POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Ingegneria Informatica

RISE: UN SISTEMA DI GESTIONE ENERGETICA PER APPLICAZIONI VINCOLATE ALLA QUALITÀ DEL SERVIZIO

Relatore: Prof.ssa Raffaela MIRANDOLA

Correlatore: Prof. Moreno MARZOLLA

Tesi di Laurea di:

Mattia ZORDAN

Matricola n. 771487

Anno Accademico 2011–2012

A Silvana, Vito, Lucio e Barbara

Ringraziamenti

A tutta la mia famiglia, che ancora una volta mi ha permesso di arrivare dove sono.
E a Barbara, per essere stata al mio fianco, sempre.

Alla professoressa Mirandola e al professor Marzolla
per avermi aiutato durante tutto il percorso di tesi.

E a tutte le persone che mi hanno spronato, anche inconsapevolmente.

Sommario

L'incremento dei consumi energetici nei paesi industrializzati a cui si sta assistendo rappresenta una continua sfida per lo sviluppo sostenibile, in particolar modo per il settore IT. Per questa ragione sono nate discipline come il *green computing*, il cui obiettivo è quello di creare e gestire infrastrutture che riescano a consumare meno energia pur fornendo la stessa qualità del servizio.

L'obiettivo di questa tesi è quello di definire ed implementare il framework RISE (RIconfigurazione di Sistemi per il risparmio Energetico), in grado di fornire una strategia di gestione energetica che minimizzi i consumi all'interno di un'infrastruttura IT vincolata alla qualità del servizio (QoS). In particolare il vincolo per l'infrastruttura è espresso come un limite massimo del tempo di risposta relativo alle richieste che giungono dai client.

RISE fa uso dello standard ACPI (Advanced Configuration and Power Interface) e può quindi essere applicato ad ogni sistema i cui dispositivi lo supportano. Il framework è in grado di riconfigurare durante l'esecuzione gli stati ACPI di ogni dispositivo presente per rendere la potenza erogata e i relativi consumi proporzionali al suo carico, variabile nel tempo. Le riconfigurazioni possono comportare modifiche degli stati di performance dei dispositivi (P-State), che comportano un aumento o una diminuzione delle loro frequenze di clock, oppure modifiche dei loro stati globali, cioè passaggi tra gli stati di esecuzione, standby e mechanical off.

Le riconfigurazioni sono gestite tramite un ciclo di controllo del tipo *Monitor-Analyze-Plan-Execute*, in cui vengono inzialmente raccolti i dati relativi alle prestazioni del sistema durante l'esecuzione. In seguito essi vengono analizzati ed in caso il tempo di risposta non sia accettabile, vengono confrontate velocemente diverse possibili riconfigurazioni. Per ciascuna viene stimato il relativo tempo di risposta attraverso l'utilizzo di un efficiente modello di performance basato sulle reti di code, e quando la stima soddisfa i vincoli di sistema, la riconfigurazione ha effettivamente luogo. Dal momento che tutti i dati necessari per eseguire correttamente il modello di performance sono ottenibili misurando le prestazioni del sistema durante l'esecuzione, RISE non necessita di conoscenze preliminari di alcun tipo.

Il framework è definito in maniera tale da poter supportare sia sistemi omogenei, cioè formati dallo stesso tipo di dispositivi, sia eterogenei, cioè formati da blocchi di dispositivi identici al loro interno, ma differenti tra un blocco e l'altro.

Il contenuto della tesi è suddiviso in questo modo: nel capitolo 1 verrà fornita un'introduzione più estesa e dettagliata al lavoro proposto. Nel capitolo 2 verranno definiti la specifica ACPI ed il problema che si vuole risolvere. Sarà inoltre descritta l'architettura di RISE. Nel capitolo 3 verrà definita l'implementazione del sistema, ovvero saranno descritti tutti gli algoritmi utili ai fini dell'ottenimento del risparmio energetico vincolato alla qualità del servizio. Nel capitolo 4 saranno presentati i risultati delle simulazioni effettuate sia per il caso di dispositivi omogenei che per quello di sistema formato da blocchi eterogenei. Saranno inoltre definiti i parametri simulativi utilizzati per entrambi i casi. Infine nei capitoli 5 e 6 saranno rispettivamente definiti i principali related works e le conclusioni.

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'incremento dei consumi energetici nei paesi industrializzati rappresenta una continua sfida per lo sviluppo sostenibile. Secondo un'analisi di Koomey [1], l'energia elettrica totale consumata dal 2000 al 2005 è raddoppiata, con una crescita annuale del 16% a livello mondiale. Il consumo energetico totale dei server negli Stati Uniti d'America nel 2005 è stimato a circa $2.6 \times 10^{12} W$, mentre a livello mondiale si attesta attorno ai $7 \times 10^{12} W$. Aggiungendo alla stima anche l'energia richiesta dai sistemi di raffreddamento e dai dispositivi ausiliari, i consumi raddoppiano per entrambi i casi [1, Tabella 6].

Inoltre, come mostrato in [2, 3], l'interesse verso l'utilizzo efficiente della tecnologia è motivato da alcune allarmanti tendenze che mostrano come, per esempio, il consumo energetico dei dispositivi di calcolo nei soli Stati Uniti è superiore ai $20 \times 10^{6} GJ$ all'anno, l'equivalente di 4 milioni di tonnellate di anidride carbonica emessa nell'atmosfera [3]. Non sono tuttavia le sole emissioni a preoccupare: un'analisi [4] condotta da IDC stima il costo del settore IT a livello mondiale attorno ai 2100 miliardi di dollari nel 2012, e prevede una crescita annuale del 6% per gli anni futuri.

La crescita dei consumi energetici nel settore IT è quindi un problema importante, risolvibile attraverso la modifica della struttura e delle opera-

zioni svolte dai centri di servizio. Per questa ragione discipline come il *green computing* [5] stanno sempre più espandendosi per studiare come creare e gestire infrastrutture che consumano meno energia pur fornendo la stessa qualità di servizio [3]. Le rudimentali tecniche di gestione energetica basate solamente sullo spegnimento dei server, hanno forti impatti sulla capacità dei centri di servizio di mantenere gli SLA (Service Level Agreements) implicitamente o esplicitamente stipulati con i clienti. Benché lo spegnimento temporaneo dei server inutilizzati assicuri il massimo risparmio energetico, li rende però incapaci di processare alcuna richiesta. Riattivare le macchine quando necessario può portare a dei tempi di accensione molto lunghi (fino a qualche minuto, in base al tempo necessario per avviare tutte le applicazioni) e questo comportamento può seriamente compromettere la capacità dei sistemi di gestire improvvise variazioni di carico. Occorre inoltre considerare che ripetuti cicli di accensione e spegnimento possono diminuire notevolmente il tempo di vita dei componenti hardware, aumentando così i costi per il loro rimpiazzamento [6].

È necessario quindi riuscire a definire metodi e tecniche per il risparmio energetico differenti, più efficaci ed efficienti. È importante in questo senso osservare che l'utilizzo medio di un server in un datacenter è piuttosto basso, ma raramente pari a zero: come osservato in [7], per la maggior parte del tempo i server operano tra il 10 ed il 50% del loro massimo livello di utilizzo. Questa tendenza non è casuale, ma deriva dall'applicazione di appropriati principi ingegneristici: se il numero di server si riducesse per aumentare il loro utilizzo medio, allora il sistema opererebbe nei pressi del punto di saturazione. In figura 1.1 è mostrata la relazione tra utilizzo e tempo di risposta di un dispositivo generico: quanto più l'utilizzo si avvicina a uno, tanto più il tempo di risposta diverge fino a che non si viene a formare una coda di richieste infinita da processare. Quello che succede nella pratica in queste situazioni è che le richieste iniziano ad essere perse. Per

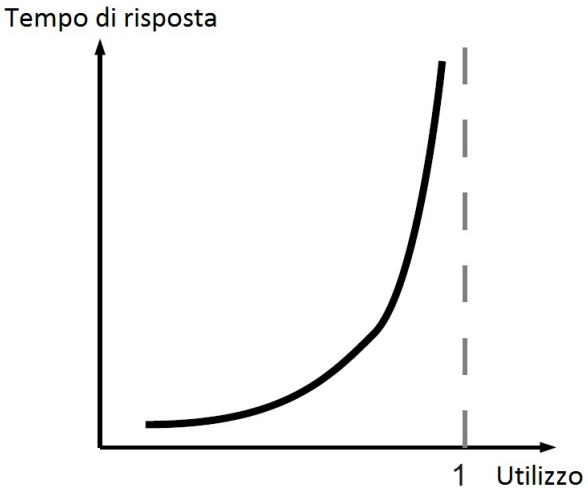

Figura 1.1: Tempo di risposta di un dispositivo in funzione al suo utilizzo

evitare ciò è necessaria della potenza di calcolo ausiliaria non utilizzata, in grado di gestire eventuali fluttuazioni del carico.

È inoltre importante osservare come una grande parte delle applicazioni Web include i cosiddetti servizi Online Data Intensive (OLDI) [8], come i motori di ricerca, giochi online multigiocatore di massa (MMOG) e altre simili che processano richieste esterne con vincoli molto restrittivi sui tempi di risposta. I carichi delle applicazioni OLDI sono definiti da richieste (generate dagli utenti) che devono interagire con enormi quantità di dati; le risposte devono poi essere consegnate entro un breve periodo di tempo, spesso inferiore al secondo. Inoltre i carichi fluttuano considerevolmente nel tempo, seguendo pattern periodici in linea con i periodi attivi umani. In figura 1.2 è mostrato come esempio il grafico relativo al numero di giocatori connessi al MMOG "EVE online" recuperato da un'analisi svolta nel 2007 [9]. Esso mostra come nelle quattro settimane del mese di Marzo 2004, i pattern di connessione siano quasi coincidenti tra loro.

Le osservazioni fatte finora suggeriscono quindi di ridurre i consumi energetici rendendo la potenza di calcolo dei dispositivi proporzionale al

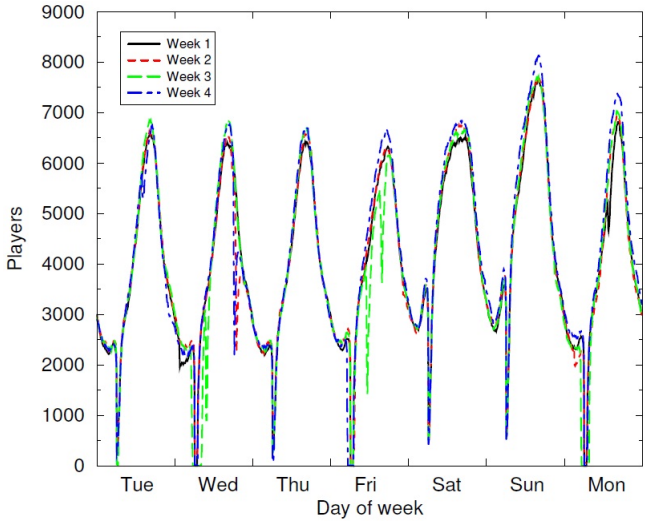

Figura 1.2: Numero di giocatori connessi a "EVE online" (Marzo 2004) [9]

loro utilizzo (*power proportional systems* [7]). Idealmente un dispositivo non dovrebbe consumare alcuna energia quando il suo utilizzo è pari a zero, e dovrebbe consumarne il massimo quando è pari a uno. Questa proporzionalità perfetta non è stata però ancora raggiunta, dal momento che gli attuali processori dei server consumano circa il 30% del loro picco massimo durante i periodi di minor attività, rendendo così disponibile un intervallo dinamico in cui spaziare di circa il 70%. Questo intervallo può essere sfruttato per ridurre i consumi osservando che la maggior parte dei processori ammette l'utilizzo del *dynamic frequency/voltage scaling* (DVS), metodo che permette di regolare la frequenza ed il voltaggio del processore così da ridurne i consumi e le prestazioni quando necessario.

L'Advanced Configuration and Power Interface (ACPI) è stato proposto come standard aperto per la configurazione e la gestione energetica dei dispositivi o di interi sistemi [11]. Esso definisce un ampio insieme di stati a basso consumo, alcuni dei quali sono completamente operativi e permettono quindi di processare richieste degli utenti ad un'eventuale ridotta frequenza. Per i processori, la riduzione dei consumi è ottenuta grazie al

dynamic frequency scaling che permette di modificare dinamicamente la loro frequenza e il loro voltaggio. Dal momento che i cambi di stato nello standard ACPI possono essere controllati dal software, l'implementazione di più sofisticate tecniche di trade-off tra performance e energia consumata può portare a significativi risparmi energetici che non era possibile ottenere con la precedente generazione di dispositivi.

In questo lavoro di tesi si cerca di perseguire questo obiettivo, proponendo una strategia di gestione energetica che minimizzi il consumo globale di un'infrastruttura IT vincolata alla qualità del servizio (QoS). In particolare verrà utilizzato il dynamic voltage scaling per i dispositivi del sistema combinato a delle dinamiche di attivazione e disattivazione che garantiranno al sistema la capacità di reagire ad eventuali variazioni del carico. È importante osservare che i vantaggi principali ottenuti dall'utilizzo del dynamic voltage scaling consistono nella rapidità con cui può essere applicato ad un processore e nel trascurabile overhead che comporta.

Sarà proposto, come estensione di EASY [10], il framework RISE (RIconfigurazione di Sistemi per il risparmio Energetico) per l'ottimizzazione dinamica dei consumi di sistemi vincolati alla qualità del servizio. Sarà considerato un sistema distribuito che utilizza un'applicazione di tipo OLDI, vincolata a limiti di tempo: in particolare il tempo di risposta dovrà essere mantenuto al di sotto di una soglia massima. Il sistema è formato da un insieme di dispositivi che supportano lo standard ACPI. Può inoltre essere suddiviso in vari blocchi eterogenei, ciascuno formato da dispositivi omogenei tra loro. È inoltre soggetto ad un carico variabile nel tempo: il carico potrà essere suddisiviso in maniera iniqua tra i vari server e potrà essere spostato da un dispositivo ad un altro in qualunque momento. RISE è un sistema di selezione a runtime dei livelli di performance dei dispositivi tale per cui il tempo di risposta medio è mantenuto al di sotto della soglia massima con il minimo consumo energetico. Facendo affidamento

allo standard ACPI, RISE può essere applicato a qualunque insieme di dispositivi compatibili con esso (non solo CPU), rendendo quindi il nostro approccio generalizzabile e applicabile a differenti sistemi.

RISE è basato sul ciclo di controllo *Monitor-Analyze-Plan-Execute*: il sistema è fornito di un monitor che si occupa di raccogliere le misurazioni delle prestazioni durante l'esecuzione. Queste informazioni sono poi usate sia per identificare gli istanti in cui i vincoli sul tempo di risposta sono violati, sia per dare inizio ad un processo di riconfigurazione. Nelle fasi di analisi e pianificazione viene utilizzato un modello di performance a reti di code per analizzare velocemente varie configurazioni alternative. I dati raccolti dal monitor sono sufficienti per poter creare un modello di performance corretto, per cui RISE non richiede nessuna conoscenza preliminare sul sistema. La soglia massima del tempo di risposta può essere poi facilmente modificata durante l'esecuzione, in modo tale che gli amministratori di sistema possano facilmente ridefinire lo SLA. Il problema di minimizzazione energetica sarà definito come un problema di programmazione intera mista (MIP). Verrà mostrato l'utilizzo di un efficace procedimento euristico per la ricerca di una soluzione al problema e infine verranno mostrati i risultati di varie simulazioni svolte sul sistema al fine di garantire la sua qualità.

Struttura del documento. Il lavoro è descritto nella seguente maniera: nel capitolo 2 verranno definiti la specifica ACPI ed il problema che si vuole risolvere. Sarà inoltre descritta l'architettura di RISE. Nel capitolo 3 verrà definita l'implementazione del sistema, ovvero saranno descritti tutti gli algoritmi utili ai fini dell'ottenimento del risparmio energetico vincolato alla qualità del servizio. Nel capitolo 4 saranno presentati i risultati delle simulazioni effettuate sia per il caso di dispositivi omogenei che per quello di sistema formato da blocchi eterogenei. Saranno inoltre definiti i parame-

tri simulativi utilizzati per entrambi i casi. Infine nei capitoli 5 e 6 saranno rispettivamente definiti i principali related works e le conclusioni.

Capitolo 2

Definizione del Problema

In questo capitolo saranno descritti i concetti principali a partire dai quali è stato svolto il lavoro di tesi. In particolare la sezione 2.1 descriverà lo standard ACPI, la sezione 2.2 indicherà il problema che si vuole risolvere ed infine la sezione 2.3 definirà l'architettura di RISE ed in che modo essa si proponga di risolverlo.

2.1 Standard ACPI

L'Advanced Configuration and Power Interface (ACPI) [11] è un'interfaccia molto utile supportata da tutte le più moderne CPU. È uno standard libero per la configurazione e la gestione energetica indipendente dalla piattaforma su cui è eseguito, che può agire sia sui singoli dispositivi che sull'intero sistema. I dispositivi ACPI-compliant permettono inoltre a componenti di più alto livello di controllare direttamente il consumo di energia all'interno del sistema. Questa è un'importante caratteristica dello standard ACPI ed un grande miglioramento rispetto alle precedenti soluzioni di risparmio energetico: è infatti possibile permettere il controllo a livello di sistema operativo, in modo da prendere decisioni più corrette grazie all'ausilio delle informazioni relative all'hardware e alle applicazioni in ese-

cuzione da esso monitorate. Ciò non era precedentemente possibile, poiché venivano utilizzate soluzioni basate su ROM BIOS.

Lo standard ACPI definisce *stati globali* relativi all'intero sistema, *stati delle periferiche* relativi ad un componente individuale e *stati del processore*. Gli *stati globali*, definiti da **G0** a **G3**, si possono elencare in ordine decrescente di consumo energetico nel seguente modo:

- **G0-Esecuzione(Working).** In questo stato il sistema opera normalmente, i thread applicativi sono eseguiti e gli stati dei singoli componenti possono essere modificati dinamicamente.
- **G1-Addormentato(Sleeping).** In questo stato il sistema richiede un basso consumo energetico, tuttavia i thread applicativi non sono eseguiti. Il sistema può essere riportato allo stato di esecuzione senza bisogno di un riavvio.
- **G2-Soft off.** Il sistema richiede un minimo consumo energetico ma i software di sistema e applicativo non sono eseguiti. Per essere riportato allo stato di esecuzione il sistema deve essere riavviato, operazione che impiega un consistente periodo di tempo.
- **G3-Mechanical off.** In questo stato il sistema è spento e non riceve corrente. L'unico consumo energetico presente è quello relativo al real-time clock, quindi molto vicino allo zero.

Gli *stati delle periferiche*, definiti da **D0** a **D3**, interessano i singoli dispositivi connessi al sistema (come disk drive, monitor, ecc.). Essi possono essere elencati in ordine decrescente di consumo energetico nel seguente modo:

- **D0-Attivo.** Il dispositivo è in normale esecuzione e richiede il massimo consumo energetico.

- **D1-D2-D3Hot.** Questi stati richiedono sempre meno energia, partendo da D1 e arrivando a D3Hot, ma il dispositivo conserva una minore quantità di contesto di esecuzione.
- **D3-Spento.** In questo stato la corrente è rimossa dal dispositivo. Nel momento in cui si passa in questo stato, il contesto di esecuzione è perso.

Gli *stati del processore* sono definiti all'interno dello stato globale **G0** e possono essere elencati in ordine decrescente di consumo energetico nel seguente modo:

- **C0.** Il processore esegue le istruzioni normalmente.
- **C1-Halt.** Il processore non esegue istruzioni, ma può ritornare allo stato **C0** con un minimo ritardo. Tutti gli stati visibili del software sono preservati.
- **C2-Stop Clock.** Questo stato richiede meno energia del precedente, ma anche un maggior ritardo nel passare allo stato **C0**. Gli stati del software sono ancora preservati.
- **C3-Sleep.** Questo stato offre un maggior risparmio energetico, ma non garantisce *cache coherence*, che deve quindi essere controllata dal software al riavvio.

Lo standard ACPI definisce inoltre *stati di performance P(i)* (*P-States*) per il processore e i dispositivi all'interno degli stati attivi (**C0** per il processore e **D0** per i dispositivi). Fino a 16 *stati di performance* sono supportati (da **P0** a **P16**) ed ogni stato *P(i)* richiede più energia dello stato *P(i+1)* perché più performante. In figura 2.1 sono mostrate le transizioni valide tra gli stati appena definiti. In questo lavoro ci si è concentrati sugli stati **G0** (esecuzione), **G1-S1** (standby, che per comodità verrà chiamato semplicemente **G1** da ora in avanti) e **G3** (mechanical off). Inoltre sono stati utilizzati pesantemente

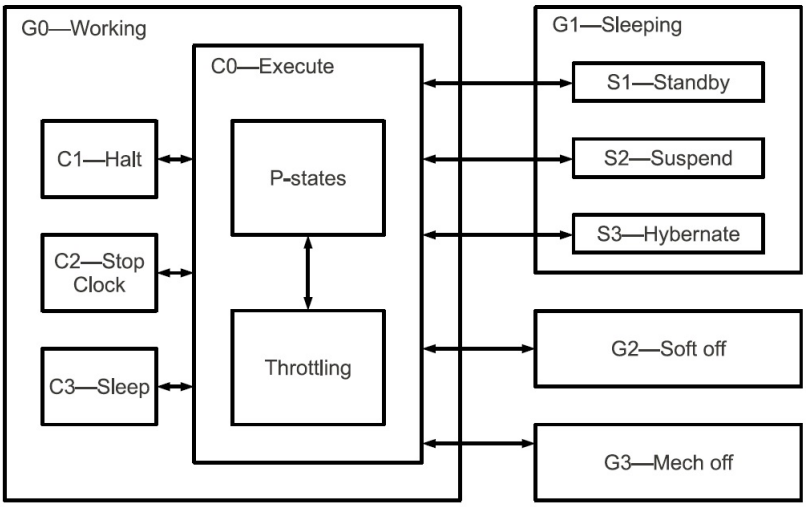

Figura 2.1: Transazioni consentite dallo standard ACPI [13]

P-State	Frequency	Voltage	Power
P0	2.8 GHz	1.40 V	92.6 W
P1	2.6 GHz	1.35 V	90.2 W
P2	2.4 GHz	1.30 V	77.0 W
P3	2.2 GHz	1.25 V	65.7 W
P4	2.0 GHz	1.20 V	55.9 W
P5	1.8 GHz	1.15 V	47.6 W
Pmin	1.0 GHz	1.10 V	36.1 W

Figura 2.2: P-States del processore AMD Opteron [1, 19]

gli *stati di perfomance del processore*: essi garantiscono una diminuzione consistente del consumo energetico ed il tempo necessario per passare da uno all'altro comporta un overhead trascurabile. In particolare la riduzione del consumi è ottenuta modificando dinamicamente il voltaggio del processore, il clock rate, o entrambi, dal momento che la potenza P dissipata da un circuito CMOS soddisfa la relazione[12]:

$$P \propto CV^2 f$$

dove C è la capacità, V il voltaggio e f la frequenza di clock. Varie tecnologie sono utilizzate da differenti produttori per abbassare i consumi energe-

P-State	Frequency	Voltage	Power
P0	2.0 GHz	1.148 V	20 W
P1	1.8 GHz	1.132 V	18 W
P2	1.6 GHz	1.100 V	15 W
P3	1.4 GHz	1.052 V	13 W
P4	1.0 GHz	1.004 V	10 W
P5	800 MHz	0.844 V	7 W
P6	600 MHz	0.844 V	6 W
P7	400 MHz	0.844 V	5 W

Figura 2.3: P-states del processore 2GHZ VIA C7-M [1, 20]

tici, come Intel SpeedStep [14], AMD PowerNow! [15] o IBM EnergyScale [16], e nuovi tipi di misurazione sono introdotti per migliorarne l'efficienza, come nel caso del metodo ACP di AMD [17]. Oltre a questi, AMD CoolCore [18] è in grado di spegnere singole parti del processore per ridurre ulteriormente il consumo di energia e la temperatura della CPU. In figura 2.2 sono mostrati il voltaggio, la frequenza ed il consumo energetico del processore AMD opteron, che fa parte della precedente generazione di processori AMD per server e dekstop. In figura 2.3 sono invece mostrati gli stessi valori riferiti al processore VIA C7-M da 2GHz, attualmente utilizzato nel mercato mobile.

2.2 Formulazione del Problema

In questa sezione sarà introdotta la notazione usata nel resto del documento e verrà inoltre descritto formalmente il problema trattato.
Si vuole riuscire a modellare un sistema in grado di eseguire un'applicazione su un sistema distribuito di dispositivi. L'insieme dei dispositivi è $\mathcal{K} = \{1,...,K\}$ ed ogni dispositivo è identificato univocamente da un intero k in $\mathcal{K}$. Ogni singolo device può essere indifferentemente una CPU, un disco o qualunque altro dispositivo. I richiedenti (*client*) interagiscono con l'ap-

plicazione generando richieste, che vengono processate dal sistema. Durante questa fase viene richiesto il servizio di uno o più dispositivi, e infine viene restituita una risposta al richiedente.
Ad ogni dispositivo k sono associati:

- $P[k]$ stati di esecuzione (o P-State), identificati da un insieme di interi cotenuti in $\mathcal{P}[k] = \{1,...,P[k]\}$ con lo stato di esecuzione 1 associato alla massima velocità e al massimo consumo energetico, e lo stato $P[k]$ associato alla minima velocità ed al minimo consumo energetico.

- Un parametro di standby del dispositivo $St[k] \in \{0,1\}$ che indica se il dispositivo k è nello stato **G1 (standby)** della specifica ACPI ($St[k] = 1$) o in uno tra gli altri stati validi **G3 (Mechanical Off)** o **G0 (Esecuzione)** ($St[k] = 0$). Nel caso $St[k] = 1$ lo stato di esecuzione del dispositivo k definito al punto precedente è ininfluente all'interno del modello, in quanto esso non elabora thread applicativi.

- Un parametro di accensione $O[k] \in \{0,1\}$ che indica se il dispositivo k è nello stato **G3 (Mechanical Off)** della specifica ACPI ($O[k] = 0$) o in uno tra gli stati **G0 (Esecuzione)** o **G1 (Standby)** ($O[k] = 1$). Nel caso $O[k] = 0$ lo stato di esecuzione del dispositivo k definito al primo punto è ininfluente all'interno del modello, in quanto esso è spento.

È importante sottolineare che quando $O[k] = 1$ le uniche combinazioni in cui si può trovare il sistema sono $St[k] = 1$ (dispositivo in standby, $L[k]$ ininfluente) e $St[k] = 0$ (dispositivo in esecuzione, $P[k]$ influente), mentre quando $O[k] = 0$ sia $P[k]$ che $St[k]$ sono ininfluenti, in quanto il dispositivo è spento. Da questo momento in avanti, quando si parlerà di P-State $P[k]$ si sottointenderà che esso sia significativo, ovvero che il dispositivo k sia nello stato **G0**.

Definiamo con *EN*[*k,j*] il tasso di consumo energetico del dispositivo $k \in \mathcal{K}$ nello stato di esecuzione $j \in \mathcal{P}$. Come conseguenza, il consumo di un dispositivo *k* nello stato *j* per una durata temporale *T* è pari a $T \times EN[k, j]$.

Indichiamo poi con *RSP*[*k,j*] la *velocità relativa* del dispositivo *k* nello stato *j* come:

$$RSP[k, j] = \frac{\text{Velocità di elaborazione di } k \text{ nello stato } j}{\text{Velocità di elaborazione di } k \text{ nello stato } 1} \tag{2.1}$$

Intuitivamente questo valore indica la perdita di capacità elaborativa di un dispositivo *k* in uno stato *j* rispetto alla sua capacità massima (nello stato 1). Unica condizione su questo indice è che un qualunque dispositivo che lavori ad uno stato j_1, sia più veloce di ogni stato con indice $j_2 > j_1$:

$$1 = RSP[k, 1] > RSP[k, 2] > ... > RSP[k, L[k]] > 0$$

Una *configurazione di P-State* è un vettore **P** = (*P*[1],..,*P*[*K*]) in cui ogni valore rappresenta lo stato di esecuzione $P[k] \in \mathcal{P}[k]$ associato al dispositivo *k*. Similmente una *configurazione di standby* è un vettore **St** = (*St*[1],..,*St*[*K*]) in cui ogni elemento indica il parametro di standby $St[k] \in \{0,1\}$ del dispositivo *k*, e una *configurazione di esecuzione* è un vettore **O** = (*O*[1],..,*O*[*K*]) in cui ogni elemento indica il parametro di accensione $O[k] \in \{0,1\}$ del dispositivo *k*. Una *configurazione di sistema* è una terna **S** = {**P**, **St**, **O**}.

L'obiettivo di RISE è quello di modificare dinamicamente lo stato di ogni dispositivo del sistema al fine di minimizzare il consumo energetico globale mantenendo il tempo di risposta stimato al di sotto di una soglia R_{max}. Definiamo ora con la costante E_{st} il consumo energetico istantaneo di un dispositivo in standby, con *R*(**S**(t)) il tempo di risposta del sistema in configurazione di sistema **S** al tempo *t* e con *E*(**S**(t)) la quantità di energia totale dissipata dal sistema nelle stesse condizioni. Otteniamo in questo

modo il problema di ottimizzazione nel quale va identificata una terna **S**(t) tale che:

$$
\begin{aligned}
&\mathcal{P}(R_{max}) \equiv \\
&\textbf{Minimizzare:} \quad E(\mathbf{S}(t)) = \sum_{k=1}^{K} EN[k, P[k](t)] \times (O[k](t) - St[k](t)) + \\
&\qquad\qquad + \sum_{k=1}^{K} E_{st} \times St[k](t) \\
&\textbf{Con vincoli:} \quad R(\mathbf{S}(t)) \leq R_{max} \\
&\qquad\qquad P[k](t) \in \mathcal{L}[k] \text{ , } \forall k \in \mathcal{K} \\
&\qquad\qquad St[k](t) \in \{0,1\} \text{ , } \forall k \in \mathcal{K} \\
&\qquad\qquad O[k](t) \in \{0,1\} \text{ , } \forall k \in \mathcal{K} \\
&\qquad\qquad St[k](t) = 1 \implies O[k](t) = 1 \text{ , } \forall k \in \mathcal{K} \\
&\qquad\qquad \sum_{k=1}^{K} O[k](t) > \sum_{k=1}^{K} St[k](t)
\end{aligned}
\tag{2.2}
$$

All'interno della funzione obiettivo si calcolano i consumi energetici direttamente da *EN* dei soli dispositivi allo stato ACPI **G0**, ovvero quelli per cui vale $O[k](t) - St[k](t) = 1$. Per i dispositivi in standby viene invece utilizzato il parametro costante E_{st}. È inoltre importante osservare che l'ultimo vincolo è inserito solo in conseguenza del fatto che il calcolo di *R*(**S**(t)) non è esplicitato: infatti se fosse che $\sum_{k=1}^{K} O[k](t) = 0$ o che $\sum_{k=1}^{K} O[k](t) = \sum_{k=1}^{K} St[k](t)$, allora *R*(**S**(t)) = ∞ sarebbe una conseguenza logica che renderebbe di fatto il problema di ottimizzazione irrisolvibile. Tre ultime osservazioni riguardo al problema 2.2:

- Ogni dispositivo ha una potenza di calcolo finita e limitata. Il problema di ottimizzazione può di conseguenza essere irrisolvibile nel caso in cui il numero di richieste generi un carico di lavoro maggiore della capacità di calcolo massima del sistema. Nel caso in cui si presenti

questa occorrenza, RISE restituirà la soluzione in cui ogni dispositivo è attivo e **P** = (1,...,1).

- All'interno del problema di ottimizzazione è stato omesso volontariamente il dettaglio che riguarda i tempi di transizione dei dispositivi tra gli stati **G0**, **G1** e **G3** (sono infatti ipotizzati istantanei). Questi non sono trascurabili a differenza delle transizioni tra i P-State. Il motivo principale di questa scelta è quello di cercare di mantenere il problema semplice e comprensibile senza inserire le complicazioni che deriverebbero altrimenti. Nei capitoli successivi, durante la descrizione dell'implementazione del sistema, sarà comunque messo in evidenza il fatto che all'interno del codice del modello questi tempi sono stati presi in considerazione, e che i costi energetici relativi hanno un loro peso.

- Dal momento che il tempo di risposta $R(\mathbf{S}(t))$ non dipende solo dalla terna **S**(t) ma anche da parametri come l'intensità di utilizzo (ovvero il numero di utenti che interagiscono contemporaneamente con il sistema), altamente variabile nel tempo, il problema 2.2 appena definito andrebbe risolto ripetutamente durante l'esecuzione. Questo garantirebbe al sistema di mantenere i requisiti richiesti con il minimo consumo energetico. Ciò è però impossibile, poiché la continua ricomputazione diverrebbe il collo di bottiglia del sistema stesso. Per questa ragione nella prossima sezione sarà introdotto un approccio pratico per riconoscere al meglio quando è necessaria una riconfigurazione.

2.3 Architettura RISE

Il problema 2.2 rappresenta una famiglia di problemi parametrizzati dal valore di tempo continuo t. Per rendere il problema trattabile occorre considerare valori discreti di t, definiti t = 0,1,2.. , in cui ognuno di essi identifica

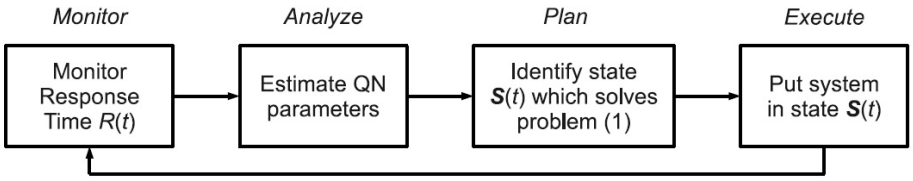

Figura 2.4: Ciclo di controllo Monitor-Analyze-Plan-Execute [10]

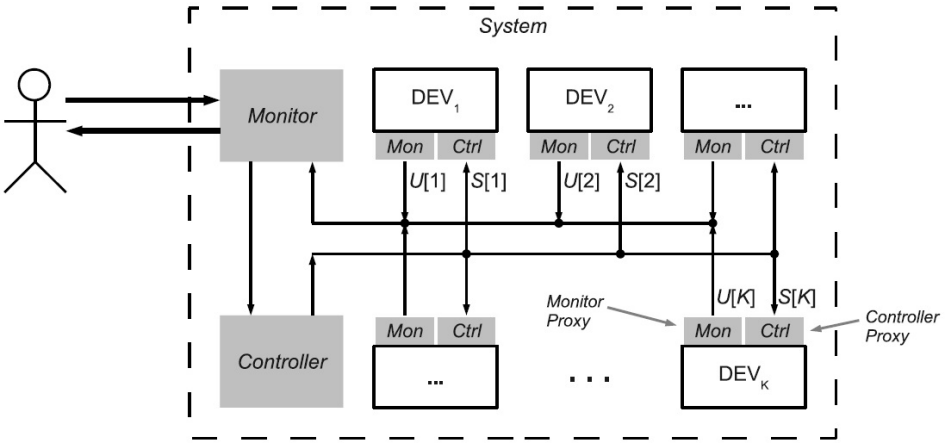

Figura 2.5: Architettura del sistema [10]

un intervallo di durata Δt all'interno del quale la configurazione di sistema non cambia. Al termine di ogni periodo invece, una nuova configurazione è identificata e applicata se necessario. La durata Δt non corrisponde ad un valore fisso, ma può variare da istante ad istante; può essere inoltre modificata per gestire il trade-off tra velocità di risposta e overhead di RISE.

In questo lavoro di tesi è proposto un sistema basato sul ciclo di controllo *Monitor-Analyze-Plan-Execute* mostrato in figura 2.4. Durante la fase *Monitor*, RISE misura i tempi di risposta, il throughput ed il tempo di utilizzo per ogni dispositivo, raccogliendo informazioni dal sistema in esecuzione nell'intervallo temporale corrente t. Alla fine di questa fase, i dati raccolti vengono utilizzati nello step *Analyze* per definire un modello prestazionale basato sulle reti di code. Questo modello è utilizzato nella fase *Plan* per analizzare diverse possibili configurazioni finché non ne viene trovata una che approssima il problema di ottimizzazione 2.2. A questo punto la solu-

zione trovata diventerà la nuova configurazione di sistema $\mathbf{S}(t+1)$ durante la fase finale *Execute*.

Per garantire l'implementazione del ciclo di controllo appena presentato, il sistema è dotato dei componenti aggiuntivi mostrati in grigio nella figura 2.5: il *monitor* è responsabile della raccolta dei dati riguardanti la performance del sistema in esecuzione, ovvero il tempo di risposta $\overline{R}(t)$, il throughput di sistema $\overline{X}(t)$ e l'utilizzo dei singoli dispositivi $\overline{U}[k](t)$ all'interno della configurazione corrente $\mathbf{S}(t)$. Il tempo di risposta del sistema e il suo throughput sono misurati grazie ad un'analisi passiva delle richieste al sistema mentre l'utilizzo dei dispositivi è calcolato grazie a misuratori integrati ad essi. Il *controller* viene attivato alla fine dell'intervallo di osservazione *t* e utilizza i dati raccolti dal *monitor* per identificare un nuovo stato $\mathbf{S}(t+1)$ che risolva il problema 2.2. I nuovi stati sono inviati ai singoli dispositivi come comandi in grado di dare inizio ad un cambio di stato ACPI.
È importante ricordare che il *monitor* ed il *controller* operano a runtime, e devono quindi essere efficienti per non diventare il collo di bottiglia del sistema. In particolare il *controller* deve riuscire ad analizzare l'impatto di differenti configurazioni sul tempo di risposta in maniera rapida. In questo lavoro viene utilizzato un modello di performance basato su reti di code [21] che possiede tecniche risolutive efficienti, ed è in grado di modellare molti tipi diversi di sistemi. Esso verrà discusso nella sezione 2.3.2.

Nella sezione successiva (2.3.1) sarà invece descritta la procedura di funzionamento del ciclo di controllo di RISE.

2.3.1 Ciclo di Controllo RISE

Verrà ora proposta una procedura euristica di tipo *greedy* per identificare una possibile soluzione al problema 2.2, analizzando una limitata quantità di alternative. Il *monitor* di figura 2.5, come già detto, si occupa di raccogliere informazioni sul sistema in esecuzione durante il periodo *t*

di durata Δt, quali il throughput, il tempo di risposta e l'utilizzo dei singoli dispositivi. Indicando con C il numero di richieste che il sistema ha processato durante il periodo t, e con $t[1], t[2], .., t[C]$ i tempi richiesti per processare le singole richieste, allora il tempo di risposta del sistema $\overline{R}(t)$ può essere stimato come $\sum_{c=1}^{C} t[c]/C$, e il throughput di sistema $\overline{X}$ come $C/\Delta t$. L'utilizzo del dispositivo *k*, indicato con $\overline{U}[k](t)$, è la parte di tempo dell'intervallo Δt in cui esso è occupato a processare una qualche richiesta. Per definizione consegue che $\overline{U}[k](t) \in [0, 1]$.

Al termine del periodo di osservazione le informazioni ottenuto dal *monitor* sono utilizzate per decidere se è necessaria una riconfigurazione del sistema. L'idea alla base della procedura è che se il tempo di risposta $\overline{R}(t)$ è al di sotto della soglia R_{max}, il *controller* cercherà di rallentare i dispositivi in esecuzione incrementandone i P-State ed eventualmente portandone qualcuno dallo stato **G0** ad uno tra **G1** e **G3**; Al contrario, se $\overline{R}(t)$ è al di sopra della soglia R_{max}, cercherà di velocizzare i dispositivi in esecuzione diminuendone i P-State ed eventualmente portandone qualcuno da uno stato inattivo allo stato **G0**. L'approccio presentato ha però il grave difetto di innescare troppe configurazioni nel momento in cui $\overline{R}(t)$ si muove continuamente al di sopra ed al di sotto di R_{max} (come è lecito aspettarsi in un contesto reale). Per risolvere il problema viene quindi introdotto un meccanismo di *isteresi* che aggiunge due ulteriori soglie, R_{low} e R_{high}, in modo tale che valga $R_{low} < R_{high} < R_{max}$. Con questa modifica una riconfigurazione è necessaria soltanto se $\overline{R}(t) < R_{low}$ o $\overline{R}(t) > R_{high}$.

L'algoritmo 1 mostra il ciclo di controllo di RISE appena descritto, comprendente il meccanismo di isteresi. Ad ogni iterazione l'utilizzo dei dispositivi $\overline{U}[k](t)$, il throughput $\overline{X}(t)$ e il tempo di risposta $\overline{R}(t)$ vengono misurati raccogliendo informazioni nell'intervallo di tempo Δt (riga 4). Se il tempo di risposta $\overline{R}(t)$ è maggiore di R_{high}, RISE identifica una nuova

Algorithm 1 Algoritmo di controllo RISE

Require: Soglie $R_{low} < R_{high} < R_{max}$

1: Sia **S** la configurazione iniziale del sistema
2: **while** (true) **do**
3: t := istante attuale
4: Misura $\overline{\mathbf{U}}(t)$, $\overline{X}(t)$, $\overline{R}(t)$ nell'intervallo Δt
5: **if** ($\overline{R}(t) > R_{high}$) **then**
6: $\mathbf{S}' = \texttt{SpeedUp}(\mathbf{S}, \overline{\mathbf{U}}(t), \overline{X}(t), \overline{R}(t))$
7: **else if** ($\overline{R}(t) < R_{low}$) **then**
8: $\mathbf{S}' = \texttt{SlowDown}(\mathbf{S}, \overline{\mathbf{U}}(t), \overline{X}(t), \overline{R}(t))$
9: **end if**
10: applica stato di sistema **S**′
11: **S** = **S**′
12: **end while**

configurazione più performante di quella attuale (riga 6). Se il tempo di risposta $\overline{R}(t)$ è minore di R_{low}, RISE identifica una configurazione meno performante di quella attuale e che quindi richiede meno energia (riga 8). Le due funzioni `SPEEDUP`() e `SLOWDOWN`() risolvono rispettivamente il problema di ottimizzazione $\mathcal{P}(R_{low})$ e $\mathcal{P}(R_{high})$.

Assegnando a Δt, R_{low} e R_{high} valori appropriati è possibile ottenere differenti tradeoff tra reattività del sistema e propensione a riconfigurare. Nello specifico Δt controlla la reattività di RISE: maggiore il suo valore, più lento è il sistema a riconfigurarsi. Minore il suo valore, più è oneroso l'overhead degli algoritmi di controllo. I parametri R_{low} e R_{high} controllano invece quanto RISE sia sensibile ai cambiamenti del tempo di risposta medio $\overline{R}(t)$.

2.3.2 Modello di Performance

Come anticipato nella sezione 2.3, RISE utilizza un modello di performance per poter stimare il tempo di risposta in configurazioni di sistema differenti. In particolare il sistema è modellato come una rete di code chiusa a singola classe in cui i dispositivi sono rappresentati come i centri di calcolo che smistano le code. Le richieste circolano nel sistema aggiungendosi alle code associate ai dispositivi di cui richiedono il servizio. Al termine del servizio di una richiesta, essa può essere inviata ad un altro dispositivo oppure spedita all'utente che l'ha generata.

In una rete chiusa esiste un numero fisso di richieste circolanti all'interno del sistema. Ciò vuole rappresentare il fatto che ogni client può avere un numero fisso massimo di richieste inviate, e aspetterà che una di esse sia completa prima di inoltrarne delle altre. È comunque permesso il variare del numero di client nel tempo, dal momento che ogni Δt unità di tempo una nuova rete è creata e analizzata.

Le reti di code sono un formalismo molto studiato e forniscono un buon tradeoff tra accuratezza ed efficienza degli algoritmi risolutivi. In particolare, le reti di code in forma prodotto hanno una semplice espressione a forma chiusa riguardante la distribuzione degli stati stazionari, così da permettere la misura delle performance in maniera efficiente [21].

Come osservato in precedenza, l'efficienza degli algoritmi di riconfigurazione è d'importanza primaria per RISE, dal momento che deve operare durante l'esecuzione e calcolare nuove configurazioni il più velocemente possibile. Nonostante le reti di code in forma prodotto debbano soddisfare determinati vincoli che riducono in qualche modo la loro accuratezza nel modellare sistemi reali (vedere [21] per ulteriori dettagli), esse sono abbastanza accurate per poter guidare il processo di identificazione di nuove configurazioni. Occorre sottolineare anche che l'accuratezza di un modello di performance è legato alla sua espressività (la capacità di catturare il mag-

gior numero possibile di dettagli comportamentali del sistema reale) e all'accuratezza dei parametri in ingresso utilizzati all'interno del modello. In RISE i parametri del modello vengono letti dal monitor, che ha una visione fugace e non sempre aggiornata del sistema in esecuzione. Ne consegue che non essendo presenti dati totalmente precisi, utilizzare tecniche e modelli più accurati (e quindi più lenti da analizzare) ha una bassa giustificazione.

Il modello a reti di code usato da RISE contiene K centri di servizio, dove il centro k corrisponde al dispositivo $k \in \mathcal{K}$. Il tempo di risposta del sistema $R(\mathbf{S})$ in configurazione **S** può essere stimato a partire dal tempo di servizio di ogni dispositivo e dal numero di richieste nel sistema.

Sia $\overline{X}(t)$ il throughput di sistema misurato al tempo t, $\overline{U}[k](t)$ l'utilizzo misurato del dispositivo k e $D[k, S[k](t)]$ il tempo di servizio del device k nello stato corrente $S[k](t)$. Il tempo di servizio è il tempo totale speso da una richiesta in un dispositivo k e può essere definito, a partire dalla Utilization Law [22], come:

$$D[k, S[k](t)] = \frac{\overline{U}[k](t)}{\overline{X}(t)}$$

Il tempo di servizio $D[k, p]$ per un generico stato $p \in \mathcal{P}$ può essere stimato come:

$$D[k, p] = \frac{\text{Velocità di elaborazione di } k \text{ allo stato } p}{\text{Velocità di elaborazione di } k \text{ allo stato } S[k](t)} \times \frac{\overline{U}[k](t)}{\overline{X}(t)}$$

Applicando a quest'ultima l'equazione 2.1, si può riscrivere come:

$$D[k, p] = \frac{RSP[k, S[k](t)]}{RSP[k, p]} \times \frac{\overline{U}[k](t)}{\overline{X}(t)} \tag{2.3}$$

Il numero di richieste nel sistema al tempo corrente $N(t)$, può essere a sua volta calcolato dal throughput di sistema $\overline{X}(t)$ e dal tempo di risposta $\overline{R}(t)$ utilizzando la legge di Little [23]:

$$N(t) = \overline{X}(t) \times \overline{R}(t) \tag{2.4}$$

Algorithm 2 EstimateRespT(**S′**,**S**,$\overline{\mathbf{U}}$,$\overline{X}$,$\overline{R}$) $\implies$ R(**S′**)

Require: **S′** configurazione arbitraria

Require: **S** = **S**(t) configurazione attuale

Require: $\overline{\mathbf{U}} = \overline{\mathbf{U}}(t)$ utilizzo corrente dei dispositivi

Require: $\overline{X} = \overline{X}(t)$ throughput di sistema corrente

Require: $\overline{R} = \overline{R}(t)$ tempo di risposta di sistema corrente

Ensure: R(**S′**) sia il tempo di risposta stimato nello stato **S′**

1: $N := \overline{X} \times \overline{R}$

2: $\mathcal{C} := \{k \in \mathcal{K} \mid \mathbf{O'}[k] = 1 \;\&\&\; \mathbf{St'}[k] = 0\}$

3: **for all** $k \in C$ **do**

4: $\quad D[k] := D[k, S'[k]] := \frac{RSP[k,S[k]]}{RSP[k,S'[k]]} \times \frac{\overline{U}[k]}{\overline{X}}$

5: **end for**

6: $K := \texttt{LENGHT}(\mathcal{C})$

7: $D_{max} := \max\{D[k] \mid k \in C\}$

8: $D_{tot} := \sum_{k \in \mathcal{C}} D[k]$

9: $D_{ave} := D_{tot}/K$

10: $R^- := \max\{N \times D_{max}, D_{tot} + (N-1) \times D_{ave}\}$

11: $R^+ := D_{tot} + (N-1) \times D_{max}$

12: **return** $(R^+ + R^-)/2$

Utilizzando le leggi appena definite, il tempo di risposta del sistema può essere stimato usando la funzione `ESTIMATERESPT`() (mostrata nell'algoritmo 2) a partire da una configurazione arbitraria **S'**, la configurazione corrente **S** = **S**(t), l'utilizzo corrente dei dispositivi $\overline{\mathbf{U}} = \overline{\mathbf{U}}(t)$, il throughput di sistema $\overline{X} = \overline{X}(t)$ ed il tempo di risposta $\overline{R} = \overline{R}(t)$. La stima è ottenuta utilizzando l'analisi Balanced System Bounds (BSB) [24] sui soli dispositivi allo stato **G0**, escludendo quindi quelli in standby, spenti o in transizione di stato (la scrematura viene effettuata definendo l'insieme $\mathcal{C}$ alla riga 2): essa produce un limite inferiore ed uno superiore (R^+ e R^- rispettivamente) del tempo di risposta del sistema. Il tempo di risposta stimato viene quindi calcolato come la media dei due limiti trovati.

È importante sottolineare che il tempo di risposta ottenuto con la funzione `ESTIMATERESPT`() è solo una stima del tempo di risposta del modello a reti di code. Il tempo di risposta esatto di un modello a reti di code in forma prodotto chiuso può essere calcolato utilizzando l'algoritmo Mean Value Analysis (MVA) [25]. Tuttavia il costo computazionale utilizzando MVA, con K centri di calcolo e N richieste, è $O(NK)$ mentre utilizzando l'analisi BSB è limitato $O(K)$. Non solo quindi i tempi di risposta delle configurazioni possono essere calcolati più velocemente, ma risultati sperimentali svolti in [10] hanno garantito l'efficienza di questa approssimazione, dimostrando risultati simili al caso in cui viene utilizzato MVA.

2.3.3 Caso di sistema eterogeneo

Fino a questo momento si sono ipotizzati i dispositivi presenti all'interno del sistema come omogenei tra loro. In particolare nella descrizione del ciclo di controllo di RISE (sezione 2.3.1), le due procedure `SPEEDUP()` e `SLOWDOWN()` sono in grado di agire su qualunque dispositivo per regolarne il P-State o lo stato globale, andando a modificare il tempo di risposta stimato.

Tuttavia occorre anche considerare il caso in cui i K dispositivi presenti non siano tutti quanti identici tra loro, ma si possano suddividere in B blocchi, ciascuno composto da device omogenei ed atto a processare una specifica parte delle richieste. Si pensi per esempio al caso in cui un datacenter presenta al suo interno un insieme di server che hanno il compito specifico di recuperare dati dallo storage, ed un secondo insieme che si appoggia al precedente che processa effettivamente le richieste. In questo modo si ottiene un sistema eterogeneo che processa le richieste all'istante t grazie alla cooperazione di blocchi di dispositivi con diverse caratteristiche, cioè diversi P-State consentiti, diversi consumi energetici e diverse potenze di calcolo.

In questa nuova configurazione, con B blocchi omogenei formati ciascuno da K_B dispositivi, è conveniente aggiungere alcune ipotesi non formulate nelle sezioni 2.3.1 e 2.3.2.

Innanzitutto le procedure `SPEEDUP()` e `SLOWDOWN()` non sono in grado di eseguire una riconfigurazione completa (modificando tutti i blocchi), ma devono essere eseguite una volta per ciascun blocco da riconfigurare.

In secondo luogo ogni blocco deve avere almeno un dispositivo operativo in ogni istante temporale, dal momento che i vari blocchi cooperano per processare le richieste. La stima del tempo di risposta di sistema $R(\mathbf{S})$ in configurazione $\mathbf{S}$ viene quindi calcolata sommando la stima dei tempi di risposta $R_B(\mathbf{S})$ di ogni blocco C. Il calcolo di $R_B(\mathbf{S})$ avviene utilizzando l'algoritmo 2 già presentato precedentemente.

Infine appare evidente la difficoltà di fissare un intervallo temporale Δt al termine del quale controllare la necessità di un'eventuale riconfigurazione per tutti i blocchi: basti pensare che ciascun blocco ha le proprie esigenze, e non è pensabile una riconfigurazione globale al termine di Δt, proprio perché verrebbe meno l'idea di intervallo temporale variabile e adattabile che si vuole perseguire. Inoltre fissare Δt in modo tale da soddisfare le esi-

genze di un determinato blocco, violerebbe probabilmente le esigenze di altri blocchi.

Si è deciso quindi di utilizzare una soluzione in cui ogni blocco ha un proprio intervallo di riconfigurazione Δt_B in linea con le sue esigenze, e quando questo scade viene innescato un tentativo di riconfigurazione solo su quello specifico blocco (e su tutti quelli per cui Δt_B è scaduto) secondo l'algoritmo di controllo 1. In questo modo si ottiene un maggior numero di riconfigurazioni, ma migliori prestazioni da parte del sistema.

Nel proseguo del documento tutti gli algoritmi presentati e le considerazioni fatte varranno per un sistema omogeneo (salvo diverse indicazioni), dal momento che il sistema multiblocco è una sua estensione secondo le ipotesi presentate in questa sezione.

Definiti il ciclo di controllo di RISE ed il modello di performance per la stima dei tempi di risposta, nel capitolo 3 verranno introdotti gli algoritmi di riconfigurazione veri e propri insieme alle meccaniche di spegnimento ed accensione dei dispositivi del sistema.

Capitolo 3

Implementazione del Modello

In questo capitolo verrà descritta l'implementazione del modello definito nel capitolo 2. Nella sezione 3.1 saranno introdotte le modalità di spegnimento e di accensione dei dispositivi, mentre le sezioni 3.2.1 e 3.2.2 descriveranno rispettivamente gli algoritmi di velocizzazione e rallentamento del sistema. Nella sezione 3.2.3 verrà definito l'algoritmo di compensazione della potenza di calcolo persa a seguito di uno spegnimento, in 3.2.4 sarà descritta la routine di gestione dei dispositivi non operativi ed infine in 3.2.5 sarà stimata la complessità di calcolo richiesta dal sistema.

3.1 Accensioni e spegnimenti

Prima di entrare nello specifico degli algoritmi che formano il corpo di RISE, quelli cioè che riguardano la soluzione vera e propria del problema di ottimizzazione 2.2, è importante soffermarsi sul dettaglio delle accensioni e degli spegnimenti. Come già visto nella sezione 2.3.1, il ciclo di controllo di RISE riesce a valutare se una riconfigurazione è necessaria grazie alle due soglie R_{high} e R_{low}. In particolare, se il tempo di risposta del sistema $\overline{R}(t)$ è al di fuori della regione definita dalle due soglie, viene innescato un meccanismo di diminuzione o incremento della capacità computazionale.

In questo contesto, occorre però specificare il criterio secondo cui sarebbe opportuno cambiare il numero di dispositivi in stato di esecuzione **G0** piuttosto che modificare i P-State di quelli che già lo sono.

Il procedimento seguito nel lavoro di tesi è simile a quello mostrato in [26] e l'idea che ne sta alla base è quella di definire due soglie, una minima ed una massima, per ogni possibile numero di dispositivi operativi (cioè allo stato **G0**) contemporaneamente. Avremo quindi due vettori $S_{min}[K]$ e $S_{max}[K]$, ciascuno di dimensione pari al numero di dispositivi presenti. Nel momento in cui fosse necessaria una riconfigurazione per un sistema con l device attivi, la media a dei loro P-State (indicati ciascuno con $P[k]$) verrebbe confrontata con le due soglie $S_{min}[l]$ e $S_{max}[l]$: nel caso in cui a non si trovi compresa tra esse, avrebbe luogo un'accensione (in caso di superamento della soglia minima), o altrimenti uno spegnimento (va ricordato che i P-State con valori alti sono i meno performanti). Ipotizzando che il numero di device attivi sia $N = 5$, che $S_{min}[5] = 2$ e che $S_{max}[5] = 6$, si avrebbe quindi un'accensione se $a < 2$ od uno spegnimento se $a > 6$. Nella prossima sezione sarà descritto nel dettaglio l'algoritmo per il calcolo delle soglie e le problematiche che esso comporta.

3.1.1 Soglie di Accensione/Spegnimento

Per poter calcolare le soglie di accensione e spegnimento $S_{min}[K]$ e $S_{max}[K]$, viene inizialmente definita una matrice $M(K, P)$ avente righe pari al numero totale di dispositivi K del sistema e colonne pari al numero di P-State P raggiungibili da ciascuno di essi. L'i-esima riga rappresenta il sistema in cui i dispositivi si trovano allo stato **G0**; la j-esima colonna indica invece in quale P-State si trova ciascun dispositivo i attivo. Ne consegue che ogni cella definisce la configurazione di sistema per cui vale che i dispositivi sono operativi al P-State $P(k) = j$.

Per ogni cella sono definiti due valori distinti: il primo, $E(i, j)$, rappre-

		P(k) = 8	P(k) = 7	P(k) = 6	P(k) = 5	P(k) = 4	P(k) = 3	P(k) = 2	P(k) = 1
k = 1	W	5	6	7	10	13	15	18	20
	MHz	400	600	800	1000	1400	1600	1800	2000
k = 2	W	10	12	14	20	26	30	36	40
	MHz	800	1200	1600	2000	2800	3200	3600	4000
k = 3	W	15	18	21	30	39	45	54	60
	MHz	1200	1800	2400	3000	4200	4800	5400	6000

Figura 3.1: Matrice risultante nel caso in cui siano presenti tre processori VIA C7-M da 2GHz

senta l'energia consumata in quella determinata configurazione, mentre il secondo, $P(i, j)$, la potenza di calcolo totale erogata. La matrice è costruita in maniera tale che nell'angolo superiore sinistro si abbia un solo dispositivo attivo al P-State meno performante, mentre nell'angolo opposto tutti i dispositivi operanti al P-State più performante. In figura 3.1 è mostrato un esempio di matrice nel caso di utilizzo di tre processori VIA C7-M da 2GHz.

È possibile definire per ogni cella un terzo parametro che indica la bontà dello stato stesso, ovvero quanto sarebbe conveniente avere il sistema in quella configurazione. Si può quindi definire $goodness(i, j) = P(i, j)/E(i, j)$, crescente al crescere della potenza erogata dal sistema e al decrescere dell'energia consumata.

Alla luce dei parametri introdotti, l'algoritmo 3 si occupa di calcolare le soglie vere e proprie. Esso riceve in ingresso la matrice M e definisce i due vettori $S_{min}[K]$ ed $S_{max}[K]$ (con K pari al numero totale di dispositivi), che conterranno i valori di soglia per ciascun numero di dispositivi operativi nel sistema. Alla riga 4 viene inizializzata la soglia massima nel caso in cui sia attivo un solo dispositivo al valore di P (P-State massimo) per impedire tentativi di spegnimento in questa configurazione. Allo stesso modo alla riga 5 viene inizializzata la soglia minima nel caso vi siano K dispositivi attivi al valore 1 per evitare tentativi di accensione. Dopodiché

Algorithm 3 Algoritmo di calcolo delle soglie

Sia $M(K, P)$ la matrice delle configurazioni di sistema
Sia $S_{min}[K]$ il vettore delle soglie minime per ogni possibile numero di dispositivi attivi
Sia $S_{max}[K]$ il vettore delle soglie massime per ogni possibile numero di dispositivi attivi
$S_{max}[1] = P$
$S_{min}[K] = 1$
for $i = 1$ to $K - 1$ **do**
  for $j = 1$ to $S_{max}[i] - 1$ **do**
    Trova $w \mid P(i+1, w) > P(i, j), \min_j(P(i+1, w) - P(i, j))$
    if $(goodness(i+1, w) > goodness(i, j+1))$ **then**
      $S_{min}[i] = j$
      $S_{max}[i+1] = w$
      break;
    end if
    if $(j == S_{max}[i] - 1)$ **then**
      $S_{min}[i] = 1 + (1/K)$
      $S_{max}[i+1] = w$
    end if
  end for
end for

		P(k) = 8	P(k) = 7	P(k) = 6	P(k) = 5	P(k) = 4	P(k) = 3	P(k) = 2	P(k) = 1
k = 1	W	5	6	7	10	13	15	18	20
	MHz	400	600	800	1000	1400	1600	1800	2000
	goodness	80	100	114,2857	100	107,6923	106,6667	100	100
k = 2	W	10	12	14	20	26	30	36	40
	MHz	800	1200	1600	2000	2800	3200	3600	4000
	goodness	80	100	114,2857	100	107,6923	106,6667	100	100
k = 3	W	15	18	21	30	39	45	54	60
	MHz	1200	1800	2400	3000	4200	4800	5400	6000
	goodness	80	100	114,2857	100	107,6923	106,6667	100	100

$S_{min} = \{4, 5, 1\}$ $S_{max} = \{8, 6, 6\}$

Figura 3.2: Esito dell'algoritmo di calcolo delle soglie nel caso in cui siano presenti tre processori VIA C7-M da 2GHz. In verde sono visualizzati gli stati leciti e in arancione quelli illeciti.

viene scandito ogni elemento di ciascuna riga partendo dal P-State più alto e muovendosi verso destra (supponiamo di analizzare l'elemento (i, j)). Viene quindi confrontato il valore $goodness$ dell'elemento successivo sulla i-esima riga $(i, j + 1)$ con l'elemento con potenza di calcolo più vicina (ma maggiore) sulla riga successiva $(i + 1, w)$. A questo punto se il valore $goodness(i + 1, w)$ è maggiore di $goodness(i, j + 1)$ siamo nella situazione in cui conviene accendere un ulteriore dispositivo piuttosto che diminuire il P-State di quelli già attivi (cioè passare alla riga successiva). Si fissa quindi la soglia minima dell'i-esima riga al valore *j*, ovvero all'altezza dell'elemento considerato inizialmente oltre il quale non conviene proseguire (riga 10). Si fissa poi la soglia massima della riga *i+1* a w, cioè all'altezza dell'elemento appena considerato nel confronto (riga 11). Infine alla riga 14 viene eseguito un controllo per evitare che si riscontrino errori nel caso in cui non sia conveniente in nessun caso aumentare il numero di dispositivi: in questa situazione si imposta la soglia minima della riga *i* (riga 15) al valore $1 + 1/K$, per forzare l'accensione di un dispositivo solo quando tutti quelli attivi sono al P-State massimo.

In figura 3.2 è possibile osservare il risultato dell'algoritmo di calcolo

delle soglie per il caso mostrato in figura 3.1. Le configurazioni di sistema con sfondo verde sono quelle lecite mentre quelle in arancione le illecite. Le soglie minime sono rappresentate dal P-State minimo lecito per un certo numero di dispositivi, mentre quelle massime dal P-State massimo.

Occorre precisare alcuni punti molto importanti riguardanti il procedimento appena descritto:

- L'algoritmo proposto per il calcolo delle soglie è di tipo greedy: infatti esso accetta sempre i risultati del primo confronto di *goodness* positivo, saltando alla riga successiva senza occuparsi dei possibili benefici che si potrebbero trarre ignorandolo e restando su quella attuale. Questo comportamento è accettabile dal momento che il valore di *goodness* non cambia da una riga all'altra a patto di prendere in considerazione una sola colonna. Ciò implica che gli stati a bassa *goodness* verranno in ogni caso percorsi su una delle righe, riducendo così la negatività della strategia greedy. Inoltre non è nota a priori la potenza di calcolo media necessaria del sistema una volta in esecuzione: ciò implica che non avrebbe molto senso preferire determinate righe piuttosto che altre lasciandogli le configurazioni con *goodness* maggiore.

- Le soglie sono definite supponendo che tutti i dispositivi abbiano lo stesso P-State, tuttavia non è raro che questa situazione non si presenti in contesti reali. Esse di conseguenza spingono ad un comportamento tanto più efficiente quanto più i dispositivi hanno P-State vicini tra loro durante l'esecuzione. Nel lavoro di tesi si è scelto di ipotizzare che ciò avvenga osservando i test numerici del modello, che hanno mostrato come effettivamente esso prediliga P-State molto simili tra i dispositivi per scongiurare l'eventualità di colli di bottiglia.

Le prossime sezioni introdurrano la soluzione euristica per il problema e gli

algoritmi di speed up e di slow down, eseguiti periodicamente dal controllore del sistema, che faranno uso delle soglie di accensione e spegnimento appena definite.

3.2 Soluzione al problema di ottimizzazione

Il problema 2.2 può essere risolto utilizzando la programmazione intera mista, tuttavia una soluzione simile è molto onerosa per quanto riguarda il costo computazionale, come dimostrato in [10]. In questo lavoro di tesi si è scelto quindi di utilizzare una soluzione euristica che fa uso del modello di performance a reti di code descritto nella sezione 2.3.2 per il calcolo delle nuove configurazioni.

L'idea per la risoluzione è la seguente: partendo da una configurazione iniziale **S** = **S**(t), viene prima controllata la necessità di accensione o spegnimento di un nuovo dispositivo, poi in caso non si sia verificata questa ipotesi viene iterativamente calcolata una nuova soluzione **S'** aumentando o diminuendo il P-State di un dispositivo per volta.

In particolare se la media a dei P-State dei dispositivi in uso oltrepassa una delle soglie $S_{min}[K]$ o $S_{max}[K]$ definite alla sezione 3.1, ha luogo un'accensione ($a < S_{min}[i]$, con *i* dispositivi attivi) oppure uno spegnimento ($a > S_{max}[i]$, con i dispositivi attivi). A seguito di uno di questi due eventi non si ha una fase di riconfigurazione dei P-State, bensì si aspetta la fine dell'intervallo di tempo Δt per evitare errori nella stima del tempo di risposta.

Nel caso in cui non abbia avuto luogo un'accensione o uno spegnimento, si possono avere due casi:

- $R(t) > R_{high}$: in questa situazione viene identificato il dispositivo B collo di bottiglia del sistema e ne viene abbassato il P-State $P(B)$, in modo da farlo operare più velocemente. Il processo viene ripetuto

finché $R(t) \leq R_{high}$ oppure quando tutti i dispositivi hanno il P-State pari a 1, cioè quando le loro performance non sono più migliorabili.

- $R(t) < R_{low}$: in questa situazione viene identificato il dispositivo con il minimo impatto sulle performance del sistema e ne viene abbassato il suo P-State, rendendolo più lento ma riducendone anche il consumo energetico. Il processo si ferma quando non sono più presenti dispositivi da rallentare o quando il tempo di risposta stimato supera R_{low}.

Per entrambi i casi appena descritti, il processo di riconfigurazione è guidato dalle stime del modello di performance a reti di code. Esso viene infatti utilizzato per identificare sia il dispositivo collo di bottiglia sia il dispositivo con il minimo impatto sul sistema, nonché per stimare il tempo di risposta del sistema.

3.2.1 Algoritmo di Speed Up

Quando $R(t) > R_{high}$, il tempo di risposta può essere incrementato portando allo stato **G0** un dispositivo che prima non lo era oppure identificando il dispositivo collo di bottiglia del sistema e aumentandone le prestazioni. Prima di modificare i P-State del bottleneck però, viene confrontata la media *a* dei P-State dei dispositivi operativi con il valore soglia $S_{min}[K]$ ($S_{max}[K]$ non è rilevante in questo frangente perché ci stiamo occupando di velocizzare il sistema).

Nel momento in cui $a < S_{min}[i]$ si è in una situazione in cui i P-State dei dispositivi sono troppo alti per garantire un buon consumo energetico. Ha quindi priorità massima l'accensione di un nuovo dispositivo, il cui stato verrà cambiato da **G1** a **G0**. Si è scelto di preferire il passaggio dallo stato di standby perchè richiede tempi minore del passaggio dallo stato **G3**. Tuttavia se non sono presenti dispositivi allo stato **G0**, RISE provvederà ad un

passaggio diretto dallo stato **G3** a **G0** per un altro di essi. Nella sezione 3.2.4 verrà descritto l'algoritmo che si occupa di gestire la quantità di dispositivi in stato **G1** in maniera tale da scongiurare l'eventualità di un'accensione completa, che richiede tempi molto lunghi.

In caso di accensione non viene eseguita nessuna riconfigurazione dei dispositivi in esecuzione, e viene impostato il valore di Δt al tempo necessario per portare il dispositivo appena acceso in stato operativo: questo tempo sarà relativamente breve per un'accensione da standby, mentre sarà maggiore per una dallo stato **G3**. Questo comportamento ha un duplice scopo:

- Evitare che vengano calcolate riconfigurazioni durante la fase transitoria. Le modifiche potrebbero portare a comportamenti inaspettati (come per esempio picchi nel tempo di risposta) perché fatte senza tenere in conto l'entrata nel sistema di un nuovo dispositivo.

- Permettere una nuova riconfigurazione nell'esatto momento in cui il dispositivo diventa operativo. Ciò permette di massimizzare i benefici di questo approccio.

Nel caso invece in cui non si è in una situazione che richiede un'accensione ($a \geq S_{min}[i]$), il tempo di risposta può essere ridotto identificando e rimuovendo il collo di bottiglia per il sistema in ciascuna iterazione. Dalla teoria delle code è noto che il collo di bottiglia è quel dispositivo con il più alto tempo di servizio [22]. Tuttavia deve essere preso in considerazione il nostro obiettivo di garantire il minimo consumo energetico. Questo si traduce nella scelta del dispositivo con il più alto tempo di servizio e il minor consumo energetico. In particolare se **S'** è la configurazione presa in considerazione, sceglieremo il dispositivo k tale per cui il rapporto $D[k, S'[k]]/EN[k, S'[k]]$ è massimo.

I dettagli sono mostrati nell'algoritmo 4. La procedura `SPEEDUP`(**S**,**U**,*X*,*R*)

Algorithm 4 SpeedUp($\mathbf{S}, \mathbf{U}, X, R) \rightarrow \mathbf{S'}$

Require: **S** configurazione corrente di sistema

Require: **U** utilizzo corrente dei dispositivi

Require: X throughput corrente del sistema

Require: R tempo di risposta corrente del sistema, $R > R_{high}$

Ensure: **S′** nuova configurazione di sistema

$N = count\{k \mid k \in \mathcal{K}, O[k] = 1, St[k] = 0\}$

if ($\frac{sum_{k=1}^{K} P[k]}{N} < S_{min}(N)$ && $N < K$ && $reconf = 1$) **then**

  if (S contiene almeno un dispositivo in standby) **then**

    S′ = `ATTIVADISPOSITIVODASTANDBY`(**S**)

    $\Delta t_{next} := T_{attivazione\_da\_standby}$

  else

    S′ = `ACCENDIDISPOSITIVO`(**S**)

    $\Delta t_{next} := T_{accensione}$

  end if

  S″ = `GESTISCIDISPOSITIVI`(**S′**)

  $reconf = 0$

  return S″

end if

Trova $D[k, j]$ per $\{k \in \mathcal{K} \mid O[k] = 1, St[k] = 0\}$, $j \in \mathcal{L}[k]$

S′ := **S**

$\mathcal{C}$:= $\{k \in \mathcal{K} \mid P'[k] > 1, O[k] = 1, St[k] = 0\}$

while $C \neq \emptyset$ **do**

  $B := \arg\max_k \{\frac{D[k,S'[k]]}{EN[k,S'[k]]} \mid k \in C\}$

  $P'[B] := P'[B] - 1$

  $R_{est} := EstimateRespT(\mathbf{S'}, \mathbf{U}, X, R)$

  if ($R_{est} < R_{high}$) **then**

    Break

  end if

  $\mathcal{C}$:= $\{k \in \mathcal{K} \mid P'[k] > 1, O[k] = 1, St[k] = 0\}$

end while

$reconf = 1$

return (**S′**, , $accensione = 0$)

riceve in ingresso la configurazione corrente **S**, l'utilizzo dei dispositivi **U**, il throughput di sistema X e il tempo di risposta $R > R_{high}$ restituendo una nuova configurazione **S'** in cui il tempo di risposta stimato è minore di R_{high}. Inizialmente viene contato il numero N di dispositivi attivi (riga 1). Questo valore viene utilizzato per il confronto della media dei P-State con il valore $S_{min}(N)$, e viene anche controllato che non sia pari al numero totale di dispositivi presenti all'interno del sistema (riga 2). L'ultimo controllo effettuato è valutare se il parametro globale $reconf$ sia pari a 1: $reconf$ è un parametro che serve ad impedire due accensioni/spegnimenti di fila, forzando una semplice riconfigurazione dei P-State quando vale 0. Nel caso in cui la media sia oltre il valore di soglia e sia possibile accendere un ulteriore dispositivo, ciò avviene dando priorità ai dispositivi in standby (righe 3-9). In questa fase viene inoltre definita la lunghezza del prossimo intervallo temporale Δt_{next}. Alla riga 10 viene eseguita la routine di controllo che garantisce che almeno un numero fisso di dispositivi non operativi sia in standby (e quindi pronti per un avvio veloce). Alla riga 11 viene aggiornato il parametro $reconf = 0$ per la logica definita poco fa ed alla riga 12 viene restituita la nuova configurazione. Nella restante parte di codice viene definito l'insieme $\mathcal{C}$ che contiene tutti i dispositivi operativi con P-State maggiore di 1 (e quindi velocizzabili). Per ogni iterazione del costrutto while viene definito il collo di bottiglia *B* e velocizzato. Il ciclo termina quando è stato raggiunto un tempo di risposta stimato minore di R_{high} o quando l'insieme $\mathcal{C}$ è vuoto.

3.2.2 Algoritmo di Slow Down

La procedura di slow down mostrata dall'algoritmo 5 agisce in modo simile a quella di speed up, però con l'obiettivo opposto. Inizialmente viene confrontata la media a dei valori P-State degli *N* dispositivi con la soglia $S_{max}[K]$. Nel caso in cui $a > S_{max}[N]$ si procede quindi al passaggio dallo

Algorithm 5 SlowDown($\mathbf{S}, \mathbf{U}, X, R) \rightarrow (\mathbf{S'})$

Require: **S** configurazione corrente di sistema

Require: **U** utilizzo corrente dei dispositivi

Require: X throughput corrente del sistema

Require: R tempo di risposta corrente del sistema, $R < R_{low}$

Ensure: **S'** nuova configurazione di sistema

1: $N = count\{k \mid k \in \mathcal{K}, O[k] = 1, St[k] = 0\}$
2: **if** $(\frac{sum_{k=1}^{K} P'[k]}{N} > S_{max}(N)$ && $N > 1$ && $reconf = 1)$ **then**
3: $B := \arg\min_k \{\frac{D[k,S[k]]}{EN[k]} \mid O[k] = 1, St[k] = 0\}$
4: **S'** = `PORTADISPOSITIVOINSTANDBY`(B)
5: $\Delta t_{next} := T$
6: `GESTISCIPOTPERSA`(**S'**,$P[B]_{MHz}$)
7: **S''** = `GESTISCIDISPOSITIVI`(**S'**)
8: $reconf = 0$
9: **return** (**S'**)
10: **end if**
11: Calcola $D[k, j]$ per ogni $\{k \in \mathcal{K} \mid O[k] = 1, St[k] = 0\}$, $j \in \mathcal{L}[k]$
12: **S'** := **S**
13: $\mathcal{C}$:= $\{k \in \mathcal{K} \mid P'[k] < P, O[k] = 1, St[k] = 0\}$
14: **while** $C \neq \emptyset$ **do**
15: $B := \arg\min_k \{\frac{D[k,S'[k]+1]}{EN[k,S'[k]+1]} \mid k \in \mathcal{C}\}$
16: $P'[B] := P'[B] + 1$
17: $R_{est} := EstimateRespT(\mathbf{S'}, \mathbf{U}, X, R)$
18: **if** $(R_{est} > R_{max})$ **then**
19: $P'[B] := P'[B] - 1$
20: $\mathcal{C} := \mathcal{C} \backslash \{B\}$
21: **else if** $(P'[B] = P)$ **then**
22: $\mathcal{C} := \mathcal{C} \backslash \{B\}$
23: **end if**
24: **end while**
25: $reconf = 1$
26: **return** (**S'**, $spegnimento = 0$)

stato **G0** allo stato **G1** per il dispositivo con il minor impatto sulle performance del sistema ed il massimo guadagno energetico, ovvero quello per cui il rapporto $D[k, S'[k]]/EN[k, S'[k]]$ è minimo. A seguito dello spegnimento viene impostato l'intervallo successivo Δt ad un valore T (riga 5): più T è basso prima verrà eseguita la successiva riconfigurazione. Alla riga 6 è presente la procedura *gestisciPotPersa*(**S'**, $P[B]_{MHz}$) il cui compito è quello di abbassare i P-State dei dispositivi rimasti attivi in maniera tale da evitare picchi nel tempo di risposta causati dalla perdita di potenza computazione. Il metodo con cui viene eseguita questa routine è descritto nella sezione 3.2.3. Alle righe 7 e 8 vengono gestiti i dispositivi in standby e restituita la nuova configurazione di sistema insieme al flag di avvenuto spegnimento. Nella restante parte dell'algoritmo si ha il ciclo dove in ogni iterazione viene tentato un innalzamento del P-State del dispositivo che meno impatta sul sistema a costo energetico maggiore. Il ciclo termina quando si svuota l'insieme $\mathcal{C}$ dei dispositivi rallentabili. Le modalità con cui vengono eliminati elementi dall'insieme sono:

- Il dispositivo *B* ha raggiunto il P-State più alto di valore *P* (riga 21).
- Rallentare ulteriormente il dispositivo *B* comporterebbe il superamento di R_{high} (riga 19).

3.2.3 Algoritmo di gestione della potenza persa

Come appena visto l'algoritmo di slow down prevede un recupero della potenza di calcolo persa in caso di spegnimento di uno dei dispositivi. Ciò è dovuto a causa del fatto che a seguito dello spegnimento il tempo di risposta del sistema potrebbe subire un picco che supera R_{max}, soprattutto nel caso in cui il dispositivo appena spento avesse un grande impatto sulle performance. Dal momento che un comportamento del genere è cri-

Algorithm 6 GestisciPotPersa(**S**, $PotPersa$) $\Longrightarrow$ **S'**

Require: **S** configurazione corrente di sistema

Require: $PotPersa$ valore numerico che indica la quantità di potenza persa

Require: $RSP[K, P]$ matrice dei consumi dei dispositivi

Ensure: **S'** nuova configurazione di sistema

1: $\mathcal{C}$:= $\{k \in \mathcal{K} \mid P'[k] > 1, O[k] = 1, St[k] = 0\}$
2: $given = 0$
3: **while** $C \neq \emptyset$ **do**
4: **for** $i = k \in \mathcal{C}$ **do**
5: $P[i]$:= $P[i] - 1$
6: $given$:= $given + RSP[i, P(i-1)] - RSP[i, P(i)]$
7: **if** ($given \leq PotPersa - \Delta P$) **then**
8: break;
9: **end if**
10: **end for**
11: $\mathcal{C}$:= $\{k \in \mathcal{K} \mid P'[k] > 1, O[k] = 1, St[k] = 0\}$
12: **end while**
13: **return S**

tico, è importante compensare la perdita almeno fino alla riconfigurazione successiva agendo sui dispositivi rimasti accesi.

L'algoritmo 6 si occupa di questo problema, ricevendo in ingresso la configurazione **S** a seguito dello spegnimento e $PotPersa$, che rappresenta il valore numerico di potenza persa. Restituisce infine una nuova configurazione in cui è stata recuperata (o si è cercato di recuperare) $PotPersa - \Delta P$ potenza di calcolo, con ΔP valore arbitario che varia in base alle esigenze del progettista. Inizialmente viene definito l'insieme $\mathcal{C}$ dei dispositivi attivi (riga 1) e la variabile incrementale $given$ (riga 2). Viene poi eseguito un ciclo nel quale iterativamente tutti i P-State dei dispositivi vengono abbassati in maniera orizzontale: ogni dispositivo vede il proprio P-State abbassato di 1 ed esso non può più essere modificato finché ogni altro dispositivo attivo non ha subito lo stesso procedimento (righe 4-6). Questo garantisce che il sistema non si sbilanci. Ad ogni modifica di un P-State viene aggiornato il valore di potenza di calcolo guadagnata dall'inizio dell'intero procedimento e, se è soddisfatta la condizione $PotPersa - \Delta P$ (riga 7), il ciclo si interrompe. Il ciclo si può inoltre interrompere se l'insieme $\mathcal{C}$ si svuota, ovvero se tutti i dispositivi hanno raggiunto P-State 1 (riga 11).

È importante osservare che questo procedimento rappresenta soltanto una soluzione temporanea in attesa che una vera e propria riconfigurazione abbia luogo. Quindi non è ricercata la perfezione della stima, ma è semplicemente utilizzato un metodo rapido per cercare di non far innalzare eccessivamente il tempo di risposta globale.

3.2.4 Gestione dei dispositivi non operativi

Come ultima procedura viene presentata quella relativa alla gestione dei dispositivi in standby. Essa è critica perché un buon sistema deve essere sempre pronto a reagire ad un picco di richieste. Per questo motivo è necessario che vi siano dei dispositivi pronti all'attivazione immediata dallo

Algorithm 7 GestisciDispositivi(**S**) $\Longrightarrow$ **S′**

Require: **S** configurazione corrente di sistema

Require: n numero di dispositivi richiesti in standby

Ensure: **S′** nuova configurazione di sistema

1: $N_{sb} = -1$
2: **while** ($N_{sb} \neq n$) **do**
3: $\mathcal{A}$:= $\{k \in \mathcal{K} \mid O[k] = 1, St[k] = 1\}$
4: N_{sb} =COUNT($\mathcal{A}$)
5: $\mathcal{B}$:= $\{k \in \mathcal{K} \mid O[k] = 0\}$
6: N_{off} =COUNT($\mathcal{B}$)
7: **if** ($N_{sb} < n$) **then**
8: **if** ($N_{off} > 0$) **then**
9: Porta un dispositivo dallo stato **G3** allo stato **G1**
10: **else**
11: break;
12: **end if**
13: **else if** ($N_{sb} > n$) **then**
14: Porta un dispositivo dallo stato **G1** allo stato **G3**
15: **end if**
16: **end while**
17: **return S**

stato di standby piuttosto che da quello di mechanical off: i tempi tra l'uno e l'altro passaggio possono infatti variare moltissimo tra loro, penalizzando di gran lunga il secondo.

L'algoritmo 7 riceve in ingresso una configurazione di sistema, restituendone una nuova in cui, se possibile, sono presenti esattamente n dispositivi in standby (con n parametro arbitrario proprio del sistema). Esso consiste in un ciclo while che prosegue fintanto che l'obiettivo non viene raggiunto. Tra le righe 2 e 5 vengono contati i dispositivi correnti in standby e spenti. Nel caso in cui il numero di dispositivi in standby sia minore di n e ci sia almeno un dispositivo spento, quest'ultimo viene fatto passare allo stato **G1**. Nel caso invece in cui sono presenti più dispositivi in standby di n, semplicemente uno di questi viene fatto passare allo stato **G3**. Infine nel caso in cui il numero di dispositivi in standby non sia sufficiente, ma non sono presenti dispositivi spenti, l'algoritmo termina.

3.2.5 Considerazioni sulla complessità

È possibile stimare la complessità delle procedure di `SPEEDUP` e `SLOWDOWN` attraverso l'analisi degli algoritmi 4 e 5. Dal momento che la loro struttura è identica, l'analisi varrà per entrambi indifferentemente. Essi sono divisi in due parti, e l'esecuzione di una preclude quella dell'altra.

Analizzando prima il blocco di codice relativo all'accensione o lo spegnimento di un dispositivo, si vede come la complessità massima sia $O(K \times L)$: infatti la funzione di gestione della potenza persa prevede un ciclo di al più $K - 1$ iterazioni eseguito al più L volte (nel caso in cui tutti i $K - 1$ dispositivi passino dallo stato L allo stato 1).

Similmente la seconda parte ha una complessità massima di $O(K \times L)$: presenta infatti la stima del tempo di risposta di uno stato **S'**, il cui costo è dato dal calcolo dei singoli tempi di serivizio stimato $O(K)$. La stima

del tempo di risposta è eseguita al più L volte, nel caso in cui tutti i K dispositivi si spostino dallo stato 1 allo stato L (o viceversa).

Si può quindi concludere che la complessità totale degli algoritmi appena definiti è $O(K \times L)$. Inoltre, dal momento che i diversi P-State supportati dallo standard ACPI raggiungono un numero massimo di sedici, non sarebbe illecito considerare la complessità dipendente principalmente dal numero di dispositivi del sistema K.

Occorre infine ricordare che nel caso questi algoritmi siano eseguiti all'interno di un sistema multiblocco (2.3.3), la complessità sarebbe maggiore. In particolare sarebbe massima nel caso di rincofigurazione di ognuno dei B blocchi presenti. In questa ipotesi verrebbero eseguite al più $\sum_{b=1}^{B} K_b \times L_b$ iterazioni, con K_b pari al numero di dispositivi presenti nel blocco b e P_b pari al numero di P-State ammessi nel blocco b.
Definendo poi $KB_{max} = \arg\max_b\{K_b \times P_b \mid \forall b \in B\}$, la complessità massima totale nel caso di sistema multiblocco sarebbe quindi $O(B \times KB_{max})$.

Nel prossimo capitolo verranno descritti i test numerici effettuati sul sistema RISE appena descritto ed i loro risultati.

Capitolo 4

Risultati

In questo capitolo verrà mostrata l'efficacia di RISE attraverso test numerici eseguiti sul sistema. Nella sezione 4.1 verranno definiti i parametri simulativi utilizzati per il caso di sistemi omogenei e saranno mostrati i relativi risultati che garantiscono la bontà del lavoro proposto sia a livello di singoli algoritmi, sia a livello di architettura completa. Nella sezione 4.2 verranno invece mostrati i principali risultati per il caso di sistema formato da blocchi eterogenei.

4.1 Simulazioni di sistemi omogenei

Nei vari esperimenti eseguiti per il caso di sistema composto da dispositivi omogenei saranno considerati K dispositivi che supportano lo stesso numero P di stati di performance ACPI. Questa semplificazione, non obbligatoria per RISE, permette di limitare il numero di parametri di simulazione. Gli esperimenti saranno eseguiti utilizzando valori di K e P differenti a seconda dello scopo della simulazione. In particolare nel primo esperimento, il cui fine è quello di mostrare l'efficacia degli algoritmi di RISE, saranno utilizzati $K = 20$ processori VIA C7-M. Nelle successive simulazioni para-

metriche saranno invece considerate tutte le possibili combinazioni tra K dispositivi e P stati di performance, con $K \in \{5, 20\}$ e $P \in \{2, 4, 8, 12\}$.

Nel caso dei test parametrici, la velocità di elaborazione $RSP[k, j]$ del dispositivo k nel P-State $j \in \{1, ..., P\}$ è definita come una funzione lineare di j con valore minimo $RSP[k, P] = 0.3$ e massimo $RSP[k, 1] = 1$. Ne consegue che:

$$RSP[k, j] = 1 - \frac{j - 1}{P - 1} \times 0.7$$

Per quanto riguarda invece l'utilizzo di processori VIA C7-M la velocità di elaborazione per ogni P-State è stata scalata partendo dall'intervallo reale $[400, 2000]$ MHz (mostrato in figura 2.3) in modo da ricoprire proporzionalmente il range $[0.2, 1]$. È stato associato il P-State 1 (con velocità di elaborazione pari a 2GHz) a $RSP[k, 1] = 1$ ed il P-State 8 (con velocità di elaborazione pari a 400MHz) a $RSP[k, 8] = 0.2$.

Il consumo energetico $EN[k, j]$ per i test parametrici è normalizzato nell'intervallo $[0.4, 1]$ per ottenere risultati assoluti che non siano dipendenti da valori di consumo energetico di uno specifico processore. $EN[k, j]$ è stata definita come una funzione lineare di j con $EN[k, 1] = 1$ e $EN[k, P] = 0.4$:

$$EN[k, j] = 1 - \frac{j - 1}{P - 1} \times 0.6$$

L'assunzione che il rapporto tra velocità di elaborazione e consumo energetico sia lineare, è consistente con i valori d'esempio mostrati nelle figure 2.2 e 2.3. Come $RSP[k, j]$, anche $EN[k, j]$ per la simulazione con i processori VIA C7-M è stato scalato dall'intervallo reale $[5, 20]$ W fino al range $[0.25, 1]$, con $EN[k, 1] = 1$ (consumo reale 20W) e $EN[k, 8] = 0.25$ (consumo reale 5W).

Per ogni esperimento è stata eseguita una simulazione a tempo discreto con $t = 600$ passi. Il ciclo di controllo di RISE (algoritmo 1) è eseguito ogni $\Delta t = 5$ passi se non sono avvenute accensioni allo step precedente, altrimenti viene eseguito non appena l'ultima accensione è portata a termi-

ne (cioè quando il dispositivo diventa operativo). Il tempo necessario per un'accensione (così come per uno spegnimento) è un parametro variabile. All'interno degli esperimenti il tempo di transizione dallo stato **G3** a **G0** o viceversa è pari $\Delta t_{acc} = 5$, mentre quello per le transizioni da **G1** a **G0** (riattivazione da standby) o viceversa è $\Delta t_{attsb} = 1$. Il tempo necessario invece per passare da **G1** a **G3** o viceversa, dal momento che richiede il passaggio forzato per **G0** in entrambi i casi, è pari alla somma di Δt_{acc} e Δt_{attsb}.

I costi energetici durante le fasi transitorie sono anch'essi parametri variabili. Per comodità e correttezza sono stati fissati a valori verosimili:

- Per il consumo in fase di spegnimento si è deciso di utilizzare il valore $EN[k, j]$, con j pari al P-State in cui si trovava il dispositivo k appena prima dello spegnimento. Nel caso in cui il cambio di stato abbia inizio da **G1**, viene utilizzato il valore di consumo energetico dello stato di standby del dispositivo $EN_{sb} = 0.1$.

- Per il consumo in fase di accensione, è stato utilizzato un valore prefissato $EN_{acc} = 0.5$.

Le transizioni sono tracciate dal sistema attraverso l'utilizzo di una matrice di transizione che per ogni dispositivo indica il tipo di transizione che sta effettuando, la sua durata ed il suo costo istantaneo.

Per l'implementazione dell'algoritmo di gestione dei dispositivi non operativi si è scelto di utilizzare il valore obiettivo $K_{sb} = 2$: nel momento in cui il sistema si trova al di sopra o al di sotto di K_{sb}, l'invocazione del processo riporta il numero di dispositivi in standby a due, secondo le modalità dell'algoritmo 7.

Nelle simulazioni è stato utilizzato un numero di richieste al sistema $N(t)$ variabile nel tempo. Sono state considerate varie curve $N(t)$ e per semplicità sono tutte definite come somme di sinusoidi con valore medio pari a 100 (un valore ragionevole per essere gestito per ogni K e P). Le

differenze tra le varie curve considerate si riscontrano nell'ampiezza delle sinusoidi e nella velocità con cui crescono e decrescono. Le singole curve utilizzate negli esperimenti saranno comunque mostrate all'interno della loro descrizione.

È stato utilizzato un tempo di servizio aggregato $D(t)$, calcolato per ogni istante di simulazione t, con valori distribuiti uniformemente nell'intervallo $[0.2, 1]$; ciò è servito a garantire una domanda di servizio variabile. Per rendere i valori dei tempi di servizio più realistici, essi hanno subito un doppio filtraggio attraverso l'utilizzo di medie mobili con intervallo pari a 10. Per calcolare i tempi di servizio dei singoli dispositivi $D[k](t)$ al tempo t si è scelto di utilizzare un load balancer che ripartisce equamente il tempo di servizio aggregato $D(t)$ al tempo t tra tutti i dispositivi attivi.

Per ogni esperimento è stata definita una soglia R_{max} tale per cui esiste una configurazione $\mathbf{S}(t)$ per ogni t che risolva il problema di ottimizzazione $\mathcal{P}(R_{max})$. Per ottenere questo risultato si è eseguito l'algoritmo MVA per calcolare R_{max} nell'istante di tempo t in cui il numero di richieste $N(t)$ è massimo, ipotizzando tutti i dispositivi attivi al P-State P più lento. Come conseguenza si osserva che all'aumentare dei dispositivi K presenti nel sistema (a parità di numero di richieste $N(t)$) la soglia R_{max} si riduce proporzionalmente. Sono state poi definite le due soglie di controllo $R_{high} = R_{max} \times 0.9$ e $R_{high} = R_{min} \times 0.8$.

RISE agisce online e calcola una nuova configurazione di sistema in ogni istante t considerando solamente la configurazione all'istante precedente $t-1$ e il numero di richieste attuali $N(t)$. Il tempo di risposta osservato $\widehat{R}(t)$ all'istante t è calcolato in due fasi: prima viene utilizzato l'algoritmo MVA sul modello a reti di code per avere il valore preciso, poi questo viene moltiplicato per un numero casuale distribuito uniformemente su $[0.9, 1.1]$. Questa accortezza è utilizzata per simulare il fatto che nella realtà il modello a reti di code non calcolerà mai il tempo di risposta attuale corretto, in

quanto il sistema non raccoglie informazioni del tutto complete.

Tutti gli algoritmi presentati nel capitolo 3 sono stati implementati in GNU Octave [27], un linguaggio interpretato per il calcolo numerico. I test sono stati effettuati mediante l'utilizzo di un processore Intel Core 2 Duo T7100 da 1.8GHz con 2GB di RAM. Il sistema operativo utilizzato è Ubuntu 12.10, una distribuzione che si basa sul kernel Linux 3.2.0. La versione utilizzata di GNU Octave è la 3.2.3.

4.1.1 Impatto degli algoritmi

Lo scopo di questa prima serie di simulazioni è quello di mostrare l'efficacia dei singoli algoritmi introdotti nel capitolo 3. In questa fase è stato utilizzato un numero fisso $K = 20$ di processori VIA C7-M, la cui tabella prestazionale è mostrata in figura 2.3. Il numero di P-State presente nelle simulazioni è pari a quelli supportati da questo tipo di processore, cioè $P = 8$.

Per far risaltare al meglio l'utilità degli algoritmi vengono mostrati i risultati secondo un approccio incrementale: viene inizialmente considerata l'analisi del sistema che non implementa alcuna funzione relativa al risparmio energetico propria di RISE, e successivamente viene analizzata la sua variazione con l'aggiunta di uno specifico algoritmo.

In figura 4.1 è mostrato il caso in cui il sistema non esegue riconfigurazioni. Nella prima parte è mostrata la variazione del tempo di risposta $R(t)$ (linea blu) e della sua media mobile (linea verde). Nella seconda parte è mostrato il consumo energetico in ogni istante, mentre nella terza la curva relativa al numero di richieste $N(t)$. In questa simulazione i processori non abbandonano mai il P-State 1 e quindi il consumo energetico è massimo. Come si vede chiaramente dal grafico il tempo di risposta non è controllato e resta quindi molto al di sotto di R_{max}, fluttuando in relazione al numero di richieste $N(t)$: come ci si aspetta, al crescere di $N(t)$, il tempo di risposta

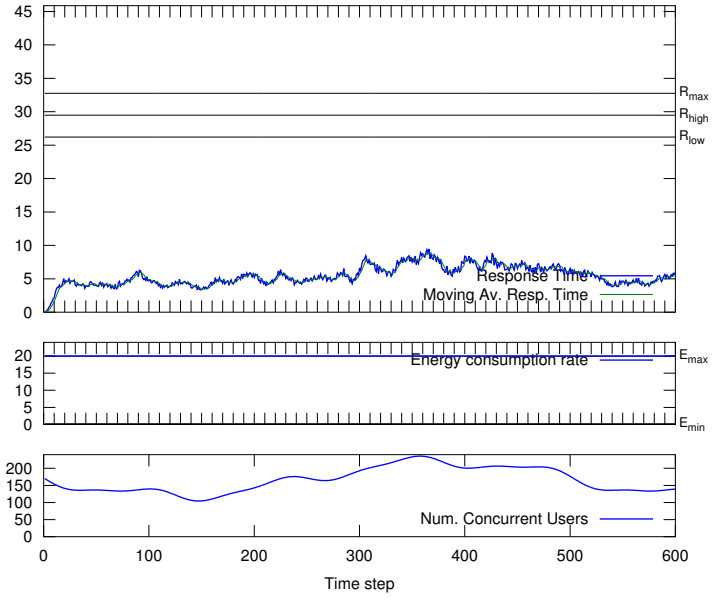

Figura 4.1: Tempo di risposta del sistema che non implementa algoritmi RISE. Consumo energetico totale $E_{tot} = 12000$

del sistema $R(t)$ cresce in maniera del tutto simile.

Riconfigurazioni

In questa fase viene garantita al sistema la capacità di eseguire riconfigurazioni dei P-State dei dispositivi attivi, senza però l'utilizzo di accensioni e spegnimenti. Come mostrato dai risultati in figura 4.2, il risparmio energetico è estremamente elevato rispetto al caso precedente: i processori tendono infatti a vedere decrementato il proprio P-State quando non è necessaria un'alta potenza di calcolo per mantenere $R(t)$ al di sotto di R_{high}. Si nota come la curva $R(t)$ non riesce tuttavia a rimanere all'interno delle due soglie R_{high} e R_{low}, ma tende a restare al di sotto di quest'ultima: ciò è dovuto al fatto che nel momento in cui il numero di richieste $N(t)$ è basso, i 20 processori al P-State più lento erogano comunque una potenza di calcolo maggiore di quella necessaria.

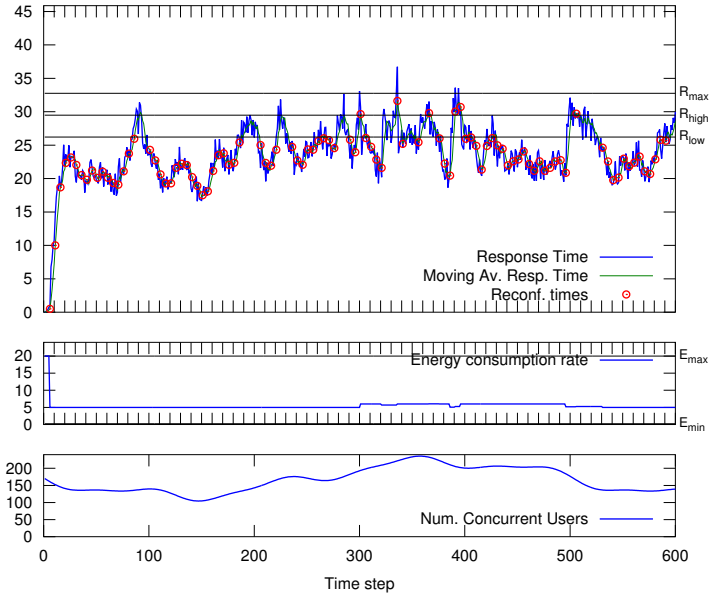

Figura 4.2: Tempo di risposta del sistema che implementa solo riconfigurazioni senza accensioni e spegnimenti. Consumo energetico totale $E_{tot} = 3265.75$

Accensioni e spegnimenti

In questa fase viene garantita al sistema la capacità di riconfigurarsi utilizzando anche le accensioni e gli spegnimenti. Ciò che non è ancora permesso è utilizzare gli algoritmi di gestione della potenza di calcolo persa a seguito di uno spegnimento e di gestione dei dispositivi non operativi. È importante sottolineare che i dispositivi che subiscono uno spegnimento non passano allo stato **G3**, ma allo stato **G1**: questa precauzione è presa per evitare che i lunghi tempi di riaccensione portino $R(t)$ oltre la soglia massima per troppo tempo in caso di crescita repentina di $N(t)$.

In figura 4.3 sono mostrati i risultati della simulazione. È subito evidente come grazie alle accensioni e agli spegnimenti la curva del tempo di risposta $R(t)$ riesca a portarsi per molti più istanti all'interno dell'intervallo $[R_{low}, R_{high}]$ dopo una prima fase di assestamento. Si nota anche come questa maggiore vicinanza comporti più violazioni della soglia massima R_{max} rispetto al caso precedente, garantendo al contempo un consumo energeti-

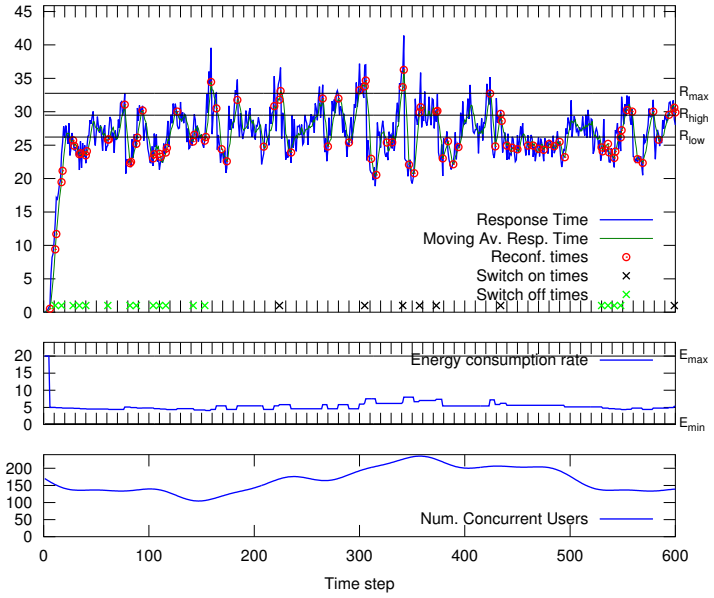

Figura 4.3: Tempo di risposta del sistema che implementa accensioni e spegnimenti. Consumo energetico totale $E_{tot} = 3221.55$

co inferiore. Come è lecito aspettarsi le violazioni di R_{max} sono concentrate nell'intervallo temporale in cui $N(t)$ è crescente, così come le accensioni dei dispositivi.

Gestione della potenza di calcolo

Dando al sistema la possibilità di compensare la potenza di calcolo persa dopo uno spegnimento, si ottiene il risultato mostrato in figura 4.4 molto simile al precedente. Va tuttavia notato che negli istanti di tempo immediatamente successivi ad uno spegnimento si assiste ad un calo del tempo di risposta $R(t)$, mentre nel caso precedente si assisteva ad un incremento. Questo fenomeno è ben visibile nei momenti in cui lo spegnimento avviene mentre il numero di richieste $N(t)$ è crescente, ovvero quelli in cui è più pericoloso spegnere un dispositivo: confrontando i grafici si può notare come viene chiaramente rimosso un picco a circa un terzo del tempo di simulazione.

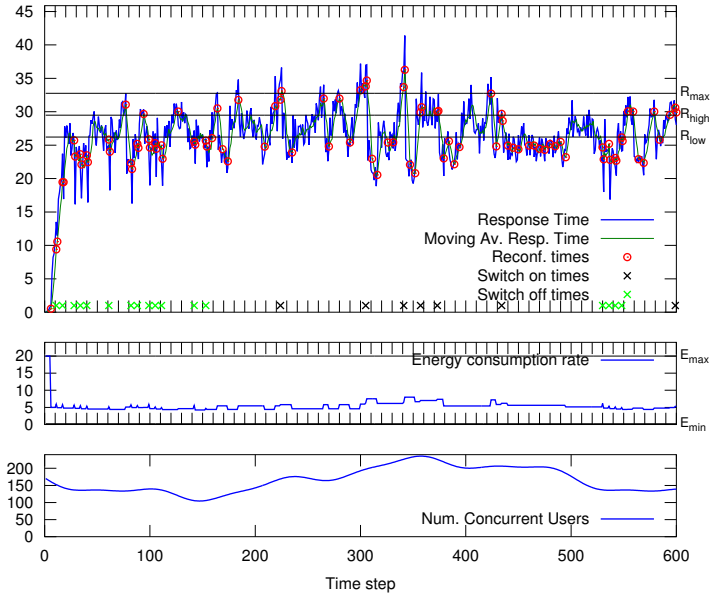

Figura 4.4: Tempo di risposta del sistema che implementa la gestione della potenza di calcolo persa. Consumo energetico totale $E_{tot} = 3234.3$

Gestione dei dispositivi non operativi

Aggiungendo quest'ultimo algoritmo il sistema RISE è ora completo. Come si nota confrontando i risultati dell'esperimento in figura 4.5 con quelli del precedente, il tempo di risposta del sistema non cambia minimamente. Quello che si osserva è però un grande calo del consumo energetico. Questo è dovuto al fatto che si è scelto di mantenere solo due dispositivi in standby contemporaneamente: nel caso precedente invece, ogni dispositivo che subiva uno spegnimento veniva portato allo stato **G1** per poterlo riattivare velocemente in caso di bisogno. Ora questa esigenza non è più necessaria perché RISE controlla il numero di dispositivi in standby a seguito di ogni riconfigurazione, e quindi molti dispositivi possono essere portati allo stato **G3** senza correre il rischio di non poter far fronte ad eventuali emergenze.

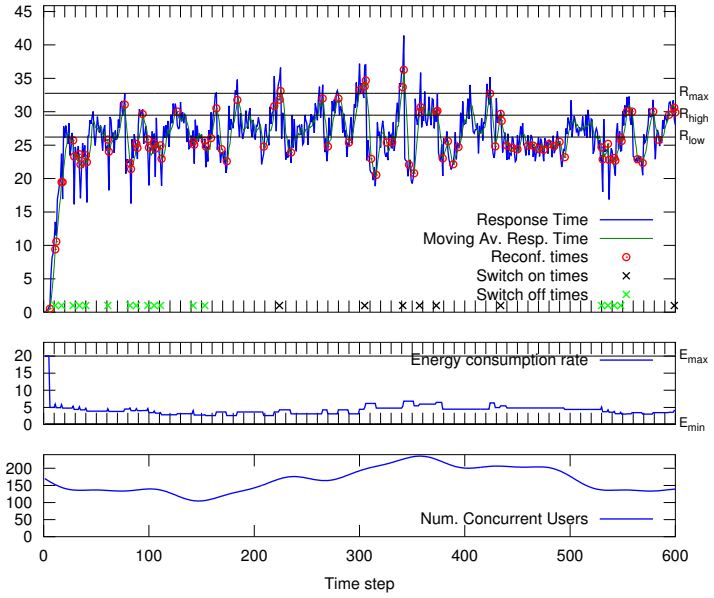

Figura 4.5: Tempo di risposta del sistema completo. Consumo energetico totale $E_{tot} = 2560.05$

4.1.2 Considerazioni

Come mostrato dai risultati delle simulazioni precedenti, il risparmio energetico ottenuto grazie all'utilizzo di RISE è enorme. In figura 4.6 sono mostrati due grafici: il primo mostra come l'introduzione della possibilità di riconfigurare i dispositivi porti ad un risparmio energetico di oltre il 70% per il caso simulato, mentre il secondo fa risaltare come l'introduzione delle accensioni e degli spegnimenti porti ad un ulteriore risparmio di circa il 20%. In particolare l'introduzione della sola possibilità di accendere e spegnere i dispositivi abbatte il consumo energetico operativo (cioè quello dei processori in esecuzione), aumentando però il consumo di standby (cioè quello causato dai processori in standby). Grazie all'algoritmo di gestione dei dispositivi si nota però come il consumo di standby venga ridotto, portando infine ad un consumo globale ottimo.

All'interno della figura 4.7 è mostrato il grafico relativo al numero di violazioni normalizzato e alla percentuale di superamento di R_{max} per vio-

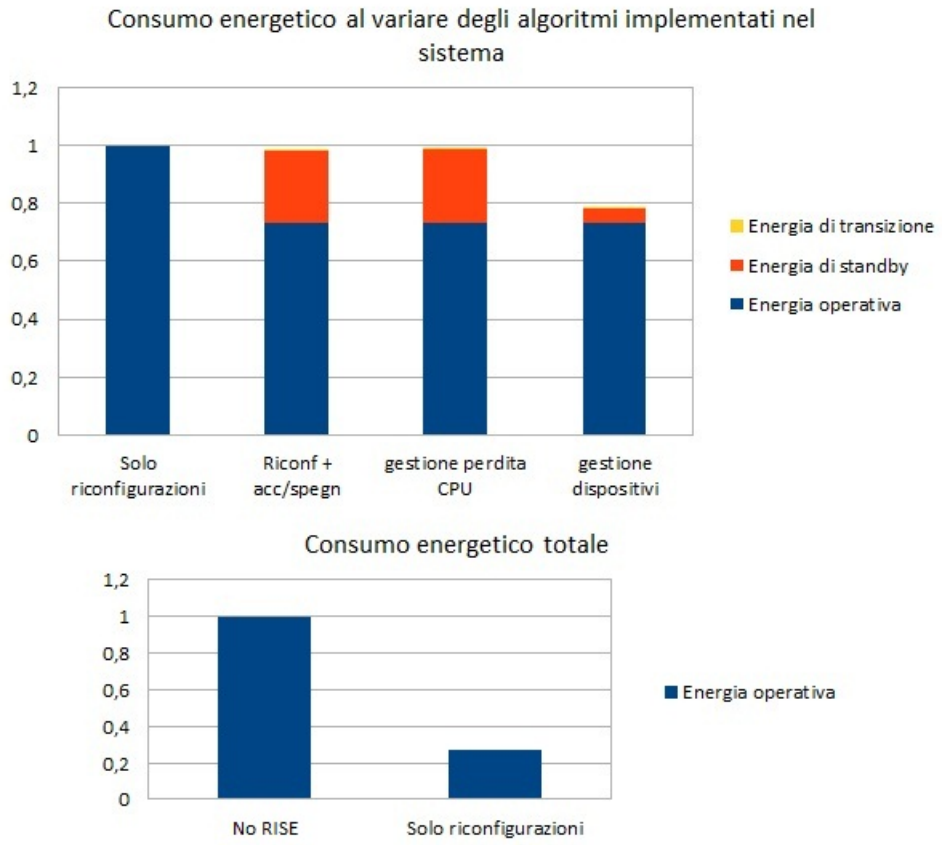

Figura 4.6: Confronto dei consumi energetici con varie implementazioni degli algoritmi

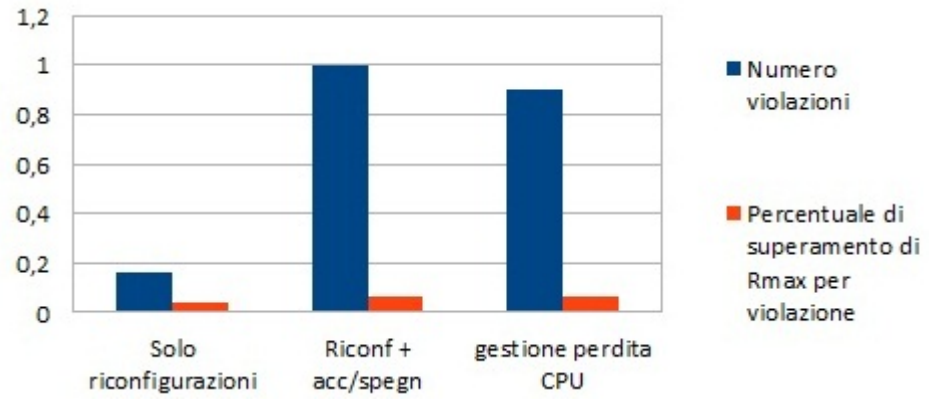

Figura 4.7: Numero di violazioni al variare degli algoritmi implementati nel sistema

lazione. Nel caso in cui il sistema può accendere e spegnere i dispositivi notiamo un aumento delle violazioni rispetto al caso delle sole riconfigurazioni: ciò, come già detto, è dovuto al fatto che grazie agli spegnimenti e alle accensioni il sistema riesce a mantenere il proprio tempo di risposta molto più vicino a R_{max}, ed è quindi anche più probabile che una variazione di $N(t)$ comporti il suo superamento. Va anche notato che l'introduzione della gestione della potenza di calcolo persa riesce in questo caso a diminuire di circa il 10% il numero di violazioni, riducendo i picchi successivi agli spegnimenti. Infine si osserva che la percentuale di superamento di R_{max} per violazione presenta differenze minime tra un caso e l'altro: questo indica che il sistema reagisce sempre con la stessa prontezza ad una violazione, indipendentemente dal metodo utilizzato per diminuire $R(t)$.

4.1.3 Test parametrici

In questa sezione verranno mostrati i risultati relativi ai test parametrici eseguiti. Le prime simulazioni saranno mirate a mostrare l'impatto sul sistema dell'algoritmo di sogliatura delle accensioni e degli spegnimenti. Verrà in particolare analizzata la situazione in cui l'algoritmo è attivo ed una in cui le soglie sono fisse per qualunque numero di dispositivi attivi. Le simulazioni finali avranno invece come scopo quello di mostrare la versatilità di RISE e alcuni comportamenti peculiari riscontrati. Come già detto in questi esperimenti RSP e EN sono stati definiti come funzioni lineari.

Algoritmo di sogliatura

Saranno ora mostrati i risultati relativi all'utilizzo dell'algoritmo che genera le soglie $R_{min}[i]$ e $R_{max}[i]$ per ogni numero di dispositivi attivi $i \in \{1, .., K\}$, definito nella sezione 3.1.1. In particolare verranno confrontati i casi in cui l'algoritmo è normalmente attivo all'interno del sistema e quello in cui le soglie sono fisse. Dal momento che nel caso di utilizzo dell'algo-

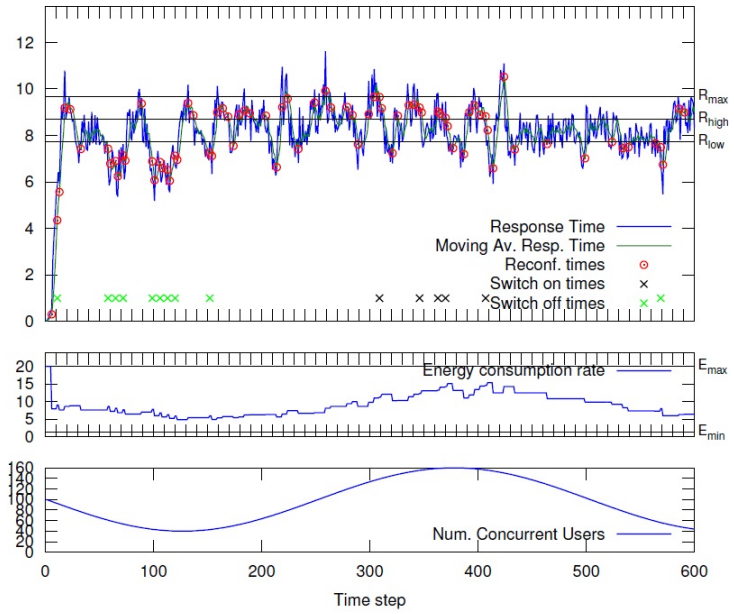

Figura 4.8: Risultati della simulazione nel caso di utilizzo di soglie fisse. Il consumo energetico totale è pari a $E_{sf} = 5397.96$.

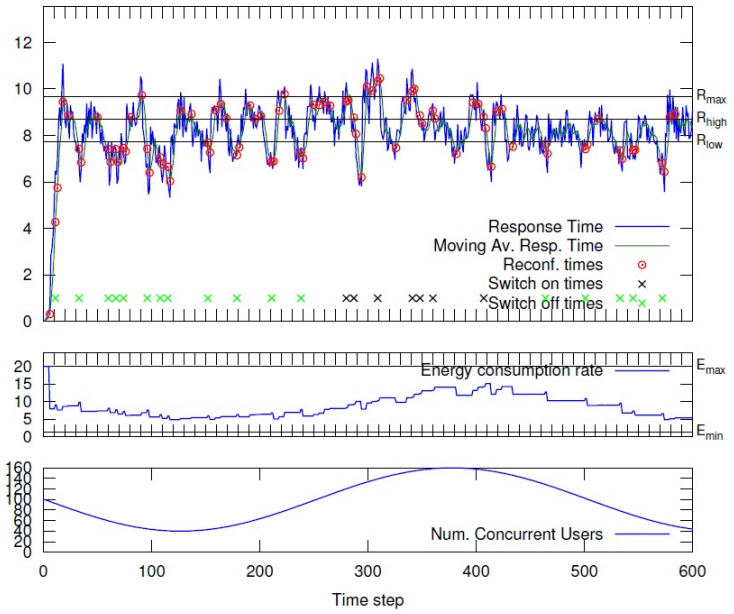

Figura 4.9: Risultati della simulazione nel caso di utilizzo dell'algoritmo di sogliatura. Il consumo energetico totale è pari a $E_{alg} = 5193.5$.

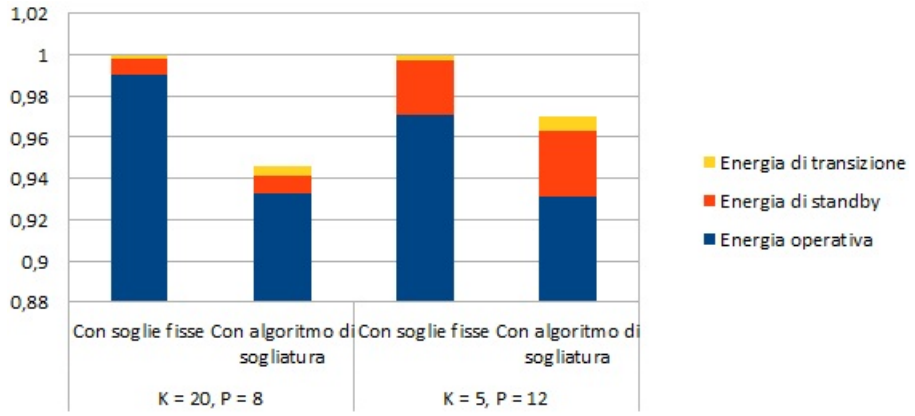

Figura 4.10: Confronto tra i consumi energetici in caso di utilizzo dell'algoritmo di sogliatura e in caso di utilizzo di soglie fisse.

ritmo di sogliatura esse sono calcolate considerando principalmente fattori di consumo energetico, capita spesso di avere valori di i tali per cui $R_{min}[i] = R_{max[i]}$, cioè una sola configurazione valida per i dispositivi attivi. Si è scelto quindi per il caso delle soglie fisse di impostare la soglia minima come $R_{min}[i] = 2 \, \forall i \in \{1, .., K\}$ e quella massima come $R_{max}[i] = P - 1$ $\forall i \in \{1, .., K\}$: così facendo si è creato un sistema in cui le accensioni e gli spegnimenti sono utilizzati il meno possibile, ovvero una sogliatura il più differente possibile da quella generata dall'algoritmo (e più adatta al confronto).

Sono state eseguite simulazioni con tutte le combinazioni di K e P, dove $K \in \{5, 20\}$ e $P \in \{4, 8, 12\}$. Nelle figure 4.8 e 4.9 si osserva a titolo di esempio il confronto delle esecuzioni nel caso di presenza e assenza dell'algoritmo con $K = 20$ dispositivi e $P = 12$ P-State. Si può notare come il costo energetico totale aumenti nel caso in cui vengono utilizzate le soglie fisse. Si nota anche come i tempi di risposta nel caso di utilizzo dell'algoritmo subiscano variazioni più brusche, fenomeno dovuto alla maggior quantità di accensioni e spegnimenti che avvengono nel sistema. Questo comportamento è risultato tipico per tutte le simulazioni effettuate.

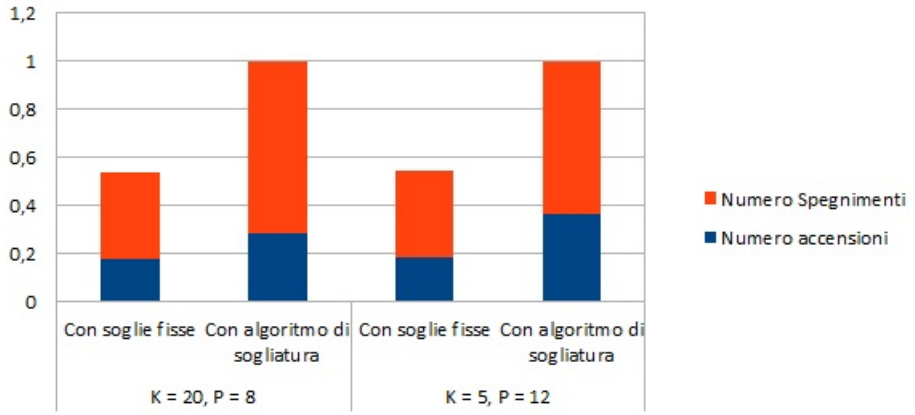

Figura 4.11: Confronto tra il numero di accensioni e spegnimenti in caso di utilizzo dell'algoritmo di sogliatura e in caso di utilizzo di soglie fisse.

In figura 4.10 è mostrato il confronto tra i consumi energetici nei due casi confrontati, sia per ($K = 5, P = 12$) che per ($K = 20, P = 8$). In entrambi i casi l'utilizzo dell'algoritmo di sogliatura ha portato a benefici energetici. Si nota come ci aspetta una consistente riduzione dei consumi operativi e un raddoppio dei consumi di transizione: infatti anche il numero di accensioni e spegnimenti in entrambi i casi è raddoppiato nel caso di utilizzo dell'algoritmo, come mostrato in figura 4.11.

Test di versatilità

I risultati in questa sezione hanno lo scopo di mostrare le variazioni delle prestazioni di RISE al variare del numero di dispositivi e dei P-State ammissibili. In particolare nelle figure 4.12 e 4.13 sono mostrati i risultati delle simulazioni con tutte le combinazioni di K e P, dove $K \in \{5, 20\}$ e $P \in \{4, 8, 12\}$. I parametri di valutazione scelti sono stati quattro:

- L'energia totale consumata, comprensiva di energia operativa, di standby e di transizione. Come si può notare per entrambi i grafici essa tende a decrescere al crescere del numero di P-State ammissibili: ciò è dovuto al fatto che con un maggior numero di P-State il sistema è in

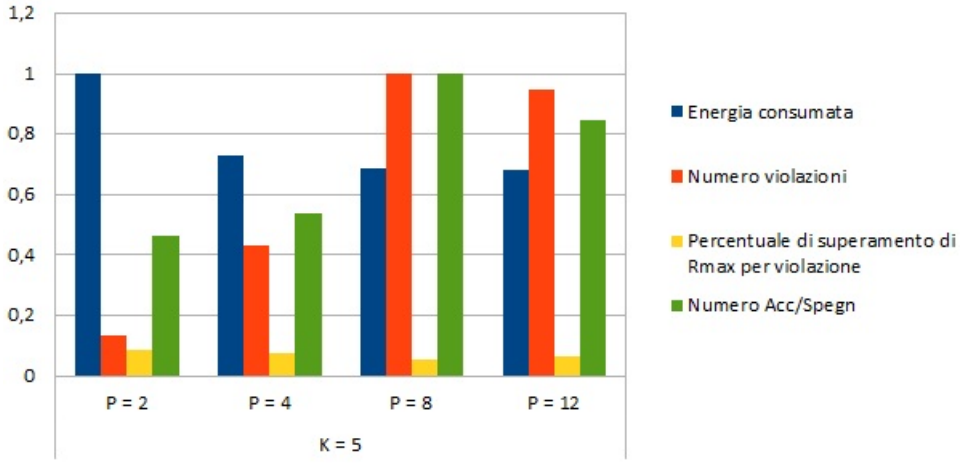

Figura 4.12: Risultati delle simulazioni con $K = 5$ dispositivi al variare del numero di P-State P.

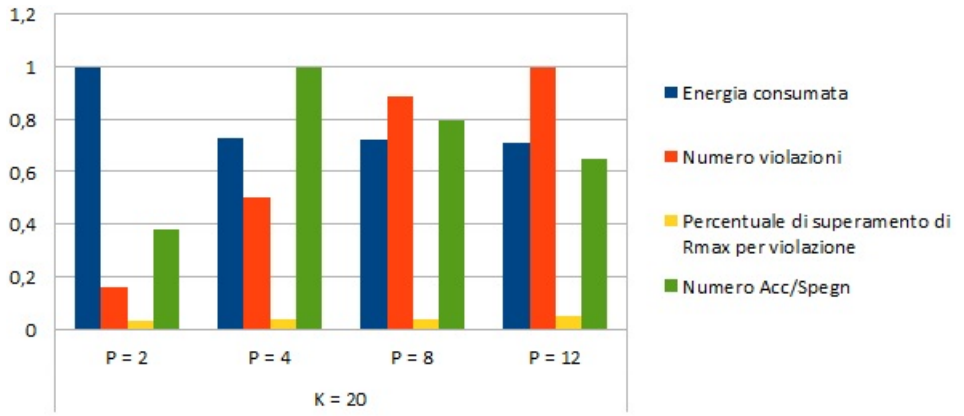

Figura 4.13: Risultati delle simulazioni con $K = 20$ dispositivi al variare del numero di P-State P.

grado di adattare al meglio i propri dispositivi e quindi di abbassare maggiormente i consumi, mantenendo il tempo di risposta all'interno dell'intervallo $[R_{high}, R_{low}]$ al meglio.

- Il numero di violazioni, che tende a crescere all'aumentare del numero di P-State ammissibili per lo stesso motivo appena descritto per l'energia totale: infatti l'algoritmo di slowdown è maggiormente frenato durante la fase di innalzamento dei P-State dei dispositivi in caso questi siano pochi, poiché è più probabile che il tempo di risposta stimato superi la soglia massima R_{high} (ricordiamo che dal momento che $RSP[k, j]$ è una variabile lineare dipendente da $j \in \{1, .., P\}$, per P piccoli la variazione tra $RSP[k, j]$ e $RSP[k, j + 1]$ è maggiore).

- Percentuale di superamento di R_{max} per violazione. Il valore tende a restare costante al variare di P, mentre subisce lievi variazioni al variare di K (per $K = 5$ si assesta attorno all'8%, mentre per $K = 20$ si assesta attorno al 5%). Questo, come già detto in precedenza, è sintomo del fatto che il sistema reagisce sempre con la stessa velocità indipendentemente dall'ampiezza del picco di superamento di R_{max}.

- Il numero di accensioni e spegnimenti. Si nota in entrambi i grafici come il suo valore sia crescente fino ad un determinato valore di P, e da quel momento in poi decresca. Il risultato si può spiegare pensando che nel caso di pochi P-State disponibili, il sistema faccia più fatica a variare il numero di dispositivi attivi poiché non riuscirebbe a compensare il guadagno (o la perdita) di potenza modificando i limitati P-State dei dispositivi attivi. Aumentando P si arriva prima o poi ad un caso in cui questa possibilità non è più preclusa (all'altezza del picco) e da quel momento in poi si osserva una decrescita delle accensioni e degli spegnimenti causata dal fatto che il sistema riesce a modificare le sue prestazioni contando maggiormente sulle ricon-

figurazioni (possibili proprio per il fatto che il numero di P-State è sempre maggiore).

4.2 Simulazioni per sistemi eterogenei

In questa sezione verranno mostrati i risultati delle simulazioni effettuate nel caso di un sistema composto da più blocchi eterogenei tra loro. Alla luce delle ipotesi formulate nella sezione 2.3.3 per questo tipo di sistemi,verrà ora descritto in che modo varino i parametri simulativi rispetto al caso di simulazione di sistemi omogenei. Nell'eventualità in cui un parametro non vari rispetto al caso relativo ai sistemi omogenei, esso sarà omesso dalla seguente descrizione.

Innanzitutto le simulazioni sono eseguite utilizzando $K = 20$ dipositivi, divisi ciascuna volta in un numero diverso di blocchi B, con $B \in \{1, 2, 5, 10\}$. Ogni blocco contiene al suo interno un numero di dispositivi K_b con $b \in B$, tale per cui $K_b \geq 1$ e $K_b \leq 20$.

Ad ogni blocco è assegnato un numero di P-State ammessi dai suoi dispositivi P_b con $b \in B$, tale per cui $6 \leq P_b \leq 10$. All'interno della stessa simulazione ogni blocco ha un valore P_b differente dagli altri, tranne per il caso con $B = 10$, in cui ogni valore è stato ripetuto due volte.

$RSP[k, j]$ e $EN[k, j]$ sono stati definiti per ogni $k \in K_b$ come funzioni lineari di $j \in \{1, ..., P_b\}$ come già visto per il caso di sistema omogeneo. Ne consegue che la velocità di elaborazione e i consumi istantanei dei dispositivi variano da blocco a blocco.

Il ciclo di controllo di RISE (algoritmo 1) è eseguito ogniqualvolta almeno un Δt_b relativo ad un blocco $b \in B$ si esaurisce: in questo contesto viene eseguita la riconfigurazione, se necessario, dei soli blocchi per cui vale $\Delta t_b = 0$. Al termine della riconfigurazione, ad ogni Δt_b con valore pari a zero viene riassegnato il valore di default $\Delta t_b = 5$. Per quanto riguarda

i tempi necessari per le accensioni e gli spegnimenti, così come per i passaggi in standby, valgono gli stessi parametri utilizzati nel caso di sistemi omogenei (si è scelto di lasciare questi valori invariati tra tutti i blocchi per mantenere ridotto il numero di variabili da considerare).

Per l'implementazione dell'algoritmo di gestione dei dispositivi non operativi si è scelto questa volta il valore obiettivo $K_{sb} = 1$ per permettere anche ai blocchi con un basso numero di dispositivi K_b di eseguire spegnimenti verso lo stato **G3**.

È stato utilizzato questa volta un tempo di servizio aggregato $D(t, b)$, diverso per ogni blocco $b \in B$ in ogni istante temporale t, definito secondo le stesse modalità del caso di sistemi omogenei. Il meccanismo di load balancing viene applicato questa volta all'interno di ogni blocco (a partire da $D(t, b)$) piuttosto che all'intero sistema.

Infine il tempo di risposta osservato $\widehat{R}(t)$ all'istante t è calcolato in tre fasi: prima viene utilizzato l'algoritmo MVA sul modello a reti di code per calcolare il tempo di risposta $\widehat{R_b}(t)$ relativo a ciascun blocco. Poi viene calcolato il tempo di risposta totale $\widehat{R}(t) = \sum_{b=1}^{B} \widehat{R_b}(t)$ e infine questo viene moltiplicato per un numero casuale distribuito uniformemente su [0.9, 1].

4.2.1 Test Parametrici

Nelle figure 4.14 e 4.15 sono mostrati i risultati delle simulazioni utilizzando $K = 20$ dispositivi nei due casi in cui sono presenti $B = 10$ e $B = 1$ blocchi. Osservando la variazione del tempo di risposta del sistema si nota immediatamente come avendo un maggior numero di blocchi il sistema riesca a mantenersi molto meglio nella zona compresa tra le soglie R_{high} e R_{low}. Questo comportamento è dovuto al fatto che può essere modificata più volte la configurazione del sistema ad ogni iterazione del ciclo di controllo (al più una volta per blocco), ed è quindi più semplice renderla adatta al numero di richieste in quel determinato istante. Come esempio si

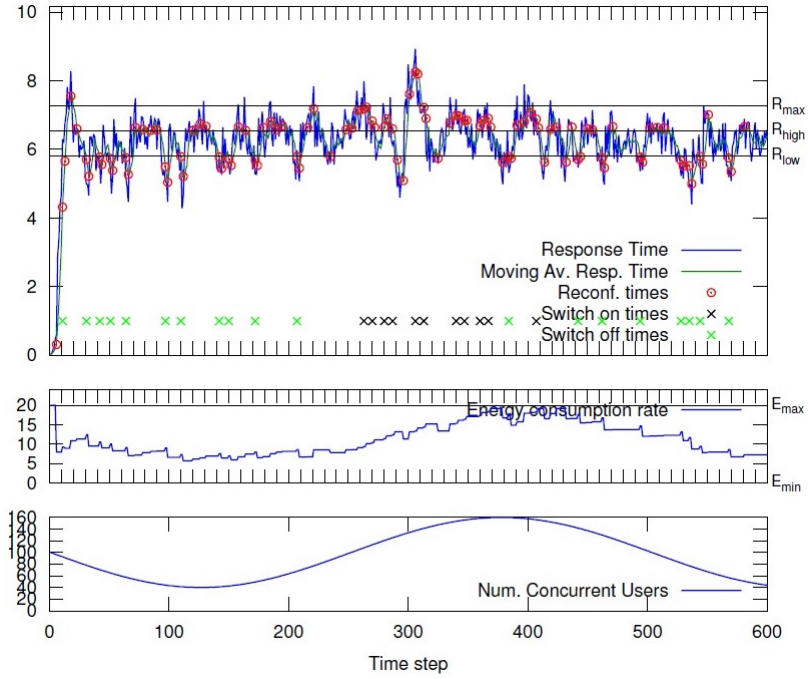

Figura 4.14: Risultati della simulazione nel caso in cui sono presenti $B = 1$ blocchi.

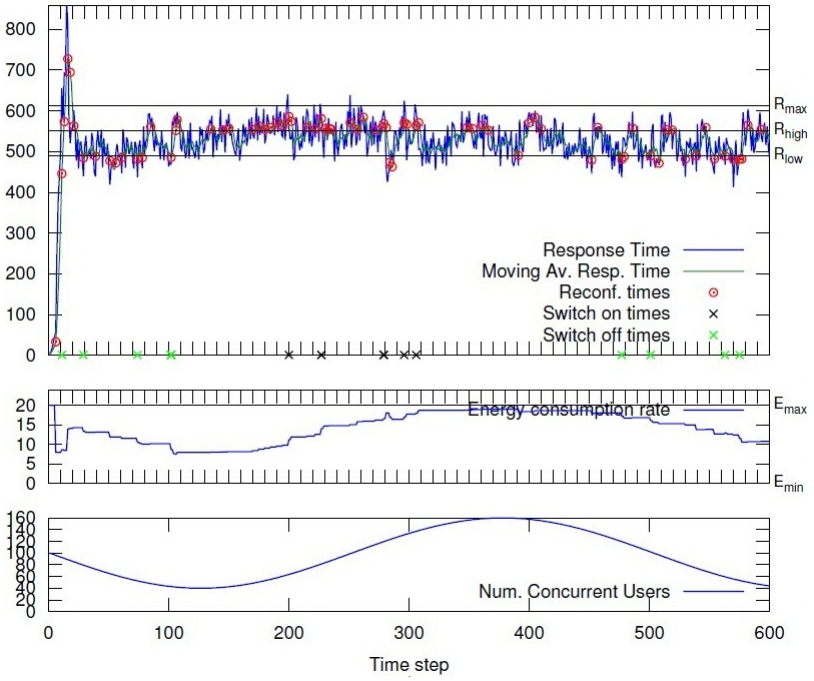

Figura 4.15: Risultati della simulazione nel caso in cui sono presenti $B = 10$ blocchi.

pensi al semplice caso in cui in un blocco si assista ad uno spegnimento di un dispositivo: in presenza di più blocchi, la perdita di potenza può essere compensata nello stesso istante temporale t dello spegnimento agendo sulla configurazione di un diverso blocco. Nel caso in cui il blocco sia soltanto uno invece, il sistema deve attendere che si esaurisca il suo nuovo Δt prima di poter eseguire una nuova riconfigurazione.

È importante poi notare la variazione di R_{max} nei due casi mostrati nelle figure: nonostante le sue modalità di calcolo siano identiche al caso di sistema omogeneo (che non presentava variazioni per K costante), R_{max} è variabile poiché l'esito dell'algoritmo MVA utilizzato dipende non solo dal numero K di dispositivi presenti nel sistema, ma anche dal numero B di blocchi. Questo perché con l'inserimento di ogni blocco oltre al primo, viene aggiunto anche un ulteriore tempo di servizio aggregato relativo a quel blocco. In questo modo si ha un incremento del tempo di servizio totale del sistema che si riflette sui suoi tempi di risposta. La proporzionalità di R_{max} rispetto al numero di blocchi presenti nel sistema rappresenta quindi la garanzia per avere risultati comparabili tra loro nonostante il variare di quest'ultimo.

Si nota inoltre un minor numero di spegnimenti e di accensioni nel caso di utilizzo di molti blocchi, causato dal fatto che per il sistema diventa sempre più difficile spegnere dispositivi tanto più il numero di quelli presenti all'interno dei blocchi diminuisce. Questo perché ogni blocco deve sempre essere operativo e performante (non ne è infatti ammesso uno in cui non sono presenti dispositivi allo stato **G0** in un istante temporale generico).

A conferma di quanto appena detto, viene mostrato il grafico riassuntivo in figura 4.16: in esso sono mostrati in verde il numero di accensioni e spegnimenti, decrescente al crescere del numero di blocchi. È poi mostrato il numero di violazioni, che tende a diminuire al crescere di B: questo perché, come già detto, un maggior numero di blocchi garantisce un mag-

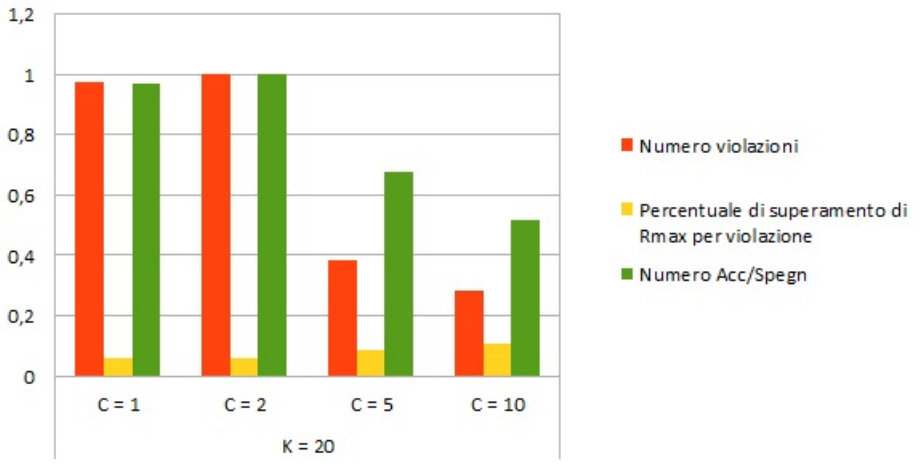

Figura 4.16: Confronto delle violazioni e del numero di accensioni e spegnimenti al variare del numero di blocchi presenti nel sistema.

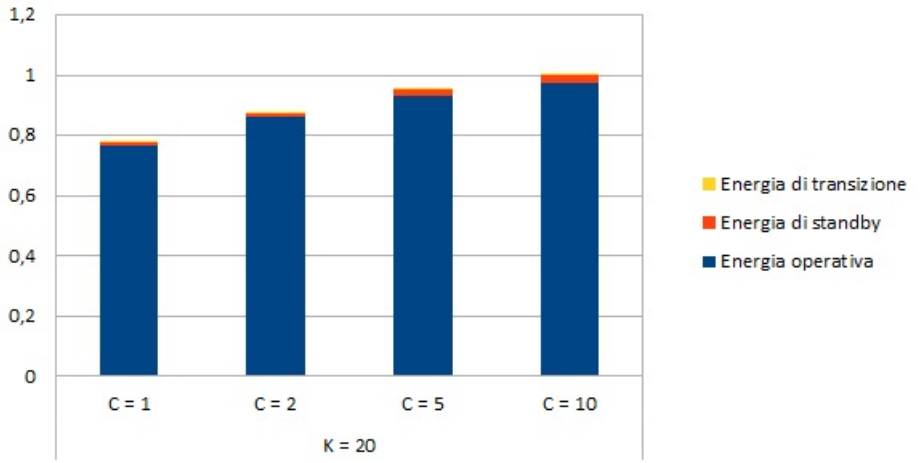

Figura 4.17: Confronto dei consumi energetici al variare del numero di blocchi presenti nel sistema.

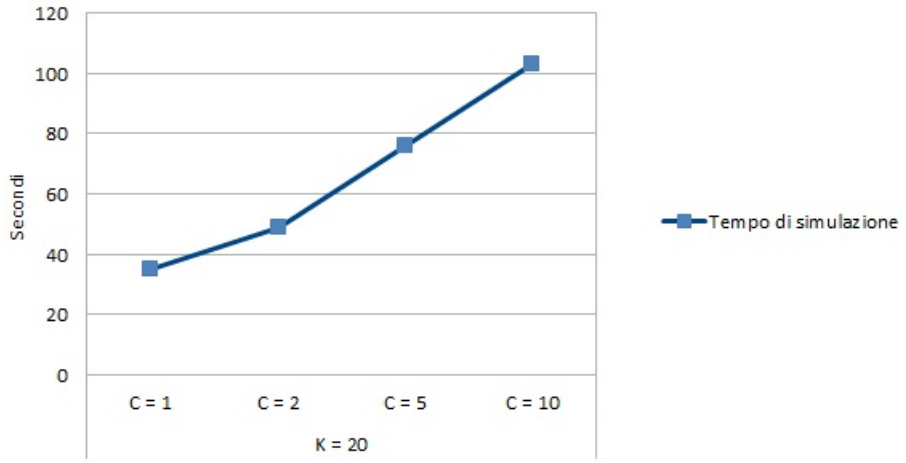

Figura 4.18: Confronto dei tempi di simulazione al variare del numero di blocchi presenti nel sistema.

gior numero di riconfigurazioni nello stesso istante temporale, e quindi un adeguamento del tempo di risposta stimato migliore. Infine è mostrata la percentuale di superamento di R_{max} per violazione, che tende a mantenersi costante nonostante una lieve crescita.

Per quanto riguarda i consumi energetici invece, il grafico in figura 4.17 riassume i risultati ottenuti dalle simulazioni: come già si notava dalle figura 4.14 e 4.15, l'energia totale consumata tende a crescere all'aumentare dei blocchi a causa del fatto che aumenta il numero minimo di dispositivi operativi. Dal grafico si può però anche apprezzare l'incremento del consumo di energia di standby, dovuto allo stesso motivo: infatti nel caso di un unico blocco composto da venti dispositivi, il sistema ne mantiene solamente uno alla volta in stato di standby, mostrando così dei consumi relativi bassi; nel caso invece in cui sono presenti molti blocchi, il sistema dovrà mantenere un dispositivo in standby per ciascun blocco in cui si è reso necessario uno spegnimento. Si osserva anche una diminuzione dei consumi di transizione all'aumentare dei blocchi, che passa dallo 0,35% rispetto al totale del caso $B = 1$ allo 0,14% del caso $B = 10$.

È infine importante mostrare come il numero di blocchi presenti all'interno del sistema incida sul tempo di esecuzione delle simulazioni. Nonostante esso non sia quantitativamente affidabile, come già detto, a causa dell'utilizzo di un linguaggio come Octave, riesce comunque a fornire un indice qualitativo della complessità computazionale per il caso multiblocco. In figura 4.18 sono mostrati i tempi di simulazione al variare del numero di blocchi B presenti: si osserva una crescita dovuta principalmente al fatto che il numero di riconfigurazioni per iterazione del ciclo di controllo è crescente, seppur sempre limitato. Ciò comporta una ripetizione delle procedure di speedup e di slowdown che vanno ad impattare sui tempi totali di esecuzione.

Capitolo 5

Related works

Negli ultimi anni i sistemi adattativi sono stati studiati da molte comunità e da prospettive differenti [28]. Il framework *autonomic computing* è un importante esempio di approccio generale alla progettazione di questo tipo di sistemi [29, 30]. Il lavoro di tesi presentato rientra nell'area dei sistemi adattativi in grado di garantire una qualità del servizio seppur soggetti ad un carico variabile, mirando parallelamente all'efficienza energetica, e di conseguenza alla riduzione dei consumi.

Da questo momento l'attenzione sarà focalizzata separatamente sui lavori che hanno contribuito alla (i) gestione dinamica della potenza ed al (ii) trade-off tra consumi energetici e performance.

L'obiettivo della gestione dinamica della potenza (GDP) all'interno del settore IT è quello di riconfigurare dispositivi o interi sistemi in modo tale che i loro consumi energetici siano minimi. Molte tecniche per l'allocazione delle risorse al variare del carico sono state studiate e applicate [31], tra cui l'*admission control* e il *load balancing*. L'admission control è un meccanismo di prevenzione da sovraccarichi di richieste che è in grado di rifiutarle sotto determinate condizioni di picco, in modo da garantire l'esecuzione di quelle già accodate [31, 32]. Il load balancing punta invece a distribuire le richieste in arrivo tra più istanze della stessa applicazione in modo da

garantire un utilizzo migliore delle risorse del sistema ed una migliore qualità del servizio [33, 34]. Framework integrati come WebSphere Application Server di IBM e la suite Tivoli riescono a combinare l'admission control e il load balancing all'interno di cluster o di data center [35].

Altre tecniche di GDP sono basate sulla riconfigurazione dei sistemi e prevedono spegnimenti dei componenti inattivi, o abbassamento del loro stato operativo quando sottoutilizzati [36, 37, 38, 39]. Studi recenti [3, 39, 40] indicano che la maggior parte delle tecniche GDP sviluppate sono destinate a singoli componenti (processori o dischi) o a specifici sistemi (cluster o data center) [41, 42, 43].

Il trade-off tra consumi energetici e performance è stato ampiamente studiato nella progettazione hardware e nelle reti, così come nei dispositivi alimentati da batteria poiché hanno intrinsecamente mostrato questo tipo di problema dalla loro creazione. Tuttavia l'interesse per questo trade-off nei centri di servizio è molto più recente. Nei lavori [3, 44] è riconosciuta l'importanza del problema degli sprechi energetici: lo trattano da diversi punti di vista (dispositivi hardware, software o sistemi operativi), descrivendone le principali cause e proponendo soluzioni per risolverlo. Gli autori di [6, 36, 45, 46] hanno invece mirato al trade-off tra performance e risparmio energetico: in [45] gli autori propongono un framework per piattaforme di hosting multi-servizio che permettono la riallocazione del corretto numero di risorse per ogni servizio mantendo i requisiti di performance. Il lavoro presentato in [46] estende il precedente considerando vincoli sui consumi e situazioni in cui il carico del sistema è eccessivo.

RISE si trova alla confluenza di queste due aree di ricerca e utilizza modelli calcolati durante l'esecuzione per riconfigurare i dispositivi del sistema sottoposto ad un carico variabile: il duplice obiettivo perseguito è quello di ridurre al massimo i consumi energetici garantendo una determinata qualità del servizio. RISE è un'estensione di EASY, un framework presenta-

to nel lavoro [10], che aggiunge al sistema la possibilità di effettuare accensioni e spegnimenti dei dispositivi, come anche transizioni verso lo stato di standby, per migliorarne ulteriormente i consumi energetici.

Capitolo 6

Conclusioni

In questo lavoro di tesi è stato proposto RISE, un framework che minimizza i consumi energetici di sistemi con vincoli sulla qualità del servizio e che può essere applicato ad un qualsiasi insieme di dispositivi che supportano lo standard ACPI. Il suo obiettivo è quello di modificare lo stato (sia globale, sia di performance) di ogni dispositivo in modo tale che il tempo di risposta del sistema rimanga sempre al di sotto di una determinata soglia e il consumo energetico totale sia minimo. Il problema della minimizzazione dei consumi è risolto da un'efficiente soluzione euristica che fa uso di un modello di performance basato sulle reti di code.

RISE è stato definito in maniera tale da gestire sia sistemi omogenei, formati cioè da dispositivi identici tra loro, sia sistemi eterogenei, formati cioè da diversi blocchi di dispositivi identici al loro interno ma differenti da un blocco all'altro.

Il framework proposto può essere esteso in differenti modi. Può essere innanzitutto utile adottare tecniche di previsione della variazione del carico del sistema, così da garantire accensioni e spegnimenti per tempo e quindi una maggior capacità di far fronte a sue improvvise variazioni.

Un altro interessante sviluppo consisterebbe nel rendere il ciclo di controllo di RISE totalmente decentralizzato, in modo tale da evitare che di-

venti esso stesso il collo di bottiglia per il sistema o il suo single point of failure.

Si potrebbe inoltre migliorare l'efficienza dell'algoritmo di controllo per il caso di sistema eterogeneo, dal momento che, come descritto in 3.2.5, la sua complessità computazionale è proporzionale sia al numero di dispositivi che al numero di blocchi presenti nel sistema.

Bibliografia

[1] J. G. Koomey, Estimating total power consumption by servers in the U.S. and the world 5/2/2007. URL http://sites.amd.com/de/Documents/svrpwrusecompletefinal.pdf

[2] R. Das, J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, H. Chan, Autonomic multi-agent management of power and performance in data centers, in: AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2008, pp. 107-114.

[3] P. Ranganathan, Recipe for efficiency: principles of power-aware computing, Commun. ACM 53 (4) (2010) 60-67.

[4] http://www.idc.com/getdoc.jsp?containerId=prUS23687112 (10/9/2012).

[5] San Murugesan, Harnessing Green IT: Principles and Practices, in IEEE IT Professional, (Feb 2008)

[6] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, N. Gautam, Managing server energy and operational costs in hosting centers, SIGMETRICS Perform. Eval. Rev. 33 (1) (2005) 303-314. doi:10.1145/1071690.1064253.

[7] L. A. Barroso, U. Hölzle, The case for energy-proportional computing, Computer 40 (12) (2007) 33-37. doi:10.1109/MC.2007.443.

[8] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, T. F. Wenisch, Power management of online data-intensive services.

[9] Wu-chang Feng, David Brandt, Debanjan Saha, A Long-Term Study of a Popular MMORPG, 19-20/9/2007.

[10] Moreno Marzolla, Raffaela Mirandola, Dynamic Power Management for QoS-Aware Applications, 22/7/2011

[11] Advanced configuration and power interface specification, revision 4.0a, Available at http://www.acpi.info/ (Apr. 5 2010).

[12] J. M. Rabaey, A. Chandrakasan, B. Nikolic, Digital Integrated Circuits-A Design Perspective, Prentice Hall, 2003, 2nd Edition.

[13] L. Brown, A. Keshavamurthy, D. Shaohua Li, R. Moore, V. Pallipadi, L. Yu, ACPI in linux: Architecture, advances and challenges, in: Proceedings of the Linux Symposium-Volume One, Ottawa, Ontario, Canada, 2004. URL http://www.linuxsymposium.org/2005/

[14] Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor, White Paper, available at ftp://download.intel.com/design/network/papers/30117401.pdf (Mar. 2004).

[15] AMD PowerNow! Technology, http://amd.com/us/products/technologies/amd-powernow-technology/Pages/amd-powernow-technology.aspx.

[16] M. Broyles, C. Francois, A. Geissler, M. Hollinger, T. Rosedahl, G. Silva, J. V. Heuklon, B. Veale, IBM EnergyScale for

POWER7 Processor-Based Systems, White Paper, http://www-03.ibm.com/systems/power/hardware/whitepapers/energyscale7.html (Aug. 2010).

[17] http://www.amd.com/us/Documents/43761D-ACP_PowerConsumption.pdf

[18] AMD Cool'n'Quiet Technology, http://www.amd.com/us/products/technologies/cool-n-quiet/Pages/cool-n-quiet.aspx.

[19] AMD Opteron Processor Power and Thermal Data Sheet, Publication # 30417, Revision 3.11, available at http://support.amd.com/us/Processor_TechDocs/30417.pdf (May 2006).

[20] http://www.via.com.tw/en/initiatives/greencomputing/powersaver.jsp

[21] S. Balsamo, Product form queueing networks, in: G. Haring, C. Lindemann, M. Reiser (Eds.), Performance Evaluation: Origins and Directions, Vol. 1769 of Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 2000, pp. 377-401. doi:10.1007/3-540-46506-5_16.

[22] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik, Quantitative System Performance: Computer System Analysis Using Queueig Network Models, Prentice-Hall, 1984.

[23] J. D. C. Little, A proof for the queuing formula: L = λW, Operations Research 9 (3) (1961) 383-387. doi:10.2307/167570.

[24] J. Zahorjan, K. C. Sevcick, D. L. Eager, B. I. Galler, Balanced job bound analysis of queueing networks, Comm. ACM 25 (2) (1982) 134-141.

[25] M. Reiser, S. S. Lavenberg, Mean-value analysis of closed multichain queuing networks, Journal of the ACM 27 (2) (1980) 313-322.

[26] Diego Perez-Palacin, Raffaela Mirandola, José Merseguer, QoS and energy management with Petri nets: a self-adaptive framework, 11-05-2012.

[27] J. W. Eaton, GNU Octave Manual, Network Theory Limited, 2002.

[28] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee (Eds.), Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar], Vol. 5525 of Lecture Notes in Computer Science, Springer, 2009. doi:10.1007/978-3-642-02161-9.

[29] J. O. Kephart, D. M. Chess, The vision of autonomic computing, IEEE Computer 36 (1) (2003) 41-50.

[30] M. C. Huebscher, J. A. McCann, A survey of autonomic computing-degrees, models, and applications, ACM Comput. Surv. 40 (3).

[31] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, T. Wood, Agile dynamic provisioning of multi-tier internet applications, ACM Trans. Auton. Adapt. Syst. 3 (1) (2008) 1-39. doi:10.1145/1342171.1342172.

[32] H. Feng, Z. Liu, C. H. Xia, L. Zhang, Load shedding and distributed resource control of stream processing networks, Perform. Eval. 64 (2007) 1102-1120. doi:10.1016/j.peva.2007.06.023.

[33] R. F. Berry, Trends, challenges and opportunities for performance engineering with modern business software, IEE Proceedings - Software 150 (4) (2003) 223-229.

[34] E. di Nitto, D. J. Dubois, R. Mirandola, On exploiting decentralized bioinspired self-organization algorithms to develop real systems, in: Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems,

IEEE Computer Society, Washington, DC, USA, 2009, pp. 68-75. doi:10.1109/SEAMS.2009.5069075.

[35] B. Urgaonkar, G. Paci

ci, P. Shenoy, M. Spreitzer, A. Tantawi, Analytic modeling of multitier internet applications, ACM Trans. Web 1. doi:10.1145/1232722.1232724.

[36] E. N. Elnozahy, M. Kistler, R. Rajamony, Energy conservation policies for web servers, in: USENIX Symposium on Internet Technologies and Systems, 2003.

[37] R. Calinescu, M. Z. Kwiatkowska, Using quantitative analysis to implement autonomic it systems, in: ICSE, IEEE, 2009, pp. 100-110.

[38] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, R. Katz, Napsac: design and implementation of a power-proportional web cluster, SIGCOMM Comput. Commun. Rev. 41 102-108. doi:http://doi.acm.org/10.1145/1925861.1925878. URL http://doi.acm.org/10.1145/1925861.1925878

[39] Y. Liu, H. Zhu, A survey of the research on power management techniques for high-performance systems, Software: Practice and Experience 40 (11) (2010) 943-964. doi:10.1002/spe.952.

[40] J. B. Carter, K. Rajamani, Designing energy-efficient servers and data centers, IEEE Computer 43 (7) (2010) 76-78.

[41] D. Barbagallo, E. Di Nitto, D. J. Dubois, R. Mirandola, A bio-inspired algorithm for energy optimization in a self-organizing data center, in: Proceedings of the First international conference on Self-organizing architectures, SOAR'09, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 127-151.

[42] L. Bertini, J. C. Leite, D. Moss, Power optimization for dynamic configuration in heterogeneous web server clusters, Journal of Systems and Software 83 (4) (2010) 585-598. doi:10.1016/j.jss.2009.10.040.

[43] M. Marzolla, O. Babaoglu, F. Panzieri, Server consolidation in clouds through gossiping, Technical Report UBLCS-2011-01, Department of Computer Science, University of Bologna, Italy (Jan. 2011). URL http://www.cs.unibo.it/pub/TR/UBLCS/ABSTRACTS/2011.bib?ncstrl.cabernet//BOLOGN UBLCS-2011-01

[44] R. Bianchini, R. Rajamony, Power and energy management for server systems, Computer 37 (11) (2004) 68-76. doi:10.1109/MC.2004.217.

[45] B. Abrahao, V. Almeida, J. Almeida, A. Zhang, D. Beyer, F. Safai, Self-fadaptive sla-driven capacity management for internet services, 2006, pp.557-568. doi:10.1109/NOMS.2006.1687584.

[46] I. Cunha, I. Viana, J. Palotti, J. Almeida, V. Almeida, Analyzing security and energy tradeoffs in autonomic capacity management, 2008, pp. 302-309. doi:10.1109/NOMS.2008.4575148.