttavia non sussistono più le garanzie sull'ottimalità della politica trovata.
Nel caso in esame, il problema della schedulazione adattativa verrà modellato come processo decisionale di tipo semi-Markov. La trattazione verrà impostata su un approccio logico/intuitivo piuttosto che teorico. Approcci più formali si deducono anche dai titoli dei lavori [YT91] e [Yih92] in cui il problema della schedulazione in tempo reale viene modellata espressamente come SMDP. L'impossibilità di reperire le pubblicazioni [YT91] e [Yih92] non ha permesso di poter sfruttare il lavoro svolto da questi ricercatori, anche se i lavori citati forniscono un supporto indiretto a sostegno all'approccio SMDP seguito all'interno di questa tesi.
Nel proseguo del capitolo, verranno trattate nel dettaglio le caratteristiche salienti del sistema adattativo, quali azioni a disposizione dell'agente, spazio degli stati e struttura della funzione rinforzo, motivando man mano le scelte effettuate.

4.1 Scheduling adattativo in tempo reale come processo decisionale di tipo semi-Markov

Consideriamo un generico insieme i processi Γ in cui le priorità dei processi sono assegnate secondo lo schema Rate Monotonic (fig.4.1). Supponiamo che all'istante

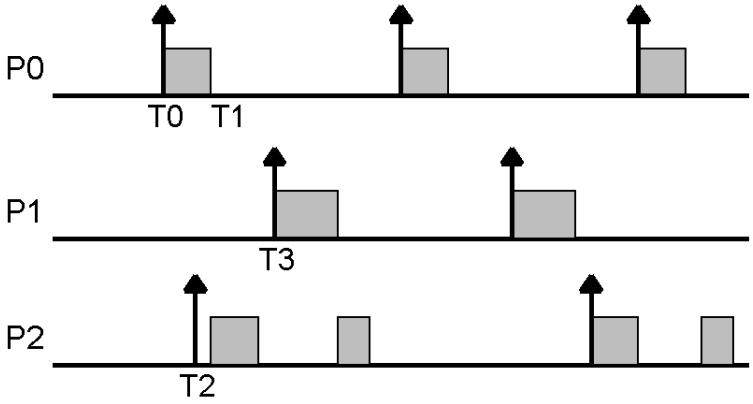

Figura 4.1:

T_0 il processo P_0 diventi attivo e che venga schedulato come processo corrente per essere elaborato dalla CPU. Al tempo T_1 arriva la richiesta di attivazione da parte del processo P_2: in base alle priorità dei task, il processore rimane assegnato a P_0 mentre P_2 viene inserito nell'insieme dei processi pronti per l'esecuzione. Al tempo T_2 il processo P_0 è stato elaborato: P_0 viene rimosso dall'insieme dei processi pronti e il processo P_2 va in esecuzione. Quando P_1 al tempo T_3 diventa attivo, P_2 subisce una preemption ad opera di P_1 da parte dello scheduler: P_1 passa in esecuzione mentre P_2 rimane in attesa di essere assegnato alla CPU.
Da questo semplice esempio è possibile osservare come, al generico istante T_k, all'interno del sistema real-time avvengano dei cambiamenti. In particolare, in corrispondenza dell' *attivazione* e della *terminazione* di un processo avvengono due modificazioni:

attivazione: aggiunta del nuovo processo attivo all'insieme dei processi pronti ed eventuale preemption.

terminazione: rimozione del processo elaborato dall'insieme dei processi pronti ed assegnazione del task a maggior priorità alla CPU.

Lo schedulatore, quindi, agisce sul sistema in modo discreto in corrispondenza degli istanti T_k che possono essere associati ad epoche di decisione. È importante osservare inoltre che tra due eventi T_k e T_{k+1}, lo stato del sistema inteso come insieme dei processi pronti e task assegnato al processore non cambia; cambiano invece altre variabili, ad esempio il tempo di esecuzione del processo corrente. Inoltre, la distanza $T_k - T_{k+1}$ è una variabile casuale di tipo continuo, che dipende dal tempo di esecuzione e dalla periodicità dei processi.
Le osservazioni esposte, mettono in evidenza lo stretto legame che esiste tra schedulazione di processi in tempo reale e i modelli decisionali di tipo semi-Markov esposti nel capitolo 3, [YT91] e [Yih92].
Qualora un meccanismo di adattamento venga introdotto, questo rappresenta un controllo in retroazione di un sistema dinamico che, in questo caso, può essere modellato mediante processi stocastici di tipo Markov. L'apprendimento con rinforzo, quindi, non rappresenta altro che un meccanismo di "self-tuning" del controllore: in base allo stato del sistema, la politica di regolazione cambia dinamicamente nel tempo. Inoltre, il controllo non è costretto ad avvenire solamente ad intervalli di tempo regolari, bensì anche qualora un evento di rilevanza per l'apprendimento si verifichi.
Un possibile esempio di processo decisionale applicato ai sistemi real-time può essere lo stesso meccanismo di ammissione/reiezione dei processi introdotto nel capitolo 2. Il controllo di ammissione rappresenta infatti una politica $\pi : \mathcal{S} \rightarrow \mathcal{A}$ che associa a ciascun stato s del sistema real-time un'azione a, corrispondente all'accettazione o meno del processo. Le epoche di decisione avvengono in corrispondenza della sola richiesta di attivazione di un singolo task mentre tra due interventi, variazioni dello stato interno del sistema, quali terminazione dei processi ed eventuali deadline miss, possono avvenire senza che questo meccanismo entri in azione.
In questo caso, la politica di accettazione è già assegnata, tuttavia nulla toglie che sia possibile pensare di apprenderla dinamicamente. Approcci di questo tipo, per

applicazioni multimediali e di allocazione dinamica delle celle in sistemi cellulari, sono stati presentati ad esempio in [SD97].

Intuitivamente, quindi, il problema dell'adattamento dello scheduling dei sistemi in tempo reale sembra poter essere descritto in modo efficace dai modelli semi-Markov. Ciononostante, è possibile che l'ipotesi fatta possa risultare non appropriata. Qualora questo si verifichi, il problema potrà essere considerato di tipo Markoviano solo in prima approssimazione e non sarà possibile garantire la convergenza dell'algoritmo utilizzato alla politica ottima.

4.2 Come intervenire: le azioni sui processi

Nell'architettura di schedulazione proposta, l'intervento dell'agente all'interno dell'ambiente (sistema) è limitato alla sola variazione del periodo di un sottoinsieme di task $\tau_1, \ldots, \tau_m$. L'insieme delle azioni disponibili per ciascun task è limitato a due sole possibilità:

- incrementare il periodo del task
- decrementare il periodo del task

L'intervento dell'agente sul periodo di ciascun task si realizza attraverso scostamenti di *piccola entità* di modo che l'adattamento sia di tipo graduale.Inoltre tale scostamento è non deterministico: la variazione relativa tra il vecchio e il nuovo periodo è una variabile casuale, distribuita uniformemente tra due valori v_l e v_h. Oltre che con l'azione sul singolo task, l'insieme delle frequenze di lavoro può essere modificato durante l'apprendimento mediante due ulteriori azioni:

- portando il periodo del task al suo valore massimo
- portando il periodo del task al suo valore minimo

Interventi del tipo proposto forniscono la possibilità, durante una prima fase di esplorazione, di visitare alcuni stati difficilmente raggiungibili da parte dell'agente. Non solo, permettono anche di ottenere soluzioni di adattamento brusco seguite da un aggiustamento successivo del periodo dei vari processi. Inoltre, l'insieme di

azioni consente di limitare il numero delle operazioni da parte del controllore.
La scelta porta con sè anche uno svantaggio: l'instaurarsi durante le prime fasi di apprendimento di politiche adattative di tipo *on-off*.

Utilizzando le sole azioni indicate, il controllore ad ogni iterazione modifica in un modo o nell'altro il periodo di uno o più processi. Questa soluzione, pur essendo in linea di principio possibile, da un punto di vista pratico non è nè ammissibile nè utile. Variare il periodo di un processo è un'operazione costosa in termini computazionali ed inoltre non presenta alcun vantaggio qualora il sottosistema realtime stia funzionando in modo corretto.
Pertanto, è prevista la presenza di un'azione nulla, che garantisce il mantenimento delle frequenze di lavoro.Il numero delle possibili azioni, quindi, risulta essere $|A| = 3 + 2 \cdot k$, funzione lineare del numero k dei task controllati, risultando dunque di dimensioni trattabili.

Azione	**Descrizione**
NOTHING	-
ALLUP	Collettiva, tutti i task con periodo al massimo
ALLDOWN	Collettiva, tutti i task con periodo al minimo
UP	Periodo del Singolo task aumentato dell'X%
DOWN	Periodo del Singolo task diminuito dell'X%

Tabella 4.1: Funzione di Rinforzo

4.3 Quando intervenire: una base di eventi discreti per l'adattamento

Il processo di apprendimento procede per passi discreti (epoche decisionali o decision epoch). Quando e in funzione di quali eventi l'agente si troverá in condizione di dover intervenire è una scelta progettuale importante.

In un sistema realtime ci sono cinque eventi che possono essere definiti critici:

1. occorrenza di un Miss

2. occorrenza di un Hit
3. attivazione di un Task
4. terminazione di un Task
5. tempo di esecuzione maggiore del WCET stimato

Ciascuno di questi eventi rappresenta un possibile punto di decisione per l'agente preposto all'apprendimento. L'occorrenza di un Miss non condiziona di per sè il comportamento del sistema (per ipotesi soft real-time) ma è la manifestazione di un qualche malfunzionamento, interno o esterno. L'agente in questo caso deve rispondere a questo cambiamento ed eseguire l'opportuna azione di conseguenza.

L'occorrenza di un Hit è invece un evento di tipo opposto. Questo è ovviamente un sintomo di salute del sistema, pertanto le uniche azioni sensate potrebbero essere quella nulla o, al limite, quella di incremento della frequenza di lavoro.

Va posta altresì attenzione riguardo all'ultimo evento dell'elenco, la rilevazione di un tempo di esecuzione maggiore del WCET stimato. La sua importanza risiede nella capacità di prevenzione di un imminente o probabile stato di sovraccarico del sistema. Poichè la risorsa CPU viene impiegata più del previsto incombe infatti il rischio di deadline miss per i task a minor priorità. Forzare l'agente a prendere una decisione consentirebbe di regolare il periodo dei vari task preventivamente, limitando il danno riportato.

Infine, ammettere la possibilitá di prendere decisioni in fase di attivazione e/o terminazione di un determinato task fornisce spunti per politiche preventive.

Il prezzo da pagare per un controllo di questo tipo è un aumento nella dimensione dello spazio degli stati necessari all'agente per distinguere le possibili situazioni. Infatti occorre tenere traccia nello stato del sistema di qualche variabile che sia ricollegabile agli eventi suddetti: per l'evento Miss indicatori pertinenti sono sicuramente la sorgente dell'evento oppure il rapporto tra il numero miss e numero di attivazioni del singolo task (Miss Ratio).

Più complesso è trovare valori misurabili collegati agli eventi di attivazione, terminazione e superamento del WCET. Un' idea potrebbe essere quella di misurare lo slack-time di tutti i task a priorità minore del task terminato. Questa idea suggerisce immediatamente quali sono i task a rischio e permetterebbe al controllore di agire di conseguenza. Si può anche osservare che, poichè la misura del tempo di esecuzione può essere fatta solo quando il task è terminato, i due eventi terminazione e tempo di esecuzione maggiore del WCET vengono accorpati in uno solo.

Alla luce delle considerazioni precedenti, la base di eventi che costituisce le epoche di decisione è composta da due elementi, ovvero, DeadlineMiss e tempo di esecuzione maggiore del WCET. L'includere nelle epoche di decisione l'attivazione di un task è parsa una scelta ridondante. Questa riduzione nel numero di interazioni da parte dell'agente stesso con l'ambiente esterno (il sistema)non favorisce di certo l'apprendimento, anzi, lo rallenta. Questa considerazione spinge quindi ad aggiungere un ulteriore intervento, questa volta di natura periodica, da parte del controllore. Il periodo di attivazione giocherà un ruolo importante.
Riassumendo, l'agente interviene sul sistema su una base di tre possibili eventi "critici"

1. Occorrenza di una DeadlineMiss;
2. Tempo di esecuzione maggiore del WCET;
3. Scadenza periodica di P secondi.

4.4 Lo spazio degli stati

Consideriamo il singolo task

$$\tau = \{WCET, BCET, P_{min}, P_{max}, D\}$$

Il suo stato interno all'istante t è rappresentato da

$$\phi(t) = \{ET(t), P(t), MISS(t), A(t)\}$$

dove $ET(t)$ è il tempo di CPU consumato al tempo t, P l'attuale periodo, $MISS(t)$ una possibile misura "utile" del numero di Deadline Miss del task mentre $A(t)$ indica lo stato di attivazione del processo. L'insieme

$$\Phi = \{\phi_1(t), \phi_2(t), \ldots, \phi_n(t)\}$$

rappresenta quindi lo stato interno dell'intero task set al tempo t. Consideriamo ora l'insieme delle variabili

$$\Theta = \{PU(t), MR(t), \overline{PU}(t), \overline{MR}(t), \dot{PU}(t), \dot{MR}(t)\}$$

Le sigle PU e MR indicano, rispettivamente, l'Utilizzo del Processore e il Miss Ratio misurati all'istante t: in particolare,

$$PU(t) = \frac{CPU(t-\delta t) - CPU(t)}{\delta t} \quad MR(t) = \frac{\sum_{i=0}^{n} MISS(t-\delta t) - \sum_{i=0}^{n} MISS(t)}{\sum_{i=0}^{n} A(t-\delta t) - \sum_{i=0}^{n} A(t)}$$

$$\bar{PU}(t) = \frac{CPU(t)}{t} \qquad \bar{MR}(t) = \frac{\sum_{i=0}^{n} MISS(t)}{\sum_{i=0}^{n} A(t)}$$

$$\dot{PU}(t) = \frac{PU(t-\delta t) - PU(t)}{\delta t} \qquad \frac{MR(t-\delta t) - MR(t)}{\delta t}$$

L'insieme delle possibili configurazioni del sistema è dato da $\Psi = \Phi \cup \Theta$: è uno spazio continuo, quindi di dimensione infinita e pertanto intrattabile. Anche effettuando una discretizzazione *grossolana*, una rappresentazione tabellare è impossibile. Volendo utilizzare metodi approssimati basati su reti neurali o altre forme di approssimazione di funzione il problema della dimensione può essere risolto; tuttavia ne nascono altri legati alla difficoltà di convergenza dell'algoritmo e, soprattutto, all'aumento del tempo necessario all'apprendimento.
Per contenere la dimensione dello spazio degli stati entro valori accettabili, nell'ordine dell migliaia, è necessario:

- eliminare o accorpare parte delle variabili di interesse in stati *simili* ai fini del controllo.
- effettuare una opportuna discretizzazione;

- scegliere la combinazione di parametri più "sensibile" agli eventi.

4.4.1 Riduzione dello spazio

Passo 1

Allo scopo di ridurre le variabili in gioco, dallo stato del task ϕ è possibile scartare la variabile $A(t)$ in quanto non strettamente necessaria all'atto pratico. Accorpiamo inoltre la coppia $\{pu_i(t), P_i(t)\}$ nell'unica variabile $pu_i(t) = \frac{C_i(t)}{P_i(t)}$, fattore di utilizzo del processo. Dal punto di vista del controllo, conoscere l'esatto tempo di esecuzione del task consente una distinzione fine dello stato tuttavia, il parametro di utilizzazione, consente immediatamente di stabilire il peso computazionale del task in termini di schedulazione. Lo stato del task viene ad essere

$$\phi = \{pu(t), MISS(t)\}$$

Passo 2

Riduciamo la ridondanza all'interno di Θ eliminando le variabili meno sensibili alla variazioni di stato dei processi e agli eventi, quali le misure medie $\overline{PU}$ e $\overline{MR}$. Occorrono invece alcune precisazioni riguardo le misure sul Miss-Ratio. MR e $\dot{M}R$ forniscono una misura globale sull'andamento di questi indicatori nel sistema; qualora l'obbiettivo dell'apprendimento fosse quello di contenere il valore del Miss-Ratio entro una certa soglia, occorrerebbe tener traccia nello stato di entrambe le variabili. Poichè una indicazione più fine è già presente in ϕ, in questo caso ulteriori misure a carico di MR sono del tutto superflue.

La variazione positiva o negativa di carico misurata dalla variabile $\dot{PU}(t)$ può risultare importante in quanto variazioni consistenti nella derivata dell'utilizzo stanno a significare un possibile sovraccarico. Nel paragrafo 4.3 è stato detto che uno degli eventi che forzano l'agente a prendere una decisione è quando il tempo di esecuzione di un task supera il valore stimato WCET. Quindi la condizione di sovraccarico è già stata determinata dall'evento stesso; inoltre, anche PU ne tiene conto, presentando un valore più elevato. Poichè stiamo cercando un insieme minimo di parametri che ci permetta di identificare lo stato del sistema, eliminiamo

per ora anche $\dot{PU}$. Qualora effettivamente non si tenga conto dell'attivazione del processo di apprendimento in corrispondenza all'evento suddetto, potrebbe essere utile tenere traccia di questo indicatore.

Lo stato del sistema dopo le modifiche apportate risulta essere dato da:

$$\Psi = \{PU(t), pu_1(t), \cdots, pu_n(t), MISS_1(t), \cdots MISS_n(t)\}$$

Passo 3
Anche se ridotto, lo spazio degli stati è ancora di dimensione troppo grande e non gestibile. In realtà, è lo stato del singolo task ad essere la parte critica, in quanto la sua dimensione è di tipo combinatorio: $|\Phi| = |\phi|^m$ con m il numero dei task.

Punto 1. Al momento sono presenti due misure dell'utilizzazione: una complessiva del sistema e l'altra, più fine, del singolo task.
Essendo $PU = \sum_i pu_i$ possiamo inglobare conglobare gli utilizzi dei singoli task nell'unica variabile $PU(t)$. Dal punto di vista della schedulatore è sicuramente la somma dei singoli utilizzi a fornire una condizione di schedulabilità dell'insieme dei processi.[1] In questo caso l'uso del singolo utilizzo avrebbe permesso di stabilire con maggiore precisione i processi in sovraccarico e avrebbe portato benefici al controllo. Tuttavia, anche operando sul solo utilizzo complessivo del processore è sempre possibile riportare il sistema in condizioni operative soddisfacenti. Pertanto, in prima analisi, terremo conto della sola variabile $PU(t)$.

Punto 2. Non occorre tenere traccia delle miss per quei task il cui rispetto dei vincoli è garantito. Così facendo si ottiene $\Phi = \{MISS_1(t), MISS_2(t), \ldots MISS_k(t)\}$ con k il numero dei task soft real-time del sistema in esame.

Punto 3. Dato che l'utilizzo complessivo é dato dalla somma $PU_{hard} + PU_{soft}$ possiamo evidenziare queste due componenti e indicarle entrambe nello stato del sistema per permettere una distinzione migliore tra stato della parte

[1] La condizione è anche necessaria nel caso EDF, mentre è solo sufficiente nel caso RM.

controllabile(task soft) e stato della parte *non controllabile*(task hard).

Lo spazio degli stati viene allora ad essere ridotto a due possibili scenari:

$$\Omega = \{PU_h(t), PU_s(t), MISS_1(t), MISS_2(t), \dots\ MISS_k(t)\} \tag{4.1}$$

$$\Psi = \{PU(t), MISS_1(t), MISS_2(t), \dots\ MISS_k(t)\} \tag{4.2}$$

All'inizio del paragrafo è stato detto che $MISS(t)$ rappresenta una misura utile del numero dei Deadline Miss da parte di un task. Le scelte possibili sono diverse e verranno indicate nel capitolo dei risultati sperimentali.
Inoltre, anche se ridotto, lo spazio degli stati é ancora continuo: è indispensabile una sua discretizzazione opportuna allo scopo di contenerne le dimensioni entro i valori indicati precedentemente. Diverse proposte al riguardo verranno analizzate nel capitolo dei risultati sperimentali.

4.5 Struttura della funzione di rinforzo

La definizione della funzione di rinforzo viene spesso definita come la "black art" del Reinforcement Learning; questo per la difficoltà nell'ottimizzare il rinforzo in modo da indirizzare l'apprendimento nella giusta direzione.
Bisogna innanzitutto porsi dei chiari obiettivi: che cosa si vuole ottenere, o meglio, qual'è il risultato ultimo desiderato ? A questa domanda abbiamo già in parte risposto in 1 e in 2.4 presentando alcune delle alternative presenti in letteratura.
Conviene chiarire gli obbiettivi schematizzando per punti:

- Minimizzare il numero dei Deadline Miss complessivi;
- Raggiungere con l'adattamento, per quanto possibile, configurazioni stabili in cui tutti i processi completano la loro esecuzione secondo i tempi previsti;
- Mantenere, per quanto possibile, la frequenza di lavoro di ciascun task prossima alla frequenza nominale;
- Minimizzare il numero degli interventi.

Minimizzare il numero di Deadline Miss si traduce nel fornire un *rinforzo negativo* qualora questo evento si verifichi. Il raggiungimento di uno stato stabile senza miss (che coincide quindi al goal) si puó tradurre applicando un *rinforzo nullo*. È possibile fornire, eventualmente, un ulteriore rinforzo, questa volta *positivo*, in corrispondenza hit, l'evento che corrisponde ad uno stato di "salute" del sistema. Per favorire politiche che mantengano una frequenza di lavoro prossima alla nominale, un'idea è quella di utilizzare un rinforzo non lineare, basato ad esempio sulla distanza $|f| = (f_1^{max} - f_1)^2 + \cdots + (f_n^{max} - f_n)^2$; questo rinforzo deve essere *negativo* in quanto l'allontanamento dalla frequenza massima costituisce una condizione non desiderata.

Per quanto riguarda il numero degli interventi, minimizzarli significa anche minimizzare gli overhead derivanti sia dal calcolo delle nuove priorità, sia da quelli dovuti alla rischedulazione dei vari processi. Associando ancora una volta un rinforzo *negativo* qualora venga effettuata un'azione diversa da quella nulla si ottiene il risultato cercato. Questo segnale può eventualmente essere suddiviso eventualmente in due componenti: una derivante dall'overhead del calcolo delle priorità, l'altro dovuto alla rischedulazione dei task per associare questo rinforzo a motivazioni pratiche.

La tabella 4.2 sintetizza quanto detto.

Rinforzo	**Valore**
Deadline Miss	-
Distanza Frequenze	-
Esecuzione Azione	-
Task Hit	+

Tabella 4.2: Criteri di sintesi per la funzione di rinforzo

Capitolo 5

Simulatore di algoritmi di scheduling in tempo reale

Per poter studiare nuovi algoritmi adattativi, comportamento e prestazioni degli algoritmi di apprendimento, l'utilizzo diretto di un sistema operativo diventa un collo di bottiglia durante la fase di sviluppo.
Infatti non è possibile pensare di eseguire le prove direttamente a caldo su di un sistema operativo: un tale approccio non è flessibile in quanto non permette di controllare le condizioni operative al contorno con la dovuta precisione, come ad esempio particolari condizioni di sovraccarico. Inoltre non consente di creare con facilità insiemi di processi di prova di determinate caratteristiche quali carico complessivo, distribuzione dei tempi di esecuzione etc.
Ultima ma non meno importante nota anche la necessità di ricevere un feedback da parte delle soluzioni adattative, positivo o negativo che sia, con una certa velocità. Durante la fase di sviluppo di algoritmi di questo tipo è importante non dover aspettare ore prima di poter apportare una leggera modifica e di doverne aspettare altre per ottenere il successivo risultato.
La realizzazione di un simulatore, quindi, si rende necessaria. Il tipo di simulazione realizzata rientra nella categoria della simulazioni ad eventi discreti. Esistono diverse librerie, sia freeware che commerciali, che forniscono tutti gli strumenti necessari alla realizzazione di simulatore di questo tipo. La scelta è legata al tipo di linguaggio e alla flessibilità fornita. È stata utilizzata a questo scopo una libreria

non commerciale, C++SIM, di ottima fattura e ricca di funzionalità.
I paragrafi successivi forniranno una descrizione di massima del software realizzato senza entrare troppo nel dettaglio. L'enfasi sarà posta sull'architettura e su particolari scelte progettuali.

5.1 Modello del carico di lavoro

La definizione di carico di lavoro (o workload) può assumere diverse forme e sfumature. In questo contesto per *carico di lavoro* si intende l'insieme dei task sottomessi al sistema e il loro carico computazionale complessivo.
Il carico del sistema è modellato secondo un approccio a *processi periodici non interagenti*. Questo significa che ogni processo è indipendente dagli altri ed esegue le sue funzioni come se fosse l'unico task all'interno del sistema.

5.1.1 Specifiche sul tempo di esecuzione

Il tempo di esecuzione di ciascun processo viene specificato utilizzando quattro parametri:

1. caso peggiore $WCET$
2. caso migliore $BCET$
3. tempo di esecuzione medio stimatoEET
4. tempo di esecuzione medio AET

AET e EET vengono calcolati partendo dal WCET e dal BCET

$$EET = (WCET + BCET) \cdot \tfrac{1}{2}$$
$$AET = EET \cdot etf$$

Il parametro *etf* (execution time factor) può essere regolato a piacimento.
Il tempo di esecuzione di ciascun processo viene quindi calcolato come realizzazione di una variabile aleatoria uniformente distribuita nell'intervallo [AET,WCET] oppure nell'intervallo [BCET,AET]; la scelta dell'intervallo viene fatta a caso, con

probabilità $P = \frac{AET-BCET}{WCET-BCET}$. È importante notare come, variando *etf*, sia possibile alterare il tempo di esecuzione medio dei vari processi così come la distribuzione statistica dei tempi di esecuzione. Cambiando infatti l'AET, cambia anche la probabilità P e di conseguenza la frequenza con cui il tempo di esecuzione cade in ognuno dei due intervalli sopracitati. La classe **Job**, figura 5.1.1, provvede a fornire la necessaria interfaccia per la gestione del tempi di esecuzione del singolo task.

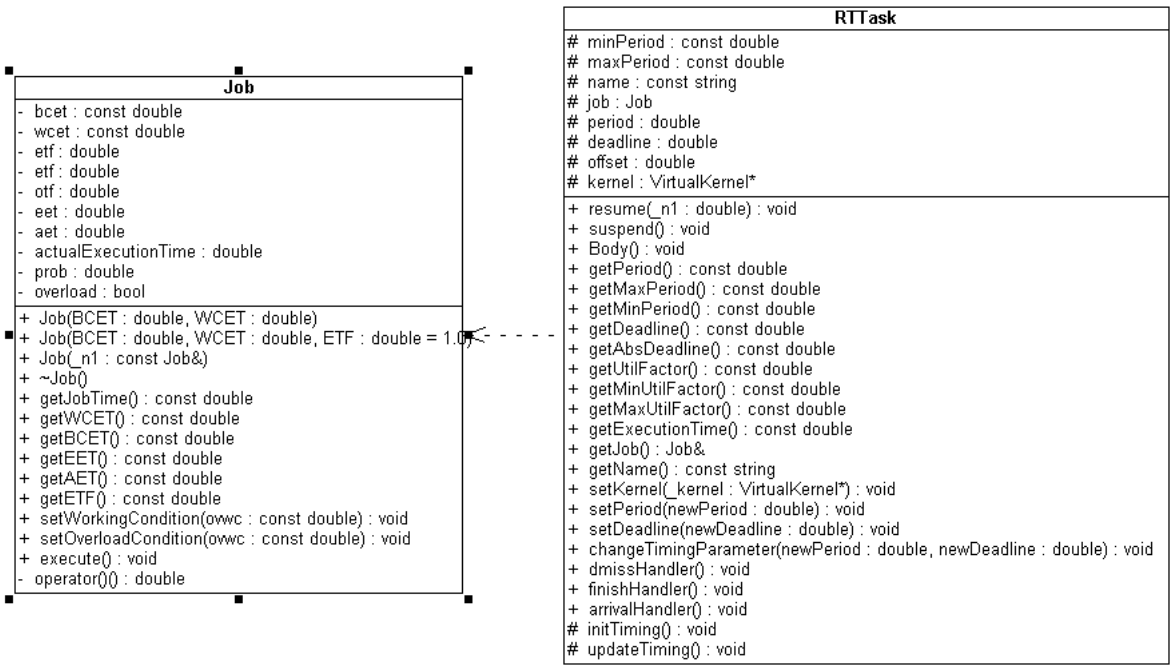

Figura 5.1: Diagramma UML delle classi Job e RTTask

5.1.2 Modello dei task

Un task di tipo hard realtime oltre che dal tempo proprio tempo di esecuzione, specificato come sopra, è caratterizzato da un periodo T e una deadline D, non nec-

essariamente coincidenti, e da un ritardo di attivazione iniziale (offset) opzionale. Formalmente un task hard realtime è rappresentato dalla n-pla $Task_H = \{P, D, O, WCET, BCE$
In aggiunta ai parametri presenti nel task hard real-time , per il generico task di tipo soft deve essere specificato anche un intervallo di valori $[T_{min}, T_{max}]$ all'interno del quale il periodo del task può essere fatto variare con continuità. Questo intervallo consente di modificare, a parità di tempo di esecuzione, il fattore di utilizzazione del processo, in accordo con $U_f = \frac{C_i}{T_i}$. Formalmente un task soft realtime è quindi una n-pla $Task_S = \{T_{min}, T_{max}, P, D, O, WCET, BCET, AET, EET\}$.

La classe **RTTask**, figura 5.1.1, fornisce un interfaccia comune per entrambe le categorie.

5.2 Progetto del nucleo

Il nucleo di qualunque sistema uni-processore è il livello più elementare dell'intero sistema operativo. Il compito del nucleo è quello di implementare i meccanismi per la sincronizzazione dei processi, gestione delle risorse, schedulazione etc.

Da questo punto di vista, quindi, l'emulazione di un nucleo di un sistema a processi deve almeno comprendere:

- Gestione e salvataggio dei contesti dei processi
- Realizzazione dei meccanismi di scheduling
- implementare le primitive di sincronizzazione, gestendo le transizioni di stato dei processi
- gestire le eccezioni, quali ad esempio eventuali Deadline Miss
- deve implementare una politica di assegnazione dei processi pronti all'unità di elaborazione (funzione di dispatching)
- gestire la creazione e distruzione dei processi.

In figura 5.2 sono rappresentate le principali classi che realizzano il nucleo del sistema.

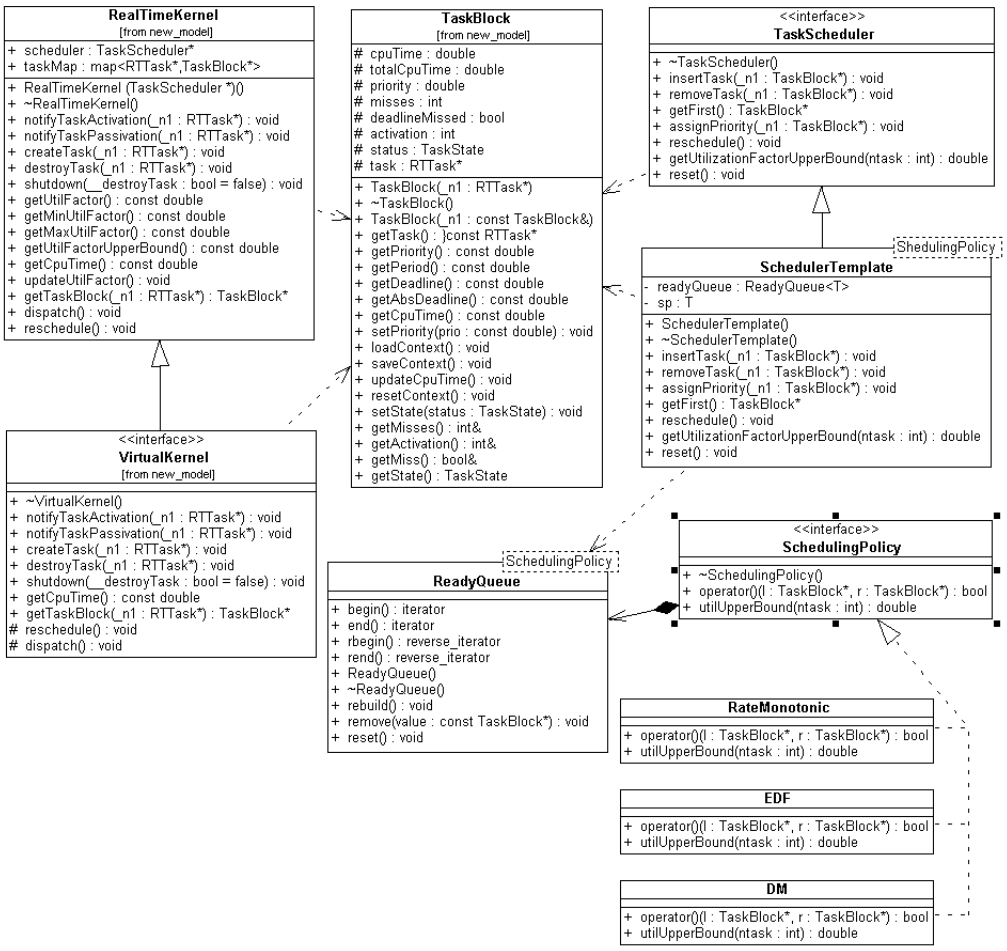

Figura 5.2: Diagramma UML delle classi componenti il nucleo

Il compito di assegnare le priorità ai vai processi e di gestire quindi l'ordinamento della *coda dei processi pronti*(ready queue) è stato assegnato alla classe TaskScheduler.
All'interno del simulatore lo scheduler viene visto come unico gestore della ready queue, il cui accesso non è consentito se non attraverso i metodi (eventualmente) forniti dall'interfaccia dallo scheduler stesso.
L'interfaccia TaskScheduler è astratta, ed è quindi necessaria una realizzazione concreta. Questa è rappresentata dalla classe SchedulerTemplate.
L'assegnazione delle priorità avviene specificando il tipo di ordinamento all'interno

della ready queue. Come scelta progettuale gli algoritmi di schedulazione implementati sono stati solo quelli di tipo preemptive. In particolare le classi EDF,RM, realizzano due possibili ordinamenti e contestualmente due differenti tipi di schedulazione.
I servizi quali creazione e gestione dello stato dei task, gestione del contesto e dispatching sono invece forniti dalla interfaccia VirtualKernel. Anche in questo caso l'interfaccia è astratta, la sua realizzazione concreta è rappresentata dalla classe RealTimeKernel.

5.2.1 Gestione dei contesti

A ciascun processo viene assegnato un descrittore definito dalla class TaskBlock che contiene tutte le informazioni sul suo stato attuale: tempo di esecuzione, priorità, e così via). La presenza di questa struttura non è del tutto necessaria in un simulatore, ma il vantaggio di poter gestire un TaskBlock al posto del processo stesso è evidente: processi e scheduler ora sono indipendenti, e tutte le operazioni eseguite durante schedulazione, ripristino di contesto non interessano la struttura dati RTTask.
L'operazione di cambio contesto, essendo il kernel preemptive, procede nel modo usuale:

1. Salvataggio dello stato del processo
2. Inserimento del TaskBlock all'interno della coda dei processi pronti o in attesa
3. Caricamento del TaskBlock a maggior priorità e ripristino dell'esecuzione del task.

Il cambio di contesto viene realizzato all'interno del metodo **dispatch** della classe RealTimeKernel.

5.3 Supporto per l'adattamento

Finora non si è parlato di adattamento ed in effetti il sistema sin qui realizzato è completamente privo di qualunque meccanismo. L'inserimento di un sistema di adattamento può essere visto come l'introduzione di un sistema di controllo in retroazione. Il sistema operativo simulato rappresenta il modello da controllare mentre il meccanismo di adattamento il regolatore, il quale agisce sul sistema in due modi:

1. Analizza il segnale di uscita del sistema generando un opportuno segnale di controllo.

2. Il segnale di controllo viene tradotto in un opportuno livello di servizio modificando in accordo il periodo di alcuni task.

5.3.1 Livello di servizio

Consideriamo il sistema K. L'insieme dei task T_K è composto da due sottoinsiemi T_H e T_S , rispettivamente l'insieme dei task hard-realtime e dei task soft-realtime. Il fattore di utilizzo $U_F(K)$ è dato dalla solita espressione
$\sum_{\imath=0}^{N} \frac{WCET_\imath}{P_\imath}$
dove N è il numero totale dei task.

Per livello o qualità di servizio di servizio di un task Γ si intende la coppia $\{WCET(\Gamma), T(\Gamma)\}$. L'insieme di tutti i livelli di servizio di Γ è dato da
$SL(\Gamma) = \{WCET_\jmath, T_\imath | \imath, \jmath \in I, J \subset \Re\}$. Si possono distinguere diversi tipologie di insiemi:

1. Livelli Di Servizio Discreti: in questo caso dei possibili livelli di servizio è finito. Si individuano tre diverse tipologie:

   (a) varia sia WCET che T. L'insieme SL è determinato da tutte le possibili combinazioni.

   (b) varia solo WCET oppure T

   (c) variano sia WCET che T ma in modo accoppiato, ovvero $\imath \equiv \jmath$

2. Livelli Di Servizio Continui: in questo caso si considera fisso il WCET e varia solamente T ma con continuità. L'insieme dei livelli è un continuo.

Si evince subito che per il generico task hard realtime esiste un unico livello di servizio. Per il generico soft-task invece, l'insieme dei livelli di servizio è di tipo continuo. Ovviamente a ciascun livello di servizio corrisponde un ben determinato fattore di utilizzo del sistema.
Un generico algoritmo di adattamento può quindi agire soltanto sul sottoinsieme dei task soft realtime, modificando il periodo di ciascun task secondo una propria politica.
Il controllo del sistema viene assegnato alle classi ServiceLevelController, Learn-

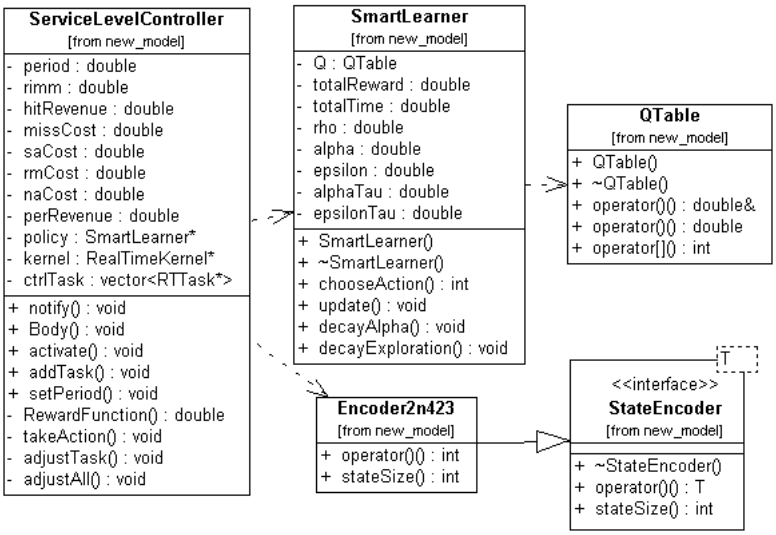

Figura 5.3: Diagramma UML della struttura di controllo e apprendimento

er e StateEncoder, figura 5.3.1 ServiceLevelController rappresenta il controllore all'interno del sistema e si avvale dell'uso delle altre due classi. StateEncoder fornisce un'interfaccia comune dedicata alla codifica dello stato del sistema controllato. La classe Learner è la realizzazione dell'algoritmo di apprendimento SMART(3.3.2) utilizzando per la funzione Q una forma tabellare, rappresentata dalla classe QTable.

5.4 Generazione dei sovraccarichi e gestione delle condizioni operative

Il sovraccarico del sistema può essere simulato in due modi differenti:

1. introduzione di un task fittizio: Il carico del sistema viene alterato introducendo un task con una certa priorità e carico computazionale prefissato.

2. modificando i tempi di esecuzione dei task. In questo caso il numero e tipo di task rimane lo stesso, ma i tempi di esecuzione non rispettano più i valori stimati.

Si è deciso di optare per la seconda soluzione. Nei paragrafi precedenti abbiamo visto come modificando il valore di etf possiamo modificare la distribuzione dei tempi di esecuzione dei processi. In particolare, per valori di etf minori dell'unità sono favoriti tempi di esecuzione compresi nell'intervallo [BCET,AET] con AET decrescente, metre per valori di etf superiori all'unità i tempi di esecuzione medi diventano prossimi al WCET.

Un sovraccarico in linea di principio può a questo punto essere simulato portando etf a valori tali da rendere AET maggiore di WCET. In questo modo il tempo di esecuzione del task è compreso nell'intervallo $[WCET, AET]$ con probabilità pari ad uno.
Il valore critico di etf è presto calcolato e varia da processo a processo, rendendo necessaria l'introduzione di due nuovi metodi: setWorkingCondition e setOverloadCondition.
Il primo permette di gestire variazioni del carico all'interno di $[BCET, WCET]$ utilizzando una notazione percentuale riferita al EET, denotata come condizione di lavoro operativa, *owc*. Il valore unitario coincide con un tempo di esecuzione sempre uguale a WCET, valori prossimi allo zero indicano tempi di esecuzione vicini a EET mentre valori negativi simulano tempi di esecuzione che si portano man mano verso il BCET.
Il secondo permette di indicare quanto deve essere il massimo incremento relativo di carico da parte del processo, inteso come aumento del proprio WCET. Il tempo di esecuzione del processo sarà quindi vincolato all'intervallo $[WCET, xWCET]$

con valor medio $.5 \cdot (1 + x) \cdot WCET$. È infatti logico pensare che , seppure in condizioni di sovraccarico, un processo non esegua sempre con lo stesso tempo di esecuzione purchè vincolato ad essere sempre superiore ai valori stimati.

5.4.1 Generatori di sovraccarichi

Dal punto di vista controllistico la condizione di sovraccarico può essere assimilata ad un segnale di disturbo d entrante nel sistema in esame. Descritto in questo modo, viene naturale associare al segnale una funzione F_{ov}, che indicheremo come funzione di overload, che ne descrive l'entità e la tipologia. Sono stati considerati cinque diversi tipi di disturbo

- funzione impulso
- funzione gradino
- funzione rampa
- funzione lineare a tratti (brevemente pwl)
- funzione random

La classe astratta OverloadFunction è l'interfaccia comune a tutte le funzioni, concretizzata poi dalla classi OverloadPulse, OverloadStep, OverloadRamp, OverloadPWL e OverloadRandom.

La generazione di questi segnali di disturbo è stata assegnata alla classe Generator. Ciascuna istanza di questa classe può essere configurata per emettere uno di questi segnali mediante il metodo *configure*. La creazione dell'instanza di una funzione di overload è subordinata alla classe Generator stessa. In un certo senso Generator e OverloadFunction costituiscono un patter di tipo strategy.

Poichè sono previste due modalità di gestione dei tempi di esecuzione è sembrato opportuno prevedere la presenza di due generatori dedicati: uno alla gestione di sovraccarichi, l'altro alla variazione delle condizioni operative.

Resta da stabilire quali siano i task interessati dal sovraccarico. La scelta più semplice è quella di gestire direttamente quali processi alterare, collegando alla stregua di un generatore di segnale la classe Generator ai task desiderati.

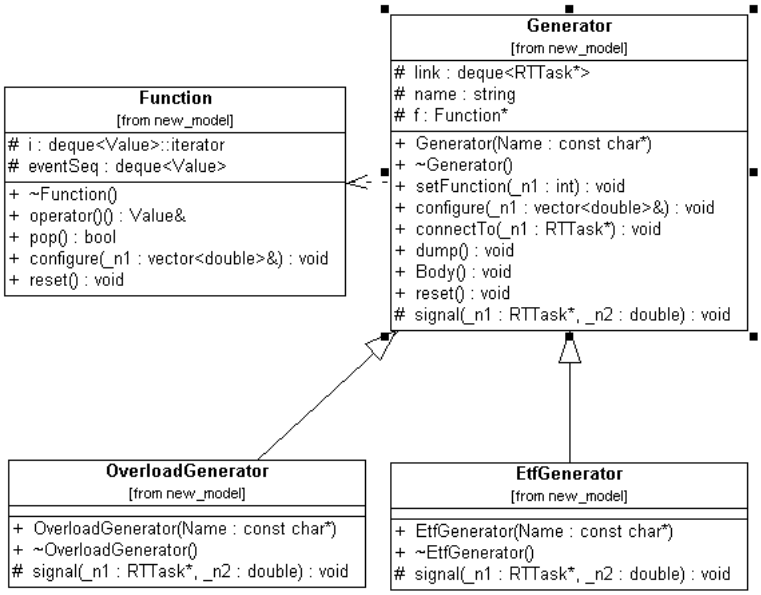

Figura 5.4: Generatori di segnale per la gestione dell'overload e del tempo di esecuzione

Una visione d'insieme del software è raffigurata in figura 5.4.1

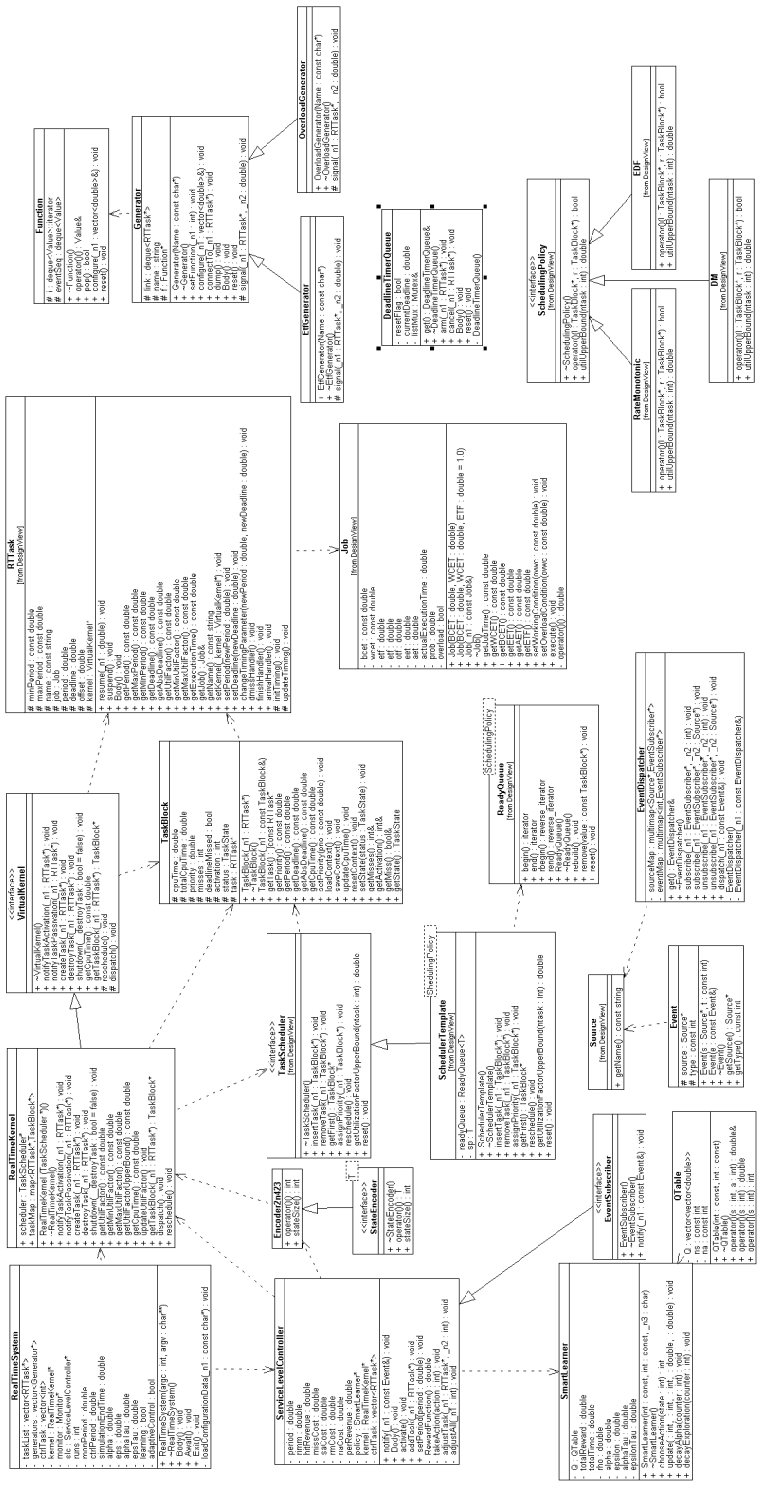

Figura 5.5: Diagramma UML delle principali classi che costituiscono il simulatore

Capitolo 6

Risultati Sperimentali

In questo capitolo verranno analizzati i risultati degli esperimenti simulativi condotti con il prototipo del simulatore presentato nel capitolo 5. Il meccanismo di apprendimento con rinforzo utilizzato è costituito dall'algoritmo SMART (sezione 3.3.2) e realizzato sulla base delle considerazioni esposte nel capitolo 4. Dopo una discussione preliminare delle problematiche legate alla misura delle variabili di interesse e alla discretizzazione dello stato del sistema, saranno presentati i risultati di alcuni di problemi di schedulazione con diverse caratteristiche allo scopo di valutare l'efficacia del modulo di controllo.

6.1 Premesse

Gli esperimenti sono stati effettuati utilizzando come problemi di ingresso insiemi di processi di piccola entità e la politica di adattamento appresa viene applicata soltanto ad un sottoinsieme ristretto di processi soft real-time. Tali scelte sono state adottate allo scopo di contenere inizialmente la complessità del problema in termini di dimensione dello spazio degli stati e numero di azioni disponibili.

In tabella 6.1 vengono riassunti tutti i parametri "liberi" riguardanti il processo di apprendimento. Il numero dei parametri che occorre regolare al fine di ottenere una politica appropriata è piuttosto elevato. Inoltre, non sempre esiste una relazione diretta tra la politica appresa e l'impostazione dei parametri scelta. I primi problemi sottoposti al simulatore cercheranno di esplicitare maggiormente

Descrizione	**Parametro**
Rinforzo deadline miss	r_{miss}
Rinforzo hit	r_{hit}
Rinforzo azione sul singolo processo	r_{asp}
Rinforzo cambio priorità	r_{rm}
Rinforzo periodo processi	r_{per}

Tabella 6.1: Parametri relativi al tasso di apprendimento ed esplorazione

Descrizione	**Parametro**
Velocità di apprendimento	α
Velocità di decadimento α	τ_α
Probabilità azione casuale	ϵ
Velocità di decadimento ϵ	τ_ϵ

Tabella 6.2: Parametri relativi alla funzione rinforzo

la sensibilità dell'apprendimento a varie combinazioni dei parametri in gioco.

La funzione rinforzo r ,come indicato nel capitolo 3, è definita come

$$\begin{aligned} r(i,j,a) = &- r_{miss} \cdot Miss + r_{hit} \cdot Hit \\ &- r_{asp} \cdot TaskAdattati \\ &- r_{rm} \cdot TaskControllati^2 \\ &- r_{per} \cdot \mathcal{F}(P_1, \ldots, P_k) \end{aligned}$$

dove la funzione $\mathcal{F}(P_1, \ldots, P_k)$ genera un rinforzo positivo dipendente dalla configurazione dei periodi dei vari task. Per la funzione $\mathcal{F}$ è stata utilizzata una formulazione basata sulla distanza tra il periodo corrente e quello nominale dei vari processi che, per semplicità di analisi, si suppone coincidere con il periodo minimo

$$\mathcal{F} = \sum_i (P_i^{min} - P_i)^2$$

Le epoche di decisione avvengono in corrispondenza di due determinati istanti temporali:

- Occorrenza di un Deadline Miss
- Ogni T secondi dall'ultimo intervento dell'agente

È stato scelto per ora di non attivare l'agente in corrispondenza del superamento del WCET da parte di un processo. Eventualmente tale condizione di attivazione verrà ripresa in considerazione qualora l'interazione dell'agente risulti troppo ridotta. D'altra parte non è possibile prevedere altri meccanismi di intervento se non l'aggiunta di qualche altro evento di attivazione, sulla base di quanto indicato nel capitolo 4.3.

6.1.1 Misura e discretizzazione dello stato del sistema

Come indicato nel capitolo 4 lo stato del sistema è rappresentato dall'insieme Ω o, in alternativa, dall'insieme Ψ

$$\Omega = \{PU_h(t), PU_s(t), MISS_1(t), MISS_2(t), \dots\ MISS_k(t)\}$$
$$\Psi = \{PU(t), MISS_1(t), MISS_2(t), \dots\ MISS_k(t)\}$$

dove $MISS(t)$ rappresenta una misura di qualche tipo del numero delle miss del singolo processo. È stato scelto di utilizzare per questo scopo una misura basata su miss-ratio. Una possibile alternativa era l'indicazione diretta del numero dell miss tra due eventi di attivazione, tuttavia, questo tipo di misura non dà nessuna indicazione riguardo alla frequenza con cui le deadline miss si verificano. In questo senso, la scelta del miss ratio rappresenta un'ottima soluzione, in quanto fornisce una percentuale degli insuccessi relativa alle attivazioni del processo.

Il periodo T del processo preposto all'apprendimento (controllore) diventa quindi un parametro particolarmente delicato: quando il controllore si attiva vengono misurate tutte le grandezze d'interesse che determinano lo stato del sistema. In particolare, la misura dell'utilizzo del processore e del miss ratio dei vari task dipendono fortemente dalla frequenza con la quale vengono osservate. Entrambe le grandezze, infatti, sono definite mediante rapporti incrementali secondo

le formulazioni presentate in 4 e di seguito riportate:

$$PU(t) = \frac{CPU(t-T) - CPU(t)}{T}$$
$$MR(t) = \frac{\sum miss(t) - \sum miss(t-T)}{\sum A(t) - \sum A(t-T)}$$

Per ottenere una misura utile di $PU(t)$ e $MR(t)$ teoricamente l'intervallo T deve essere grande, comunque maggiore del minimo periodo di tutti i task. All'aumentare della frequenza infatti, le quantità così misurate assumono connotazioni completamente diverse: $PU(t)$ indica al tempo t se il processore è occupato oppure libero, mentre $MR(t)$ indica se al tempo t è avvenuta o meno un deadline miss per il singolo processo. La misura ottimale del reale utilizzo del processore si otterrebbe utilizzando come T un multiplo o sottomultiplo dell'intero iper-periodo, scelta non ammissibile in quanto andrebbe a gravare ulteriormente sulla già limita interazione da parte del controllore. Occorre inoltre considerare che la misura non avviene a T costante ma variabile, in conseguenza dell'attivazione prevista anche a seguito del singolo miss. Questo insieme di fattori è causa di una misura estremamente "rumorosa" di $MR(t)$ o $PU(t)$.
Per risolvere questo problema è possibile stimare l'utilizzazione del processore (o del miss-ratio) mediante una misura *filtrata* che si ottiene tramite la seguente espressione:

$$\widetilde{PU}(t) = \widetilde{PU}(t-T) \cdot \rho + (1-\rho) \cdot PU(t)$$

con $0 \leq \rho < 1$. Per valori di ρ prossimi ad 1, $\widetilde{PU}(t)$ è una funzione morbida che tende ad approssimare il valore medio dell'utilizzazione. Per valori di ρ prossimi a zero si ottiene invece una misura più rumorosa ma al contempo più rapida di $PU(t)$. Utilizzando questa stima "filtrata" è possibile ricavare misure comunque reattive ai cambiamenti del carico e al contempo prive del problema del campionamento esposto in precedenza.

Una discussione analoga può essere fatta per quanto riguarda le misure di PU_h e PU_s. Entrambe le quantità soffrono dei problemi descritti precedentemente, pertanto devono essere utilizzati gli stessi accorgimenti. L'utilizzazione della parte controllata viene ottenuta calcolando il tempo di CPU consumato da ciascun processo e divendendolo per il tempo trascorso tra le due misurazioni, alla stregua

di quanto fatto per l'utilizzazione PU. Sebbene questa tecnica funzioni, è possibile sfruttare una misura più precisa, in cui viene stimato il tempo di esecuzione complessivo di ciascun processo al tempo t secondo la formulazione

$$\widetilde{C_i(t)} = \widetilde{C}_i(t-T) \cdot \rho + (1-\rho) \cdot C_i(t)$$

A questo punto l'utilizzazione del processore può essere calcolata per entrambe le componenti mediante la formula usuale $\sum_i \frac{\widehat{C}_i}{P_i}$.

Discretizzazione dello stato

Lo stato del sistema presenta due parti distinte: la prima riguarda variabili che possiamo definire "globali" (utilizzazione complessiva del processore), mentre la seconda rappresenta informazioni locali del singolo processo (miss ratio). La dimensione dello stato è quindi di natura *esponenziale*. Ad esempio, se consideriamo Ω:

$$|\Omega| = |PU| \cdot |MR|^k = \infty \cdot \infty^k$$

Per contenere la dimensione dello stato entro limiti accettabili è necessario ridurre il più possibile l'impatto della misura del miss ratio del singolo processo, in quanto poi la dimensione dello spazio discretizzato di PU agisce da ulteriore fattore moltiplicativo.

In figura 6.1 è raffigurato la dimensione di Ω_{task} al variare del numero dei processi controllati, k. La variabile MR viene discretizzata in un numero n di intervalli, nel caso rappresentato in figura gli intervalli sono due,tre e quattro. Già per valori di k non troppo elevati, $k \in [3, 6]$, il numero degli stati subisce un fortissimo incremento sebbene la discretizzazione di MR sia molto grossolana.
In una generica applicazione real-time, ragionevolmente realistica, in cui il numero dei processi controllati dal sistema adattativo si attesta attorno alla decina, non è possibile pensare di discretizzare il miss-ratio di ciascun task con un numero di intervalli superiore a tre. I risultati presentati sono stati ottenuti mediante una suddivisione in due soli intervalli, una scelta che limita molto il dettaglio a favore di un numero di stati ristretto. Eventualmente, questa suddivisione può anche essere adattata caso per caso variando il numero delle suddivisioni in base al numero dei processi che devono essere adattati.

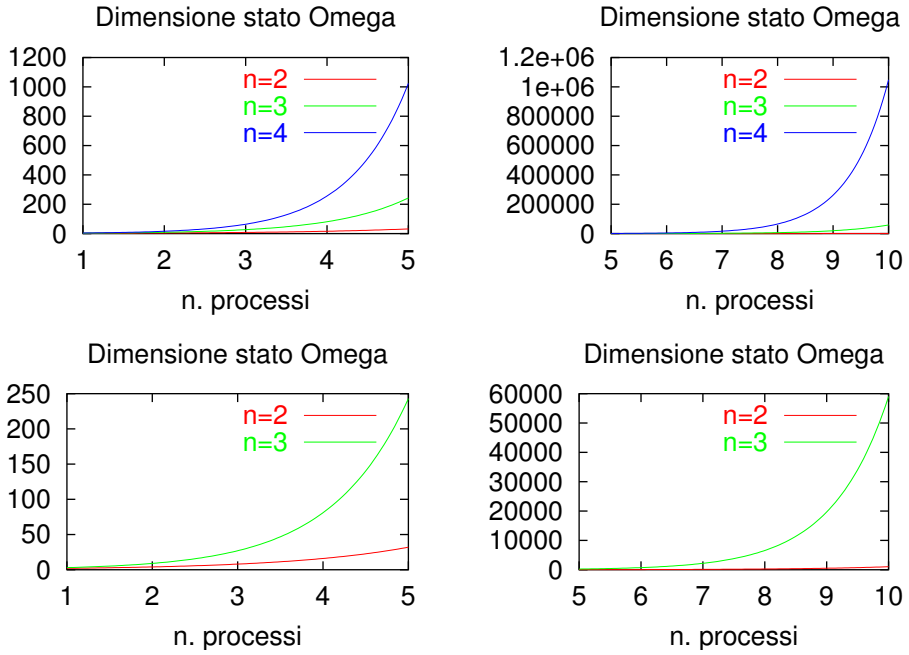

Figura 6.1: Dimensione spazio degli stati in funzione della discretizzazione del parametro MR e del numero di processi

L'utilizzazione del processore PU è stata infine discretizzata in quattro intervalli, la cui definizione e disposizione dipende dal singolo insieme di processi esaminato. Anche in questo caso si potrebbe optare per aumentare il numero delle suddivisioni qualora il numero dei processi controllati sia basso. Sebbene scelta possibile, per uniformità di analisi all'interno di questa tesi, il numero degli intervalli rimane fissato a quattro in tutti i casi esaminati.

Le considerazioni precedenti riguardanti la rappresentazione dello stato fornita da Ω possono essere trasferite direttamente al caso di Ψ, applicando lo stesso tipo di segmentazione utilizzato per PU anche per PU_h e PU_s. In questo caso, lo stato si presenta però molto più grande, di un fattore quattro: quale sia l'impatto dell'accresciuta dimensione e della diversa rappresentazione verrà esaminata nei vari esempi trattati. La dimensione dello spazio degli stati per Ω e Ψ è rappresentata in figura 6.2.

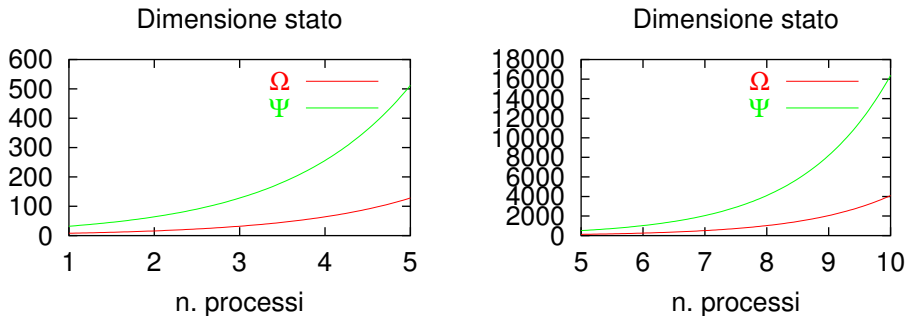

Figura 6.2: Dimensione finale dello spazio degli stati al variare del numero di processi

6.2 Apprendimento della politica in casi semplici

Gli esperimenti seguenti sono stati eseguiti utilizzando l'insieme di processi **Simple1** descritto in tabella 6.3.

Processo	**Periodo**		**WCET**	**BCET**
	Max	Min		
hard0	-	5 ms	1ms	0.8 ms
soft0	18 ms	12 ms	3 ms	2 ms
soft1	24 ms	14 ms	5 ms	3.5 ms

Carico massimo	80.71%
Carico minimo	49.60%
Carico medio	69.19%
Lyu Layland B.	77.98%

Tabella 6.3: Caratteristiche processi Simple1

L'insieme dei processi Simple1 è schedulabile sebbene l'utilizzazione complessiva calcolata sulla base del periodo minimo di ciasun processo sia superiore al limite di Liu Layland(77.98 per 3 processi).
Il primo problema sottoposto al modulo di adattamento è stato quello di controllare l'evoluzione di Simple1 in condizioni di non sovraccarico del sistema per un tempo t pari a 5000s. L'intervallo T di attivazione del controllo è stato impostato a $60ms$ mentre i parametri legati all'apprendimento utilizzati inizialmente per l'esperimento sono contenuti in tabella 6.4. L'andamento dell'utilizzazione del processore da parte dei processi è invece analizzato in figura 6.3.

r_{miss}	r_{hit}	r_{asp}	r_{rm}	r_{per}
-1	1	1	1	10
α	τ_α	ϵ	τ_ϵ	
0.9	1.0e3	0.1	1.0e8	

Tabella 6.4: Parametri iniziali utilizzati nel primo esperimento.

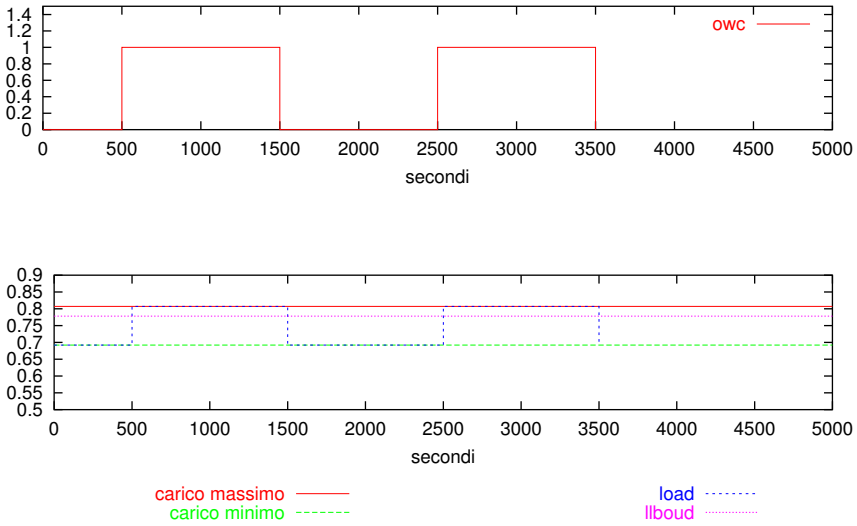

Figura 6.3: Andamento del carico per il problema di schedulazione sottoposto al modulo di controllo.

Il modulo di controllo percepisce lo stato del sistema utilizzando la rapprensentazione Ω che si compone in questo caso di 16 stati osservabili. È stata utilizzata per α una formulazione locale $\alpha(stato, azione)$ come indicato nella sezione **??**. La discretizzazione dello stato è fatta suddividendo PU in quattro fasce: $[0 - 0.65) \cup [0.65 - 0.75) \cup [0.75 - 0.85) \cup (0.85 - 1]$. In questo caso la scelta della soglia per MR non è importante in quanto il caso esaminato è esente da deadline miss. La variazione percentuale del periodo operata del controllore è contenuta(come specificato nel capitolo 4) nell'intervallo $[2 - 3]\%$.
Il comportamento del sistema è rappresentato in figura 6.4.

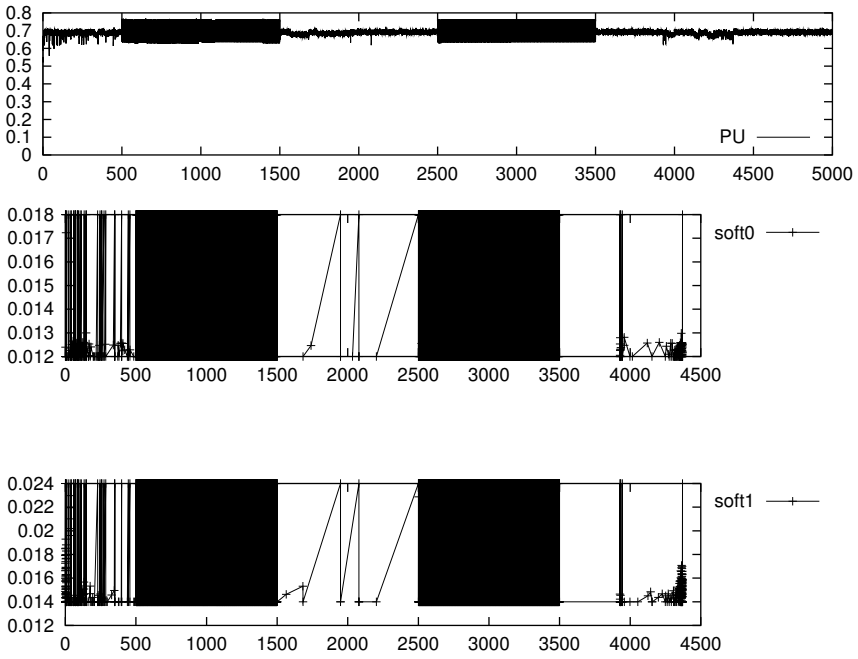

Figura 6.4: Esempio di apprendimento di una politica di governo errata per Simple1 in assenza di sovraccarico.

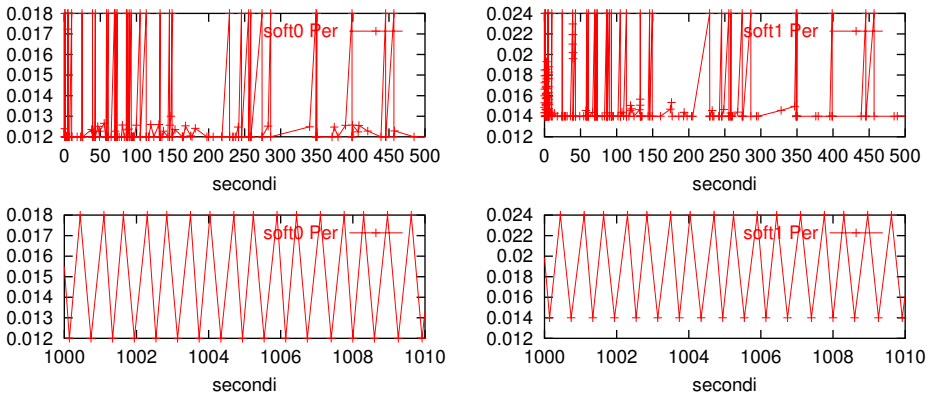

Figura 6.5: Dettagli relativi all'oscillazione del periodo dei processi in assenza di sovraccarico e parametri definiti in tab **??**.

Con questa particolare configurazione dei parametri, la soluzione trovata non è per nulla soddisfacente. Il controllo apprende velocemente che nell'intervallo $0.65 < PU < 0.75$ la soluzione migliore è quella di non modificare il periodo dei

processi (figura 6.5). Gli eventuali cambiamenti di periodo, sono principalmente dovuti alla esplorazione che, in questa fase, è largamente presente. Tuttavia quando all'istante $t = 500s$ si presenta lo scenario peggiore (tutti i processi vengono eseguiti con un tempo di esecuzione pari al loro WCET), il periodo dei processi comincia ad oscillare tra il minimo e il massimo. Quando all'istante $t = 1500$ i processi ritornano ad essere elaborati con tempi di esecuzione medi, il controllo ricomincia ad operare nel modo corretto. Al ripresentarsi del caso peggiore a $t = 2500s$, il comportamento oscillatorio si ripresenta.
L'errato controllo anche in presenza di un caso così semplice può essere imputato ad una funzione di rinforzo errata. Una funzione alternativa in cui si elimina il rinforzo positivo ad opera del singolo hit (r_{hit}) e lascia inalterato il resto, produce il risultato in figura 6.6, ugualmente insoddisfacente.

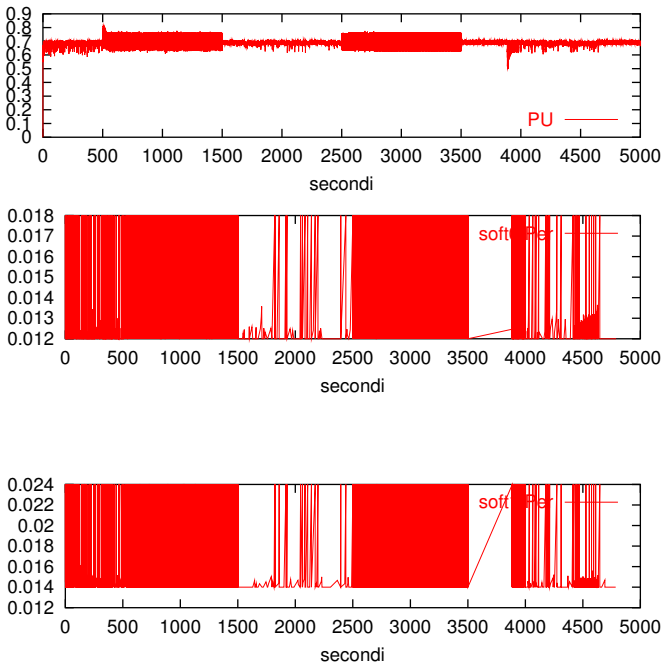

Figura 6.6: Apprendimento della politica di governo errata per Simple1 nel caso $r_{hit} = 0$.

È stato indagato quindi l'effetto della riduzione del rinforzo negativo riguardante le azioni, portando r_{asp} e r_{rm} dal valore 1 al valore 0.1. In questo modo, il contributo negativo per la singola azione scende da $1+4 = 5$ a 0.5, mentre nel caso di una

azione globale (ALLUP oppure ALLDOWN) si riduce il rinforzo da $3+4 = 7$ a 0.7. Il sistema questa volta si comporta decisamente meglio (figura 6.7 e figura 6.8).

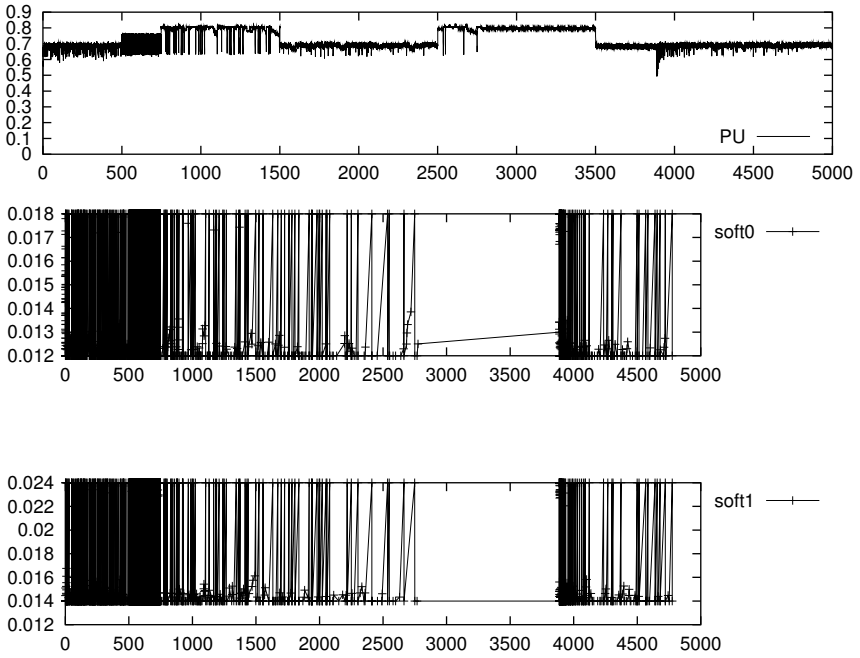

Figura 6.7: Apprendimento della politica di adattamento dopo la riduzione di r_{asp} e r_{rm} nel caso Simple1.

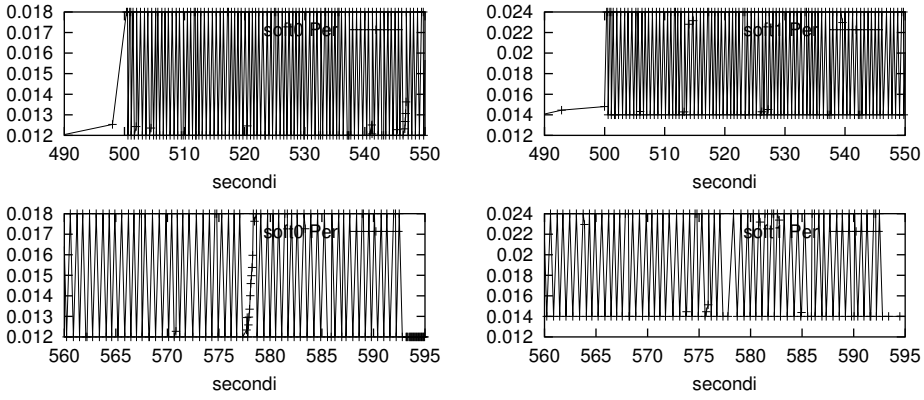

Figura 6.8: Oscillazione iniziale del periodo dei processi durante l'apprendimento della politica dopo la riduzione di r_{asp} e r_{rm}.

In fase di apprendimento permane ancora una certa oscillazione in presenza del cambiamento di stato a $t = 500$, che però non si verifica più a $t = 2500$, segno che il sistema ha appreso come comportarsi. Sfruttando però esplicitamente la politica appresa (quindi in assenza di esplorazione) il comportamento del sistema a seguito dell'apprendimento è tuttaltro che perfetto (figura 6.9).

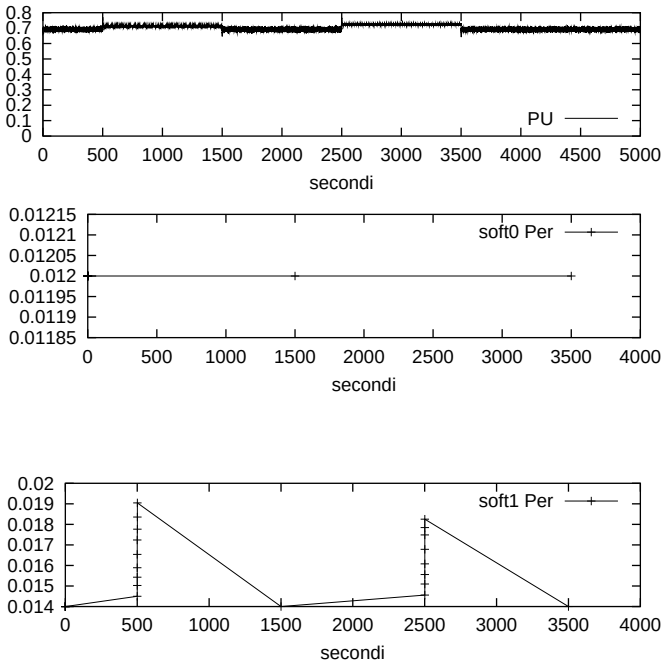

Figura 6.9: Sfruttamento della politica appresa mediante riduzione di r_{asp} e r_{rm}. Il comportamento non è ottimale.

Al fine di ottenere mediante l'apprendimento una politica ottimale non è risultato abbastanza valutare altre combinazioni di parametri per il rinforzo. Occorre quindi prendere in esame i parametri α, τ_α, ϵ, τ_ϵ. L'impatto dei valori assegnati per $\tau_\star$ sull'andamento di α e ϵ è analizzato in figura 6.10.

Mentre per ϵ il decadimento è lento, permettendo l'esplorazione anche dopo un elevato numero di cicli, il tasso di apprendimento α decresce velocemente e già dopo sole mille iterazioni il suo valore è sufficientemente basso da consentire un apprendimento trascurabile per quegli stati frequentemente visitati. Essendo il periodo del controllo pari a 60ms, in 5000s il numero di cicli ammonta a circa 85000, pertanto, è necessario aumentare τ_α al fine di consentire apprendimento significa-

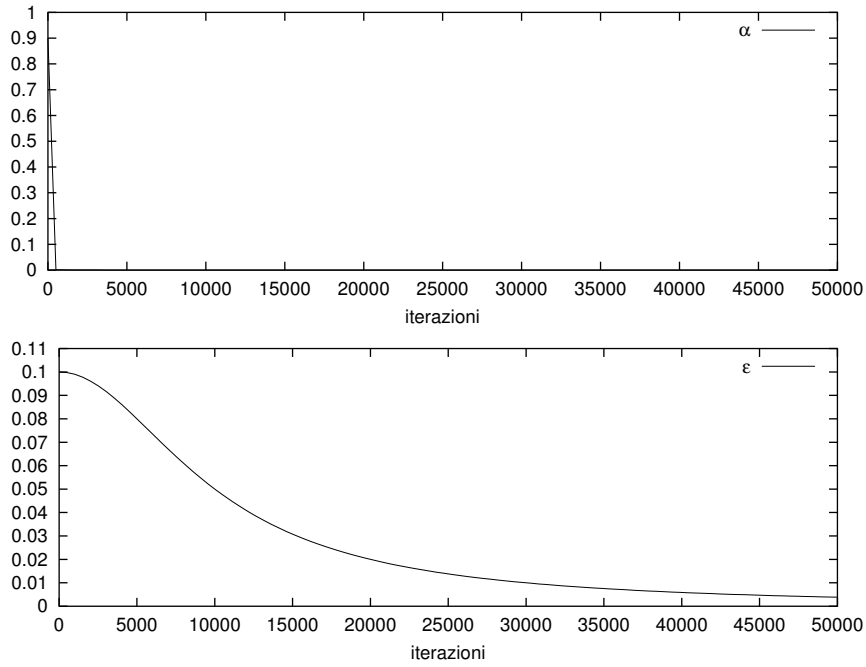

Figura 6.10: Esplorazione e tasso di apprendimento al variare del numero di iterazioni del controllo

tivo anche in fasi più avanzate dell'esperienza dell'agente. Operando questa scelta e modificando i rinforzi precedentemente impostati, si ottengono dei miglioramenti evidenti, primo fra tutti la scomparsa dell'oscillazione prolungata del periodo (figura 6.11).

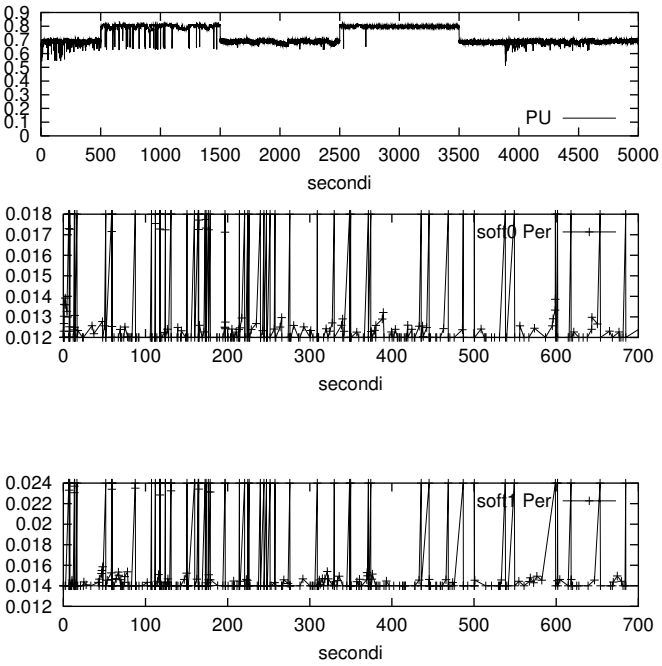

Figura 6.11: Apprendimento politica di governo per Simple1 dopo la modifica dei parametri relativi al tasso di apprendimento.

Sfruttando la politica appresa dal controllo, da un punto di vista applicativo l'apprendimento fornisce una risultato questa volta ottimale. (figura 6.12).

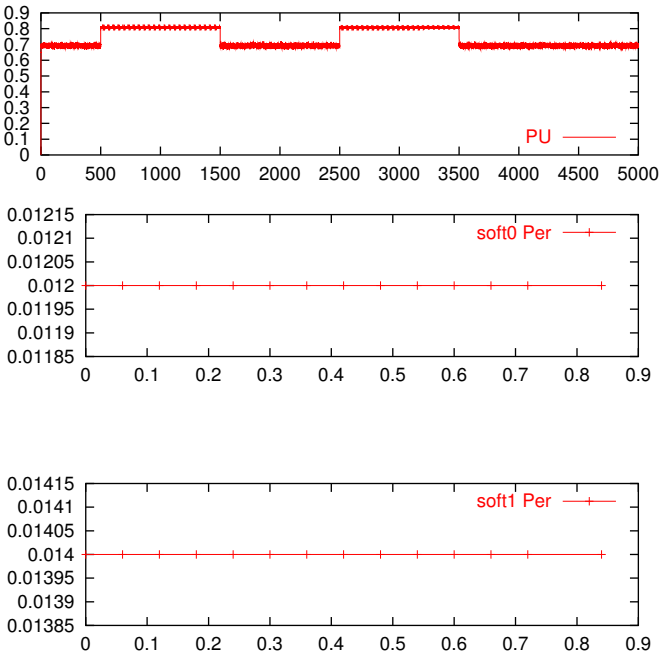

Figura 6.12: Sfruttamento della politica appresa. Questa volta la politica è ottimale.

Questo semplice esempio mette in evidenza soltanto alcuni dei diversi aspetti critici del modulo di controllo. In effetti, anche se non trattato direttamente in questo esempio, anche la scelta della discretizzazione ha una notevole influenza sul risultato finale dell'apprendimento. Nel caso esaminato, le soglie utilizzate sono quelle che hanno dato i risultati migliori e sono state ricavate sulla base di semplici considerazioni sul carico computazionale massimo, minimo e medio dell'insieme dei processi. Per poter ottenenere dal processo di apprendimento una politica di gestione dei processi che sia il quanto più vicino possibile alle esigenze, è necessaria una discreta quantità di lavoro da parte del progettista, volta in particolar modo ai seguenti aspetti:

- Regolazione dei parametri legati all'apprendimento.
- Scelta del tipo di discretizzazione dello spazio degli stait da utilizzare.

6.2.1 Schedulazione adattativa di Simple1

Il caso esaminato in precedenza fornisce qualche ragguaglio riguardo alla necessità di uno sforzo in direzione della regolazione dei vari parametri al fine di ottenere comportamenti interessanti da parte dell'agente preposto all'apprendimento. Un secondo caso esaminato è stato l'adattamento dello scheduling di Simple1 in presenza di condizioni di sovraccarico. L'esempio non verrà trattato evidenziando tutti i tentativi effettuati ma verranno commentati soltanto alcuni dei risultati più significativi.
Il primo esempio esaminato presenta alcune condizioni di sovraccarico che interessano però solo la parte critica di Simple1. L'andamento del carico del sistema è rappresentato in figura **??**.

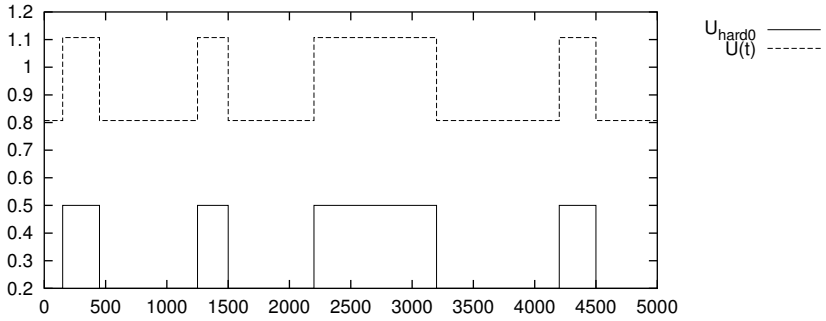

Figura 6.13: Problema di schedulazione Simple1 in presenza di sovraccarico della parte critica.

I parametri in tabella 6.5 sono stati ricavati partendo dalle impostazioni relative al caso di politica *nulla* esaminato in precedenza.

r_{miss}	r_{hit}	r_{asp}	r_{rm}	r_{per}
30	0.1	0.1	1	10
α	τ_α	ϵ	τ_ϵ	
0.01	1.0e6	0.05	1.0e6	

Tabella 6.5: Parametri apprendimento per il caso di sovraccarico

Il risultato è stato incoraggiante (figura 6.14 e 6.15), infatti, il modulo di controllo apprende abbastanza facilmente una politica di adattamento per i processi

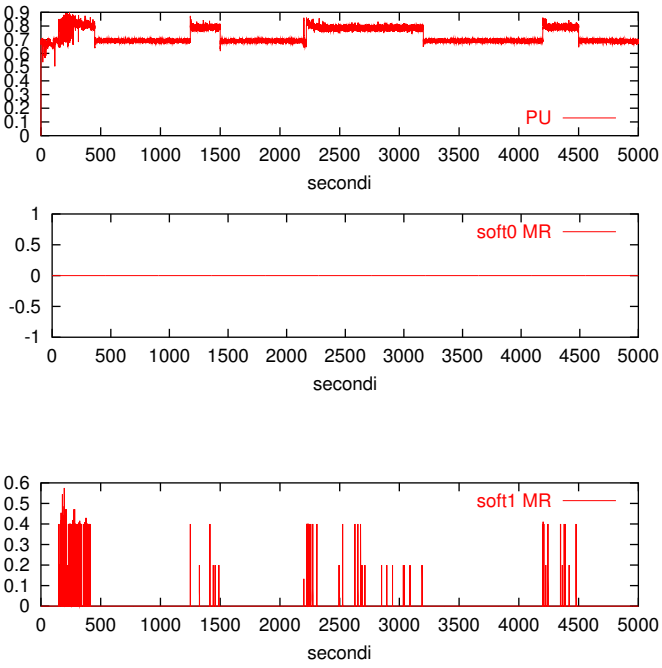

Figura 6.14: Utilizzazione del processore e miss ratio durante l'apprendimento in condizioni di sovraccarico della parte critica per Simple1

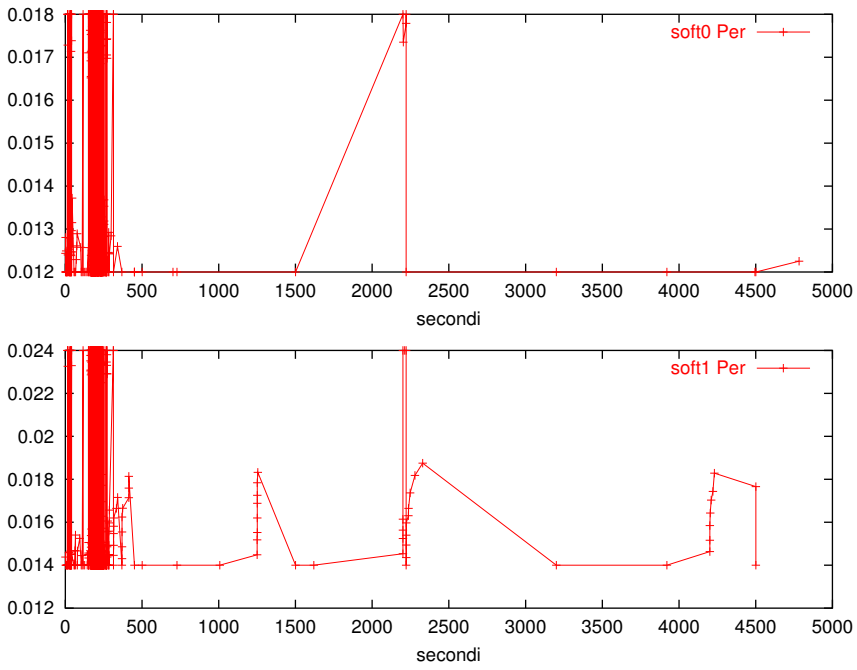

Figura 6.15: Periodo dei processi durante l'apprendimento in condizioni di sovraccarico della parte critica per Simple1. Ottimizzazione del periodo di soft1

che si può definire molto soddisfacente. A parte la fase iniziale di forte esplorazione, la politica appresa, che comincia a manifestarsi già attorno a $t = 3000$, lentamente adatta il periodo del processo soft1 in modo da minimizzarne le miss e al contempo mantenere per quanto possibile il periodo in condizioni prossime a quelle ottimali. Interessante osservare come l'apprendimento sia in grado di ottenere politiche diametralmente opposte per lo stesso caso, applicando opportune variazioni dei parametri(figura 6.16 e 6.17).

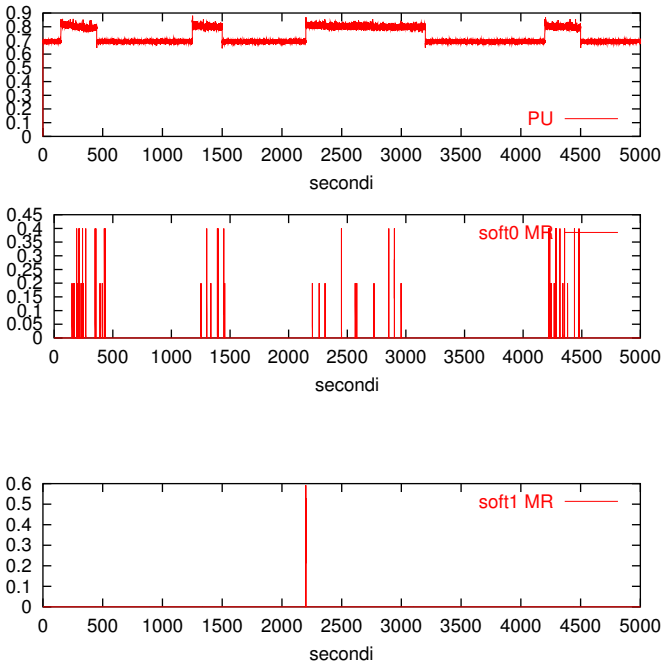

Figura 6.16: Utilizzazione del processore e miss ratio durante l'apprendimento in condizioni di sovraccarico della parte critica per Simple1 variando i parametri

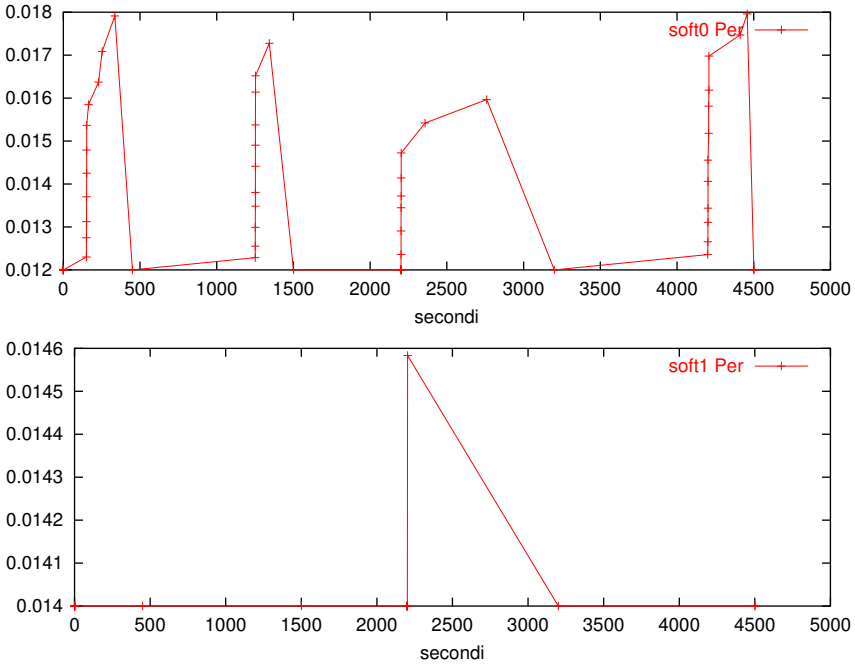

Figura 6.17: Periodo dei processi durante l'apprendimento in condizioni di sovraccarico della parte critica per Simple1 dopo il cambio di parametri. Ottimizzazione del periodo di soft0

Sullo stesso esempio Simple1 è stato inoltre sperimentato l'apprendimento dell'adattamento nel caso in cui un sovraccarico di entità pari al 40% interessi tutti i processi(figura 6.18).

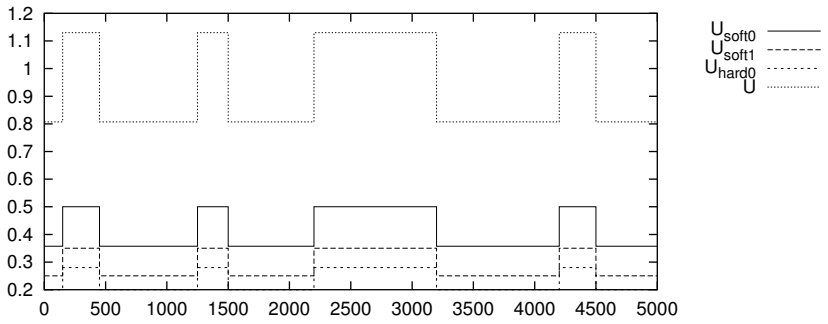

Figura 6.18: Problema di schedulazione Simple1 in presenza di sovraccarico della parte critica.

La soluzione rappresentata in figura 6.19 e figura 6.20 ha richiesto un numero molto elevato di tentativi prima di trovare la configurazione ottima dei parametri. L'adattamento trovato però è molto interessante, infatti nel caso esaminato esistono diverse soluzioni che garantiscono la schedulabilità dei processi con queste condizioni di carico. Una in particolare consiste nell'incrementare il periodo di soft1 fino al valore $P^* = 22ms$: con questa nuova configurazione dei periodi Simple1 è di nuovo schedulabile. Sebbene il modulo di controllo continui a voler diminuire il periodo di soft0 anche in condizioni nominali, durante il sovraccarico il suo periodo rimane fissato al minimo mentre quello di soft1 aumenta lentamente fino al valore ottimale (nell'intorno dei 22ms).

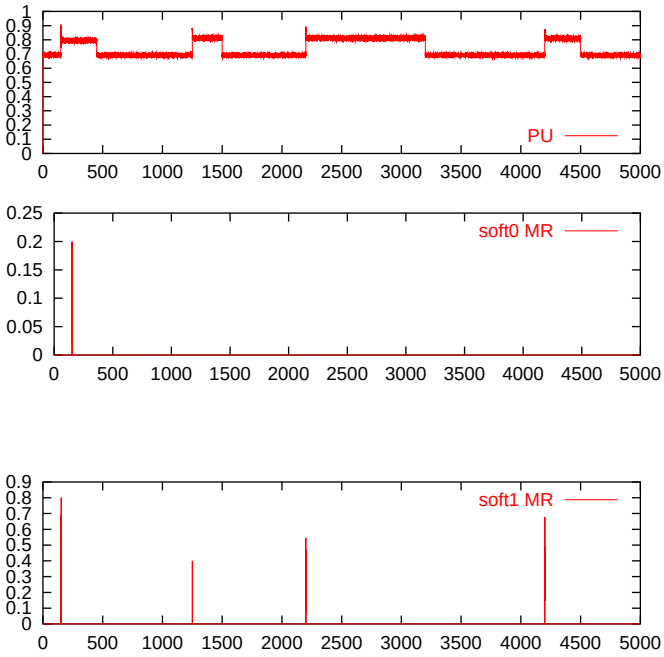

Figura 6.19: Utilizzazione del processore e miss ratio durante l'apprendimento in condizioni di sovraccarico per Simple1 (caso A)

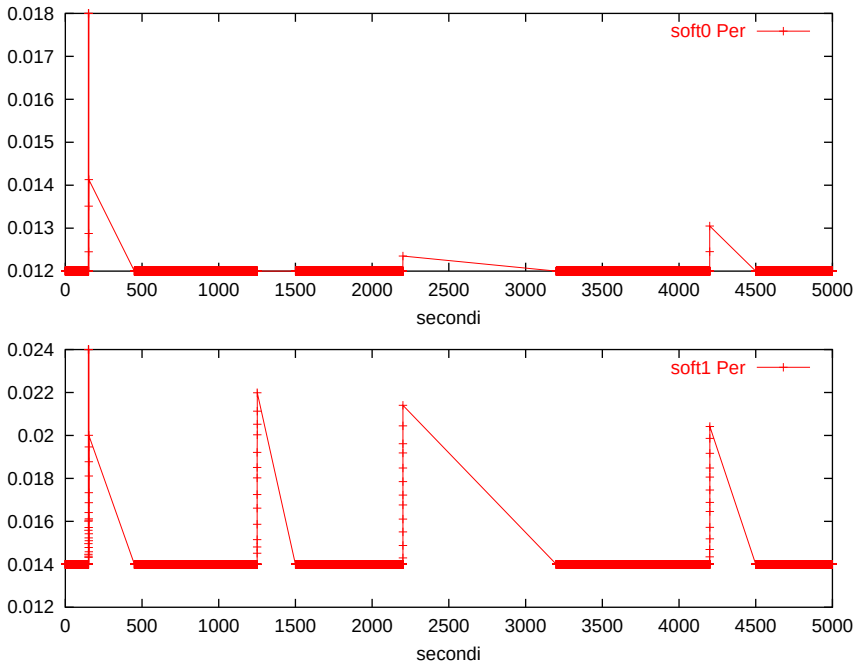

Figura 6.20: Periodo dei processi durante l'apprendimento in condizioni di sovraccarico per Simple1 (caso A)

Sempre su questo caso è stato possibile ottenere un altra soluzione interessante, questa volta ottenuta variando il periodo di soft0 in condizioni di sovraccarico fino al suo valore massimo e incrementando leggermente il periodo di soft1. Il numero di miss viene così ridotto a soli sporadici casi(figura 6.21 e 6.22).

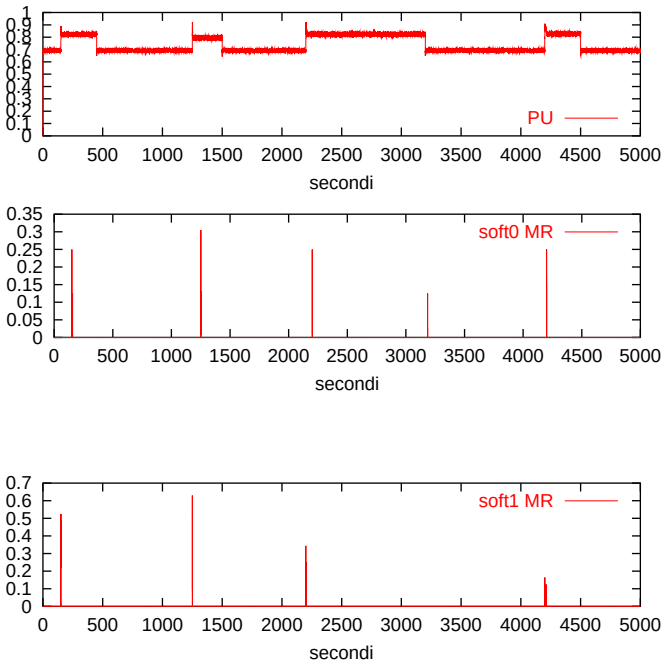

Figura 6.21: Utilizzazione del processore e miss ratio durante l'apprendimento in condizioni di sovraccarico per Simple1 (caso B)

6.2.2 Osservazioni sui risultati ottenuti

Gli esperimenti condotti con il simulatore hanno posto in evidenza come le prestazioni del modulo di controllo sia fortemente influenzate dalla particolare configurazione dei parametri e dal valore del rinforzo. Sebbene questo tipo di dipendenza fosse prevista, il sistema ha richiesto un certo impegno nel tarare i vari attributi, tuttavia , per i semplici esempi trattati, è stato comunque possibile ottenere soluzioni interessanti, alcune ottimali, altre prossime all'ottimalità. Non risulta ben chiaro però, sulla base delle conoscenze finora acquisite, stabilire quale sia il legame tra la combinazione dei vari rinforzi e la politica di adattamento appresa.

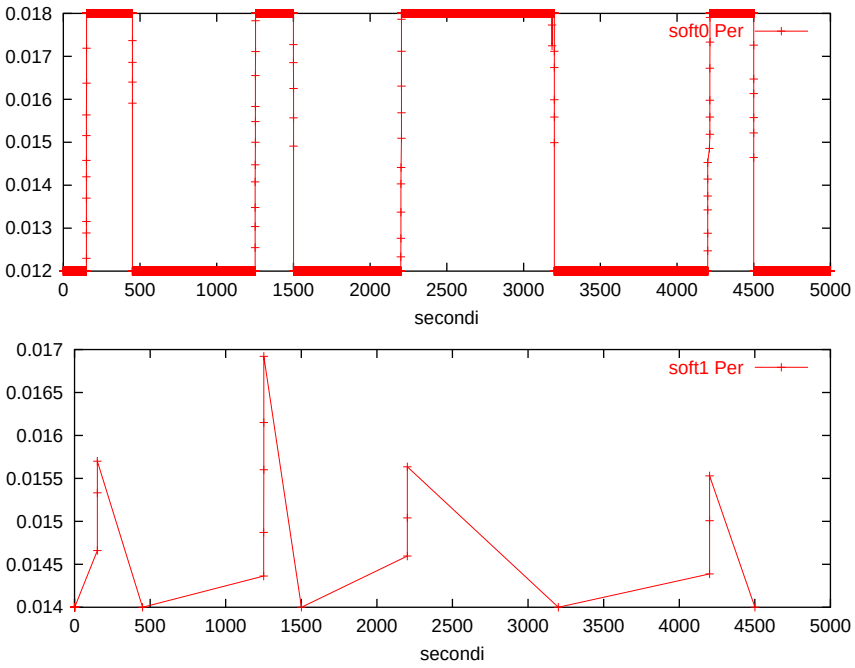

Figura 6.22: Periodo dei processi durante l'apprendimento in condizioni di sovraccarico per Simple1 (caso B)

L'efficienza dell'apprendimento dipende inoltre dalla particolare scelta di rappresentazione dello spazio degli stati. Le scelta di utilizzare Ω puó non essere sempre adeguata. La riduzione dello spazio degli stati operata nel capitolo 4 ha permesso di ridurre la dimensione di questo spazio in termini di numero di attributi, tuttavia nasconde molti dettagli riguardo lo stato del singolo processo, in particolare il valore del periodo dei vari processi. Utilizzando Ω infatti, l'agente non è in grado di definire quando un processo o l'insieme controllato è adattato. Questo può essere un limite per l'apprendimento, così come un limite forte è la necessità di discretizzare lo stato in un numero ristretto di intervalli. Poter utilizzare rappresentazioni basate ad esempio su alberi binari o reti neurali può risolvere questo problema , presentando al contempo tempi di apprendimento molto più lunghi di quelli esposti in questi semplici casi. È bene osservare comunque che l'apprendimento nei casi trattati è stato ottenuto in un tempo pari 5000s che si traduce in 85000 iterazioni da parte dell'agente. Il processo di apprendimento è generalmento molto lento anche nel caso dell'apprendimento con rinforzo, pertanto il numero delle iterazioni lasciate a disposizione dell'agente è stato in un certo senso vincolante. Inoltre, poichè l'evoluzione del sistema (in termini di tempi di esecuzione dei processi) non dipende soltanto dalla propria configurazione interna ma anche da eventi esterni, spesso non caratterizzati e sporadici quali i sovraccarichi, le possibilità di apprendere in queste situazioni è vincolata al verificarsi del dato evento. Forse, in esperimenti di natura differente, in cui il sovraccarico è di maggior durata, l'agente ha modo di sperimentare maggiormente, anche se, su una base di tempi di 60ms, già sovraccarichi di 1000s danno all'agente la possibilità di sperimentare diverse strategie per un numero elevato di iterazioni. Il limite sembra quindi più volto verso la rappresentazione dello stato che non all'effettivo tempo messo a disposizione all'agente per apprendere la politica di adattamento.

Anche la scelta di utilizzare α in formulazione locale puó aver influenzato notevolmente l'apprendimento. Utilizzando α come una singola variabile, è possibile controllarne il decadimento in modo più controllato.

Capitolo 7

Conclusioni

Questa tesi ha avuto come obiettivo l'analisi e la valutazione di algoritmi di scheduling di tipo adattativo con caratteristiche di apprendimento.
Il problema è stato trattato in modo estensivo nel capitolo 4 introducendo una formulazione del problema basata su processi di tipo semi-Markov. È stato inoltre sottolineato come questa ipotesi sia frutto sopprattutto di considerazioni di tipo intuitivo e di come la presenza di alcuni approcci più formali ([YT91] e [Yih92]) valorizzi l'ipotesi fatta. È stato discusso nel dettaglio il problema della dimensione dello spazio degli stati ed è stata avanzata una sua possibile riduzione in termini di attributi costituenti lo stato stesso.

Al fine di agevolare lo sviluppo e il collaudo del sistema di controllo, nell'ambito di questa tesi è stato sviluppato un programma di simulazione di algoritmi di scheduling basato sulla libreria non commerciale C++SIM. Il simulatore è molto flessibile e permette di scegliere tra diverse politiche di scheduling. Il framework realizzato simula prevalentemente politiche di scheduling di tipo preemptive ma può essere esteso facilmente a nuove politiche di schedulazione e alla modellazione di carichi di lavoro con processi non periodici.

È stata infine condotta un'analisi sulle prestazioni del modulo di controllo dell'adattamento utilizzando come dati di ingresso insiemi di processi dalle diverse caratteristiche. I risultati ottenuti dalle simulazioni sono stati promettenti ed hanno consentito di osservare alcune interessanti politiche di adattamento autonomamente apprese dal sistema. Si é osservato, tuttavia, che l'ottenimento di soluzioni

efficaci dipende fortemente dai diversi parametri utilizzati per orientare il processo di apprendimento. Risulta infatti necessaria una attenta fase di configurazione di tali parametri per assicurare all'algoritmo la convergenza verso una politica adattativa efficiente.

Il contributo dato da questa tesi non è quindi quello di presentare uno strumento completo per la progettazione di sistemi soft real-time adattivi, ma, piuttosto, quello di fornire una prima analisi sulla possibilità di integrare in modo efficace meccanismi di apprendimento all'interno del contesto degli algoritmi di scheduling adattativi. Ulteriore lavoro in termini di studio del problema è necessario per pervenire alla sintesi di una politica ottimale ed alla risoluzione delle problematiche evidenziate.

Appendice A

Generazione del carico

Avere a disposizioni opportuni set di prova su cui effettuare la misura (o la verifica) delle prestazioni è di vitale importanza. L'ideale è rappresentato da un insieme di dati di input che riesca a coprire un ampio spettro di tutte le possibili combinazioni. Nel nostro caso, identificare diversi insiemi di processi rappresentativi di una certa classe di problemi non è cosa semplice; da un lato perchè è difficile definire cosa si intende, in questo contesto, per classe di problemi; dall'altro perchè le possibili combinazioni di insieme di task sono pressochè infinite. Diventa importante,allora,stabilire alcune metriche che consentano di suddividere l'insieme delle configurazioni in classi di equivalenza; in una fase successiva si cercherà di realizzare uno strumento che generi in modo automatico insiemi di task con le desiderate caratteristiche.

A.1 Suddivisione in base all'utilizzo complessivo

Definizione 1: Per Carico Normale si intende un insieme di processi T_{SL} tale che il fattore di utilizzo complessivo sia pari al 90% del carico limite imposto dal fattore di utilizzo limite dell'algoritmo di schedulazione
$T_{SL} = \{T_i | \sum_{i=1}^{N} \frac{W_i}{P_i} = 0.9 \cdot N \cdot (2^{\frac{1}{N}} - 1)\}$
Definizione 2: Per Carico Leggero si intende un insieme di processi T_{LL} tale che il fattore di utilizzo complessivo sia inferiore al 90% del valore di utilizzazione limite
$T_{LL} = \{T_i | \sum_{i=1}^{N} \frac{W_i}{P_i} < 0.9 \cdot N \cdot (2^{\frac{1}{N}} - 1)\}$

Definizione 3: Per Carico Saturato, si intende un insieme di processi T_{QL} tale che il fattore sia maggiore compreso tra il 90% del limite sopracitato e il limite stesso.
$T_{QL} = \{T_i | 0.9 \cdot UB < \sum_{i=1}^{N} \frac{W_i}{P_i} < UB, UB = \cdot N \cdot (2^{\frac{1}{N}} - 1)\}$
Definizione 4: Per Carico Armonico Unimodale, si intende un insieme di processi T_{AUML} tale che le frequenze di lavoro siano tutti multipli interi di una frequenza F_0.
Definizione 5: Per Carico Armonico Multimodale si intende un insieme di processi T_{AMML} tale per cui esistono due o più frequenze prime F_i, rispetto alle quali l'insieme può essere suddiviso in due o più sottoisiemi Armonici Unimodali.
definizione 6: Qualora il fattore di utilizzo complessivo sia superiore all'utilizzazione limite, l'insieme dei processi viene detto **Carico A Rischio**.

Notiamo subito che alla definizione di carico a rischio corrisponde all'impossibilità di una schedulazione efficace solo per l'algoritmo EDF, in cui il limite equivale a 1. Nel caso Rate Monotonic, utilizzando il limite di Liu e Layland stiamo solo controllando una condizione sufficiente ma non necessaria. Va inoltre osservato che le condizioni di armonicità unimodale e multimodale possono essere associate alle definizioni precedenti, formando quindi ad esempio un Carico Armonico Unimodale Standard. In questo caso però occorre modificare il limite imposto considerando come numero di task il numero di modi fondamentali presenti nell'insieme.

La motivazione delle suddette definizioni va ricercata nel tentativo di dare un prima suddivisione alla pletora di configurazioni disponibili.

A.2 Organizzazione dei task all'interno del set

Oltre alla suddivisione in classi rispetto al tipo di carico, è possibile distinguere l'aggregazione di un certo numero di task anche a seconda della classe di appartenenza dei task, alla percentuale di ciascuna classe all'interno del set e e in base a quanto questo influiscono sul carico complessivo.

- Carico Hard : tutti i task sono di tipo hard-realtime; caso non di interesse

- Carico Soft : tutti i task sono di tipo soft realtime
- Carico Misto : sia hard che soft con un certa proporzione.

Nel caso Carico Soft possiamo distinguere due diverse tipologie:

- Carico ad intervalli sovrapposti : l'intervallo $[T_{min}, T_{max}]$ di ciascun task può essere in parte sovrapposto con quello di uno o più task appartenti al set.
- Carico ad intervalli non sovrapposti : tutti gli intervalli $[T_{min}, T_{max}]$ dei vari task sono separati.

Questa suddivisione permette di isolare due grandi sottoinsiemi soft realtime: gli insiemi a *priorità invariante*, associato con il caso intervalli separati, e gli *insiemi a priorità variabile*. Questa distinzione credo sia importante in quanto non sempre tutti i task possono essere considerati equivalenti in quanto ci sono diversi problemi in cui la priorità assegnata deve essere rispettata ma anche altri in cui, ogni task è equivalente all'altro.

Nel caso a Carico Misto, una prima considerazione riguarda la proporzione tra le due classi, che suddivide lo spazio delle soluzioni in tre regioni distinte. Una seconda osservazione va invece fatta sulla distribuzione delle frequenze operative: fermo restando l'eventuale sovrapposizione degli intervalli, sottolineata nel paragrafo precedente, occorre precisare se le frequenze di lavoro dei subset soft realtime possa o meno intersecare quello del subset hard realtime. Il rispetto della deadline deve sempre essere garantito per il sottoinsieme hard, quindi mi sembra ovvio che le priorità assegnate ai task del gruppo debbano essere sempre le più alte. Questo elimina subito il dubbio di una eventuale intersezione.

Un'ulteriore specifica riguarda la percentuale con cui uno dei due gruppi all'interno dell'insieme misto influisce sul carico complessivo. Non sempre infatti il carico può considerarsi equamente suddiviso; alcune volte il computo hard realtime può essere preponderante altre volte,invece,solo una piccola porzione del tempo di CPU necessita garanzie stringenti.

A.3 Specifiche del generatore

1. Carico complessivo del set : questo permette di scegliere liberamente Carico Standard,Limitato e Saturato
2. Carico minimo del singolo task: questo è utile per evitare di generare con un fattore di utilizzo troppo basso. Se viene specificato pari a 0, il parametro viene ignorato.
3. Proporzione libera tra task hard e soft.
4. Possibilità di indicare quanta parte del carico associare a ciascuno dei due sottoinsiemi
5. Permettere o meno l'intersezione tra gli intervalli di periodicità per i task soft realtime

Di seguito sono illustrate le relazioni che legano i vari parametri dei task. Le relazioni sono del tutto generali; la scelta delle variabili fisse e di quelle vincolate verrà illustrata più avanti.

Hard task

$$\frac{bcet + wcet}{2} = met \quad \frac{wcet}{bcet} = \beta_i$$
$$\frac{wcet}{T} = U_i$$

Soft task

$$\frac{bcet + wcet}{2} = met \quad \frac{wcet}{bcet} = \beta_i$$
$$\frac{T_{max}}{T_{min}} = \gamma_i \star$$
$$\frac{met}{T_{tmin}} = U_i \star$$

β_i, γ_i indicano rispettivamente il rapporto tra il wcet e bcet e quello tra il periodo massimo e minimo di ciascun task.
È stato scelto quindi di fissare un intervallo $[MET_{min}, MET_{max}]$ all'interno del quale scegliere il tempo medio di esecuzione *met* e di fissare i rapporti γ e β.
Le equazioni contrassegnate con $\star$ necessitano di qualche spiegazione: la scelta di fissare γ deriva dall'ipotesi che, generalmente, il rapporto tra periodo minimo e massimo di un task debba essere una scelta progettuale ben definita; è infatti abbastanza logico che la frequenza di un certo task sia fissata a priori in dipendenza sia dal met (oppure dal wcet a cui è legato tramite β), che del degrado massimo di prestazioni che si è disposti a tollerare.
La particolare variante adottata per il calcolo del fattore di utilizzo può essere giustificata pensando che, in un ottica soft-realtime, l'utilizzo della cpu è una risorsa preziosa e quindi va massimizzato. Continuare ad utilizzare il wcet come misura del tempo di esecuzione risulta in un grosso spreco di risorse in quanto il task mediamente viene eseguito in tempi inferiori. Specificando invece il met non garantiamo l'esecuzione nel caso peggiore (o almeno non è garantita sempre), tuttavia il comportamento del sistema si avvicina al concetto di best-effort.

Fissati quindi questi parametri, bcet,wcet,T_{min} e T_{max} sono elencati nella tabella A.1.

HARD	SOFT
$bcet = \frac{2 \cdot met}{1+\beta}$	$bcet = \frac{2 \cdot met}{1+\beta}$
$wcet = \frac{2 \cdot \beta \cdot met}{1+\beta}$	$wcet = \frac{2 \cdot \beta \cdot met}{1+\beta}$
$T = \frac{wcet}{U_i}$	$T = T_{min}$
	$T_{min} = \frac{2 \cdot met}{U_i}$
	$T_{max} = \gamma \cdot T_{min}$

Tabella A.1: Funzioni per la creazione dei task

A.4 Processo di creazione dei task

Il generatore viene configurato tramite file, specificato come descritto nella sezione seguente. Viene quindi utilizzato un apposito parser, realizzato utilizzando lex e yacc, tramite il quale vengono ricavati i vari campi. A questo punto il compito di generare i task viene demandato alla classe JobGenerator.
La creazione avviene in due passi: il primo passo è quello di calcolare il fattore di utilizzo di ciascun task, suddividendo il carico totale tra tutti i task. Questa suddivisione non avviene in modo equo, quindi, non tutti i task hanno lo stesso carico computazionale.

Prima di eseguire questo calcolo, vengono effettuati alcuni test sui parametri di ingresso al fine di stabilire se sia possibile oppure no generare il carico richiesto. In particolare:

1. se specificato $hslw < 1$: L'HardSoftLoadWeight ,la percentuale di carico associata ai task hard, deve essere almeno minore dell'unità.

2. $minload \cdot numtask < 1$: anche il carico minimo deve essere sufficientemente piccolo.

3. se il numero di task hard è maggiore di 0, è necessario verificare l'hslw. Se è stato specificato, vengono controllate le disequazioni

$$maxload \cdot hslw < llbound$$

   e

$$\frac{maxload \cdot hslw}{wbr} > minload \cdot hardtask$$

   per gararantire la schedulabilità del set e il vincolo sul carico minimo di ciascun task rispettivemente. Qualora hslw non venga specificato, il generatore provvede ad assegnarne uno preso a caso all'interno di un opportuno intervallo.

4. se il numero dei task soft è maggiore di 0, viene verificato che la parte di carico lasciata al subset dalla quota (1-hslw) sia sufficiente a garantire il

vincolo sul carico minimo. La disequazione

$$\frac{maxload \cdot (1 - hslw)}{mmr} < minload \cdot softtask$$

serve a questo scopo.

Verificate le condizioni necessarie, ma non sufficienti, vengono calcolati $softtask+$ $hardtask$ numeri casuali compresi nell'intervallo $[0 \dots 1]$ e due somme parziali *hload,sload*, che identificano rispettivamente il carico non normalizzato dei due sottoinsiemi, hard e soft. A questo punto vengono testate le due condizioni sufficienti

$$\sum_{i=0}^{hardtask} < \frac{min_{i \in hard}(U_i) \cdot hslw}{minload \cdot wbr}$$

$$\sum_{j=0}^{softtask} < \frac{min_{j \in soft}(U_j) \cdot (1 - hslw)}{minload \cdot mmr}$$

Una volta soddisfatte le due condizioni non rimane che normalizzare i due sottoinsiemi, dividendo ciascun membro per il carico complessivo del proprio subset e moltiplicandolo poi per il prodotto $maxload \cdot hslw$ oppure $maxload \cdot \ (1 - hslw)$ a seconda della classe di appartenenza:

$$H_i = \frac{H_i \cdot hslw \cdot maxload}{hload}$$

$$S_i = \frac{S_i \cdot (1 - hslw) \cdot maxload}{sload}$$

Abbiamo quindi stabilito tutti i fattori di utilizzo. La creazione dei task hard realtime procede senza particolari problemi, utilizzando le formule della tabella A.1. Più complessa la generazione dei task di tipo soft in quanto occorre prevedere la possibilità che gli intervalli di periodicità dei vari processi siano disgiunti.
In questo caso è necessario procedere per tentativi. Vengono dapprima calcolati il periodo minimo e massimo del task i-esimo, in particolare T_{min} deve essere

maggiore del $min_{i \in hard}(T_i)$; a questo punto il generatore controlla se l'intervallo precedentemente calcolato si interseca con quelli dei task già presenti. In caso affermativo vengono ricalcolati due nuovi intervalli e si ricomincia. Maggiori dettagli sono forniti nei file sorgenti. Trovata la coppia T_{min} , T_{max}, il calcolo procede spedito utilizzando le formule di tabella A.1.

L'output del generatore viene salvato su file, già pronto per essere utilizzato con il simulatore; vengono inoltre specificati i semi del generatore di numeri casuali, in modo che il task set possa essere riprodotto.

A.5 Sintassi del file di configurazione

Anche il generatore viene configurato utilizzando un opportuno file; anche questo file è suddiviso tre sezioni contraddistinte dalla seguente sintassi:
$config \rightarrow loadsec\ setsec\ constrainsec$
$loadsec \rightarrow' [' LOAD ']' ENDLINE\ loadpar\ sec\_sep$
$loadpar \rightarrow\ maxload\ minload|minloadmaxload$
$minload \rightarrow\ MINLOAD\ value\ ENDLINE$
$maxload \rightarrow\ MAXLOAD\ value\ ENDLINE$
$setsec \rightarrow\ '[' SET ']' ENDLINE\ setcfg\ sec\_sep$
$setcfg \rightarrow softtask|hardtask|softtask\ hardtask|hardtask\ softtask$
$softtask \rightarrow'' soft''\ INTEGER\ ENDLINE$
$hardtask \rightarrow'' hard''\ INTEGER\ ENDLINE$
$constrainsec \rightarrow' [' CONSTRAIN ']' ENDLINE\ parameters\ sec\_sep$
$parameters \rightarrow met\_c\ wbr\_c\ mmr\_c\ optionalparams|met\_c\ wbr\_c\ mmr\_c$
$optionalparams : hslw_coverlap|overlaphslw_c$
$met_c \rightarrow'' met''\ range\ ENDLINE$
$mmr_c \rightarrow'' mmr''\ value\ ENDLINE$
$wbr_c \rightarrow'' wbr''\ value\ ENDLINE$
$hslw_c \rightarrow'' hslw''\ value\ ENDLINE|$
$overlap \rightarrow\ ''overlap''\ ENDLINE|$
$range \rightarrow value\ ' -'\ value$

$value_type \rightarrow INTEGER|DOUBLE$
$value \rightarrow value\_type\ tag$
$tag \rightarrow''\ s''|''ms''|''us''|\%|$
$sec\_sep \rightarrow |sec\_sepENDLINE$
$comment \rightarrow''\ \#''STRING$

Di seguito riporto un esempio

```
[LOAD]
maxload 80%    #must be in range 0..1
minload 2.5%    #must be in range 0..maxload

[SET]
soft 5
#hard 2

[CONSTRAIN]
met 2us - 200us  #mandatory
wbr 2.3
mmr 2.0
hslw  20%
overlap   #if present,
          #the varius task operating area can overlap each other.
```

bibliografia

Bibliografia

[AA98] A.Atlas and A.Bestavros. Statistical rate monotonic scheduling. Technical Report 1998-010, Boston University, 1998.

[ABR+93] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284292, 1993.

[A.C03] A.Cervin. *Integrated Control and Real-Time Scheduling*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, April 2003.

[AL97] A.Mok and G. Liu. Early detection of timing constraint violation at runtime. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1997.

[AMS97] A.Y.Zomaya, M.Clements, and S.Olariu. Randomized reinforcement based scheduling in parallel processor systems. In *30th HawaiiInternational Conference on System Sciences*, volume 1, pages 556–565, USA, 1997. IEEE.

[Be99] G. Beccari and et.al. Rate modulation of soft real-time tasks in autonomous robot control systems. In *EuroMicro Conference on Real-Time Systems*, June 1999.

[BL94] J.A. Boyan and M.L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems*, volume 6, pages 671–678. Morgan Kaufmann Publishers, Inc., 1994.

[BLA98] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *IEEE Real-Time Systems Symposium*, pages 286–295, Dec 1998.

[BPB+00] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46:305–325, 2000.

[BSL89] B.Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1989.

[CBS00] M. Caccamo, G. Buttazzo, and L. Sha. Elastic feedback control. In *Proceedings of 12th IEEE Euromicro Conference on Real-Time Systems*, pages 121–128, June 2000.

[C.D82] C.D.Locke. *Best effort decision making for real-time scheduling*. PhD thesis, Computer Science Department of Carnagie Mellon University, Pittsburgh, Pennsylvania, 1982.

[CLTS99] J.A. Stankovic C. Lu, G. Tao, and S. H. Son. Design and evaluation of a feedback control EDF scheduling algorithm. In *IEEE Real-Time Systems Symposium*, pages 56–67, 1999.

[CRM96] C.Lee, R. Rajkumar, and C. Mercer. Experiences with processor reservation and dynamic qos in real-time mach. In *Proceedings of Multimedia*, Japan, 1996.

[CST+00] C.Lu, J.A. Stankovic, T.F.Abdelzaher, G.Tao, S.H.Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *IEEE Real-Time Systems Symposium*, Dec 2000.

[DGMM99] T. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck. Solving semi-markov decision problems using average reward reinforcement learning. *Management Science*, 45(4):560–574, 1999.

[DK95] D.Shasha and G. Koren. D-over: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *Siam Journal of Computing*, page 318339, 1995.

[DLSS96] D.Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, 1996.

[DVJB02] S.G. Dykes, D.C. Varner, J.Pricea, and B.Abbot. Adaptive scheduling for real-time network communication, 2002.

[EG01] E.Bini and G.Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In 13th *Euromicro Conference on Real Time System*, Delft, Netherlands, June 2001.

[EG02] E.Bini and G.Buttazzo. The space of rate monotonic schedulability. In *Proceedings of the* 23rd *IEEE Real-Time Systems Symposium*, Austin,Texas, December 2002.

[G.B97] G.Buttazzo. *Hard RealTime Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[G.J94] G.J.Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[III95] L.C. Baird III. Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, pages 30–37, 1995.

[JA96] S. J.Bradtke and A.G.Barto. Linear least–squares algorithms for temporal difference learning. *Machine Learning*, 1996.

[JPr00] J.Eker, P.Hagander, and K.E.Årzén. A feedback scheduler for real-time controller tasks. In *IFAC Control Engineering Pratice*, 2000.

[JR91] J.A.Stankovic and K. Ramamritham. The spring kernel: A new paradigm for real-time systems. *IEEE Software*, 8(3):62–72, May 1991.

[KA91] T.-W. Kuo and A.K.Mok. Load adjustment in adaptive real-time systems. In *Proceedings of the 12th IEEE Real-Time Systems Symposium*, Dec 1991.

[LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, 20(1):46–61, 1973.

[LLS+91] J. W.S. Liu, K. J. Lin, W. K. Shih, J. Y. Chung, A. Yu, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, page 5868, May 1991.

[L.P97] L.Pettersson. Real-time scheduling using fuzzy techniques, 1997.

[LRL90] L.Sha, R. Rajkumar, and J. Lehoczy. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transaction on Computers*, 1990.

[LSD89] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, 1989.

[LTY94] J. Lee, A. Tiao, and J. Yen. A fuzzy rule-based approach to real-time scheduling. In *Proceedings of IEEE International Conference on Fuzzy System*, pages 1394–1399, 1994.

[Lu01] C. Lu. *Feedback Control Real-Time Scheduling*. PhD thesis, University of Virginia, Charlottesville, May 2001.

[MCM96] L. Mendonca, C. Cardeira, and P.M. Martins. Fuzzy concepts applied to preemtive real-time tasks scheduling. In *Fourth IFAC Workshop on Algorithms and Architectures for Real-time Control*, 1996.

[M.K99] M.K.Gardner. Probabilistic analysss and scheduling of critical soft real-time systems. Master's thesis, University of Illinoise, Urbana-Champaign, 1999.

[M.L94] M.L.Putterman. *Markov decision processes-Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[MLA96] M.L.Littman, L.P.Kaelbling, and A.W.Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[PG93] P.Dayan and G.E.Hinton. Feudal reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems Conference*, pages 271–278, 1993.

[PS99] P.Richardson and S.Sarkar. Adaptive scheduling: Overload scheduling for mission critical systems. In *Fifth IEEE Real-Time Technology and Application*, pages 14–23, 1999.

[RA96] R.H.Crites and A.G.Barto. Improving elevator performance using reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems Conference*, pages 1017–1023, 1996.

[RA98] R.S.Sutton and A.G.Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Ros96] S. M. Ross. *Simulation*. Academic Press Inc., second edition, 1996.

[R.S95] R.S.Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Proceedings of Advances in Neural Information Processing Systems Conference*, pages 1038–1044, 1995.

[SCST99] J. A. Stankovic, C.Lu, S.H.Son, and G. Tao. The case for feedback control real-time scheduling. In *EuroMicro Conference on Real-Time Systems*, June 1999.

[SD97] S.Singh and D.Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems*, volume 9, page 974. The MIT Press, 1997.

[S.J94] S.J.Bradtke. *Incremental Dynamic Programming for On-Line Adaptive Optimal Control*. PhD thesis, University of Massachusetts, Amherst, 1994.

[SM95] S.J.Bradtke and M.O.Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Proceedings of Conference Of Advances in Neural Information Processing Systems*, pages 393–400, 1995.

[S.M96] S.Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1-3):159–195, 1996.

[SNDA97] S.Mahadevan, N.Marchalleck, T.K. Das, and A.Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proc. 14th International Conference on Machine Learning*, pages 202–210, 1997.

[Tao99] Z. Tao. A probabilistic metric for real-time embedded systems. Master's thesis, Univ. Notre Dame, Notre Dame, 1999.

[TC94] F. Terrier and Z. Chen. Fuzzy calculus applied to real time scheduling. In *IEEE Conference on Fuzzy Systems*, 1994.

[TEK00] T.F.Abdelzahera, E.M.Atkins, and K.G.Shin. Qos negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers*, 49(11):1170–1183, 2000.

[TRC95] F. Terrier, L. Rioux, and Z. Chen. Real time scheduling under uncertainty. In *IEEE Conference on Fuzzy Systems*, 1995.

[TSDS+95] T.-S.Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C.Wu, and J.W-.S. Liu. probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the Real-Time Technology and Applications Symposium*, pages 164–173. IEEE, May 1995.

[WRJ96] H.G. Wang, J.E. Rooda, and J.F.Haan. Solve scheduling problems with a fuzzy approach. In *IEEE Conference on Fuzzy Systems*, 1996.

[XTS01] X.S.Hu, T.Zho, and E.H.-M. Sha. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Transaction On Very Large Scale Integration (VLSI) Systems*, 9(6):833, December 2001.

[Yih92] Y. Yih. Learning real-time scheduling rules from optimal policy of semi-markov decision processes. *Int. Journal of Computer Integrated Manufacturing*, 5(3):171–181, 1992.

[YT91] Y. Yih and A. Thesen. Semi-markov decision models for real-time scheduling. *Int. Journal of Production Research*, 29:2331–2346, 1991.

[ZCBO91] D. N. Zhou, V. Charkassky, T. R. Baldwin, and D. E. Olson. A neural network approach to job-shop scheduling. *IEEE Trans. Neural Networks*, 2(1):175–179, 1991.

[ZD95] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intellience*, 1995.

[ZHS95] T. Zhou, X. Hu, and E. H.-M. Sha. A probabilistic performance metric for real-time system design. In *Proceeding of International Conference of Computer Design (ICCD95)*, Austin, Texas, Oct 1995.

[Zim96] H-J. Zimmermann. *Fuzzy Set Theory and its applications.* Kluwer Academic Publishers, third edition, 1996.