UNIVERSITÀ CA' FOSCARI – VENEZIA
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea Triennale

Laureando: Luigi Bovino

Modelli a rete di code multiclasse

Relatore: Chiar.ma prof.ssa Simonetta Balsamo

Anno Accademico 2002-03

Indice

Capitolo 1

Introduzione

Nel progettare e nell'analizzare sistemi informatici è fondamentale avere degli strumenti di valutazione che permettano l'analisi delle prestazioni e del comportamento di ciò che abbiamo di fronte, in modo da poter valutare la bontà delle scelte fatte. Nasce così l'esigenza di creare modelli astratti in grado di rappresentare determinati aspetti del sistema che si desidera analizzare, e le reti di code multiclasse con blocco nascono appunto con questo scopo. Questa tipologia di modelli è infatti pensata per rappresentare sistemi composti da risorse condivise in grado di fornire servizi ad una certa popolazione di clienti. Ogni utente che richiede l'utilizzo di una risorsa, può essere servito immediatamente o, se questa è occupata, essere inserito in una coda d'attesa di lunghezza finita. Inoltre, in questo particolare tipo di reti di code, la popolazione servita dal sistema può essere suddivisa in più tipologie di utenti, e questo permette

una maggiore caratterizzazione del modello ed un'analisi più accurata del sistema, a scapito, però, di un aumento della complessità computazionale degli algoritmi volti ad analizzarlo.
L'elaborato si pone come primo obbiettivo il definire questi modelli a rete di code multiclasse. Per porre le basi su cui poi costruire la definizione, vengono introdotti nei capitoli dal 2 al 5 modelli a code via via più complessi, in modo da portare il lettore ad una conoscenza adeguata delle nozioni a lui utili per la comprensione dei capitoli 6 e 7, in cui vengono spiegati i modelli a rete di code multiclasse con e senza blocco.
La seconda parte dell'elaborato è dedicata agli algoritmi di risoluzione dei modelli a rete di code. Il capitolo 8 si occupa dei metodi per la risoluzione delle reti in forma prodotto senza blocco. Nelle prime due sezioni vengono descritti i celebri algoritmi di Convoluzione e MVA per reti a singola classe, mentre, nell'ultima sezione, vengono illustrati alcuni metodi capaci di risolvere in tempo polinomiale anche reti in forma prodotto multiclasse. Nel capitolo 9 vengono introdotti alcuni algoritmi di risoluzione per diverse tipologie di reti di code con blocco. Per le reti in forma prodotto vengono descritte le modifiche da apportare agli algoritmi di Convoluzione e MVA, per permetterne l'utilizzo anche in presenza di nodi a buffer finito, mentre, per le altre tipologie di reti vengono introdotti dei metodi di risoluzione approssimati. Il capitolo 10 è dedicato ai metodi per la risoluzione delle reti di code con blocco multiclasse, ed è diviso in due punti: nel primo vengono illustrati i metodi già presenti

in letteratura per la risoluzione di questo tipo di reti, mentre, al secondo punto, viene proposto, come nuovo metodo, una versione dell'algoritmo Acyclic Network Decomposition estesa alle reti multiclasse con blocco. In appendice, infine, viene fornita una possibile implementazione in Java di questo nuovo algoritmo.

Parte I

Modelli di code

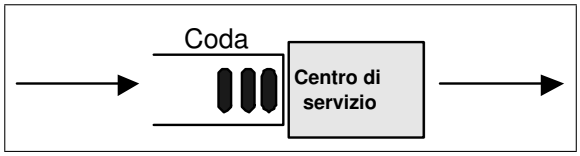
Capitolo 2

Modelli

Un modello è una rappresentazione astratta di un sistema e viene creato allo scopo di considerare ed analizzare meglio determinati aspetti di un problema, per studiarne meglio il comportamento. Sarà quindi possibile considerare nel proprio modello solo una parte del sistema o analizzarlo a diversi livelli di astrazione, per isolare meglio il problema che vogliamo prendere in considerazione. Un modello potrà essere analitico o di simulazione. Nel primo caso le componenti e il carico del sistema saranno rappresentate da variabili e parametri, e le relazioni fra di esse mostreranno le interazioni fra le componenti. Nel secondo caso invece sarà necessario l'utilizzo di un programma di simulazione, che riprodurrà il comportamento dinamico del sistema nel tempo.

I modelli a rete di code sono dei modelli analitici e vengono utilizzati per analizzare le performance dei sistemi che posseggono un insieme limi-

Figura 2.1: Esempio di modello a singola coda



tato di risorse e un certo numero di clienti che ne vuole usufruire. Nella figura 2.1 possiamo vedere un esempio di questa tipologia di modelli in cui abbiamo una sola coda e un solo centro di servizio. I centri di servizio rappresentano delle determinate risorse mentre le code contengono i clienti in attesa di poterle utilizzare. La figura in esame potrà rappresentare il modello ad alto livello di diverse situazioni, quali la coda di utenti di una rete locale che aspetta di poter usufruire di una stampante o la coda formata da coloro che vogliono ritirare un assegno in banca. I centri di servizio, inoltre, possono non rappresentare un'unica risorsa, ma anche tutte le risorse dello stesso tipo (quindi nell'esempio della banca, un centro di servizio può rappresentare più sportelli). Le diverse interpretazioni proposte della figura 2.1 evidenziano la duttilità di un modello, che una volta pensato ed analizzato, può essere poi riutilizzato in un contesto totalmente differente da quello che ne aveva determinato la nascita.
Per definire completamente un sistema a code è necessario descrivere:

Il processo di arrivo al sistema Se i clienti arrivano con dei tempi $t_1, t_2, ..., t_j$ all'interno del sistema, il tempo di interarrivo viene de-

finito come $\Delta = t_j - t_{j-1}$. Assumiamo che i tempi di interarrivo siano variabili casuali stocasticamente indipendenti con la stessa distribuzione di probabilità. Ad esempio il processo di arrivo sarà un processo di Poisson, qualora i tempi di interarrivo siano variabili random esponenziali. Oltre a Poisson, vengono anche utilizzate spesso distribuzioni di Erlang e iperesponenziali;

Il processo di servizio Il tempo di servizio è il tempo speso dai clienti nell'usufruire di una determinata risorsa. I tempi di servizio sono quindi delle variabili casuali IID (Independent and Identically Distributed) per le quali valgono le stesse considerazioni fatte per i tempi di interarrivo;

Il numero di server Come detto in precedenza, ogni centro di servizio puo' contenere al suo interno diversi server. Conoscerne il numero è fondamentale per un buon calcolo delle prestazioni di quel sistema;

La disciplina di coda (o di servizio) E' l'ordine in cui i clienti in coda vengono serviti:

- FCFS (First Come First Served)
- LCFS (Last Come First Served)
- RR (Round Robin con un quantum di dimensione fissata)
- PS (equivale a RR ma con un quantum tendente a zero)
- IS (Infinite Server)

- altre discipline dinamiche dipendenti dal tempo t richiesto (Shortest Processing Time First, Shortest Remaining Process Time First, ...)
- RAND (determina in modo casuale l'utente da estrarre dalla coda)

Proseguendo con l'esempio dei clienti in una banca, si può facilmente immaginare che la coda da loro formata abbia una disciplina di tipo FCFS.
Ogni disciplina può anche considerare gli utenti con diversi gradi di importanza. Ad esempio una disciplina con prelazione potrà interrompere l'utilizzo di una risorsa da parte di un utente, a favore di un cliente che possiede un livello di precedenza maggiore;

Dimensione della coda E' il numero di utenti che una coda riesce contemporaneamente a contenere. E' molto importante valutare in modo corretto questo parametro per evitare i blocchi del sistema (o di parte di esso) dovuti a code troppo corte.

Cardinalità della popolazione E' il numero totale di clienti che il sistema dovrà servire.

2.1 Notazione di Kendall

La notazione di Kendall serve a descrivere, in modo sintetico, un sistema a code, specificando i 6 parametri appena illustrati in questa forma:

A/B/c/n/p/Z

dove i simboli denotano:

- A: distribuzione del tempo di interarrivo
- B: distribuzione del tempo di servizio
- c: numero di server m
- n: dimensione della coda
- p: dimensione della popolazione
- Z: disciplina di servizio

Tale notazione può essere semplificata in A/B/c qualora coda e popolazione siano illimitate e la disciplina di servizio sia FCFS. Le distribuzioni di probabilità del tempo di interarrivo e di quello di servizio sono generalmente indicate dai seguenti simboli:

- M Esponenziale
- E_k Erlang con parametro k
- H_k Iperesponenziale con parametro k

- D Deterministico
- G Generale

Da notare che la distribuzione esponenziale viene indicata con la lettera M per evidenziare il suo essere *memoryless*, cioè il suo non essere influenzata nel comportamento dagli stati passati.
Per fare un esempio M/M/3/20/1500/FCFS rappresenterà una sistema a coda singola con rispettivamente: una distribuzione esponenziale sia per il tempo d'arrivo che per il tempo di servizio, 3 server, una coda con un buffer che può contenere 20 clienti, una popolazione di 1500 clienti e una disciplina di servizio del tipo First Come First Served. Da notare che dire che il buffer della coda è di lunghezza 20, vuol dire in questo caso che al massimo ci potranno essere 3 utenti ad occupare le risorse e 17 in attesa.

2.2 Indici di Prestazione

E' utile specificare subito la notazione con cui indicheremo in seguito gli indici di prestazione che ci serviranno per l'analisi dei sistemi a coda:

- q: numero di utenti nel sistema
- w: numero di utenti in coda
- s: numero di utenti in servizio

- t_q: tempo di risposta
- t_w: tempo di attesa
- t_s: tempo di servizio
- U: utilizzazione
- X: throughput

Inoltre con λ e μ si è soliti rappresentare rispettivamente i parametri delle distribuzioni esponenziale del tempi di interarrivo e di quelli di servizio e con $\rho = \lambda/\mu$ l'intensità di traffico.

2.3 Processi stocastici

Un processo stocastico (p.s.) è definito come una famiglia di variabili casuali $X = \{X(t) \mid t \in T\}$ definite su uno spazio di probabilità Ω e dove t varia in un insieme indice T. Con t comunemente si intende il tempo. Ogni variabile casuale assume dei valori che variano all'interno di un insieme E, detto spazio degli stati del processo stocastico.
Per fare un'esempio, se consideriamo n(t) come il numero di job in una CPU ad un determinato istante t, possiamo vedere il numero n(t) come una variabile casuale.

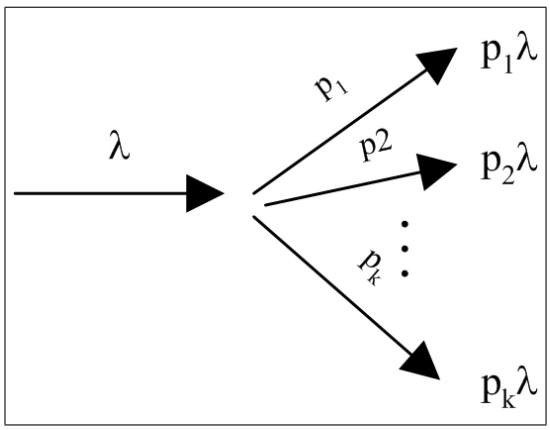
Un processo viene chiamato *processo di Markov* quando gli stati futuri dipendono solamente dallo stato presente e sono indipendenti da quelli

passati. Questa proprietà semplifica notevolmente l'analisi di un processo perché non necessita della conoscenza della sua storia passata.
Quando si lavora in uno spazio di stati discreto allora si parla di catena di Markov.
Un esempio di modellizzazione tramite un processo di Markov si può vedere ripensando alle code del tipo M/M/m, in cui con M si indica una distribuzione di tipo esponenziale. Come già visto, questo tipo di distribuzione ha la proprietà di essere memoryless, e proprio per questo motivo si adatta perfettamente ai processi markoviani. Il tempo speso da un utente in una determinata coda è un processo di Markov ed il numero di utenti in attesa di ottenere il servizio sono invece una catena di Markov.
Le catene di Markov di nascita-morte sono dei processi in cui le transizioni da uno stato all'altro sono limitate agli stati vicini. Quindi da uno stato i si potrà passare solamente agli stati $i+1$ o $i-1$.
Le code agli sportelli bancari sono in questo senso dei processi nascita-morte.
Se i tempi di interarrivo sono IID ed esponenzialmente distribuiti, allora il numero di arrivi n in un dato intervallo $(t, t+x)$ è una distribuzione di Poisson, e quindi un processo di arrivo può essere considerato come un processo di Poisson.
I processi di Poisson godono delle seguenti utili proprietà:

- Se si uniscono k processi di Poisson con parametro λ_i, si ottiene un

processo di Poisson con un parametro $\lambda = \sum_{i=1}^{k} \lambda_i$.

Figura 2.2: Una proprietà dei processi di Poisson. $p_1, p_2, .., p_k$ sono tali che $\sum_{i=1}^{K} p_i = 1$



- Se un processo di Poisson viene diviso in k sotto-processi, tali che un utente possa andare nel sotto-processo i-esimo con probabilità p_i, allora ogni sotto-processo avrà come parametro $p_i\lambda$. (v. figura 2.2)

- Se il parametro di interarrivo λ è minore del parametro esponenziale di servizio μ, allora le partenze dal centro di servizio avranno la stessa frequenza degli arrivi. Per convicersi di questo basta pensare all'esempio dello sportello postale. Se i clienti arrivano con una frequenza λ minore della frequenza con cui gli sportelli riescono a soddisfare le richieste, allora il parametro di servizio μ si adatterà

alla frequenza con cui i clienti si presentano nella coda. Se così non fosse, allora avremmo che gli sportelli completerebbero più compiti di quanti ne verrebbero loro assegnati, facendoci cadere in un assurdo.

Capitolo 3

Modelli a singola coda

I modelli a singola coda sono i più semplici che considereremo. Possono essere usati, per esempio per rappresentare con un modello il funzionamento di un processore o di una singola risorsa. Molte di queste code possono essere viste come processi nascita-morte e per questo, nella sezione che segue, daremo ulteriore attenzione a questo tipo di processi, già introdotti nel capitolo precendente.
Successivamente analizzeremo i basilari modelli a singola coda M/M/1, M/M/m/, M/M/1/K.

3.1 Processi nascita-morte

Come già visto nel capitolo precedente, i processi nascita-morte servono a rappresentare dei sistemi in cui gli stati cambiano solamente di un'u-

nità alla volta. Quindi, se all'istante t una risorsa si trova allo stato n, all'istante t+1 il suo stato potrà diventare n-1 o n+1.
Il teorema seguente mostra un'importante proprietà di questa tipologia di processi, che permette di calcolare facilmente la probabilità stazionaria di stato π.

Teorema 1 *La probabilità stazionaria di stato π_n che un processo nascita-morte si trovi in uno stato n, è data dalla formula*

$$\pi_n = \frac{\lambda_0 \lambda_1 ... \lambda_{n-1}}{\mu_1 \mu_2 \mu_n} \pi_0, \quad con \quad n = 1, 2, ... \infty \tag{3.1}$$

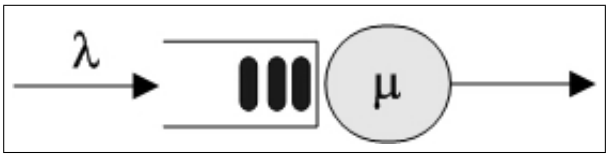
Dove π_0 indica la probabilità di trovarsi nello stato 0 e λ e μ rappresentano rispettivamente il tasso di nascita e il tasso di morte.

Per la dimostrazione di questo teorema consigliamo di guardare i testi indicati in bibliografia (v. [R91])

3.2 Sistemi M/M/1

Il modello M/M/1 è formato da una singola coda con tempi di interarrivo e di servizio esponenziali, con parametri rispettivamente λ e μ.
Si considera inoltre che la popolazione e la memoria siano infiniti e una disciplina di servizio del tipo FCFS. La figura 3.1, che indica un diagramma di transizione dello stato per la coda in esame, mostra chiaramente

Figura 3.1: Sistema M/M/1



che il processo stocastico considerato è di tipo nascita-morte con tassi di crescita $\lambda_k = \lambda, \kappa \geq 0$ e con tassi di morte $\mu_k = \mu, \kappa \geq 1$.
Considerando quanto detto nel teorema 1, la probabilità che n clienti siano nel sistema sarà uguale a:

$$\pi_n = \left(\frac{\lambda}{\mu}\right)^n \pi_0, \qquad n = 1, 2, ..., \infty \tag{3.2}$$

La quantità λ/μ viene chiamata intensità di traffico e viene indicata con il simbolo ρ. L'equazione precedente può assumere quindi la forma più compatta:

$$\pi_n = (1-\rho)\rho^n \qquad con, \quad \pi_0 = \frac{1}{1+\rho+\rho^2+\rho^\infty} = 1-\rho \tag{3.3}$$

Ora elencheremo una serie di indici utili per il calcolo delle prestazioni dei modelli M/M/1, fornendo anche le formule per calcolarli. Questo testo, soprattutto per quanto riguarda questa prima parte, non possiede l'ambizione d'essere completo ma vuole fornire solamente un'introduzione ai risultati ottenuti in questa disciplina. Per questo motivo non ci si concentrerà nel giustificare tutte le formule riportate e si lascerà poi al lettore interessato l'onere di consultare per approfondimenti i libri citati in bibliografia.

- Numero medio di clienti nel sistema: $N = E[q] = \rho/(1-\rho)$
- La varianza del numero di utenti nel sistema: $Var[q] = \rho/(1-\rho)^2$
- Il numero medio di utenti in coda: $W = E[w] = N - \rho = \rho^2/(1-\rho)$
- La probabilità di avere k utenti in coda:

$$P(n_q = k) = \begin{cases} 1-\rho^2, & \text{k=0} \\ (1-\rho)\rho^{k+1}, & \text{k}\geq 1 \end{cases}$$

- Il tempo medio di risposta $R = E[t_q] = (1/\mu)/(1-\rho)$
- Il tempo medio d'attesa: $T_w = (\rho/\mu)/(1-\rho)$
- Utilizzazione: $U = 1 - \pi_0 = \rho$

Il sistema è considerato stabile quando il tasso di arrivo è minore del tasso servizio. Quindi la condizione di stabilità può essere scritta come $\lambda/\mu < 1$.

3.3 Sistemi M/M/m

Il sistema M/M/m è caratterizzato da processi d'arrivo e di servizio esponenziali e da un centro di servizio formato da m componenti. Inoltre, si considera sempre una coda di lunghezza infinita e una disciplina di servizio FCFS.

Questo modello può essere pensato come un M/M/1 dove il tasso medio

di servizio μ dipende dallo stato k in questo modo:

$$\mu(k) = \begin{cases} k\mu, & 0 \leq \text{k} \leq \text{m} \\ m\mu, & \text{k} \geq \text{m} \end{cases}$$

Avremo quindi che il processo stocastico associato al sistema M/M/m sarà un processo di nascita-morte in cui i tassi di nascita sono $\lambda_k = \lambda, k \geq 0$, e i tassi di morte $\mu_k = min(k, m)$, $\mu, k \geq 1$. L'intesità di traffico sarà $\rho = \lambda/m\mu$ e la condizione di stazionarietà sarà $\rho < 1$.
La probabilità di non avere nessun utente nel sistema sarà:

$$\pi_0 = \left[1 + \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{n=1}^{m-1} \frac{(m\rho)^n}{n!}\right]^{-1}$$

Mentre la probabilità che all'interno del sistema ci siano n utenti sarà:

$$\pi_n = \begin{cases} \pi_0 \frac{(m\rho)^n}{n!}, & \text{n<m} \\ \pi \frac{\rho^n m^m}{m!}, & \text{n} \geq \text{m} \end{cases}$$

Il numero medio di server occupati E[s] è dato dalla formula:

$$E[s] = \sum_{k=0}^{m-1} k\pi_k + \frac{m\pi_m}{1-\rho} = m\rho = \frac{\lambda}{\mu}$$

Mentre il numero medio di utenti nel sistema è:

$$N = m\rho + \pi_m \frac{\rho}{(i-\rho)^2}$$

Inoltre il tempo medio di risposta è uguale a:

$$R = \frac{1}{\mu} + \frac{\pi_m}{(m\mu(1-\rho)^2}$$

E il tempo medio di attesa è:

$$T_W = \frac{\pi_m}{m\mu(i-\rho)^2}$$

Infine, la probabilità che un utente in arrivo trovi tutti i server occupati e finisca in coda è:

$$Prob\{coda\} = \sum_{k=m}^{\infty} \pi_k = \pi_0 \frac{(m\rho)^2}{m!(1-\rho)}$$

3.4 Sistemi M/M/1/K

Può essere considerato come un sistema M/M/1, capace di ospitare un numero non superiore a K clienti. Per fare un esempio, se K=5, il sistema potrà contenere un massimo di 5 clienti, di cui 4 rimarranno in coda ad aspettare che il server si liberi.

Un caso particolare di questo sistema si avrà quando K=1. In questo caso non ci sarà alcuna coda e quindi, quando il server sarà occupato, il sistema ignorerà le nuove richieste.

Il processo stocastico associato al sistema M/M/1/K è un processo nascita-morte con uno spazio di stati E finito e tassi di nascita $\lambda_k = \lambda$ per $0 \leq k < K$ e tassi di morte $\mu_k = \mu$ per $0 \leq k < K-1$. In intervalli diversi da quelli specificati, i tassi di nascita e di morte saranno uguali a zero, poiché il sistema avrà raggiunto la saturazione.

Alcuni dei principali indici per questo sistema saranno:

- La probabilità che ci siano n clienti nel sistema:

$$\pi_n = \begin{cases} \frac{(1-\rho)}{1-\rho_{K+1}}\rho, & 0 \leq \text{k} \leq \text{K} \\ 0, & \text{k} > \text{K} \end{cases}$$

- Il numero medio di clienti nel sistema:

$$E[n] = \frac{\rho}{1-\rho} - \frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}}$$

- Il numero medio di utenti in coda:

$$E[n_q] = \frac{\rho}{1-\rho} - \rho\frac{1+B\rho^K}{1-\rho^{K+1}}$$

- Il tempo medio di risposta:

$$R = E[n]/[\lambda(1-\pi_K)]$$

- Il tempo medio d'attesa

$$T_W = E[n_q]/[\lambda(1-\pi_K)]$$

Capitolo 4

Modelli a rete di code

Nel capitolo precedente abbiamo visto alcuni modelli che utilizzano un'unica coda. Molti sistemi però consistono in diverse code, e, proprio per questo, devono essere modellizzati in una rete di code.
In generale, viene chiamato modello a rete di code, ogni modello in cui sono presenti più code collegate tra loro. Avremo quindi esempi in cui clienti, appena usciti da una coda, entreranno in un'altra, per utilizzare una differente risorsa.
Le reti di code possono essere aperte, chiuse oppure miste. Sono aperte quando la cardinalità della popolazione è variabile, e quindi quando ci sono degli utenti che entrano ex novo nel sistema, ed altri che invece ve ne escono. Una rete è chiusa quando ogni cliente rimane costantemente all'interno del sistema e quindi quando la popolazione rimane costante. Una rete, infine, si dice mista quando è chiusa per alcuni utenti, ma è

aperta per altri. In questo capitolo analizzeremo solamente modelli in cui gli utenti appartengono ad un'unica classe e per questo motivo le reti di code potranno essere solamente aperte o chiuse. Non vi è infatti alcun carattere distintivo capace di distinguere il comportamento degli clienti del sistema. Un rete potrà essere considerata mista solo se multicatena (v. capitolo 6), in quanto, come vedremo, in reti di questo tipo gli utenti che appartengono ad una catena potranno avere un comportamento del tutto distinto da altri appartenenti a catene diverse.

Un modello a rete di code viene definito da un insieme di centri di servizio, un insieme di utenti e da una struttura di interconnessione. Per definire a pieno il modello occorrerà quindi definire gli insieme da cui è composto. La struttura indica la topologia della rete e quindi le possibili transizioni che potrà compiere un utente. Il cliente potrà muoversi all'interno della rete seguendo una determinata distribuzione di probabilità. La topologia della rete sarà quindi caratterizzata da una matrice di probabilità P, tale che p_{nm} rappresenterà la probabilità che un cliente, dopo essere stato servito dal nodo n, scelga di mettersi in coda al nodo m. Un utente al nodo i uscirà dalla rete con probabilità p_{i0}. Naturalmente la matrice dovrà essere tale da soddisfare la seguente proprietà che ne garantisce la coerenza:

$$\sum_{j=1}^{M} p_{ij} + pi0 = 1, \qquad 1 \leq i \leq M$$

Numerando opportunamente i nodi, la matrice di routing P risulterà essere una matrice triangolare superiore.
Un modello a rete di code si dice Markoviano se è rappresentabile da un processo stocastico di Markov, e se quindi il processo gode della proprietà di assenza di memoria. Anche per modelli di reti di code molto semplici lo spazio degli stati cresce in modo esponenziale al crescere del numero dei nodi, degli utenti o considerando più classi d'utenza. Questo rende il metodo d'analisi del modello a rete di code spesso molto difficile da un punto di vista computazionale, e quindi poco applicabile. Esiste però una classe di modelli per cui è possibile utilizzare algoritmi efficienti, ed è formata dai modelli a rete di code in forma prodotto. Il nome deriva dal fatto che la probabilità congiunta dell'intero sistema, per questi modelli, è data dal prodotto delle funzioni degli stati dei singoli nodi.
Nei punti successivi illustreremo due tipologie di reti a singola classe a cui è applicabile la forma prodotto: vedremo le reti aperte di Jackson, le reti di chiuse di Gordon-Newell.

4.1 Reti di Jackson

Le reti di code di Jackson [JAC63] sono reti aperte, con centri di servizio esponenziali, arrivi Poissiani e topologia probabilistica arbitraria indipendente dall stato della rete.
La matrice P rappresenta la matrice di probabilità di diramazione tra gli M nodi della rete, dove p_{ij} rappresenta la probabilità che un utente

uscito dal nodo i entri nel nodo j, con 1≤i,j≤M. La probabilità che un cliente esca dal sistema dopo essere stato servito nel nodo i è dato da $p_{i0} = 1 - \sum_j p_{ij}$.

Ogni nodo è formato da m_i server e possiede un tempo di servizio esponenziale di parametro μ_i. λ_i, il tasso medio d'arrivo totale al nodo i, è definito dall'equazione.

$$\lambda_i = \gamma_i + \sum_{j=1}^{M} \lambda jpji, \qquad 1 \leq i \leq M$$

Con γ_i inteso come il parametro del processo di Poisson d'arrivo dall'esterno al nodo i.

Per calcolare la distribuzione stazionaria di questo tipo di rete, si utilizzano le equazioni definite nel teorema di Jackson:

Teorema di Jackson 4.1.1 *Se è soddisfatta la condizione di stazionarietà*

$$\rho_i = \lambda i/mi\mu i < 1 \qquad 1 \leq i \leq M \tag{4.1}$$

allora la distribuzione stazionaria di stato di una rete di Jackson si esprime come:

$$\pi(n_1, n_2, ..., n_M) = \prod_{i=1}^{M} \pi i(n_i) \tag{4.2}$$

dove

$$\pi_i(n) = \pi_i(0)(m_i\rho_i)^n/n! \qquad 1 \leq n \leq m_i$$

$$\pi_i(n) = \pi_i(0)m_i^{m_i}\rho_i^n/m_i! \qquad n > m_i$$

e $\pi_i(0)$ *è ottenuto dalla condizione di normalizzazione* $\sum_n \pi_i(n) = 1$.

4.2 Reti di Gordon-Newell

Le reti di Gordon-Newell [GN67] sono reti chiuse, con una topologia probabilistica arbitraria indipendente dallo stato della rete. Gli M nodi della rete sono del tipo $./M/m_i$, dove ogni nodo i è formato da m_i server, possiede una disciplina di servizio FCFS ed una distribuzione di servizio esponenziale di parametro μ_i. Essendo reti chiuse, la cardinalità K della popolazione sarà costante e gli utenti si sposteranno tra i nodi secondo la matrice di diramazione $P = [p_{ij}]$, già introdotta nella sezione precedente. Il tasso medio e_i di arrivo totale al nodo i è definito come:

$$e_i = \sum_{j=1}^{M} e_j p_{ji} \qquad 1 \leq j \leq M$$

Il sistema così definito possiede infinite soluzioni, e questo porta all'introduzione di un fattore arbitrario per poterlo risolvere.
Il teorema che segue definisce la distribuzione stazionaria di stato per questa tipologia di reti.

Teorema di Gordon-Newell 4.2.1 *La distribuzione stazionaria di stato per reti di Gordon Newell si può esprimere come:*

$$\pi(n_i, ..., n_M) = \frac{1}{C(K)} \prod_{i=1}^{M} g_i(n_i)$$

Dove $g_i(n)$ è la soluzione non normalizzata del nodo i risolto come un sistema $M/M/m_i$, con parametri e_i, μ_i definita come:

$$g_i(n) = (m_i \rho_i)^n / n! \qquad 0 \leq n \leq m_i$$

$$g_i(n) = m_i^n \rho_i^k / m_i! \qquad n > m_i$$

e C(K) è una costante di normalizzazione sulle probabilità definita come:

$$C(K) = \sum_{n \in E} \prod_{i=1}^{M} g_i(n_i)$$

Capitolo 5

Modelli a rete di code con blocco

Nel capitolo precedente consideravamo reti composte da nodi con un buffer infinito e tale da permettere ad un numero infinito di utenti di mettersi coda per attendere di poter usufruire di un servizio. E' più realistico considerare spesso però reti con nodi a capacità limitata, ed è quello che faremo in questo capitolo. Con reti di questo tipo (chiamate anche QNB: Queueing Networks with blocking) ci si potrà trovare nella situazione in cui un cliente voglia passare dal nodo i al nodo j, ma trovi che il nodo j possieda la coda dei clienti in attesa di servizio piena. E' evidente che in situazioni di questo tipo l'utente non potrà entrare subito nel nodo di destinazione ma dovrà attendere che si liberi, e dovrà quindi in qualche modo bloccarsi. Nella sezione che segue vedremo alcune tec-

niche di blocco, pensate proprio per gestire situazioni di questo tipo. Al punto successivo, invece, introdurremo un importante problema derivato dal blocco dei nodi: il deadlock. Infatti in reti a buffer finito potranno verificarsi combinazioni di blocchi di nodi tali da portare il sistema in una situazione di stallo, risolvibili solamente con opportune tecniche.

5.1 Tipologie di blocco

Come già visto nell'introduzione al capitolo, in reti formate da nodi con code di lunghezza limitata, possono verificarsi situazioni in cui un utente del nodo i non possa trasferirsi al nodo j da lui richiesto, poiché quel nodo è pieno. Per gestire queste situazioni sono state pensate diverse tecniche di blocco dei nodi. La tecnica *Blocking After Service (BAS)*, gestisce il problema facendo rimanere il cliente al nodo i fino a quando si libera dello spazio nella coda del nodo j. In questo modo il nodo i è costretto a bloccare il suo lavoro e ad attendere che si risolva la situazione di saturazione della coda del nodo j, facendo rimanere in attesa i clienti presenti nella sua coda. Se più clienti, in nodi differenti, richiedono il servizio da un nodo saturo, bisognerà pensare all'ordine in cui questi clienti dovranno sbloccarsi per accedere al nodo; un metodo semplice e molto utilizzato è il First Blocked First Unblocked, che fa accedere al nodo i clienti in ordine d'anzianità della richiesta.

Blocking Before Service (BBS) impone ad ogni cliente di dichiarare la sua destinazione j prima di usufruire del servizio del nodo i in cui si tro-

va. Inoltre il nodo i si bloccherà qualora la destinazione risulti satura al momento della dichiarazione da parte dell'utente. Se la coda del nodo di destinazione si riempie mentre l'utente usufruisce del servizio al nodo i, allora il nodo i stesso interrompe il suo lavoro fino a quando j non si libera, e, una volta che questo accade, ricomincia a fornire il servizio all'utente dall'inizio, perdendo tutto ciò che aveva già fatto prima di bloccarsi.
Con il metodo *Repetitive Service Blocking (RS)*, in caso di destinazione satura, l'utente comincia un nuovo servizio nel nodo i di partenza. Una volta terminato il servizio ripetuto nel nodo i, l'utente potrà provare ad entrare nella stessa destinazione scelta in precedenza (in questo caso si parla di RS-FD: fixed destination) o scegliere una nuova destinazione (RS-RD: random destination) basandosi sulla matrice probabilistica di routing della rete.
Una scelta semplice (anche se molto drastica) in caso di destinazione satura, potrebbe essere Rejection Blocking, dove l'utente che si trova nella situazione di non poter entrare nella coda del nodo di destinazione, viene fatto uscire dalla rete. Questo metodo potrà essere utilizzato naturalmente solamente in reti aperte (in quanto nelle reti chiuse il numero dei clienti deve rimanere costante).

5.2 Deadlock

Quando si considerano reti di code con blocchi, allora bisogna prendere anche in considerazione la possibilità che si manifesti un deadlock. Il

deadlock è una condizione di stasi in cui un sistema può venirsi a trovare quando più nodi vengono bloccati. Per spiegare questo concetto facciamo un esempio: consideriamo il caso in cui un utente a sia bloccato al nodo i, aspettando che il nodo j si liberi. Se anche un utente b sta bloccando contemporaneamente il nodo j in attesa di poter usufruire del nodo i, allora può verificarsi una situazione di deadlock, poiché entrambi i nodi potranno rimanere in attesa per un tempo indefinito. Per risolvere il problema del deadlock sono state pensate diverse tecniche che utilizzano o un approccio di tipo preventivo, o tentano di scoprire e risolvere il problema una volta che si presenta. Una condizione necessaria per evitare il deadlock per tecniche di bloccaggio BAS, BBS e RS è che $p_{ii} = 0$, $1 \leq i \leq M$, poiché altrimenti, banalmente, ci si potrà trovare nella condizione in cui un utente richiede di poter usufruire per una seconda volta dello stesso servizio ma trova la coda piena. Se ciò accade è facile immaginare che il nodo non potrà mai sbloccarsi, poiché ne avrebbe la possibilità solo nel caso in cui l'utente che lo tiene in stasi possa migrare sul nodo di destinazione, che in questo caso è anche quello da cui proviene. Condizione che non si potrà mai verificare.

Una semplice tecnica di prevenzione per i blocchi di tipo BAS, BBS e RS-FD consiste nel mantenere il numero di utenti minore delle capacità totali dei buffer dei nodi durante ogni possibile ciclo nella rete. Per il blocco di tipo RS-RD è sufficiente invece che la matrice di routing P sia irriducibile e che la popolazione di clienti sia minore della capacità totale

fornita dai buffer dei nodi della rete.

In termini più strettamente matematici, avremo che con una tipologia di blocco RS-RD non potrà verificarsi del deadlock se

$$N < \sum_{j=1}^{M} B_j$$

con B_j capacità della coda del nodo j e M la cardinalità dei nodi della rete.

Per le tipologie di blocco BAS, BBS e RS-FD dobbiamo prendere in considerazione ogni possibile ciclo $C = (i_j, i_{j+1}, ..., i_j)$, dove $\forall j \in C, p_{i_j i_{j+1}} \neq 0$. Premesso ciò, avremo che le tipologie di blocchi considerate non potranno mai subire deadlock se

$$N < \min_{\forall c} \sum_{i_j \in c} B_{i_j}$$

Capitolo 6

Modelli a rete di code multiclasse

Nelle pagine precedenti abbiamo considerato solamente esempi in cui i clienti di una rete appartenevano tutti ad una stessa tipologia. Nessuna distinzione avevano infatti nelle loro matrici probabilistiche di routing tra i nodi, oppure nella frequenza con cui terminavano d'essere serviti da un nodo. In alcuni casi però questa tipologia di modelli è limitante quando si vuol sintetizzare il comportamento di un sistema reale, in quanto spesso è più efficace considerare delle classi distinte di utenti.

Per comprendere meglio quest'affermazione si può prendere come esempio un sito web che dà la possibilità di giocare ad un gioco a quiz. Questo sito inoltre è frequentato da due tipi di utenti: dai giocatori, che si limitano a rispondere alle domande a loro proposte, e dagli utenti autori,

che si occupano di inserire continuamente all'interno dei database del sito nuove domande. In una rappresentazione a rete di code di questo sito non sarebbe corretto considerare gli utenti come un'unica tipologia di clienti, perché è facile comprendere che autori e giocatori avranno comportamenti differenti tra loro. Un autore, accederà ai database (rappresentati da dei nodi in un ipotetico modello) per compiere per lo più delle scritture, mentre un giocatore lo farà per leggere le nuove domande, e letture e scritture occupano un database per tempi differenti. Inoltre un giocatore è plausibile pensare che rimanga per molto più tempo all'interno del sito e che visiti con maggior frequenza pagine differenti rispetto ad un autore, e che debba, quindi, possedere matrici di routing differenti. Differenze, insomma, tra le due tipologie di utenti, non è difficile trovarne. Bisogna ora utilizzare nuovi strumenti che ci permettano di considerare, all'interno di un modello, più tipi di clienti, ed è quello che tenteremo di fare all'interno di questo capitolo.

6.1 Catene e Classi

Per rappresentare adeguatamente un modello che consideri più tipologie di utenti, dobbiamo introdurre il concetto di catena. Una catena è una rappresentazione che caratterizza fortemente un cliente inserendolo all'interno di una determinata categoria. Un utente che appartiene ad una catena, non potrà mai appartenerne ad un'altra. Riprendendo l'esempio precedente, avremo quindi che giocatori e autori apparterranno a catene

differenti.

Ogni catena è formata da classi, che rappresentano lo stato in cui un utente si trova. Consideriamo, per esempio, il comportamento di un autore. Egli potrà compiere il log-in, per farsi riconoscere all'interno del sito, inserire una domanda, e dopo uscire dal sistema. In questo caso l'autore attraverserà due stati: quello in cui sta compiendo il log-in e quello in cui inserisce una domanda. Questi due stati potranno essere rappresentati nella rete da due classi distinte che apparterranno alla catena autore e che saranno in qualche modo collegate tra loro.

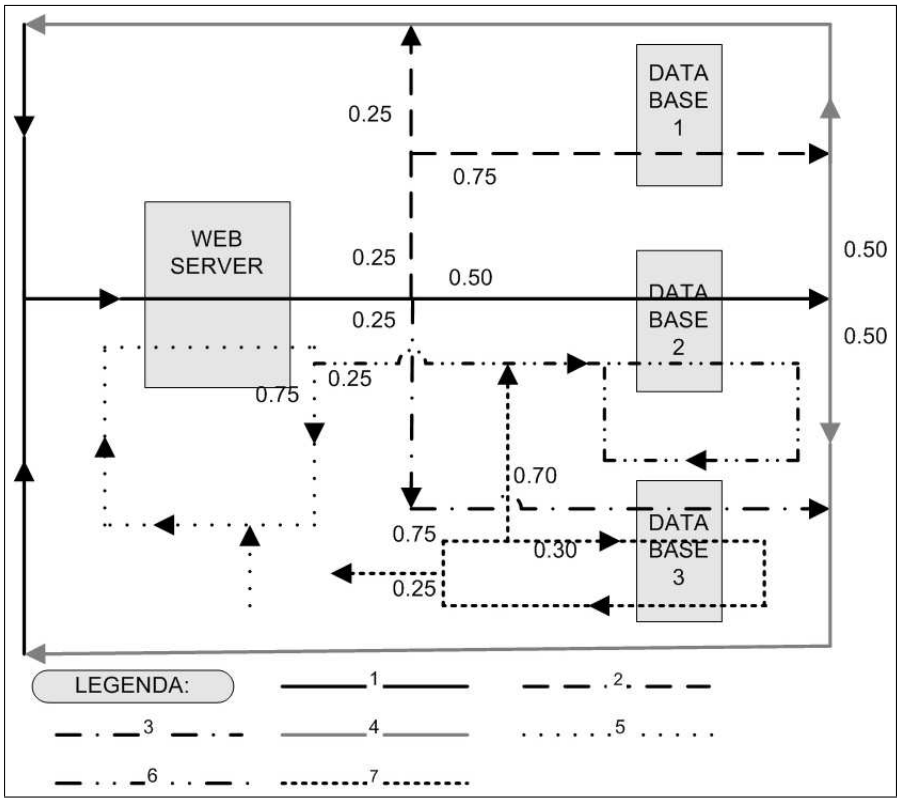
Con questi due nuovi concetti possiamo ora considerare una versione estesa dei modelli a rete di code fino ad ora analizzati, che consideri più tipologie di utenti. E' fondamentale però specificare che all'aumentare delle catene e delle classi, aumenta anche la complessità del modello e i tempi necessari per risolverlo. Sarà quindi necessario spesso trovare un compromesso tra accuratezza del modello e velocità di risoluzione.

6.2 Caratterizzazione di classi e catene

Dopo aver astratto un sistema con un'ipotetica soluzione contenente classi e catene, il progettista può trovarsi di fronte ad un modello graficamente molto complesso, in cui risulta difficile rilevare errori. In questo sottocapitolo forniremo qualche utile strumento per valutare la bontà della rete multiclasse che è stata creata. Per semplificare la spiegazione adotteremo il modello in figura come esempio, e lo proporremo come pos-

sibile soluzione del problema del sito internet del gioco a quiz proposto nella sezione precedente.

Figura 6.1: Esempio di modello multiclasse



Analizziamo la figura: possiamo notare che ogni classe è distinta dalle altre da una grafica differente e che, tra alcune classi, esistono delle congiunzioni che permettono ad un utente di passare da una all'altra. Seguendo il percorso della classe 1, possiamo notare che dopo aver attra-

versato il web-server incontra un bivio a 3 vie, ed accanto ad ognuna di esse c'è un numero. Un utente che passa per quel bivio potrà rimanere in classe 1 con una probabilità dello 0.50, potrà passare alla classe 2 con probabilità 0.25 o passare alla classe 3 con probabilità 0.25. Sempre analizzando la figura, si può notare che la classe 5 riceve un input dall'esterno mentre un utente in classe 7 può decidere di uscire dal sistema.

Introduciamo ora un nuovo concetto: una rete si dice *consistente* se, presi due cammini x e y all'interno di essa, che appartengano alla stessa classe, è possibile trovare un percorso capace di congiungerli. Guardando l'esempio in figura si può notare che il modello da noi proposto è consistente, ma non lo sarebbe se la classe 5 e la classe 2 appartenessero alla stessa classe. Infatti, e si invita il lettore a provare, non esiste alcun percorso capace di collegare le due classi.

Ma non è assolutamente sicuro che una rete consistente si comporti in realtà come vogliamo. Nulla nega che una rete consistente possa avere, a lungo termine, fenomeni di saturazione di una classe (o di una parte di essa) per mancanza di uscite verso altre classi, o che una classe possa diventare irrimediabilmente vuota, perché magari possiede solamente uscite e nessun rifornimento di utenti. Dobbiamo, insomma, trovare il modo per riconoscere una rete consistente e well-formed. Per raggiungere il nostro scopo costruiamo un grafo di raggiungibilità (RG: reachability graph) utilizzando come nodi le classi della nostra rete. In questo grafo due nodi sono uniti tra loro da un arco solo se le due classi, a cui corri-

spondono i nodi, sono direttamente collegate tra loro. Naturalmente la direzione dell'arco sarà quella del flusso d'utenti tra le due classi. Consideriamo inoltre $S_1, ..., S_k$ come delle componenti fortemente connesse (SCC: strongly connected components) dell'RG appena creato. Un SCC può appartenere ad una delle seguenti quattro categorie:

Chiuso Un SCC è chiuso se il numero di utenti al suo interno rimane sempre costante.

Aperto Un SCC è aperto se può ricevere nuovi arrivi dall'esterno e se gli utenti possono uscire dal sistema una volta terminato il servizio.

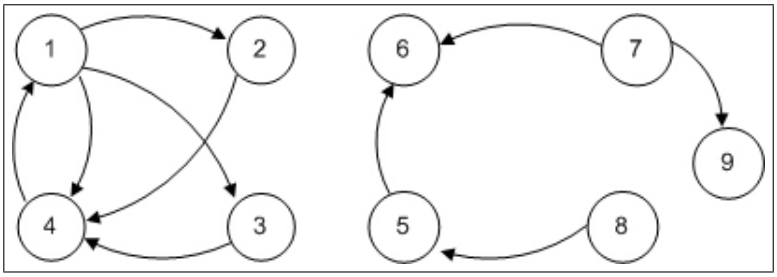
Transiente Un SCC si dice transiente se la cardinalità della popolazione al suo interno tende, a lungo termine, a zero.

Instabile Un SCC si dice instabile se riceve in input utenti dal mondo esterno o da nodi appartenenti ad SCC confinanti, senza avere la possibilità poi di farli uscire da sè. A lungo termine, in un SCC avremo quindi un accumularsi di utenti fino alla completa saturazione.

Utilizzando il concetto di SCC possiamo ora definire il concetto di rete well-formed: una rete si dice well-formed se e soltanto se non contiene al suo interno SCC transienti o instabili. In una rete così definita un SCC chiuso può essere interpretato come una catena chiusa mentre un SCC aperto come una catena aperta.

Proviamo ora a costruire un RC per l'esempio considerato. La figura

Figura 6.2: Grafo di raggiungibilità



6.2 mostra che il grafo da noi ottenuto è formato da due componenti connesse. La componente a sinistra è formata dalle prime quattro classi ed è isolata dal mondo esterno e da classi di altre componenti. Potremo considerare quindi la componente in esame come una SCC chiusa. Per analizzare la parte sinistra invece faremo uso di una accorgimento tecnico: una volta disegnata la componente con le classi appartenenti ad essa, aggiungeremo delle classi dummy per sostituire gli arrivi e le partenze dal mondo esterno. Avremo quindi che la classe 5 riceverà gli utenti provenienti da un'immaginaria classe 8, mentre i clienti della classe 7, al posto di uscire dal sistema, migreranno all'interno della classe 9. Nella parte sinistra del grafo non esistono SCC non banali, poiché non esistono componenti strettamente connesse (SCC banali sono naturalmente quelle composte da una sola classe). Potremo ora etichettare gli SCC da noi trovati secondo le tipologie precedentemente definite. Come già detto la componente a sinistra è chiusa, ma nella parte destra si possono notare delle anomalie sintomatiche di una rete che non è well-formed (come noi

desideriamo che sia). Si può notare, osservando il grafo, che la classe 6 non ha nessuna uscita (nessuna freccia infatti esce da quel nodo) ma può ricevere i clienti in output delle classi 5 e 7; riguardando le definizioni potremo quindi dedurre che la classe 6 è un SCC instabile. Concentrando ora l'analisi sul noto 7, possiamo notare che da esso gli archi escono solamente e, per questo, apparterrà alla categoria degli SCC transienti. Per finire considereremo il nodo 5 poiché possiede sia archi entranti che archi uscenti, e l'arco entrante porta all'interno della classe utenti provenienti dall'esterno del sistema. Le classi 7 e 8 (che ricordiamo essere fittizie) saranno per definizioni aperte.
Una volta analizzato il grafo è facile comprendere che il modello da noi considerato come esempio sarà consistente ma non well-formed, e quindi non sarà adatto per una successiva corretta analisi del sistema in esame. Per un approfondimento su questa tecnica di analisi di modelli a rete di code multiclasse si consiglia la lettura del riferimento bibliografico [Ka92].

6.3 Reti multiclasse

Nella sezione precedente abbiamo visto che una rete well-formed può essere formata da catene aperte o chiuse. Ricordiamo che una rete viene chiamata aperta se è formata solamente da catene aperte, mentre chiusa se è composta esclusivamente da catene chiuse. Infine, una rete viene chiamata mista, se composta sia da catene aperte che chiuse.
La popolazione di una rete non potrà essere più rappresentabile da un

numero, ma dovrà essere definita da un vettore $\mathbf{N} = (N_1, .., N_k)$, dove N_i, con $1 \leq i \leq k$ e k inteso come il numero di catene della rete, sarà la cardinalità della popolazione della catena i. Una catena aperta avrà $N_i = \infty$. In una catena chiusa invece la popolazione rimarrà costante, perché, per definizione, un utente appartenente ad una determinata catena non potrà mai appartenere ad un'altra.

Molto spesso, per semplicità, è utile considerare catene formate da una sola classe, poiché spesso l'uso di più classi può non essere fondamentale per l'accuratezza della rappresentazione ed appesantisce di molto le notazioni. Per fare un esempio consideriamo il parametro λ che, come più volte detto nei capitoli precedenti, rappresenta il tempo di interarrivo ad un nodo della rete. In una rete con catene formate da una sola classe avremo che $\lambda_{ir}(\mathbf{N})$, che denota il tempo d'interarrivo di una catena r al nodo i con una $\mathbf{N}$ vettore della popolazione della rete, sarà uguale a:

$$\lambda_{ir} = \gamma_{ir} + \sum_{j=1}^{M} \lambda_{jr} p_{jir}$$

dove p_{jir} l'elemento della matrice di routing P_r della catena r che indica la probabilità di un utente di passare dal nodo j al nodo i, e γ_{ir} è la frequenza di interarrivo dall'esterno della rete al nodo i di clienti appartenenti alla catena r. Una catena chiusa avrà, naturalmente, sempre che $\gamma_{ir} = 0$.

Se una catena r possiede più classi c, allora avremo che λ_{irc}, dove i

rappresenta il nodo, r la catena e c la classe, sarà uguale a:

$$\lambda_{irc} = \gamma_{irc} + \sum_{j=1}^{M} \sum_{d=1}^{nc(r)} \lambda_{jrd} p_{jdicr}$$

dove nc(r) rappresenta il numero di classi nella catena r, λ_{jrd} il parametro esponenziale d'interarrivo del nodo j appartenente alla classe d e alla catena r, e p_{jdicr} la probabilità che un cliente, uscito da un nodo j appartenente alla classe d, vada al nodo i appartenente alla classe c, con d e c classi della catena r. Come si può notare l'inserimento di più classi differenti in una catena appesantisce molto la notazione e complica i calcoli inserendo una doppia sommatoria. Per semplicità noi considereremo nelle sezioni successive per lo più catene con una sola classe ed inoltre considereremo una sola coda per nodo e verrà condivisa dagli utenti di ogni catena.

6.4 Modelli a rete di code Markoviane multiclasse

Un modello a rete di code si dice Markoviano se è rappresentabile da un processo stocastico di Markov. L'evoluzione di un modello di questo tipo può essere definita da una catena di Markov ergodica[1] a tempo conti-

[1]Una catena di Markov è ergodica se è irriducibile (e cioè se ogni stato può essere raggiunto da ogni altro in un tempo finito) e se è formata da stati aperiodici e ricorrenti non nulli (cioè se considerato un nodo i, il tempo medio di ricorrenza in i è finito).

nuo, uno spazio E di stati **S** discreti e una matrice, chiamata generatore infinitesimale, Q. Definiti questi elementi è possibile, qualora la matrice di diramazione P sia irriducibile, trovare la distribuzione di probabilità $\pi = \{\pi(\mathbf{S}), \mathbf{S} \in E\}$ risolvendo il sistema di equazioni di bilanciamento globale:

$$\pi Q = \mathbf{0} \qquad con \sum_{\mathbf{S} \in E} \pi(\mathbf{S}) = 1 \tag{6.1}$$

dove 0 indica un vettore formato da soli zeri. Ampliamo ora il concetto di catena di Markov ad un modello con una popolazione formata da R catene differenti, tutte monoclasse. Avremo che ogni catena r, con $1 \leq r \leq R$ possiederà un matrice di routing P_r e μ_{ir} indicherà il service rate del nodo i per gli utenti della catena r. Consideriamo per il momento che i nodi siano di grandezza infinita e che quindi non si possa verificare nessun tipo di blocco. Con questo presupposto, possiamo allora considerare l'insieme degli stati E come

$$E = \{\mathbf{S} : \mathbf{S} = (\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3, ..., \mathbf{n}_M), e \sum_{i=1}^{M} \mathbf{n}_i = \mathbf{N}\} \tag{6.2}$$

dove M indica il numero dei nodi della rete, $\mathbf{n}_i$, con $1 \leq i \leq M$, sono i vettore delle popolazioni delle catene nel nodo i e **N** è il vettore della popolazione totale. Per fare un esempio, n_{ir} indicherà il numero di utenti della catena r presenti nel nodo i.

La matrice di transizione di stato Q sarà formata da elementi $q(\mathbf{S}, \mathbf{S}')$, con$\mathbf{S}, \mathbf{S}' \in E$, definiti nel seguente modo:

$$q(\mathbf{S}, \mathbf{S}') = \mu_{jr} p_{jir}$$

Per fare un esempio, consideriamo una a rete composta da sue sole catene e i due stati:

$$\mathbf{S} = \begin{pmatrix} (2,1) \\ (3,2) \\ (2,2) \\ (4,1) \\ (3,4) \end{pmatrix} \qquad \mathbf{S}' = \begin{pmatrix} (2,1) \\ (3,2) \\ (2,1) \\ (4,2) \\ (3,4) \end{pmatrix}$$

In questo caso il cambiamento nei due stati, che avviene ai nodi 3 e 4 e nella catena 2 sarà segnalato nell'elemento $q(\mathbf{S}, \mathbf{S}') = \mu_{32}\pi_{342}$.

6.5 Reti di code multiclasse in forma prodotto

Le reti di code in forma prodotto sono una classe particolare di modelli, e possiedono la caratteristica d'avere una soluzione esprimibile con una formula del tipo

$$\pi(\mathbf{S}) = \frac{1}{G} d(n) \prod_{i=1}^{M} f_i(n_i) \tag{6.3}$$

dove con G si indica la costante di normalizzazione e con n la popolazione totale della rete.

Abbiamo già visto esempi di questo tipo di modelli nelle sezioni 4.1 e 4.2, dove però abbiamo preso in considerazione solamente popolazioni di utenti appartenenti ad un'unica classe. Ora, in questo capitolo, analizzeremo una classe di modelli multiclasse. Avremo quindi che la soluzione

Tipo	Disciplina di servizio	Distribuzione del tempo di servizio
1	FIFO	esponenziale
2	PS (Processor Sharing)	Coxiana
3	IS (Server Infiniti)	Coxiana
4	LIFO con Prelazione	Coxiana

Tabella 6.1: Tipi di nodi BCMP.

in forma prodotto di modelli con popolazioni appartenenti a R catene differenti prenderà questa forma

$$\pi(\mathbf{S}) = \frac{1}{G} \prod_{r=1}^{R} d_r(N_r) \prod_{i=1}^{M} \prod_{r=1}^{R} f_{ir}(n_{ir}) \tag{6.4}$$

dove con N_r si intente la popolazione della rete appartenente ad una catena r. Per una questione di leggibilità, nell'equazione appena scritta, consideriamo solamente catene monoclasse.

6.5.1 Reti BCMP

I modelli a rete di code BCMP sono un'estensione delle reti di Jackson e Gordon-Newell introdotte nelle sezioni 4.1 e 4.2 di questo elaborato. A questa classe di modelli appartengono reti aperte, chiuse o miste, multiclasse e di topologia probabilistica arbitraria. Inoltre i centri di servizio possono essere di 4 tipologie, che si differenziano per disciplina di servizio e per distribuzione del tempo di servizio (v. tabella 6.1)

A differenza di quanto fatto nei paragrafi precedenti, per questa tipologia di modelli considereremo catene con più classi al loro interno, e per questo ora dovremo introdurre delle ulteriori notazioni. Con C_r intenderemo l'insieme delle classi appartenenti alla catena r, mentre con C_{ir} l'insieme delle classi della catena r appartenenti al nodo i. La già vista matrice di routing P_r dovrà essere composta da elementi che indichino anche la classe di partenze e quella di destinazione, e quindi dovranno essere scritti nella forma p_{icjdr}, dove i e j sono i nodi di partenza e destinazione, $c \in C_{ir}$ e $d \in C_{jr}$ sono le classi e r è la catena. Il tasso di servizio di denota con μ_{icr}, mentre il tasso di arrivo dall'esterno sarà indicato con γ_{icr}, dove, in entrambi i casi, i indica il nodo, c la classe e r la catena. Il tasso medio relativo di arrivo totale al nodo i λ_{icr}, con $c \in C_{ir}$, $1 \leq i \leq M$, $1 \leq r \leq R$, si ottiene dalla seguente equazione:

$$\lambda_{icr} = \gamma_{icr} + \sum_{j=1}^{M} \sum_{d \in C_{jr}} \lambda_{jdr} p_{jdicr} \tag{6.5}$$

Vista la varietà di nodi utilizzabili per questi modelli, e considerato l'utilizzo di catene multiclasse, è facile intuire che lo stato di un modello così pensato dovrà essere molto dettagliato e dovrà includere il numero di utenti per ogni nodo e ogni classe, lo stadio esponenziale delle distribuzioni Coxiane e l'ordine nella coda degli utenti (definito dalla disciplina di servizio scelta).

Il teorema seguente fornisce gli strumenti per risolvere la rete calcolando la distribuzione stazionaria della probabilità di stato.

Teorema BCMP 6.5.1 *La distribuzione stazionaria di stato per reti BCMP si può esprimere come:*

$$\pi(\mathbf{n_1}, \mathbf{n_2}, ..., \mathbf{n_M}) = \frac{1}{G} d(\mathbf{n}) \prod_{i=1}^{M} f_i(\mathbf{n_i}) \tag{6.6}$$

dove G è la costante di normalizzazione:

$$G = \sum_{n \in E} \prod_{i=1}^{M} f_i(\mathbf{n_i}) \tag{6.7}$$

Le funzioni $d(\mathbf{n})$ per reti aperte o miste saranno scritte come $d(\mathbf{n}) = \prod_{k=0}^{n-1} \gamma(k)$ oppure $d(\mathbf{n}) = \prod_{r=1}^{R} \prod_{k=0}^{n_r-1} \gamma_r(k)$, dove $\gamma(n)$ indica il tasso d'arrivo dipendente dal numero totale di utenti nella rete, mentre $\gamma_r(n_r)$ denota il tasso d'arrivo dipendente dal numero d'utenti di una determinata catena r. Per le reti chiuse invece, $d(\mathbf{n}) = 1$.
Le funzioni $f_i(n_i)$ dipendono invece dalla tipologia dei nodi utilizzati:

$$\begin{aligned} f_i(n_i) &= n_i! \prod_{r=1}^{R} \frac{\rho_{ir}^{n_{ir}}}{n_{ir}!}, \quad \textit{se il nodo } i \textit{ è di tipo 1,2 o 4} \\ f_i(n_i) &= \prod_{r=1}^{R} \frac{\rho_{ir}^{n_{ir}}}{n_{ir}!}, \quad \textit{se il nodo } i \textit{ è di tipo 3} \end{aligned}$$

con $\rho_{ir} = \lambda_{ir}/\mu_{ir}$, $e_{ir} = \sum_{c \in C_{ir}} e_{icr}$, $\mu_{ir} = \sum_{c \in C_{ir}} \mu_{icr}$, $1 \leq i \leq M$, $1 \leq r \leq R$.

La dimostrazione di questo teorema (v. [BCMP75]) in questo elaborato verrà omessa. Infine è utile sottolineare che per le reti chiuse la soluzione per l'equazione (6.6) esiste sempre se le matrici P_r sono irriducibili.

Capitolo 7

Modelli a rete di code con blocco multiclasse

Nel quinto capitolo abbiamo introdotto le reti con nodi a buffer finito e le tecniche di blocco più utilizzate per gestire le situazioni di nodo saturo, mentre nel sesto capitolo abbiamo considerato le reti multiclasse, con la loro caratteristica di poter differenziare il comportamento degli utenti al loro interno. In questo capitolo uniremo le caratteristiche di queste due tipologie di reti di code considerando reti di code con blocco multiclasse. Nelle prime tre sezioni considereremo tre tipologie di blocco applicate a reti di code Markoviane multiclasse, e le modifiche che bisognerà apportare agli strumenti d'analisi visti al capitolo 6.4 per uno studio corretto di queste reti. Nell'ultima sezione, infine, analizzeremo le reti di code multiclasse in forma prodotto con diverse tecniche di blocco.

7.1 Reti di code Markoviane multiclasse con blocco RS-RD

Consideriamo ora alcuni casi di reti chiuse con code finite analizzabili con catene di Markov. Per questo tipo di reti dovremo inevitabilmente prendere in considerazioni i casi di blocchi, dovuti a nodi con code sature e quindi incapaci di ricevere nuovi utenti (si consiglia la lettura del capitolo 5). Vediamo ora l'analisi di una rete di questo tipo che utilizzi un sistema di blocco RS-RD. Per definire al meglio il modello avremo bisogno di nuovi elementi:

$\mathbf{a_i}$: E' il vettore che indica il numero minimo possibile di utenti al nodo i ed è definito come:

$$\mathbf{a_i} = \{a_{ir} : a_{ir} = max\{0, N_r - \sum_{j=1, j\neq i}^{M} B_j\}\} \tag{7.1}$$

dove r indica una catena e B_j la lunghezza della coda del nodo j. Naturalmente avremo $1 \leq R \leq K$ e $1 \leq i, j \leq M$.

$\delta(n_i)$: è una funzione definita come:

$$\delta(n_{ir}) = \begin{cases} 0, & \text{se } n_{ir} = 0; \\ 1, & \text{se } n_{ir} > 0. \end{cases} \tag{7.2}$$

$b_i(\mathbf{n_i})$: è la funzione:

$$b_i(\mathbf{n_i}) = \begin{cases} 0, & \text{se } \sum_{r=1}^{R} n_{ir} = Bi; \\ 1, & \text{se } \sum_{r=1}^{R} n_{ir} < B_i. \end{cases} \tag{7.3}$$

Si noti che questa funzione prende in considerazione la popolazione totale presente nel nodo i, e non quella appartenente ad una specifica catena r, perché le code, nei nostri ragionamenti sono condivise da tutti.

I_{ir} **:** E' una matrice, di grandezza MxR, formato da elementi tutti nulli tranne per l'elemento in posizione ir che avrà valore 1.

Potremo ora definire l'insieme degli stati E_{RS-RD} come:

$$E_{RS-RD} = \{(\mathbf{n_1}, \mathbf{n_2}, ..., \mathbf{n_M}) : \forall \text{ catena r } a_{ir} \leq n_{ir}, \sum_{r=1}^{R} n_{ir} \leq B_i, \\ 1 \leq i \leq M, \sum_{i=1}^{M} n_{ir} = N_r\} \tag{7.4}$$

La matrice Q sarà composta da elementi $q(\mathbf{S}, \mathbf{S'})$ con $\mathbf{S}, \mathbf{S'} \in E_{RS-RD}$, definiti nel seguente modo:

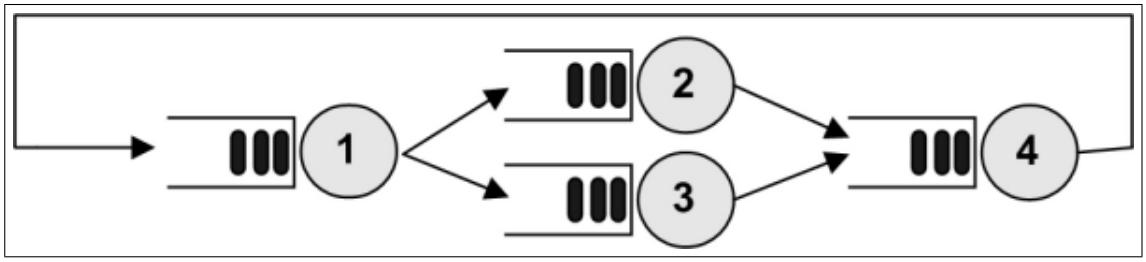
$$q(S, S') = \begin{cases} \delta(n_{jr})\mu_{jr}p_{jir}b_i(\mathbf{n_i}), & \text{se } \mathbf{S'} = \mathbf{S} + I_{ir} - I_{jr}; \\ -\sum_{\mathbf{S''} \in E_{RS-RD}, \mathbf{S''} \neq \mathbf{S}} q(\mathbf{S}, \mathbf{S''}) , & \text{se } \mathbf{S'} = \mathbf{S}; \\ 0, & \text{altrimenti.} \end{cases}$$

In questa definizione facciamo un'operazione che inizialmente può sembrare strana e cioè una sottrazione tra un vettore di stato e la matrice I_{ir}. In realtà bisogna pensare a come è costruito il vettore $\mathbf{S}$, che, da definizione, sarà composto da M vettori di lunghezza R. Un vettore di stato potrà essere dunque visto come una matrice MxR in cui le righe indicano i nodi, e le colonne le catene. La sottrazione tra un vettore S e la matrice I_{ir} non farà altro quindi che sottrarre un'unità al nodo i nel contatore

della popolazione della catena r. Questa definizione è completa solo se si considera la rete come chiusa. Nel caso di una rete aperta bisognerà prendere in considerazione anche i casi in cui $\mathbf{S}' = \mathbf{S} + I_{jr}$ e $\mathbf{S}' = \mathbf{S} - I_{jr}$, e cioè i casi in cui un utente di una catena entra nel sistema o ve ne esce.

Esempio

Figura 7.1: Esempio di rete di code



Consideriamo una rete chiusa composta da 4 nodi, con un server ciascuno e con una capienza tale da poter contenere massimo 4 utenti ciascuno, disposti come in figura 7.1 e una popolazione di utenti appartenenti a due catene e rappresentati dal vettore $\mathbf{N} = \{N_1 = 6,\ N_2 = 6\}$ e ogni catena possiederà una matrice di routing P_r:

$$P_1 = \begin{pmatrix} 0 & 0.3 & 0.7 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad P_2 \begin{pmatrix} 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Inoltre, per semplicità, consideriamo il tasso di servizio per gli utenti di una determinata catena costante, e cioè che $\mu_{i1} = 0.5$ e $\mu_{i1} = 0.7 \ \forall 1 \leq i \leq M$. Vediamo ora 3 potenziali stati del sistema:

$$\mathbf{S} = \begin{pmatrix} (2,2) \\ (1,2) \\ (2,1) \\ (1,1) \end{pmatrix} \quad \mathbf{S}' = \begin{pmatrix} (2,3) \\ (1,2) \\ (2,1) \\ (1,0) \end{pmatrix} \quad \mathbf{S}'' = \begin{pmatrix} (1,2) \\ (2,2) \\ (2,1) \\ (1,1) \end{pmatrix}$$

Posto S come stato di partenza, proviamo a considerare gli elementi della matrice Q, $q(\mathbf{S}, \mathbf{S}')$ e $q(\mathbf{S}, \mathbf{S}'')$. E' facile intuire che il primo elemento valga 0. Infatti, se si guarda la popolazione del primo nodo nello stato S', si può notare che questa supera la capienza totale disponibile poiché il nodo, secondo questo stato, dovrebbe ospitare 2 utenti appartenenti alla prima catena e due appartenenti alla seconda. La formula che calcola questo elemento sarà quindi:

$$q(\mathbf{S}, \mathbf{S}') = \delta(n_{42})\mu_{12}p_{412}b_1(\mathbf{n_1}) = 1 * 0.7 * 1 * 0 = 0$$

L'elemento $q(\mathbf{S}, \mathbf{S}'')$ potrà invece essere calcolato nel seguente modo:

$$q(\mathbf{S}, \mathbf{S}'') = \delta(n_{11})\mu_{11}p_{121}b_2(\mathbf{n_2}) = 1 * 0.5 * 0.3 * 1 = 0.15$$

7.2 Reti di code Markoviane multiclasse con blocco BAS

Se consideriamo la tecnica di blocco BAS, possiamo facilmente comprendere che non è sufficiente definire lo stato di un nodo semplicemente con il vettore degli utenti delle diverse catene presenti in esso in un determinato istante. Infatti lo stato $\mathbf{S}_i$ di un nodo i deve essere definito così:

$$\mathbf{S}_i = (\mathbf{n}_i, s_i, \mathbf{m}_i)$$

dove $\mathbf{n}_i$ indica il numero di utenti presenti al nodo i, s_i il numero di server bloccati al nodo i in attesa che si liberi un nodo di destinazione, e $\mathbf{m}_i$ la lista dei nodi bloccati in attesa che si liberi il nodo i:

$$\mathbf{m}_i = \begin{cases} \emptyset & \text{se } \sum_{i=0}^{R} n_{ir} \leq B_i \\ (m_1, m_2, ..., m_{\tau(i)}) & \text{se } \sum_{i=0}^{R} n_{ir} = B_i \end{cases}$$

dove $\tau(i)$ indica il numero di server bloccati al nodo i e m_j è il j-esimo nodo bloccato in attesa che i si liberi, con $1 \leq j \leq \tau(i)$. Prima di definire gli elementi della matrice Q, introduciamo un nuovo simbolo, che ci sarà subito utile per snellire la definizione:

$$n_{iTOT} = \sum_{r=1}^{R} n_{ir}, \qquad \text{con } 1 \leq i \leq M$$

Nella tabella 7.1 vengono analizzati gli elementi $q(\mathbf{S}, \mathbf{S}'), \mathbf{S}, \mathbf{S}' \in E_{BAS}$ della matrice Q_{BAS}. Come si può vedere, a differenza di Q_{RS-RD}, non sempre una transizione di stato consiste nel passaggio di un utente da un

nodo ad un altro. Può accadere che il nodo di destinazione sia pieno e che l'utente sia obbligato a bloccare il server da cui ha appena ottenuto un servizio. In questo caso il cambiamento di stato consiste, per il nodo di partenza i nel aumentare di un'unità il valore del contatore dei server bloccati, mentre per il nodo di arrivo j nell'aggiungere il nodo i nella lista $\mathbf{m}_j$ dei nodi in attesa di servizio. Le discipline che regolamentano l'ordine in cui j dovrà poi servire i nodi in attesa possono essere diverse, anche se la più semplice rimane quella di considerare la lista come una coda e di utilizzare quindi una disciplina FBFU (First Blocked First Unblocked).

Nella tabella vengono utilizzate delle funzioni che non sono state ancora introdotte:

Ins$(\mathbf{m}_i, j)$: inserisce j nella lista $\mathbf{m}_i$

Update(j, k) : assegna a k il valore del primo elemento di$\mathbf{m}_{ij}$ e lo rimuove dalla lista.

La funzione Update viene utilizzata nel secondo caso preso in considerazione nella tabella, ed è interessante soffermarci ad analizzare che situazione considera. In questo secondo caso si può leggere che nello stato di partenza **S** il nodo j possiede la lista dei nodi in attesa non vuota. In questa situazione, quando un utente della catena r esce dal nodo j per andare al nodo i, non avremo, come in altri casi, che $n'_{ir} = n_{ir} + 1$ e $n'_{jr} = n_{jr} - 1$, perché il nodo j ha in lista $\mathbf{m}_j$ dei nodi che attendono di mandare i propri utenti in j. Se consideriamo k, come il primo nodo che

$q(\mathbf{S}, \mathbf{S}')$	Relazioni tra **S** e **S'**
$\delta(n_{jr})\mu_{jr}p_{jir}$	$\mathbf{S} : s_j < min\{n_{jTOT}, K_j\}$, $m_j = \emptyset$, $n_{iR} < B_i$, $i \neq j \Rightarrow$ $\mathbf{S}' = \mathbf{S}'_j(\mathbf{n}'_j, s'_j, \mathbf{m}'_j)$: $n'_{jr} = n_{jr} - 1$, $s'_j = s_j$, $\mathbf{m}'_j = \mathbf{m}_j$ $\mathbf{S}'_i(\mathbf{n}'_i, s'_i, \mathbf{m}'_i)$: $n'_{ir} = n_{ir} + 1$, $s'_i = s_i$, $\mathbf{m}'_i = \mathbf{m}_i$ $\mathbf{S}'_h = \mathbf{S}_h \quad \forall h \neq i, j$ oppure $\mathbf{S} : s_j < min\{n_{jTOT}, K_j\}$, $n_{iTOT} = B_i$, $i \neq j \Rightarrow$ $\mathbf{S}' = \mathbf{S}'_j(\mathbf{n}'_j, s'_j, \mathbf{m}'_j)$: $n'_{jr} = n_{jr}$, $s'_j = s_j + 1$, $\mathbf{m}'_j = \mathbf{m}_j$ $\mathbf{S}'_i(\mathbf{n}'_i, s'_i, \mathbf{m}'_i)$: $n'_{ir} = n_{ir}$, $s'_i = s_i$, $\mathbf{m}'_i = \mathbf{Ins}(\mathbf{m}_i, j)$ $\mathbf{S}'_h = \mathbf{S}_h \quad \forall h \neq i, j$
$\delta(n_{jr})\mu_{jr}p_{jir}b_i(\mathbf{n}_{i)}$	$\mathbf{S} : s_j < min\{n_{jR}, K_j\}$, $m_j \neq \emptyset \quad \Rightarrow$ $\mathbf{S}' : Update(j, k)$, if $i \neq k$then $n'_{ir} = n_{ir} + 1$, $n'_{jr} = n_{jr} - 1$, $n'_{jo} = n_{jo} + 1$, $n'_{ko} = n_{ko} - 1$, con $r, o \in R$
$-\sum_{\mathbf{S}'' \in E_{BAS}, \mathbf{S}'' \neq \mathbf{S}} q(\mathbf{S}, \mathbf{S}'')$	$\mathbf{S} = \mathbf{S}'$
0	altrimenti

Tabella 7.1: Generatore infinitesimale Q_{BAS}.

ha diritto di mandare un proprio utente in j, allora avremo contemporaneamente anche una situazione in cui $n'_{ko} = n_{ko} - 1$ e $n'_{jo} = n_{jo} + 1$. Avremo cioè che il vettore $\mathbf{n}_j$ varierà nel valore di due suoi elementi, ma che comunque n_{jR} rimarrà costante.

Esempio

Consideriamo ora l'esempio utilizzato nella sezione 7.1, utilizzando però blocchi di tipo BAS. Un possibile stato $\mathbf{S}$ può essere:

$$\mathbf{S} = \begin{pmatrix} ((2,2),1,[4]) \\ ((0,1),0,\emptyset) \\ ((3,1),0,[1]) \\ ((1,2),1,\emptyset) \end{pmatrix}$$

Il modello nello stato $\mathbf{S}$ ha due server bloccati (s_1, s_4) e il nodo 1 ed il nodo 3 hanno una lista non nulla di nodi con utenti che attendono di potervi accedere. Un possibile sviluppo di questa situazione potrà consistere nel passaggio di un utente della catena 1 dal nodo 3 al nodo 4, e dal successivo sblocco del nodo 1 (che attendeva che il nodo 3 si liberasse), che potrà far transitare un suo utente (in questo caso appartenente alla catena 2) al nodo 3, e riceverne uno nuovo della catena 1 dal centro di servizio numero 4 (che conseguentemente, potrà sbloccarsi).
Si arriverebbe quindi allo stato $\mathbf{S}'$, dove tutti i server sono liberi e tutte

le liste dei nodi d'attesa sono vuote:

$$\mathbf{S}' = \begin{pmatrix} ((3,1),0,\emptyset) \\ ((0,1),0,\emptyset) \\ ((2,2),0,\emptyset) \\ ((1,2),0,\emptyset) \end{pmatrix}$$

7.3 Reti di code Markoviane multiclasse con blocco BBS

Consideriamo ora una rete di code Markoviana chiusa multiclasse con blocco di tipo BBS. Come già visto nel capitolo 5, questa tipologia di blocco si contraddistingue per il fatto di far dichiarare la destinazione ad ogni utente prima del servizio ad un determinato nodo.
Prima di prendere in considerazione l'insieme degli stati, introduciamo una funzione booleana che ci sarà molto utile in seguito:

$$Blocked(i) = \begin{cases} true, & \text{se } \exists j \in Dest_i : B_j < N_R \\ false, & \text{se } B_j \geq N_{TOT} \forall j \in Dest_i \end{cases}$$

dove $Dest_i = \{j : \exists r$ tale che $p_{ijr} > 0, 1 \leq i,j \leq M \, e \, 1 \leq r \leq R\}$ indica tutti i possibili nodi destinazione degli utenti del nodo i e $N_{TOT} = \sum_{r=0}^{R} N_r$. La funzione Blocked sarà quindi vera per un nodo i solamente se esistono nei nodi bloccabili (é non possono contenere l'intera popolazione), tra quelli raggiungibili dai clienti che si trovano al nodo i.

Possiamo ora definire lo stato di un nodo i nel seguente modo:

$$\mathbf{S}_i = \begin{cases} (\mathbf{n}_i, \mathbf{NS}_i), & \text{se Blocked(i)=true} \\ (\mathbf{n}_i), & \text{se Blocked(i)=false} \end{cases} \tag{7.5}$$

dove $\mathbf{n}_i$ è il vettore di popolazione al nodo i e $\mathbf{NS}_i$ è un vettore dove $NS_{i,k}$ indica il numero di utenti del nodo i che, tra quelli che si stanno servendo, hanno dichiarato come destinazione il nodo k. Come si può vedere dalla definizione (7.5), il vettore $\mathbf{NS}_i$ se il nodo non è bloccabile. Una volta definito lo stato dei singolo nodi, possiamo definire l'insieme degli stati per questa tipologia di blocco:

$$E_{BBS} = \{(\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_M) : \mathbf{S}_i = (\mathbf{n}_i, \mathbf{NS}_i) \text{ e } \forall \text{ catena r } a_{ir} \leq n_{ir}, \sum_{r=1}^{K} n_{ir} \leq B_i,\ 1 \leq i \leq M,\ \sum_{i=1}^{M} n_{ir} = N_r\}$$

Osservando la tabella 7.2, dove viene definita la matrice Q_{BBS}, possiamo evidenziare una caratteristica che distingue questa tipologia di blocco dagli altri casi presi in esame. Consideriamo il caso in cui un utente passi da un nodo j ad un nodo i e che il server liberato al nodo j si occupi di servire un cliente che dichiara come destinazione successiva il nodo k. Questo tipo di transizione di stato potrà avvenire con una frequenza pari a $\mu_{ir} p_{ikr'}$ dove l'utente che lascia il nodo i appartiene alla catena r, e quello che si appresta ad ottenere il servizio appartiene invece alla catena r'. Questo evidenzia che la tipologia di blocco BBS, per definire le diverse transizioni di stato, dovrà prendere in considerazione la matrice di routing non dell'utente che esce dal nodo, ma da quello che viene servito dal server

al suo posto, proprio per la caratteristica della BBS di far dichiarare la destinazione prima di fornire il servizio. Analizziamo ora i diversi casi presi in considerazione nella tabella. I casi più significativi sono identificati dalle etichette dall'uno al tre e considerano il passaggio di un utente dal nodo j al nodo i nelle diverse condizioni in cui i nodi si possono trovare. Guardiamo la prima riga della tabella: nello stato iniziale **S** si considera il nodo di provenienza j come non bloccabile, mentre il nodo i può essere non bloccabile oppure avere una popolazione tale da occupare tutti i server a disposizione (obbligando il nuovo utente ad entrare in coda). In questo caso avremo che la transizione non dovrà prendere in considerazione i vettori **NS** dei due nodi. Infatti j non possiede, per definizione di stato di un nodo, quel vettore e i, o non lo possiede, oppure non deve modificarlo é nessun utente ha cominciato un nuovo servizio in quanto il nuovo cliente è solamente entrato in coda. E' importante anche sottolineare che in questo caso l'inizio del servizio di un nuovo utente al nodo j, nel server appena liberato, non influenza il valore dell'elemento della matrice Q in esame, perché j non è un nodo bloccabile.
Il punto 2 considera il nodo j bloccabile, con la popolazione al suo interno che nel primo caso è minore o uguale al numero dei server, e nel secondo è strettamente maggiore. Naturalmente un numero d'utenti maggiore a quello dei server di un nodo, implica che ci sia almeno un utente in coda. Nel secondo caso del punto 2 bisognerà quindi anche prendere in considerazione l'aggiornamento del vettore $\mathbf{NS_j}$ anche per il cliente che

esce dalla coda. Avremo quindi un doppio aggiornamento di quel vettore, poiché si dovrà togliere un'unità dal contatore per la destinazione del cliente che è uscito dal nodo e aggiungere un'unità dal contatore della destinazione dichiarata dal cliente appena entrato nel server.
Nel terzo punto, infine, si prendono in considerazione i casi in cui anche il nodo i è bloccabile e possiede inoltre un numero d'utenti al suo interno minore del numero dei suoi server. In questo caso, l'utente uscito da j ed entrato in i, dovrà subito dichiarare la sua prossima destinazione, poiché può subito ricevere il servizio.
Infine, il punto 4, considera gli elementi che stanno sulla diagonale della matrice Q.

7.4 Reti di code multiclasse in forma prodotto con blocco

Come già visto nel capitolo 6.5 alcune reti di code possono essere risolte con soluzioni in forma prodotto del tipo:

$$\pi(\mathbf{S}) = \frac{1}{G} \prod_{r=1}^{R} V_r(\mathbf{n_r}) \prod_{i=1}^{M} \prod_{r=1}^{R} g_{ir}(n_{ir}) \tag{7.6}$$

in cui G è una costante di normalizzazione, $\mathbf{n_r}$ indica il vettore della popolazione mentre n_{ir} rappresenta il numero di utenti appartenenti alla catena r al nodo i.
Le reti di code con blocco possono possedere soluzioni in forma prodotto

#	$q(\mathbf{S}, \mathbf{S}')$	Relazioni tra **S** e **S'**
1	$\delta(n_{jr})\mu_{jr}p_{jir}$	$\mathbf{S}: \neg Blocked(j)$ $\neg Blocked(i) \vee (Blocked(i) \wedge n_{iTOT} \geq K_i)$ $\mathbf{S}': n'_{jr} = n_{jr} - 1,\ n'_{ir} = n_{ir} + 1$
	$\delta(n_{jr})\mu_{jr}p_{jir}p_{ikr'}$	$\mathbf{S}: \neg Blocked(j)$ $Blocked(i) \wedge n_{iTOT} < K_i$ $\mathbf{S}': n'_{jr} = n_{jr} - 1,\ n'_{ir} = n_{ir} + 1,\ NS'_{i,k} = NS_{i,k} + 1$
2	$NS_{j,i}\mu_{jr}b(\mathbf{n_i})/z_j(\mathbf{n_j})$	$\mathbf{S}: Blocked(j) \wedge n_{jTOT} \leq K_j$ $\neg Blocked(i) \vee (Blocked(i) \wedge n_{iR} \geq K_i)$ $\mathbf{S}': n'_{jr} = n_{jr} - 1,\ NS'_{j,i} = NS_{j,i} - 1,\ n'_{ir} = n_{ir} + 1$
	$NS_{j,i}\mu_{jr}p_{jhr'}b(\mathbf{n_i})/z_j(\mathbf{n_j})$	$\mathbf{S}: Blocked(j) \wedge n_{jTOT} > K_j$ $\neg Blocked(i) \vee (Blocked(i) \wedge n_{iTOT} \geq K_i)$ $\mathbf{S}': n'_{jr} = n_{jr} - 1,\ NS'_{j,i} = NS_{j,i} - 1$ $n'_{ir} = n_{ir} + 1,\ NS'_{j,h} = NS_{j,h} + 1$
3	$NS_{j,i}\mu_{jr}p_{ikr}/z_j(\mathbf{n_j})$	$\mathbf{S}: Blocked(j) \wedge n_{jTOT} \leq K_j$ $Blocked(i) \wedge n_{iTOT} < K_i$ $\mathbf{S}': n'_{jr} = n_{jr} - 1,\ NS'_{j,i} = NS_{j,i} - 1$ $n'_{ir} = n_{ir} + 1,\ NS'_{i,k} = NS_{i,k} + 1$
	$NS_{j,i}\mu_{jr}p_{ijr'}p_{ikr}/z_j(\mathbf{n_j})$	$\mathbf{S}: Blocked(j) \wedge n_{jTOT} > K_j$ $Blocked(i) \wedge n_{iTOT} < K_i$ $\mathbf{S}': n'_{jr} = n_{jr} - 1,\ NS'_{j,i} = NS_{j,i} - 1$ $NS'_{j,h} = NS_{j,h} + 1,\ n'_{ir} = n_{ir} + 1$ $NS'_{i,k} = NS_{i,k} + 1$
4	$-\sum_{\mathbf{S}'' \in E_{BBS}, \mathbf{S}'' \neq \mathbf{S}} q(\mathbf{S}, \mathbf{S}'')$	$\mathbf{S} = \mathbf{S}'$
5	0	altrimenti

Tabella 7.2: Generatore infinitesimale Q_{BBS}.

solo se soddisfano alcuni vincoli, che variano a seconda della topologia della rete del tipo di blocco adottato. Nella tabella 7.3 vengono prese in considerazione alcune topologie di rete abbinate alle tecniche di blocco utilizzate finora, e per ogni combinazione tra queste due caratteristiche vengono definite le funzioni V e g_i ed i vincoli che la rete deve soddisfare. Le condizioni presenti in tabella sono così definite:

Non-empty condition: per le reti chiuse viene richiesto che al più un nodo possa essere vuoto;

Strictly non-empty (SNE) condition: questa condizione viene soddisfatta quando nessun nodo della rete può diventare mai vuoto;

Condition A: ogni nodo i con coda a capacità finita, dev'essere tale da essere l'unica destinazione di ogni sua sorgente. Deve quindi valere il vincolo per cui se $p_{ji} > 0$ allora $p_{ji} = 1$ per ogni nodo j della rete;

Condition B: ad ogni istante, al più un solo nodo può essere bloccato.

I simboli PFi, con $1 \leq i \leq 4$ in tabella indicano quali funzioni bisogna applicare alla soluzione in forma prodotto per le diverse topologie e per le tecniche di blocco considerate. Per PF1 e PF2 abbiamo che $V(n) = 1$. La forma PF1 è applicabile a reti di code con blocco multiclasse con nodi di tipo BCMP e con classi con capacità indipendenti e le funzioni $g_i(n_{ir})$, $\forall 1 \leq i \leq M$ e $\forall 1 \leq r \leq R$, sono così definite:

$$g_i(n_{ir}) = \left(\frac{x_{ir}}{\mu_{ir}}\right)^{n_i} \tag{7.7}$$

con

$$x_{ir} = \sum_{j=1}^{M} x_{jr} p_{jir} + \lambda_r p_{0ir} \tag{7.8}$$

La soluzione in forma prodotto PF2 solitamente viene utilizzata per reti a singola classe con nodi esponenziali e con un tasso di servizio indipendente dal carico. E' ipotizzabile, comunque, che possa risultare corretta anche per modelli multicatena a coda condivisa, ma rimandiamo la dimostrazione formale di questa congettura ad un successivo lavoro di analisi. Per reti di questo tipo, la funzione $g_i(n_{ir})$ è definita come segue:

$$g_i(n_{ir}) \frac{1}{\epsilon_{ir}^{n_{ir}}} \tag{7.9}$$

con $\epsilon_{\mathbf{r}} = (\epsilon_1, ..., \epsilon_M)$ e tale che $\epsilon_{\mathbf{r}} = \epsilon_{\mathbf{r}} \mathbf{P}_{\mathbf{r}}^{'}$. Infine $\mathbf{P}_{\mathbf{r}}^{'}$ è tale che:

$$\mathbf{P}_{\mathbf{r}}^{'} = \| p_{ijr}^{'} \|, \quad \text{con } p_{ijr}^{'} = \begin{cases} \mu_{jr} p_{jir}, & \text{se } i \neq j; \\ 1 - \sum_{j \neq i} p_{jir}^{'}, & \text{se } i = j. \end{cases} \tag{7.10}$$

con $1 \leq i, j \leq M$.

La soluzione PF3 è valida per reti multiclasse con server centrale e con il tipo di classe di un utente fissato nel sistema. Inoltre il state-dependent routing dovrà dipendere dal tipo di classe e le funzioni di blocco dipenderanno dai nodi e dalle classi. I nodi di questa rete dovranno avere un distribuzione arbitraria del tempo di servizio e una disciplina di scheduling simmetrica oppure, con disciplina arbitraria, dovranno possedere una distribuzione del tempo di servizio esponenziale identica per ogni nodo e classe. Per la definizione delle funzioni $V(\mathbf{n_r})$ e $g_i(n_{ir})$ di questa

soluzione si consiglia la consultazione della bibliografia (v. [AV89])
Infine la soluzione PF4 è valida per reti multiclasse con nodi con disciplina FCFS e tassi esponenziale, e in cui le classi avranno capacità indipendenti. Le funzioni $V(\mathbf{n_r})$ e $g_i(n_{ir})$ sono identiche a quelle già definite per la soluzione PF1.

Tipo di Blocco	Topologia della rete			
	Two-Node	Ciclica	Server Centrale	Arbitraria
BAS	PF1	$\rightarrow$	$\rightarrow$	PF4 con condizione (B)
BBS-SO	PF1	PF2 con non-empty condition	PF3 solo se $B_1 < \infty$	PF2 con condizioni SNE e (A)
RS-RD	PF1	PF2 con non-empty condition	PF3	PF2 con SNE condition
RS-FD	PF1	PF2 con non-empty condition	PF3 solo se $B_1 < \infty$	PF2 con condizioni SNE e (A)

Tabella 7.3: Reti di code multiclasse forma-prodotto con blocco.

Parte II

Metodi di soluzione

Capitolo 8

Metodi per reti in forma prodotto senza blocco

In questo capitolo illustreremo i due algoritmi più utilizzati per analizzare i modelli in forma prodotto, e più nello specifico, quelli basati su reti BCMP (v. capitolo 6.5.1): l'algoritmo di Convoluzione e l'algoritmo Mean Value Analysis (MVA). Entrambi posseggono una complessità computazionale di ordine polinomiale ed entrambi sono basati su una relazione di tipo ricorsivo. E' difficile definire quale algoritmo sia migliore e da preferire in senso assoluto, perché entrambi presentano, in determinate circostanze, problemi di instabilità numerica. L'algoritmo di convoluzione, per esempio, basandosi sul calcolo della costante di normalizzazione G, subisce questo tipo di problematiche quando si presentano situazioni di overflow/underflow o errori dovuti a troncamenti e arrotondamenti dei

valori. L'algoritmo MVA invece può presentare dei problemi di instabilità numerica davanti a modelli formati da nodi a capacità dipendente dal carico (problema risolto dall'algoritmo MMVA, a discapito però di un elevato aumento della complessità spaziale e temporale). Generalmente, comunque, si tende a preferire, tra i due, l'algoritmo MVA per reti costituite da nodi a capacità fissa o da nodi di tipo 3 (cioè, con infiniti server). Per entrambi gli algoritmi prenderemo in considerazione una rete BCMP chiusa composta da M nodi, con tasso di servizio $\mu_j (1 \leq j \leq M)$, e una popolazione di K utenti, tutti appartenenti ad un'unica catena. Gli utenti si muoveranno all'interno della rete secondo i valori della matrice di routing P ed $E = \{\mathbf{n} : 0 \leq n_j \leq K, \sum_{i=1}^{M} n_i = K\}$ indicherà l'insieme degli stati. Nella rete così definita, la probabilità stazionaria di stato $\pi(\mathbf{n})$, viene calcolata utilizzando il già visto teorema BCMP (v. capitolo 6.5.1) ed equazioni in forma prodotto di questo tipo:

$$\pi(\mathbf{n}) = \frac{1}{G} \prod_{j=1}^{M} f_j(n_j) \tag{8.1}$$

8.1 Algoritmo di convoluzione

L'algoritmo di convoluzione si basa sul calcolo esplicito della costante di normalizzazione G definita come:

$$G = \sum_{n \in E} \prod_{i=1}^{M} f_i(n_i) \tag{8.2}$$

dove la funzione $f_i(n_i)$ deriva dal teorema BCMP e può essere calcolata in due modi, a seconda della tipologia dei nodi utilizzati:

$$f_i(n_i) = \rho_j^{n_j}/n_j! \qquad \text{se il nodo j è di tipo 3}$$
$$f_i(n_i) = \rho_j^{n_j} \qquad \text{altrimenti}$$

La complessità computazionale dipende quindi dal numero di utenti nella rete e dal numero dei nodi, e cioè sarà O(MK).
Con $\mathbf{G_j} = (G_j(0)G_j(1)G_j(K))$ viene indicato il vettore delle costanti di normalizzazione della rete formata dai primi j nodi e con un numero d'utenti che varia da 0 a K. Ogni elemento del vettore $G_j(k)$ viene definito come

$$G_j(k) = \sum_{n \in E(j,k)} \prod_{i=1}^{j} f_i(n_i)$$

Ora che abbiamo definito la costante $G_j(k)$, possiamo introdurre la formula ricorsiva a base dell'algoritmo:

$$G_j(k) = \sum_{n=0}^{k} f_j(n) G_{j-1}(k-n) \tag{8.3}$$

dove $j > 1$ e $0 \leq k \leq K$.
Partendo dalla formula (8.3), possiamo vedere che il vettore $\mathbf{G_j}$ può essere ottenuto per convoluzione dei vettori $\mathbf{G_{j-1}}$ e del vettore $(f_j(0)...f_j(K))$.
La base della formula ricorsiva la si ottiene per definizione:

$$per\ k = 0 \qquad G_j(0) = 1 \qquad 1 \leq j \leq M$$
$$G_0(0) = 1$$

$$\begin{aligned} &per\, j = 0 \qquad G_0(k) = 0 \qquad 0 \leq k \leq K \\ &per\, k = 0 \quad G_1(k) = f_1(k) \quad 0 \leq k \leq K \end{aligned} \tag{8.4}$$

Le formule (8.3) e (8.4) saranno alla base della prima parte dell'algoritmo di convoluzione, in cui viene calcolata la costante di normalizzazione, ma è possibile semplificare ulteriormente il procedimento differenziando le operazioni da compiere per le diverse tipologie di nodi. Per fare ciò dobbiamo ordinare i nodi basandoci sulle loro caratteristiche: avremo così che i nodi ordinati da 1 a M' saranno di tipo 3, i nodi da M'+1 a M" saranno nodi a velocità costante (ma, naturalmente, non di tipo 3) e i rimanenti nodi da M"+1 a M avranno una velocità dipendente dal carico.

Per i primi M' nodi di tipo 3 possiamo sfruttare il teorema BCMP, che per questa tipologia di nodo definisce $f_j(n_j) = \rho_j^{n_j}/n_j!$, e calcolare la costante di normalizzazione nel seguente modo:

$$G_{M'}(k) = [\sum_{j=1}^{M'} \rho_j]^k \, \frac{1}{k!} \tag{8.5}$$

Per i nodi a velocità costante sappiamo, sempre dal teorema BCMP, che $f_j(k) = \rho_j^k$, e compiendo una sostituzione alla formula ricorsiva (8.3) otteniamo:

$$G_j(k) = G_{j-1}(k) + \rho_j G_j(k-1) \tag{8.6}$$

con $0 \leq k \leq K$ *e* $M' + 1 \leq j \leq M''$.

I restanti nodi, che ricordiamo essere non di tipo 3 e con una velocità

che dipende dal carico, vengono calcolati utilizzando la formula generale (8.3) e la costante finale G risulterà essere la somma delle tre costanti ottenute. Inoltre, dall'ultimo vettore $\mathbf{G_M}$ possiamo ricavare direttamente anche gli indici di prestazione locali al nodo j, con $0 \leq j \leq M$.
La distribuzione di probabilità marginale della lunghezza di coda è data dalla formula:

$$\pi_j(k) = f_j(k)\frac{G_{M-\{j\}}(K-k)}{G_M(K)} \tag{8.7}$$

per $0 \leq k \leq K$ e dove $G_{M-\{j\}}$ indica la costante di normalizzazione della rete con k utenti e tutti i nodi eccetto il nodo j, $1 \leq j \leq M$.
Per i nodi a velocità costante, è possibile semplificare la formula (8.7), sfruttando la relazione tra $\mathbf{G_{M-1}}$ e $\mathbf{G_M}$ definita dalla formula (8.6):

$$\pi_j(k) = \rho_j^k\Big\{G_M(K-k) - \rho_j G_M(K-k-1)\Big\}/G_M(K) \tag{8.8}$$

dove $0 \leq k \leq K$, $M' \leq j \leq M''$ e $G_M(k) = 0$ se k¡0.
L'algoritmo oltre alla distribuzione di probabilità $\pi_j(k)$, calcola anche il throughput, l'utilizzazione e la lunghezza media di coda per ogni nodo j. Il throughput, per definizione, lo ricaverà tramite la seguente espressione:

$$X_j(K) = e_j\frac{G_M(K-1)}{G_M(K)} \tag{8.9}$$

mentre per calcolare l'utilizzazione di un nodo utilizzarà tre diverse espressioni, a seconda della tipologia del nodo:

$$U_j(K) = \quad 0, \qquad 1 \leq j \leq M' \tag{8.10}$$

$$U_j(K) = \quad X_j(K)/\mu_j m_j, \qquad M'+1 \leq j \leq M'' \tag{8.11}$$

$$U_j(K) = \quad \textstyle\sum_{k=1}^{K} min\{k, m_j\}\pi_j(k)/m_j, \qquad M''+1 \leq j \leq M \tag{8.12}$$

La lunghezza media di coda sarà infine data da:

$$N_j(K) = \quad X_j(K)/\mu_j, \qquad 1 \leq j \leq M' \tag{8.13}$$

$$N_j(K) = \quad \sum_{k=1}^{K} \rho_j^k \frac{G_M(K-k)}{G_M(K)}, \qquad M'+1 \leq j \leq M'' \tag{8.14}$$

$$N_j(K) = \quad \sum_{k=1}^{K} k\pi_j(k), \qquad M''+1 \leq j \leq M \tag{8.15}$$

Il tempo di risposta medio si ottiene applicando il teorema di Little ad ogni nodo j, $1 \leq j \leq M$

$$R_j(K) = N_j(K)/X_j(K) \tag{8.16}$$

Riassumendo possiamo distinguere in questo algoritmo due fasi distinte In una prima fase calcola la costante di normalizzazione distinguendo i nodi a seconda delle loro caratteristiche ed utilizzando le formule (8.3),(8.5) e (8.6). Successivamente in una seconda fase utilizza le formule da (8.9) a (8.16) per calcolare gli indici di prestazione.

8.2 Algoritmo MVA

L'algoritmo MVA (Mean Value Analysis) analizza i modelli a rete di code chiuse e, a differenza dell'algoritmo di Convoluzione, non calcola esplicitamente la costante di normalizzazione. Questa caratteristica evita i problemi di instabilità numerica causati dal calcolo esplicito di G presenti invece nell'algoritmo descritto nella sezione precedente.
Il metodo MVA è basato sul teorema degli arrivi, che permette di deri-

vare una relazione ricorsiva per il tempo medio di risposta di un nodo, in funzione del numero totale degli utenti della rete:

Teorema degli arrivi 8.2.1 *In una rete chiusa BCMP la probabilità stazionaria di stato al tempo di arrivo di un utente ad un nodo è identica alla probabilità stazionaria di stato a tempo arbitrario della stessa rete con l'utente rimosso.*

Da questo algoritmo si ricava l'espressione ricorsiva per calcolare il tempo medio di risposta al nodo j quando vi sono K utenti, in funzione della lunghezza media di coda quando ve ne sono K-1:

$$R_j(K) = 1/\mu_j(1 + N_j(K-1)) \tag{8.17}$$

se il nodo k ha una velocità costante di servizio μ_j.
Inoltre possiamo scrivere la relazione ricorsiva che calcola la probabilità marginale dello stato n del nodo j dati K utenti nella rete, in funzione della probabilità di stato n-1 del nodo j quando nella rete ci sono K-1 utenti:

$$\pi_j(n \mid K) = \frac{X_j(K)}{\mu_j(n)}\pi_j(n-1 \mid K-1) \tag{8.18}$$

per $1 \leq n \leq K$, $K > 1$.
Per calcolare gli indici di prestazione l'algoritmo ordina i nodi in base ad alcune caratteristiche, come del resto fa anche l'algoritmo di Convoluzione: i nodi compresi tra 1 e M' saranno di tipo 3, ed avranno quindi un numero di server infinito, i nodi tra M'+1 e M" avranno una velocità costante (e naturalmente non saranno di tipo 3), mentre i rimanenti tra

M" e M avranno una velocità variabile. Per ogni tipologia di nodo l'algoritmo calcola i diversi indici di prestazione:

1. Il tempo medio di risposta $R_j(K)$

$$R_j(K) = 1/\mu_j, \quad 1 \leq j \leq M' \tag{8.19}$$

$$R_j(K) = 1/\mu_j(1 + N_j(K-1)), \quad M'+1 \leq j \leq M'' \tag{8.20}$$

$$R_j(K) = \sum_{n=1}^{K} \frac{n}{\mu_j(n)} \pi_j(n-1 \mid K-1), \quad M''+1 \leq j \leq M' \tag{8.21}$$

2. Il throughput $X_j(K)$

$$X_j(K) = \frac{K}{\sum_{i=1}^{M} \frac{e_i}{e_j} R_j(K)}, \quad \forall j \tag{8.22}$$

3. La lunghezza media di coda $N_j(K)$ (ricavata dal teorema di Little)

$$N_j(K) = X_j(K) R_j(K), \quad \forall j \tag{8.23}$$

Per i nodi a velocità costante è necessario calcolare la probabilità marginale sia per trovare il tempo medio di risposta che per la lunghezza media di coda, e nell'algoritmo viene fatto utilizzando il seguente schema ricorsivo:

$$\pi_j(n \mid K) = \frac{X_j(K)}{\mu_j(n)} \pi_j(n-1 \mid K-1), \quad 1 \leq n \leq K, K > \tag{8.24}$$

$$\pi_j(0 \mid K) = 1 - \sum_{n=1}^{K} \pi_j(n \mid K) \tag{8.25}$$

$$\text{con } \pi_j(0 \mid 0) = 1 \tag{8.26}$$

Anche se l'algoritmo MVA non possiede il problema del calcolo esplicito della costante di normalizzazione, può soffrire ugualmente di problemi di

stabilità numerica quando la probabilità $\pi_j(0 \mid K)$ che il nodo k sia vuoto tende a zero. Per risolvere questo problema è stato pensato una versione modificata dell'algoritmo MVA (MMVA) che riscrive la formula (8.25) a scapito di un aumento della complessità computazionale.

8.3 Metodi per reti in forma prodotto multiclasse

Gli algoritmi di Convoluzione e MVA visti in precedenza, nell'analizzare reti multicatena richiedono una complessità computazionale che è ancora polinomiale nel numero di nodi ed utenti, ma esponenziale nel numero di catene. In questa sezione, però illustreremo molto brevemente tre algoritmi che, apportando delle modifiche ai due metodi classici di Convoluzione e MVA, riescono a rendere la complessità computazionale polinomiale anche nel numero di catene. Questi tre algoritmi sono:

- Tree Convolution Algorithm
- Tree-Structured Mean Value Analysis Algorithm
- RECAL (REcursion by Chain ALgorithm)

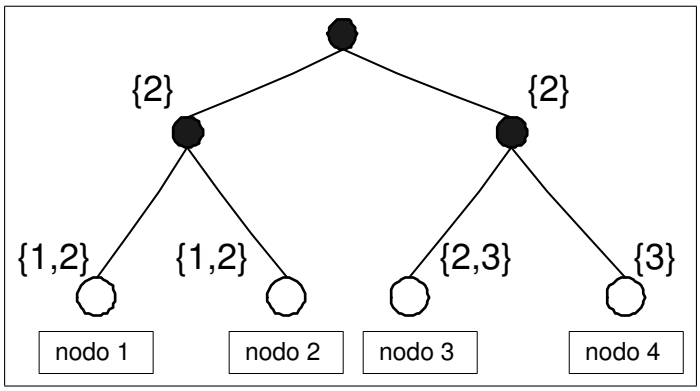
Tree Convolution Algorithm [LL83] è un metodo che nasce dall'osservare la presenza spesso nelle reti di code multiclasse di due proprietà: della *sparseness property* e della *locality property*. La prima è presente quando i percorsi possibili di un utente appartenente ad una determinata catena

comprendono un numero limitato di nodi della rete. La seconda proprietà invece si manifesta quando gli utenti di una catena agiscono solo in insiemi di nodi vicini. Queste due caratteristiche, comuni a molte reti, hanno portato gli autori di questo algoritmo a pensare ad una ottimizzazione del calcolo della costante di normalizzazione G, in modo da limitare i già citati problemi di underflow e overflow di cui soffre l'algoritmo standard di Convoluzione.

Consideriamo CENTERS(k) come l'insieme dei centri di servizio utilizzati dalla catena k e SUBNET come una sottoinsieme della rete. Avremo allora che una catena k sarà totalmente coperta da un sottoinsieme SUBNET quando $CENTERS(k) \subseteq SUBNET$, mentre totalmente non coperta quando i due insiemi risulteranno disgiunti. Infine, se CENTERS(k) risulterà essere solamente intersecata al sottoinsieme della rete, allora la catena si dirà parzialmente coperta. L'algoritmo di Convoluzione tradizionale comincia il calcolo della costante G dalla subnet composta dal solo nodo 1 e mano mano unisce gli altri nodi fino ad arrivare alla totalità della rete. Questo metodo, invece considera la rete come un albero binario, in cui le foglie sono i nodi, i rami interni sono dei sottoinsiemi di nodi e la radice rappresenta tutta la rete, e calcola la costante con un approccio bottom-up sull'albero. Consideriamo l'esempio di una rete composta da 4 catene e da 4 nodi. Ogni catena possiede un vettore $\rho_{\mathbf{k}}$ in cui ρ_{km} rappresenta l'intensità di traffico della catena k al nodo m: $\rho_1 = \{0.5, 1.0, 0, 0\}$, $\rho_2 = \{0.5, 1.0, 0.5, 0\}$, $\rho_3 = \{0, 0, 0.5, 1.0\}$,

$\rho_4 = \{5, 0, 0.5, 0\}$. L'algoritmo considererà questa rete come l'albero raffigurato nella figura 8.1, in cui, per ogni sottoinsieme, tra parentesi quadre vengono evidenziate le catene coperte parzialmente.

Figura 8.1: Esempio di albero



Il metodo, seguendo l'ordine dettato dalla struttura dell'albero, calcolerà la costante di normalizzazione partendo dai nodi-foglie e definendo ogni ramo interno per convoluzione dei due rami a lui sottostanti. Procedendo in questo modo avremo che via via tutte le catene arriveranno ad un sottoinsieme capace di coprirle totalmente, e per queste non sarà più necessario risalire ulteriormente l'albero, con grande risparmi di spazio e tempo (considerando anche le due proprietà di *sparseness property* e di *locality property* viste pocanzi). Infatti, la complessità dell'algoritmo è molto conveniente ed è $O(KM)$ per quanto riguarda lo spazio, e $O(KM^2)$ per il tempo di computazione.

L'approccio di questo algoritmo, in cui la rete viene analizzata trami-

te l'attraversamento di un albero, è stato poi ripreso da *Tree-Structured Mean Value Analysis Algorithm*[HBAK86]. Questo metodo estende l'algoritmo MVA classico alle reti multiclasse evitando che il numero di catene incida in modo esponenziale sulla complessità. L'applicare l'approccio ad albero anche al MVA ha portato ad importanti migliorie tra cui la risoluzione del problema di instabilità numerica e il mantenimento di una complessità computazionale polinomiale anche per reti multiclasse.

Consideriamo ora, per concludere, l'algoritmo *RECAL (REcursion by Chain ALgorithm)* [CG86]. Questo metodo risolve i modelli a rete in forma prodotto chiusi e multiclasse, con discipline di servizio dei nodi FCFS, LCFSPR, PS e IS, con una complessità polinomiale rispetto al numero di catene. L'algoritmo viene costruito, e ricava il proprio nome, da uno schema ricorsivo che ricava la costante di normalizzazione di una rete con r catene da un insieme di reti con r-1 catene. Inoltre divide ogni catena, in modo da ottenere solo sottocatene di un solo utente, poiché questo riesce a semplificare di molto l'espressione ricorsiva. Per quanto riguarda le prestazioni, l'algoritmo si dimostra molto efficiente quando le reti da analizzare posseggono molte catene e matrici di routing con molti elementi non nulli. Le prestazioni invece peggiorano con matrici di routing sparse, per cui, in questi casi, consigliamo l'uso degli algoritmi ad albero visti in precedenza.

Concludiamo, per completezza, citando altri due algoritmi che risolvono con complessità polinomiale le reti in forma prodotto multiclasse, e

cioè il Mean Value Analysis by Chains (MVAC)[CSL89], e il Distribution Analysis by Chain (DAC)[DSL89], che permette, però, di calcolare solo alcuni indici di prestazione globale.

Capitolo 9

Metodi per reti di code a singola classe con blocco

In questo capitolo vedremo alcuni metodi per la risoluzione delle reti di code con blocco a singola classe. Nella prima sezione vedremo le modifiche da apportare ai già visti algoritmi di Convoluzione e MVA per poterli applicare a reti in forma prodotto con buffer finito. Nella seconda sezione vedremo invece dei metodi di risoluzione approssimati per QNB chiuse e aperte.

9.1 Metodi per reti in forma prodotto con blocco

Consideriamo ora degli algoritmi che modificano quelli di Convoluzione e MVA, visti nel capitolo precedente, in modo da poterli utilizzare anche con reti di code in forma prodotto con blocco.

Algoritmo di Convoluzione per QNB

L'algoritmo di Convoluzione può essere applicato alle reti di code con blocco utilizzando due particolari funzioni $f_i(n_i)$ (v. formula (8.1)) a seconda della tipologia di rete da analizzare:

$$f_i(n_i) = \sigma_i^{n_i}, \quad n_i > 0,\ 1 \leq i \leq M \tag{9.1}$$

dove

$$CASO1: \quad \sigma_i = x_i/\mu_i$$

$$CASO2: \quad \sigma_i = 1/\epsilon_i$$

$$con \quad x_i = \sum_{j=1}^{M} x_j p_{ij} + \lambda p_{0i}$$

Il valore di σ del caso 1 è valido per le seguenti reti:

- per reti con blocco RS-RD, routing reversibile e nodi con scheduling arbitrario;
- per reti formati da due nodi e con blocchi di tipo BAS, BBS o RS, e con una disciplina di servizio FCFS.

Il secondo caso invece considera reti delle seguenti tipologie:

- reti con topologia arbitraria, con blocco RS-RD e dove il numero degli utenti N è tale da non permettere il verificarsi di stati in cui sono presenti nodi vuoti;
- reti con blocco di tipo RS-RD, RS-FD o BBS, dove la popolazione è sufficientemente grande da non poter mai lasciare un nodo vuoto, e dove i nodi con buffer finito possono mandare i proprio utenti soltanto ad un destinatario j.
- reti con topologia ciclica e blocco di tipo BBS o RS, dove al più un nodo può essere vuoto.

Come per l'algoritmo di Convoluzione originale, anche in questa modifica verrà calcolata esplicitamente la costante di normalizzazione prima di poter calcolare gli indici di prestazione. Per le formule utilizzate per il calcolo della costante e degli indici si rimanda il lettore alla consultazione dei riferimenti bibliografici (v. [BPO01] pagg. 111-119). La complessità computazionale di questo algoritmo sarà O(MC), dove $C = max\{B_i - a_i \ 1 \leq i \leq M\}$ e a_i è il numero minimo d'utenti ad un nodo i.

Algoritmo MVA per QNB

L'algoritmo MVA, leggermente modificato, può essere utilizzato anche per analizzare reti di code in forma prodotto con blocco di tipo BBS-SO o RS. La soluzione si basa sulla definizione di una rete duale senza

alcun blocco e con un'identica distribuzione degli stati rispetto alla prima, come descritto in [GN67]. Una volta trovata la rete duale e senza blocco è possibile risolverla applicando semplicemente l'algoritmo MVA standard visto nel capitolo 8.2, e, sfruttando la proprietà di dualità, risolvere così anche la rete originaria. E' utile sottolineare che per risolvere le reti con blocco, con questa tecnica, non si sfrutta direttamente il teorema degli arrivi, ma si utilizza una rete duale, che avendo come caratteristica l'assenza di blocco, può sfruttare l'algoritmo standard.

9.2 Metodi approssimati per reti di code con blocco

In questa sezione ci dedicheremo agli algoritmi che calcolano in modo approssimato le prestazioni delle reti di code con blocchi. Utilizzare soluzioni non approssimate spesso infatti è troppo dispendioso in termini di risorse e quindi in letteratura si è preferisce utilizzare approcci approssimati ma, spesso, polinomiali.
Molti algoritmi utilizzano il metodo di decomposizione e aggregazione, che si articola in 3 passi:

1. la rete viene divisa in un insieme di sottoreti
2. ogni sottorete viene analizzata separatamente in modo da definire una componente aggregata

3. viene definita e analizzata la rete aggregata, ottenuta dalla sostituzione di ogni sottorete con la sua corrispondente componente aggregata

In pratica si utilizza il principio del divide-et-impera in cui il problema, per essere risolto, viene diviso in sottoproblemi più semplici da analizzare.

9.2.1 Metodi approssimati per reti chiuse

In questa sezione introdurremo 5 popolari algoritmi approssimati per analizzare reti di code chiuse con blocco. Non esiste tra questi l'algoritmo migliore in senso assoluto e, ogni metodo, fornisce prestazioni diverse e differenti percentuali d'errore a seconda del tipo di rete data in input.

Nella tabella riassuntiva 9.1, è possibile trovare le caratteristiche peculiari e le prestazioni di questi algoritmi. I primi tre metodi analizzano reti con topologia ciclica mentre gli ultimi due possono analizzare reti con topologia arbitraria. Guardando la tabella inoltre è possibile vedere come tutti gli algoritmi funzionino con reti omogenee, cioè con reti dove la tipologia dei nodi e la tecnica di blocco rimangono costanti in tutto il modello.

Throughput Approssimation

Consideriamo una rete chiusa con un numero N di utenti. Questo metodo si basa sulla simmetria della funzione di throughput X(N) in relazione

Metodo	Caratteristiche della rete		Blocco	Accuratezza	Efficienza
	Topologia	Nodi			
Throughput Approximation	ciclica	G/M/1/B	BAS o BBS-SO	Molto buona	Scarsa per reti con meno di 5 nodi
Network Decomposition	ciclica	G/M/1/B	BBS-SO	Buona	Buona
Variable Queue Capacity Decomposition	ciclica con 1 nodo a capacità illimitata	G/M/1/B	BBS-SO	Buona per reti con più di 4 nodi	Discreta
Approximate MVA	arbitraria	G/M/1/B	BAS	Discreta per il throughput, scarsa per gli altri indici di prestazione	Molto buona
Maximum Entropy Algorithm	arbitraria	G/GE/1/B	RS-RD	Discreta	Discreta

Tabella 9.1: Metodi approssimati per l'analisi di reti di code chiuse con blocco.

agli utenti. Avremo quindi che $\forall N$, $X(N) = X(B - N)$.
Poniamo ora N^* come il numero d'utenti per cui la funzione di throughput raggiunge il suo massimo e definiamola come $N^* = \lfloor B/2 \rfloor$ se B è pari, $N^* = \lfloor B/2 \rfloor + 1$, se è dispari. Essendo simmetrica, avremo che la funzione di throughput risulterà non-decrescente fino ad N^* e, da N^* a B, sarà non-crescente. L'algoritmo calcolerà direttamente, con metodi analitici esatti, alcuni punti della funzione, e ricaverà i rimanenti tramite la seguente funzione interpolatrice:

$$X(N) = X(N+1) - yx^{N^*-N} \tag{9.2}$$

dove

$$y = [X(N^*) - X(B^-)][\sum_{i=1}^{N^*-B^-} x^{i-1}] \tag{9.3}$$

e x è il punto fisso della seguente equazione

$$X(B^- - 1) = X(B^-) - [X(N^*) - X(B^-)][x^{N^*-B^-+1}(1-x)]/[x - x^{N^*-B^-+1}] \tag{9.4}$$

L'algoritmo quindi con una rete con blocco di tipo BBS calcola esattamente i punti che vanno da $1 \leq N \leq B^-$ poiché, per quei valori, la rete può essere pensata come una rete senza blocco e può essere risolta con un algoritmo per le reti in forma prodotto, ed il punto $N = N^*$ tramite l'analisi della catena markoviana per $X(N^*)$. I punti della funzione X(N) nel range $B^- + 1 \leq N \leq N^* - 1$ verranno calcolati dei valori approssimati, ricavati dalle formule (9.2) e (9.3).
Se si considerano reti con blocco BAS non sarà più sufficiente limitarci

a questi due passi, poiché reti di questo tipo perdono la proprietà simmetrica per le funzioni di throughput. Per questa tipologia di blocco N^* dipenderà dalla capicità delle code e dai tassi di servizio e sarà inoltre necessario calcolare con precisione ulteriori punti, oltre a quelli già considerati per reti con blocco BBS. L'algoritmo produrrà comunque una buona accuratezza nei risultati per entrambe le tipologie di blocco ed un errore relativo massimo del 5%.

Network Decomposition

Questo metodo approssima il throughput di reti cicliche con blocco BBS, partizionandole in M sottoreti composte da un solo nodo. Queste sottoreti vengono poi analizzate come reti $M/M/1/B_i$ con tasso d'arrivo λ_i^* e tasso di servizio $\mu_i^*(n)$, dove $0 \leq n \leq B_i$. L'algoritmo distingue i due casi in cui tutti i nodi hanno capacità finita e in cui c'è un nodo con capacità infinita, indicato con 1 (quindi $B_1 = \infty$). Per quest'ultimo caso l'algoritmo calcola i parametri μ_i^* (il tasso di servizio del nodo i), μ_M^* e λ_i^* (il tasso d'arrivo) con le seguenti formule:

$$\mu_i^*(n) = \Big\{(1/\mu_i) + \textstyle\sum_{i=1}^{M} b_{ij}(n)\Big[\sum_{k=i+1}^{j}(1/\mu_k)\Big]\Big\}^{-1}, \qquad 1 \leq i \leq M,\ 1 \leq B_i \leq N \tag{9.5}$$

$$\mu_M^*(n) = \mu_M, \qquad 1 \leq n \leq B_M \tag{9.6}$$

$$\lambda_i^* = X/(1 - p_i^*(B_i)) \tag{9.7}$$

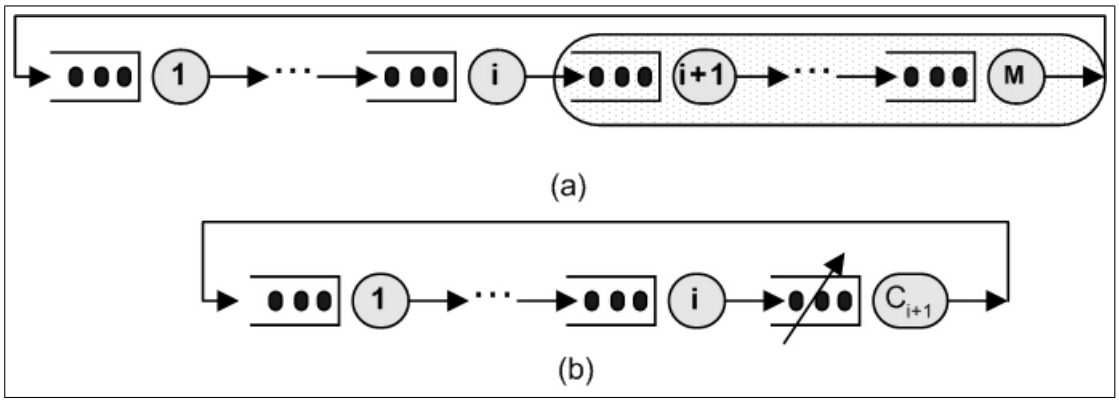
dove X è il throughput della rete e $b_{ij}(n)$ indica la probabilità che, con n utenti al nodo i, i nodi $i+1, i+2, ..., j$ siano pieni, mentre il nodo $j+1$ non lo sia. L'algoritmo è costruito in modo iterativo e partendo da un intervallo approssimato di throughput $[X_{min}(0), X_{max}(0)]$ termina quando dopo k iterazioni viene soddisfatta la condizione $(X_{min}(k) - X_{max}(k)) < \epsilon$ per un ϵ abbastanza piccolo. Ad iterazione k, l'algoritmo calcola i parametri λ_i^* e $\mu_i^*(n)$ (con $0 \leq n \leq B_i$ e $0 \leq i \leq M$) ed aggiorna la k-esima approssimazione del throughput $[X_{min}(k), X_{max}(k)]$. La complessità computazionale di questo algoritmo è $O(kM^4(B^+)^3)$, dove k indica il numero di iterazioni. Inoltre i risultati sono abbastanza accurati e si manifestano errori relativi massimi non maggiori del 7%.

Variable Queue Capacity Decomposition

Consideriamo una rete ciclica con blocco BBS e dove un nodo ha capacità finita. Per analizzare reti di questo tipo è possibile utilizzare il metodo Variable Queue Capacity Decomposition. L'idea generale alla base di questo metodo è il considerare, dato un nodo i, tutti i nodi in uscita aggregati in un unico nodo composto C_{i+1}, tale da possedere una capacità della coda variabile (v. fig. 9.1). In concetto di capacità variabile della coda migliora l'approssimazione rispetto all'utilizzo di nodi composti a capacità costante. L'algoritmo procede come segue:

- Comincia analizzando una sottorete formata dai due nodi M e M-

Figura 9.1: (a) la rete originaria (b) la rete aggregata, dove il nodo C_{i+1} è il nodo con capacità di coda variabile



1, per definire il nodo aggregato C_{M-1}, che viene poi analizzato insieme al nodo M-2.

- L'algoritmo procede scorrendo all'indietro la rete dal nodo M-2 al nodo 1, analizzando tutte le sottoreti formate da due nodi (un nodo i ed il nodo aggregato C_{i+1}) in modo da poter prendere in considerazioni nodi aggregati mano mano sempre più grandi.

- All'ultimo passo si ottiene una rete formata dal nodo 1 e dal nodo C_2, che permette di ottenere il valore di throughput approssimato.

L'analisi di ogni sottorete di due nodi si svolge considerando due corrispondenti sottoreti da due nodi con, rispettivamente, un buffer fisso (mentre quelle originarie posseggono un buffer variabile) e un buffer infi-

nito. Dai risultati ricavati da queste due sottoreti, vengono poi calcolati i valori approssimati della sottorete di partenza. L'algoritmo possiede una complessità computazionale pari a $O(MN^3)$ operazioni e si dimostra accurato per reti con meno di 5 nodi. Questo metodo però aumenta la percentuale di errore all'aumentare del numero dei nodi, perché ad ogni passo viene introdotto un errore d'approssimazione.

Approssimate MVA

Questo algoritmo modifica il già visto algoritmo MVA rendendolo utilizzabile con reti di code com blocco BAS e con tempo di servizio esponenziale. L'algoritmo MVA si basa sul teorema di Little e sul teorema degli arrivi e utilizza le seguenti formule ricorsive:

$$R_i(n) = (1/\mu_i)[1 + L_i(n-1)] \tag{9.8}$$

$$X_i(n) = nx_i/[\sum_{1 \le j \le M} x_j R_j(n)] \tag{9.9}$$

$$L_i(n) = X_i(n)R_i(n) \tag{9.10}$$

$$\text{per } \; n = 1, ..., N \, e \, 1 \le i \le M$$

dove $R_i(n)$ indica il tempo medio di risposta quando nella rete sono presenti n utenti, $X_i(n)$ il throughput del nodo i e $L_i(n)$ la lunghezza media della coda.

L'algoritmo approssimato modifica la formula (9.8) quando deve calcolare il tempo di risposta medio di nodi pieni o bloccati. La modifica si basa sull'osservazione che, se un nodo i è pieno, ogni nodo j bloccato dal nodo i

aumenterà il suo tempo di risposta, poiché dovrà attendere che i si liberi. Inoltre contribuiranno al tempo medio di risposta del nodo i solamente gli utenti già al suo interno. Avremo quindi che la formula ricorsiva (9.8) diventerà:

$$R_i(n) = (1/\mu_i)L_i(n-1) \tag{9.11}$$

$$R_j(n) = (1/\mu_j)[1 + L_j(n-1)] + (1/\mu_j)(e_j p_{ji}/e_i) \tag{9.12}$$

Il metodo applicherà le formule introdotte, all'interno di un ciclo che termirà quando $L_i(n) \leq B_i$ per ogni nodo i della rete. Questo algoritmo è semplice da costruire e possiede una complessità computazionale polinomiale pari a $O(M^3 + kMN)$, dove k indica il numero delle iterazioni all'interno del ciclo. L'algoritmo dimostra comunque spesso una scarsa precisione e, sperimentalmente, ha dimostrato i migliori risultati per reti con un sever centrale e i peggiori con reti cicliche.

Maximum Entropy Algorithm

Questo metodo si basa sul principio della massima entropia ed è definito per reti di code con blocco RS-RD. In realtà il metodo per reti chiuse è un'estensione dell'algoritmo basato sulla massima entropia per reti aperte, che vedremo in seguito. Il metodo approssima la probabilità congiunta $\pi(\mathbf{S})$ per ogni stato della rete $\mathbf{S} = (n_1, ..., n_M)$ massimizzando la funzione entropica $H(\pi) = -\sum_{\mathbf{S}} \pi(\mathbf{S}) log(\pi(\mathbf{S}))$, che è soggetta ai seguenti vincoli:

1. $\sum_{\mathbf{S}} \pi(\mathbf{S}) = 1$ (normalizzazione)

2. $\sum_{n_i>a_i} \pi_i(n_i) = u_i$ (u_i è la probabilità che il nodo i abbia più di a_i utenti)

3. $\sum_{a_i \le n_i \le B_i} h_i(n_i)\pi_i(n_i) = L_i$ (L_i è la lunghezza media della coda)

4. $\sum_{a_i \le n_i \le B_i} f_i(n_i)\pi_i(n_i) = \Phi_i$ (Φ_i è la probabilità che il nodo i sia pieno)

dove $a_i = max\{0,\, N - B + B_i\}$, $h_i(n_i) = min\{0,\, n_i - a_i - 1\}$ e $f(n_i) = max\{0,\, n_i - B_i + 1\}$. L'algoritmo determina il valore approssimato di $\pi(\mathbf{S})$ con la seguente espressione in forma prodotto:

$$\pi(\mathbf{S}) = \Big(\frac{1}{Z}\Big) \prod_{i=1}^{M} x_i(n_i) y_i^{h_i(n_i)} z_i^{f_i} \tag{9.13}$$

dove Z è una costante di normalizzazione, $x_i(n_i) = 1$ se $n_i = a_i$ e $x_i(n_i) = x_i$ se $a_i \le n_i \le B_i$, e x_i, y_i e z_i sono i coefficienti langrangiani corrispondenti ai vincoli 2, 3 e 4.

L'algoritmo approssima la rete chiusa, analizzandola come una rete aperta senza arrivi e partenze verso l'esterno ed utilizzando le tecniche d'approssimazione basate sul concetto di massima entropia utilizzate per le reti aperte (che descriveremo nel prossimo capitolo) con qualche piccola modifica nel derivare la soluzione per x_i e y_i, e con un'approssimazione per z_i. Dopo aver calcolato i coefficienti, il metodo valuta z_i in modo iterativo. La complessità computazionale dipende dai due passaggi di analisi della rete aperta e di valutazione del coefficiente z_i: la complessità del primo passaggio verrà analizzata successivamente (v. cap.

9.2.2), mentre la complessità computazionale della fase di valutazione è di $O(kM^2N^2)$, dove k indica il numero delle iterazioni. Il metodo risulta essere abbastanza accurato (errore medio relativo del 4% per il calcolo della lunghezza media della coda e del throughput), ma a volte presenta picchi di percentuale d'errore che possono anche raggiungere un massimo del 60%. Per maggiori informazioni sull'algoritmo si consiglia di consultare la bibliografia (v. [KX89]).

9.2.2 Metodi approssimati per reti aperte

In questa sezione descriveremo tre algoritmi molto utilizzati nell'analisi approssimata di reti aperte.

Nella tabella 9.2 vengono riassunte le condizioni con cui i metodi possono essere applicati e una valutazione sulle prestazioni, in termini di qualità dei risultati e di velocità nell'elaborarli. Come per i metodi d'analisi approssimata di reti chiuse, questi algoritmi lavorano solamente con reti omogenee ed con il presupposto che tutti i nodi adottino una disciplina di servizio FCFS.

Tandem Exponential Network Decomposition

Consideriamo una rete topologicamente a tandem e con blocco di tipo BAS. In questo algoritmo la rete viene partizionata in M sottoreti T(i), ognuna delle quali formata da un unico nodo. Ogni sottorete T(i) rappresenta un unico nodo isolato e può essere analizzata come una modello

Metodo	Caratteristiche della rete		Blocco	Accuratezza	Efficienza
	Topologia	Nodi			
Tandem Exponential Network Decomposition	tandem	G/M/1/B	BAS	Molto buona per tutti gli indici di prestazione	Molto buona
Acyclic Network Decomposition	aciclica	G/M/1/B	BAS	Molto buona per tutti gli indici di prestazione	Buona
Maximum Entropy Algorithm	arbitraria	G/GE/1/B	RS-RD	Buona per tutti gli indici di prestazione	Discreta

Tabella 9.2: Metodi approssimati per l'analisi di reti di code aperte con blocco.

$M/M/1/B_i + 1$ con un tasso di arrivo $\mu_u(i)$ e un tasso di servizio $\mu_d(i)$, e con una probabilità marginale $\pi_i(n)$, $\forall n$, $1 \leq i \leq M$. Gli unici valori che conosciamo sono $\mu_u(1) = \lambda$ e $\mu_d(M) = \mu_M$, poiché T(1) e T(M) sono rispettivamente formate dal primo e dall'ultimo nodo della rete originaria. Inoltre indichiamo con $p_b(i)$ la probabilità che, al momento di un nuovo arrivo, la sottorete T(i) sia piena, mentre con $p_s(i)$ la probabilità che, finito un servizio, la sottorete T(i) sia vuota. L'approssimazione si basa sulle seguenti relazioni:

$$\mu_u(i) = [(1/\mu_{i-1}) + p_s(i-1)/\mu_u(i-1)]^{-1} \qquad 2 \leq i \leq M \tag{9.14}$$

$$\mu_d(i) = [(1/\mu_i) + p_b(i+1)/\mu_d(i+1)]^{-1} \qquad 1 \leq i \leq M-1 \tag{9.15}$$

$$X_1 = X_2 = \ldots = X_M \tag{9.16}$$

Il throughput di un sistema T(i) può essere inoltre calcolato tramite la seguente formula:

$$X_i = \mu_u(i)[1 - \pi_i(B_i + 1)] = \mu_d(i)[1 - \pi_i(0)] \tag{9.17}$$

L'algoritmo può essere scritto ad alto livello con il seguente schema iterativo:

0. *Inizializzazione:* $\mu_u(i) = \lambda$, $\mu_d(i) = \mu_i \forall i$
1. do
2. forward cycle: *for* i=1,...,M-1
3. calcola la probabilità di stato π_i e la probabilità $p_s(i)$ come segue:
4. $p_s(i) = \pi_i(1)/(1 - \pi_i(0))$
5. calcola $\mu_u(i+1)$ secondo la formula (9.14)
6. backward cycle: *for* i=M,...,2
7. calcola la probabilità $p_b(i)$ nel seguente modo:
8. $p_b(i) = \pi_i(B_i)/(1 - \pi_i(B_i + 1))$
9. calcola $\mu_d(i-1)$ grazie alla formula (9.15)
10. *until* $max\{\ X\_i''\text{-}X\_j''\ , 1 \leq i, j \leq M\} < \epsilon$
11. Calcola L_i, $\pi_i(n)$, $0 \leq i \leq B_i$, $1 \leq i \leq M$ e X_1.

L'algoritmo ha una complessità polinomiale pari a O(kM$(B^+)^2$), dove k è il numero delle iterazioni al passo 1. Prove sulle prestazioni dell'algoritmo dimostrano che converge velocemente e fornisce dati con una buona accuratezza (con un errore massimo del 4.1%).

Acyclic Network Decomposition

Questo metodo è un estensione del metodo Tandem Exponential Network Decomposition, definito precedentemente su reti tandem con blocco BAS, per reti aperte acicliche. Alla base del metodo rimane la decomposizione della rete in sottoreti T(i) formate da un solo nodo ed analizzate come

reti del tipo $M/M/1/B_i$. Con λ_i e μ_i indicheremo rispettivamente il tasso di entrata nel nodo di utenti esterni al sistema ed il tasso di servizio del nodo. Sarà inoltre necessario definire due nuove funzioni binarie:

$$I_i = \begin{cases} 0, & \text{se } \lambda_i = 0; \\ 1, & \text{se } \lambda_i \neq 0. \end{cases} \qquad O_i = \begin{cases} 0, & \text{se } p_{i0} = 0; \\ 1, & \text{se } p_{i0} \neq 0. \end{cases}$$

Dovremo poi definire due insiemi che contengano gli indici dei nodi sorgente di un nodo i e gli indici dei possibili nodi di destinazione: i due insiemi saranno rispettivamente $U_i = \{j : p_{ji} > 0\} \bigcup I_i\{0\}$ e $D_i = \{j : p_{ij} > 0\} \bigcup O_i\{0\}$, dove con l'indice 0 si rappresentano gli arrivi dall'esterno della rete e le partenze.

Ogni sottorete T(i) riceverà utenti dai nodi che appartengono all'insieme U_i con un tasso $\mu_{uj}(i)$, $j \in U_i$, ed avrà un tasso di servizio $\mu_d(i)$. Assumiamo inoltre che i nodi siano ordinati in modo tale che la matrice di routing sia triangolare superiore. Definiamo $\pi_i(n)$ come la probabilità di stato del sistema T(i), $0 \leq n \leq B_i + \mid U_i \mid$. Questa probabilità può essere ottenuta da un processo markoviano di nascita-morte con un tasso di morte costante $\mu_d(n)$ per ogni stato n e un tasso di nascita $\mu^*(i)$ così definito:

$$\begin{aligned} \mu^*(i) = & \quad \textstyle\sum_{j=1}^{|U_i|} \mu_{uj}(i) \qquad per\, 0 \leq n \leq B_i - 1 \\ = & \quad (k+1)\Omega_{k-1}/\Omega_k \quad per\, n = B_{i+k},\, 0 \leq k \leq \mid U_i \mid -1 \end{aligned} \tag{9.18}$$

dove

$$\Omega_k = \sum_{j_1 < j_2 < \ldots < j_k} \prod_{s=1}^{k} \mu_{uj_s}(i),\; j_s \in U_i$$

Definiamo ora le variabili $p_{bj}(i)$ e $p_s(i)$ come, rispettivamente, la probabilità che la j-esima sorgente sia bloccata dal nodo i pieno, e la probabilità che alla fine di un servizio il sistema T(i) sia vuoto. $p_s(i)$ è definita dalla seguente formula:

$$p_s(i) = \pi_i(1)/[1 - \pi_i(0)] \tag{9.19}$$

Inoltre con $p_{bj}(k : i)$ intenderemo la probabilità che alla fine di un servizio del sistema T(i), la j-esima sorgente trovi k altre sorgenti bloccate dal nodi i:

$$p_{bj}(k : i) = [\pi_i(B_i + k) - b_{uj}(k : i)]/[1 - b_{uj}(i)] \tag{9.20}$$

con $j \in U_i$, $0 \leq k \leq \mid U_i \mid$, e con $b_{uj}(k : i)$ e $b_{uj}(i)$ definite nel seguente modo:

$$b_{uj}(k : i) = \frac{\mu_{uj}(i)\Omega^j_{k-1}}{\Omega_k}\pi_i(B_i + k) \tag{9.21}$$

$$\text{con } \Omega^j_{k-1} = \sum_{j_1<j_2<\ldots<j_{k-1}} \prod_{s=1}^{k-1} \mu_{uj_s}(i),\ j_s \in U_i$$

$$b_{uj}(i) = \sum_{k=1}^{|U_i|} b_{uj}(k : i) \tag{9.22}$$

Ora possiamo definire $\mu_{uj}(i)$ e $\mu_d(i)$, per ogni nodo i del sistema e per $j \in U_i$:

$$\mu_{uj}(i) = p_{ti}[\frac{p_s(t)}{\mu^*(t)} + \frac{1}{\mu_d(t)} - p_{ti} \sum_{k=0}^{|U_i|-1} p_{bj}(k : i)\frac{k+1}{\mu_d(i)}]^{-1} \tag{9.23}$$

$$\mu_d(i) = [\frac{1}{\mu_i} + \sum_{m\in D_i} p_{im} \sum_{k=0}^{|U_m|-1} p_{bt}(k : m)\frac{k+1}{\mu_d(m)}]^{-1} \tag{9.24}$$

dove t indica l'indice del j-esimo nodo nella lista U_i.

Sfruttando la legge di conservazione del throughput, possiamo scrivere

un sistema di equazioni per il throughput $X_{uj}(i)$ che la j-esima sorgente produce verso il nodo i:

$$X_{uj}(i) = X_d(k)p_{ki}$$

dove k è l'indice del nodo che è j-esima sorgente di i, e $X_d(k)$ indica il throughput totale che fuoriesce da k. Utilizzando le variabili già definite $\mu_{uj}(i)$ e $\mu_d(i)$ possiamo riscrivere le equazioni che calcolano i throughput nella seguente forma:

$$\begin{aligned} X_{uj}(i) &= [\frac{1}{\mu_{uj}(i)} + \sum_{k=0}^{|U_j|-1} \frac{p_{bj}(k:i)(k+1)}{\mu_d(i)}]^{-1} \\ X_d(i) &= [\frac{1}{\mu_d(i)} + \frac{p_s(k)}{\mu^*(i)}]^{-1} \end{aligned}$$

Ora che abbiamo definito tutte le equazioni e le variabili, possiamo finalmente scrivere una versione ad alto livello dell'algoritmo:

01. $\mu_{u1}(i) = \lambda_i$, $\forall i$ tale che $U_i \neq \oslash$
02. $\mu_d(i) = \mu_i$, $\forall i$ tale che $D_i \neq \oslash$
03. $p_{bj}(n : i) = 0$, $j \in U_i$, $0 \leq n \leq \mid U_i \mid$, $\forall i$
04. repeat
05. forward cycle: for $i = 1, \ldots, M$
06. calcola $p_s(i)$ utilizzando la formula (9.19)
07. calcola $\mu_{uj}(i)\, \forall j \in U_i$ utilizzando la formula (9.23)
08. backward cycle: for $i = M, \ldots, 1$
09. calcola $\mu_d(i)$ utilizzando la formula (9.24)
10. calcola $p_{bj}(k : i)\, 0 \leq k \leq \mid U_i \mid -1$, $j \in U_i$ con la formula (9.20)
11. until converge su $\mu_d(i)$, $\forall i$

La complessità dell'algoritmo è $O(kM[(U + B^+)^2 + U^3 + 2^{U+1}])$, dove con k si indicano il numero di interazioni e dove $U = max_i \mid U_i \mid$. L'algoritmo dimostra una buona accuratezza e possiede un errore relativo massimo del 5 % . Sperimentalmente, si è notato che l'algoritmo funziona con maggiore precisione quando ogni nodo possiede più di una sorgente e peggio, invece, quando si trova a lavorare con reti tandem.

Maximum Entropy per reti aperte

Questo metodo analizza in modo approssimato reti di code aperte con blocco RS-RD, decomponendole in M sottoreti ed analizzando ogni sottorete come un modello GE/GE/1/B. Similmente a quanto già visto per le

reti chiuse, questo metodo si basa sul massimizzare la funzione entropica $H(\pi) = -\sum_n \pi(n) log(\pi(n))$, soggetta ai seguenti vincoli:

1. $\sum_{0 \leq n \leq B_i} \pi_i(n) = 1$ (normalizzazione)
2. $\sum_{0 \leq n \leq B_i} h_i(n)\pi_i(n) = \rho_i$ (utilizzazione)
3. $\sum_{a_i \leq n \leq B_i} n\pi_i(n_i) = L_i$ (L_i è la lunghezza media della coda)
4. $\sum_{a_i \leq n \leq B_i} f_i(n)\pi_i(n) = \Phi_i$ (Φ_i è la probabilità che il nodo i sia pieno)

dove $a_i = max\{0,\, N - B + B_i\}$, $h_i(n_i) = min\{1,\, max(0, n)\}$ e $f(n_i) = max\{0,\, n_i - B_i + 1\}$.
La distribuzione di probabilità congiunta della lunghezza di coda è data dalla seguente espressione in forma prodotto:

$$\pi(\mathbf{n}) = \Big(\frac{1}{Z}\Big) \prod_{i=1}^{M} x_i^{h_i(n_i)} y_i^{n_i} z_i^{f_i(n_i)} \tag{9.25}$$

dove Z è una costante di normalizzazione, $x_i = e^{-\beta_{i1}}$, $y_i = e^{-\beta_{i2}}$, $z_i = e^{-\beta_{i3}}$. La distribuzione marginale per ogni nodo i è invece data dalla seguente formula:

$$\pi(\mathbf{n}) = \Big(\frac{1}{Z'}\Big) x_i^{h_i(n_i)} y_i^{n_i} z_i^{f_i(n_i)} \tag{9.26}$$

dove Z' è una costante di normalizzazione. Ora che abbiamo definito le formule utilizzate dall'algoritmo, possiamo descriverne il funzionamento (nel pseudocodice c_{di} e c_{si} indicano rispettivamente i coefficienti di variazione dei tempi di interarrivo e di interpartenza da un nodo i):

1. Inizializzazione
2. repeat per ogni sottorete i, con $1 \leq i \leq M$
3. calcola il tasso effettivo di arrivo λ_i' dalle equazioni di traffico
4. calcola la probabilità che dopo un completamento di servizio al nodo i, j sia pieno $\forall j$
5. calcola la probabilità π_i utilizzando la formula (9.26)
6. calcola i nuovi valori per c_{di}
7. until i valori di c_{di} non convergono

La complessità computazionale complessiva di questo algoritmo è di $O(\Omega^3)$, dove Ω rappresenta la cardinalità dell'insieme di probabilità calcolate al passo 4. Inoltre questo metodo a volte risulta essere poco accurato, soprattutto nell'analizzare reti che posseggono cicli.

Capitolo 10

Metodi per reti di code con blocco multiclasse

Dopo aver analizzato, al capitolo precedente, i metodi per la risoluzione delle reti di code con blocco a singola classe, ci occuperemo ora delle soluzioni per QNB multiclasse. Nella prima sezione vedremo quanto finora è stato fatto in letteratura e mostreremo una modifica dell'algoritmo Entropy Maximisation, che ne consente l'utilizzo anche in quest'ambito. Nella seconda sezione invece verrà proposto un nuovo metodo che, estendendo l'algoritmo Acyclic Network Decomposition, si porrà come obbiettivo la risoluzione di reti di code aperte e multiclasse.

10.1 Metodi di risoluzione per QNB multiclasse

A differenza della vasta letteratura che è possibile trovare su algoritmi in grado di risolvere altre tipologie di reti, per l'analisi per reti di code multiclasse con blocco finora sono state proposte poche soluzioni. In questa sezione mostreremo un'estensione del Maximum Entropy Algoritm che ne consente l'utilizzo con QNB chiuse con blocco repetitive service. Come nelle precedenti versione, l'algoritmo calcola valori approssimati di $\pi(\mathbf{S})$, massimizzando la funzione entropica soggetta a determinati vincoli (v. cap. 9.2.1), ma in queste reti dovremo ridefinire lo stato $\mathbf{S}$ come segue:

$$\mathbf{S} = (\mathbf{S_1}, \mathbf{S_2}, ..., \mathbf{S_M})$$

dove

$$\mathbf{S_k} = (n_{k1}, n_{k2}, ..., n_{kR}), \qquad 1 \leq k \leq M$$

con M e R che indicano rispettivamente la cardinalità di nodi e classi. Lo stato infatti dovrà differenziare all'interno di ogni nodo, il numero d'utenti delle diverse classi.

Dovremo ridefinire anche la formula per il calcolo della probabilità $\pi(\mathbf{S})$:

$$\pi(\mathbf{S}) = \frac{1}{Z(\mathbf{L_c}, M)} \prod_{k=1}^{M} \omega_k(\mathbf{S_k}) \tag{10.1}$$

dove

$$\omega_k(\mathbf{S_k}) = \begin{cases} 1, & \sum_{i=1}^{R} n_{ki} = K_k \\ \prod_{i=1}^{R} g_{ki} x_{ki}^{n_{ki}} y_{ki}^{f_{ki}(S_k)}, & K_k < \sum_{i=1}^{R} n_{ki} \leq L_k \end{cases} \tag{10.2}$$

dove g_{ki}, x_{ki} e y_{ki}, per $k = 1, 2, ..., M$ e $i = 1, 2, ...R$, sono i coefficienti lagrangiani corrispondenti ai vincoli 2, 3 e 4 (v. cap. 9.2.1), $\mathbf{L_c}$ è il vettore che rappresenta il numero totale di utenti di ogni classe nella rete, e dove L_k e K_k indicano rispettivamente la capacità effettiva massima e minima di un nodo k. Infine, la costante di normalizzazione $Z(\mathbf{L_c}, M)$ è data dalla seguente formula:

$$Z(\mathbf{L_c}, M) = \sum_{\mathbf{S} \in \mathbf{S}(\mathbf{L_c}, M)} \prod_{k=1}^{M} \omega_k(\mathbf{S_k}) \tag{10.3}$$

Definite queste modifiche, l'algoritmo esteso utilizza la stessa tecnica vista per le reti a singola classe, e cioè risolve la rete chiusa analizzando una corrispondete pseudo-rete aperta senza arrivi nè partenze verso l'esterno, per poi ricavarne i risultati desiderati. Il metodo è efficiente e, nelle verifiche sperimentali, dimostra una grande accuratezza dei risultati. Per maggiori informazioni si consiglia la consultazione della bibliografia (v. [KA95]).

10.2 Proposta di un metodo per reti di code con blocco multiclasse

In questa sezione proveremo ad estendere l'algoritmo Acyclic Network Decomposition (v. 9.2.2) a reti di code aperte e acicliche, con blocco BAS e multiclasse. Per semplicità, utilizzeremo solamente reti multiclasse composte da catene monoclasse, come del resto è stato fatto anche in fase di definizione dei modelli multiclasse nel capitolo 6.4, e da nodi con una sola coda condivisa.

Consideriamo un modello a rete di code formato da M nodi e con una popolazione divisa in R catene. Ogni nodo i riceve utenti di una catena r dall'esterno del sistema con un tasso λ_{ir} ed ha tassi di servizio μ_{ir}, con $r \in R$. Ogni utente si muove all'interno della rete seguendo le probabilità della matrice di routing P_r, composta da elementi p_{ijr}, tali che i indica il nodo di partenza, j il nodo di destinazione e r la catena a cui appartiene l'utente. Inoltre i nodi sono ordinati in modo tale che ogni matrice di routing risulti essere triangolare superiore.

Le funzioni I_i e O_i, già viste nell'algoritmo originale, dovranno essere ridefinite:

$$I_i = \begin{cases} 0, & \text{se } \sum_{r=1}^{R} \lambda_{ir} = 0; \\ 1, & \text{se } \sum_{r=1}^{R} \lambda_{ir} \neq 0. \end{cases} \qquad O_i = \begin{cases} 0, & \text{se } \sum_{r=1}^{R} p_{i0r} = 0; \\ 1, & \text{se } \sum_{r=1}^{R} p_{i0r} \neq 0. \end{cases}$$

Gli insiemi U_i e D_i, che contengono rispettivamente gli indici delle sorgenti e delle destinazioni di un nodo i saranno definite nel seguente

modo:

$$\begin{aligned} U_i &= \{j : \exists\ r \text{ tale che } p_{jir} > 0,\ 0 \leq j, i \leq M\, e\, 0 \leq r \leq R\} \bigcup I_i\{0\} \\ D_i &= \{j : \exists\ r \text{ tale che } p_{ijr} > 0,\ 0 \leq j, i \leq M\, e\, 0 \leq r \leq R\} \bigcup O_i\{0\} \end{aligned}$$

Come l'algoritmo originale, anche questo metodo analizza la rete come M sottoreti distinte T(i) formate da un unico nodo i, con tassi d'arrivo dai nodi $j \in U_i$ indicati con il simbolo $\mu_{uj}(i)$ e con tassi di servizio $\mu_{dr}(i)$. La probabilità di stato $\pi_i(n)$, dove con n si intende la popolazione totale del nodo indipendentemente dalle catene d'appartenenza degli utenti, può essere ottenuta da un processo markoviano di nascita-morte con tasso di morte $\mu_d(n)$, costante per ogni stato n, e un tasso di nascita $\mu^*(i)$ già definito dalla formula (9.18). Rimangono invariate anche le definizioni delle variabili $p_s(i)$ e $p_{bj}(k : i)$ (v. le formule (9.19) e (9.20)), che indicano rispettivamente la probabilità che alla fine di un servizio il sistema T(i) sia vuoto e la probabilità che la j-esima sorgente trovi k altre sorgenti bloccate dal nodo i pieno. Oltre alla variabile $p_s(i)$ è utile però anche definire $p_{sr}(i)$, che riguarda solamente gli utenti di una determinata catena r:

$$p_{sr}(i) = \pi_{ir}(1)/[1 - \pi_{ir}(0)] \tag{10.4}$$

dove $\pi_{ir}(n)$ indica la probabilità che al nodo i ci siano n utenti della catena r. I tassi d'arrivo $\mu_{uj}(i)$ ed i tassi di servizio $\mu_d(i)$ sono definiti dalle seguenti formule:

$$\mu_{uj}(i) = \sum_{r=1}^{R} \mu_{ujr}(i) \tag{10.5}$$

$$\mu_{ujr}(i) = p_{tir}[\frac{p_{sr}(t)}{\mu^*_r(t)} + \frac{1}{\mu_{dr}(t)} - p_{tir} \sum_{k=0}^{|U_i|-1} p_{bj}(k:i)\frac{k+1}{\mu_{dr}(i)}]^{-1} \quad (10.6)$$

$$\text{con } \mu^*_r(i) = \sum_{j=1}^{|U_i|} \mu_{ujr}(i)$$

$$\mu_{dr}(i) = [\frac{1}{\mu_{ir}} + \sum_{m \in D_i} p_{imr} \sum_{k=0}^{|U_m|-1} p_{bt}(k:m)\frac{k+1}{\mu_{dr}(m)}]^{-1} \quad (10.7)$$

dove t indica l'indice del j-esimo nodo dell'insieme U_i.

Abbiamo quindi che $\mu_{uj}(i)$ risulta essere la sommatoria dei tassi d'arrivo degli utenti delle diverse catene che dal nodo j giungono al nodo i. Per la legge di conservazione del throughput, avremo che il throughput $X_{uj}(i)$ prodotto dalla j-esima sorgente verso il nodo i è definibile nel seguente modo:

$$X_{uj}(i) = \sum_{r=1}^{R} X_{dr}(k) p_{kir}$$

dove $X_{dr}(k)$ indica il throughput di utenti dalla catena r in uscita dal nodo k, che corrisponde alla j-esima sorgente del nodo i. Come già fatto per l'algoritmo originale, possiamo ora utilizzare le variabili $\mu_{uj}(i)$ e $\mu_d(i)$ per riscrivere le equazioni che calcolano i throughput:

$$X_{uj}(i) = [\frac{1}{\mu_{uj}(i)} + \sum_{k=0}^{|U_j|-1} \frac{p_{bj}(k:i)(k+1)}{\sum_{r=1}^{R} \mu_{dr}(i)}]^{-1}$$

$$X_{dr}(i) = [\frac{1}{\mu_{dr}(i)} + \frac{p_{sr}(k)}{\mu^*_r(i)}]^{-1}$$

Ora che abbiamo definito tutte le formule per questo algoritmo multiclasse, possiamo descriverne il comportamento:

01. $\mu_{u1}(i) = \sum_{r=1}^{R} \lambda_{ir}$, $\forall i$ tale che $U_i \neq \oslash$
02. $\mu_{dr}(i) = \mu_{ir}$, $\forall i$ tale che $D_i \neq \oslash$ e $\forall r$
03. $p_{bj}(n : i) = 0$, $j \in U_i$, $0 \leq n \leq \mid U_i \mid$, $\forall i$
04. repeat
05. forward cycle: for $i = 1, \ldots, M$
06. calcola $p_{sr}(i)$ utilizzando la formula (10.4) $\forall r$
07. calcola $\mu_{uj}(i) \forall j \in U_i$ utilizzando la formula (10.5)
08. backward cycle: for $i = M, \ldots, 1$
09. calcola $\mu_{dr}(i)$ utilizzando la formula (10.7), $\forall r$
10. calcola $p_{bj}(k : i)\, 0 \leq k \leq \mid U_i \mid -1$, $j \in U_i$ con la formula (9.20)
11. until $max_i \mid \sum_{r=1}^{R} updated\mu_{dr}(i) - \sum_{r=1}^{R} \mu_{dr}(i) \mid \leq \epsilon$

La complessità di questo algoritmo è $O(kMR[(U+B^+)^2+U^3+2^{U+1}])$, dove con k si indica il numero delle iterazioni che portano alla convergenza e dove $U = max_i \mid U_i \mid$. Da notare che si differenzia dall'algoritmo originario solo per la presenza di R, che indica il numero di code presenti nella rete. Con R=1 infatti si tornerebbe alla complessità dell'Acyclic Network Decomposition a singola classe, ed inoltre, essendo l'AND classico espansione dell'algoritmo Tandem Exponential Network Decomposition, si otterrebbe con R=1 e U=1 proprio la complessità di quest'ultimo. Infatti U=1 per ogni nodo soltanto quando la rete è di tipo tandem.
Dagli esperimenti fatti su diverse reti (utilizzando un programma sviluppato in Java, il cui codice si può trovare in appendice) l'algoritmo ha

sempre trovato convergenza in un numero limitato di passi.

10.2.1 Sperimentazioni

Questa versione dell'algoritmo Acyclic Network Decomposition esteso alle reti multiclasse con blocco vuole, in questo elaborato, essere soltanto una proposta, e, in quest'ottica, non sono ancora stati fatti test approfonditi sulla sua accuratezza. I test compiuti finora si sono concentrati sulle prestazioni dell'algoritmo e sul suo corretto funzionamento con reti multiclasse senza blocco utilizzando l'implementazione in Java consultabile in appendice e, per confrontare i dati, il programma RAQS[1]. Nelle tabelle che seguono sono stati riportati alcuni significativi risultati sulla velocità di convergenza dell'algoritmo (e cioè sul numero di iterazioni di cui necessita per fornire una soluzione), e ne emerge globalmente una buona efficienza. E' stato notato, durante questi test, che il numero di iterazioni non dipende strettamente dalla quantità di nodi o di catene da cui è formata la rete, ma piuttosto dalla capienza dei buffer e dal rapporto tra tassi d'arrivo e tassi di servizio: più cresce la probabilità di un blocco al nodo e più iterazioni l'algoritmo impiega per terminare. Il numero di iterazioni naturalmente dipende anche dalla grandezza della variabile ϵ, che influenza la condizione di uscita: nei nostri test abbiamo utilizzato un valore molto piccolo (1.0E-9) e sarà compito dei test sull'accuratezza scoprire se è possibile utilizzare anche valori più grandi. In ogni prova da

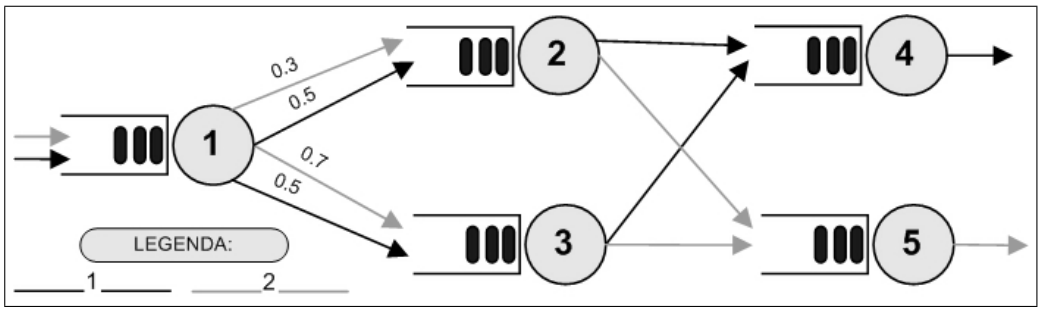
[1]RAQS - A Queueing Solver, Url: http://www.okstate.edu/cocim/raqs/

noi compiuta, comunque, l'algoritmo tende in modo costante a convergere.

Utilizzando il programma citato pocanzi, è stata poi provata la correttezza di questo algoritmo con reti multiclasse senza blocco, basando il confronto su indici come l'utilizzazione, il numero medio d'utenti ad un nodo ed il throughput. Con le reti utilizzate, l'algoritmo si è sempre dimostrato preciso e ha sempre fornito risultati corretti.

Come già detto, per terminare i test su questa estensione del metodo Acyclic Network Decomposition, sarà necessario compiere dei test sull'accuratezza dei risultati forniti analizzando reti di code multiclasse con blocco. Quest'ulteriore lavoro, che comporterà il confronto tra i risultati dell'algoritmo con quelli di un simulatore, viene rimandato ad un successivo lavoro di verifica più approfondito.

Figura 10.1: Rete di code multiclasse con 5 nodi e 2 catene



M	R	λ = tassi d'arrivo	μ = tassi di servizio	Capacità	Iter.
5	2	λ_1=(2,5;0;0;0;0) λ_2=(3;0;0;0;0)	μ_1=(3;4;3,5;4;0) μ_2=(3;2;3;0;5)	(5;5;5;5;5)	10
5	2	λ_1=(1,5;0;0;0;0) λ_2=(1;0;0;0;0)	μ_1=(3;4;3,5;4;0) μ_2=(3;2;3;0;5)	(10;10;10;10;10)	6
5	2	λ_1=(3;0;0;0;0) λ_2=(3,5;0;0;0;0)	μ_1=(3;4;3,5;4;0) μ_2=(3,5;2,5;3;0;5)	(5;5;5;5;5)	12
5	2	λ_1=(3;0;0;0;0) λ_2=(3,5;0;0;0;0)	μ_1=(3;4;3,5;4;0) μ_2=(3,5;2,5;3;0;5)	(10;10;10;10;10)	8
5	2	λ_1=(3;0;0;0;0) λ_2=(3,5;0;0;0;0)	μ_1=(5;5;5;5;5) μ_2=(5;5;5;5;5)	(10;10;10;10;10)	8
5	2	λ_1=(3;0;0;0;0) λ_2=(3,5;0;0;0;0)	μ_1=(5;5;5;5;5) μ_2=(5;5;5;5;5)	(10;1;1;10;10)	8

Tabella 10.1: il numero di iterazioni compiute dall'algoritmo con la rete di code con la topologia di fig. 10.1, al variare dei tassi d'arrivo, dei tassi di servizio e della capacità dei nodi.

Figura 10.2: Rete di code multiclasse con 3 nodi e 3 catene

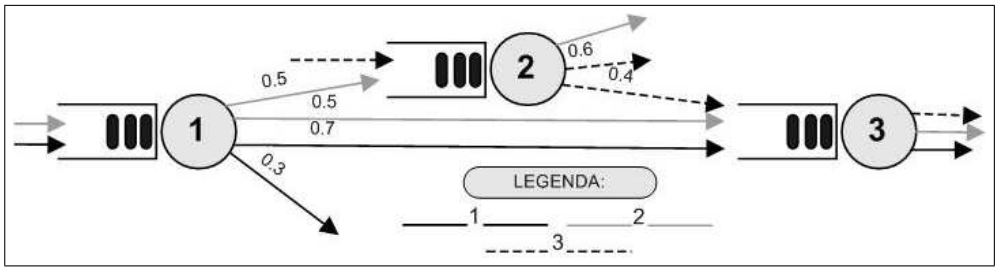
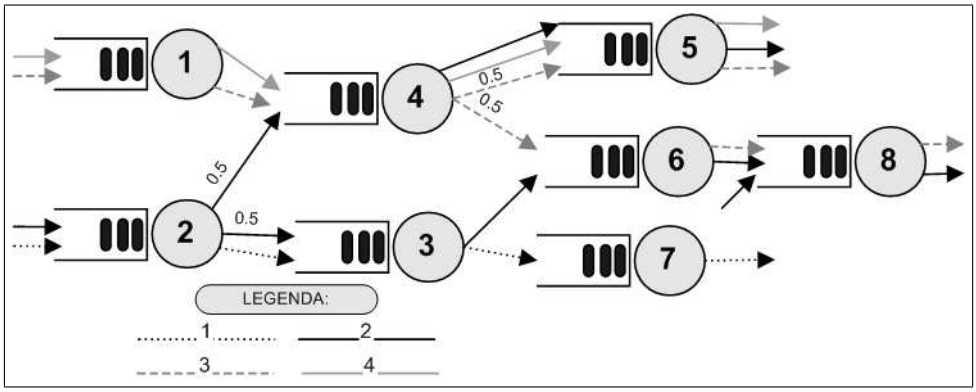


Figura 10.3: Rete di code multiclasse con 8 nodi e 4 catene



M	R	λ = tassi d'arrivo	μ = tassi di servizio	Capacità	Iter.
3	3	λ_1=(3;0;0)	μ_1=(4;0;3)	(10;10;10)	5
		λ_2=(2,5;0;0)	μ_2=(3,5;3;2,5)		
		λ_3=(0;2;0)	μ_3=(0;3;4)		
3	3	λ_1=(3;0;0)	μ_1=(3,1;0;3)	(10;10;7)	13
		λ_2=(2,5;0;0)	μ_2=(3,5;3;2,5)		
		λ_3=(0;2;0)	μ_3=(0;3;2)		
3	3	λ_1=(1,5;0;0)	μ_1=(2,5;0;3)	(10;10;10)	3
		λ_2=(2;0;0)	μ_2=(3;3,2;3,5)		
		λ_3=(0;1,5;0)	μ_3=(0;3;2)		
3	3	λ_1=(1,5;0;0)	μ_1=(2,5;0;3)	(20;20;20)	2
		λ_2=(2;0;0)	μ_2=(3;3,2;3,5)		
		λ_3=(0;1,5;0)	μ_3=(0;3;2)		

Tabella 10.2: il numero di iterazioni compiute dall'algoritmo con la rete di code con la topologia di fig. 10.2, al variare dei tassi d'arrivo, dei tassi di servizio e della capacità dei nodi.

M	R	λ = tassi d'arrivo	μ = tassi di servizio	Capacità	It.
8	4	λ_1=(0;2;0;0;0;0;0;0)	μ_1=(0;3;3;0;0;0;3,5;0)	(13;13;10;15;10;8;5;12)	6
		λ_2=(0;2,5;0;0;0;0;0;1)	μ_2=(0;2,5;2;2;2;3;0;4)		
		λ_3=(1,5;0;0;0;0;0;0;0)	μ_3=(3;0;0;3,2;2,5;3;0;3)		
		λ_4=(1,7;0;0;0;0;0;0;0)	μ_4=(3,5;0;0;4;3,5;0;0;0)		
8	4	λ_1=(0;2;0;0;0;0;0;0)	μ_1=(0;3;3;0;0;0;3,5;0)	(13;13;10;5;10;8;5;12)	11
		λ_2=(0;2,5;0;0;0;0;0;1)	μ_2=(0;2,5;2;2;2;3;0;4)		
		λ_3=(1,5;0;0;0;0;0;0;0)	μ_3=(3;0;0;3,2;2,5;3;0;3)		
		λ_4=(3;0;0;0;0;0;0;0)	μ_4=(3,5;0;0;3;3,5;0;0;0)		
8	4	λ_1=(0;1;0;0;0;0;0;0)	μ_1=(0;3;3;0;0;0;3,5;0)	(10;10;10;7;10;8;5;12)	6
		λ_2=(0;0,5;0;0;0;0;0;1)	μ_2=(0;2,5;2;2;2;3;0;3)		
		λ_3=(1;0;0;0;0;0;0;0)	μ_3=(3;0;0;3,2;2,5;3;0;3)		
		λ_4=(2;0;0;0;0;0;0;0)	μ_4=(3,5;0;0;3;3,5;0;0;0)		
8	4	λ_1=(0;1;0;0;0;0;0;0)	μ_1=(0;4;4;0;0;0;4,5;0)	(10;10;10;7;10;8;5;12)	5
		λ_2=(0;0,5;0;0;0;0;0;1)	μ_2=(0;3,5;3;3;3;4;0;4)		
		λ_3=(1;0;0;0;0;0;0;0)	μ_3=(4;0;0;4,2;3,5;4;0;4)		
		λ_4=(2;0;0;0;0;0;0;0)	μ_4=(4,5;0;0;4;4,5;0;0;0)		

Tabella 10.3: il numero di iterazioni compiute dall'algoritmo con la rete di code con la topologia di fig. 10.3, al variare dei tassi d'arrivo, dei tassi di servizio e della capacità dei nodi.

Capitolo 11

Conclusioni

I due obbiettivi su cui si è intenso concentrare la ricerca in questo elaborato sono la definizione dei modelli a rete di code multiclasse con blocco e la loro risoluzione. Il primo obbiettivo è stato raggiunto nella prima parte, fornendo una definizione chiara delle reti Markoviane e delle reti in forma prodotto con utenza multiclasse e con diverse tipologie di blocco. Per far ciò, è stato scelto un approccio al problema di tipo progressivo, introducendo concetti via via più complessi, fino a portare il lettore ad una facile comprensione della definizione di questi modelli. Il secondo obbiettivo è stato raggiunto invece nella seconda parte, dove, oltre a descrivere i metodi di risoluzione già presenti in letteratura, viene proposto un nuovo algoritmo, che, estendendo il metodo Acyclic Network Decomposition, riesce a fornire, con un complessità polinomiale, una soluzione approssimata delle reti di code aperte, acicliche, multiclasse e con nodi

a buffer finito. Gli iniziali test finora svolti sull'algoritmo, hanno dato buoni risultati sia in termini di prestazioni che di accuratezza, ma sarà necessario proseguire, nei successivi lavori, la fase di sperimentazione per poter valutare a pieno le potenzialità di questo nuovo metodo.

Appendice A

Implementazione di AND per QNB multiclasse

In questa appendice riportiamo una possibile implementazione, scritta in linguaggio Java, dell'algoritmo Acyclic Network Decomposition per reti di code con blocco multiclasse spiegato al capitolo 10.2. Per implementare questo metodo sono state create tre classi:

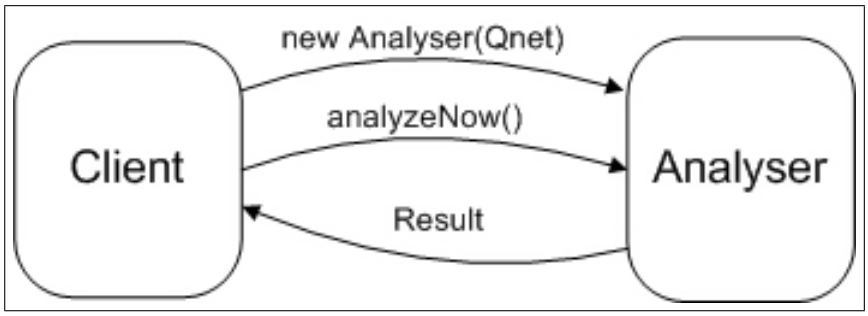
Qnet: questa classe contiene la struttura della rete di code che l'utente desidera far analizzare. Il programma dell'utente infatti dovrà utilizzare oggetti di questa classe per interagire con l'analizzatore;

Analyser è la classe che ha compito di analizzare la rete seguendo la linea guida dell'algoritmo proposto al capitolo 10.2. In input potrà ricevere solamente reti che siano oggetti di classe Qnet e restituirà in output oggetti Result;

Result contiene i risultati dei calcoli fatti da Analyser. Conterrà i throughput, i tassi d'entrata e i tassi d'uscita d'ogni nodo.

Il cuore del programma, dove verrà effettivamente applicato l'algoritmo, sarà quindi Analyser. Le altre due classi forniscono solamente delle strutture adeguate per la rappresentazione dei dati in input e dei risultati in output. Nella figura A.1 è possibile vedere schematicamente come un oggetto di classe Analyser interagisca con i programmi clienti. Ogni oggetto analizza una sola rete Qnet a cui viene abbinato in fase di inizializzazione. Una volta che il cliente ha creato l'oggetto, può chiamare il metodo analyzeNow() che analizzerà la rete e restituirà in output un oggetto Result, contenente i risultati. Nelle pagine seguenti riportiamo il codice dei tre file corrispondenti alle tre classi sopra descritte.

Figura A.1: Schema di funzionamento del programma



```
package netAnalyser;

//Questa classe definisce l'oggetto rete di code
class QNet {

    private double service[][];  //tassi di servizio per ogni nodo e catena
    private double arrives[][];  //tassi di arrivo per ogni nodo e catena
    private double routing[][][];//tabelle di routing di ogni catena
    private int nodes, chains;   //cardinalita' dei nodi e delle catene
    private int [] capacity;     //capacita' dei nodi

    public QNet(int nodi, int nchains, double rout[][][], double servRate
[][],double arRate[][],int cap[]){
    nodes = nodi;
    chains = nchains;
    service = servRate;
    routing = rout;
    arrives=arRate;
    capacity=cap;
    }


    public int getCapacity(int node){
        return capacity[node];
    }

    public double getServiceRate(int node, int chain){
        return service[node][chain];
    }
    public double getArriveRate(int node, int chain){
        return arrives[node][chain];
    }

    public double getProb(int chain, int row, int column)
    {
        return routing[chain][row][column];
    }

    public int getChains()
    {
        return chains;
    }

        public int getNodes()
    {
        return nodes;
    }
}
```

```
package netAnalyser;

/*Questa classe contiene l'algoritmo vero e proprio, che, ricevuta una rete
 *in input (sotto forma di oggetto QNet) l'analizza producendo i risultati
 *desiderati.
 */
class Analyser {
    //mu[i][j] contiene il tasso d'arrivo dal nodo j alla subnetwork
composta
    //dal nodo i
    private double mu[][];
    //mu[i][j][r] e' il tasso d'arrivo dal nodo j al nodo i degli utenti
della
    //catena r
    private double mur[][][];
    //mdr[i][r] e' il tasso di servizio del nodo i per la catena r
    private double mdr[][];
    //ps[i][r] e' la probabilita' che alla fine di un servizio ad un utente
    //della catena r, la sottorete T(i) rimanga priva di utenti di quella
catena
    private double ps[][];
    //pb[i][j][k] e' la probabilita' che la j-esima sorgente trovi k altre
    //sorgenti bloccate al nodo i
    private double pb[][][];
    private QNet qnetwork;
    private int[][] Ui, Di;
    private double mrstar[][];
    private double xuj[][]; //throughput entrante al nodo i per ogni
sorgente j
  private double xdr[][]; //throughput uscente dal nodo i per ogni catena r
    private int M,R; //M e' il numero di nodi e R il numero di catene
    private double epsilon=0.000000001;

    //il costruttore inizializza alcune variabili e alloca la memoria
    public Analyser(QNet net) {
        int i;
        qnetwork = net;
        M=qnetwork.getNodes();
        R=qnetwork.getChains();
        Ui = new int[M+1][0];
        Di = new int[M+1][0];
        initUi();
        initDi();
        M = qnetwork.getNodes();
        R = qnetwork.getChains();
        mu = new double[M][];
        mur = new double[M][][];
        mdr = new double[M][R];
        ps = new double[M][R];
        mrstar = new double[M][R];
        xuj = new double[M][];
        xdr = new double[M][R];
        pb = new double[M][][];
        for(i=0;i<M;i++){
            pb[i]= new double[Ui[i].length][Ui[i].length];
```

```
            xuj[i] = new double[Ui[i].length];
            mu[i]=new double[Ui[i].length];
            mur[i]= new double[Ui[i].length][R];
        }

    }

//inizializza l'insieme Ui che contiene tutte le possibili sorgenti di ogni
    //nodo
    private void initUi(){
        int i,j,r,l,h;
        boolean test=false;
        int [] temp;
        for(i=1;i<M+1;i++) {
            for(j=0;j<M+1;j++) {
                for(r=0;r<R;r++) {
                //controlla nella matrice di routing di ogni catena r se il
                    //nodo j e' una possibile sorgente del nodo i
                    if(qnetwork.getProb(r,j,i)>0)
                        test = test || true;
                    }
                    //se j puo' essere sorgente, lo inserisce nell'insieme
                    if(test){
                        l= (Ui[i-1].length)+1;
                        temp = new int[l];
                        for(h=0;h<Ui[i-1].length;h++)
                            temp[h]=Ui[i-1][h];
                        temp[h]=j;
                        Ui[i-1]=temp;
                        }
                    test=false;
                }
            }
    }

    //inizializza l'insieme Di che contiene tutte le possibili destinazioni
    //di ogni nodo
    private void initDi(){
        int i,j,r,l,h;
        boolean test=false;
        int [] temp;
        for(i=0;i<M+1;i++) {
            for(j=0;j<M+1;j++) {
                for(r=0;r<R;r++) {
                //controlla nella matrice di routing di ogni catena r se il
                    //nodo j e' una possibile destinazione del nodo i
                    if(qnetwork.getProb(r,i,j)>0)
                        test = test || true;
                    }
                    if(test){
                        l= Di[i].length+1;
                        temp = new int[l];
                        for(h=0;h<Di[i].length;h++)
                            temp[h]=Di[i][h];
                        temp[h]=j;
```

```
                        Di[i]=temp;
                        }
                    test=false;
                }
            }
    }
//permette di impostare la precisione da avere nella condizione di uscita
    //di default epsilon=1.0E-9
    public void setEpsilon(double newEpsilon){
        epsilon=newEpsilon;
    }
    //e' il metodo principale ed applica l'algoritmo per il calcolo delle
    //prestazioni
    public Result analyzeNow(){

        int i,j,u1,u2,r,k,count=0,unit;
        double sum,diff=0,sum1=0,mdtemp=0;
        double throughputIn=0,throughputOut=0;
        boolean test;
        double mdrTot=0,mdrMed=0,rho=0,muTot=0;
        double verifica [][] = new double[M][2];

        //Inizializzazione
        for(i=0;i<M;i++){
            sum=0;
            verifica[i][0]=0;
            verifica[i][1]=0;
            for(r=0;r<R;r++){
                mur[i][0][r] = qnetwork.getArriveRate(i,r);
                sum+=mur[i][0][r];
            }
            mu[i][0]=sum;
            for(j=1;j<Ui[i].length;j++){
                mu[i][j]=0;
                for(r=0;r<R;r++){
                    mur[i][j][r]=0;
                        }
            }
        }
        for(i=0;i<M;i++){
            for(r=0;r<R;r++)
                mdr[i][r]=qnetwork.getServiceRate(i,r);
        }
        for(i=0;i<M;i++){
            for(u1=0;u1<Ui[i].length;u1++){
                for(u2=0;u2<Ui[i].length;u2++)
                    pb[i][u1][u2]=0;
                }
        }
        //Fine Inizializzazione
        do{
            //forward cycle
            for(i=0;i<M;i++){
                for(r=0;r<R;r++)
                    ps[i][r]=(piGreco(i, r,1))/(1-piGreco(i,r,0));
```

```
                for(j=0;j<Ui[i].length;j++){
                    mu[i][j]=0;
                    for(r=0;r<R;r++)
                        mu[i][j]+=setMur(i,j,r);
                }
            }
            //backward cycle
            for(i=M-1;i>=0;i--){
                for(r=0;r<R;r++)
                    setMdr(i,r);
                if(Ui[i][0]==0)
                    unit=1;
                else unit=0;
                for(j=(0+unit);j<Ui[i].length;j++){
                    if(Ui[i][j]!=0)
                        for(k=0;k<(Ui[i].length-unit);k++){
                            setPb(i,j,k);
                            }
                }
            }

            //calcolo il throughput in entrata e in uscita per ogni nodo.
            //il throughput in uscita viene calcolato per ogni catena.
            for(i=0;i<M;i++){
                for(j=0;j<Ui[i].length;j++)
                    xuj[i][j]=setXuj(i,j);
                for(r=0;r<R;r++)
                    xdr[i][r]=setXdr(i,r);
            }

            test=false;
            diff=sum1;
            sum1=0;
            //verifica la condizione d'uscita
            for(i=0;i<M;i++){

                //calcolo il throughput per ogni nodo
                throughputIn=0;
                for(j=0;j<Ui[i].length;j++)
                    throughputIn+=xuj[i][j];
                throughputOut=0;
                for(j=0;j<Ui[i].length;j++){
                    for(r=0;r<R;r++){
                        u1=Ui[i][j]-1;
                        if(Ui[i][j]==0)
                            throughputOut+=qnetwork.getArriveRate(i,r);
                        else throughputOut+=(xdr[Ui[i][j]-1][r]*qnetwork.
getProb(r,Ui[i][j],i+1));
                }   }
                sum1+=throughputIn+throughputOut;
                diff-=(throughputIn+throughputOut);
                //verifica della variazione di md
                mdtemp=0;
                for(r=0;r<R;r++)
                    mdtemp+=mdr[i][r];
```

```
                verifica[i][1]= Math.abs(1/mdtemp-verifica[i][0]);
                verifica[i][0]=1/mdtemp;
            }

        for(i=0;i<M;i++)
            if(verifica[i][1]<epsilon)
                test=true;
        count++;


        }while(count<1000 && (!test || (Math.abs(diff)>=epsilon || diff==(
Double.NaN))));

        System.out.println("La rete e' stata risolta in "+(count)+"
iterazioni");

        return new Result(mu,mur,mdr,xuj,xdr);

    }

//calcola il throughput in entrata che node riceve dalla sorgente source
    private double setXuj(int node,int source){

        int k,i;
        double temp=0,mdrTot=0;
        for(i=0;i<R;i++)
            mdrTot+=mdr[node][i];
        for(k=0;k<Ui[node].length-1;k++)
            temp += pb[node][source][k] * ((k+1)/mdrTot);
        return 1/((1/mu[node][source])+temp);
    }

    //calcola il throughput in uscita di node per la catena chain
    private double setXdr(int node, int chain){
        setMrstar(node,chain);
        if(mrstar[node][chain]==0)
        return 0;
    return 1/((1/mdr[node][chain])+(ps[node][chain]/mrstar[node][chain]));
    }

    //imposta la variabile pb[node][source][k] secondo la formula 9.18
    private void setPb(int node, int source, int k){
        int j;
        double pi,buj=0,bujpartial;
        if(Ui[node].length==1 && Ui[node][source]==0){
            pb[node][source][k]=0;
            return;
        }
        pi=piGrecoB(node,(k+qnetwork.getCapacity(node)));
        if(k!=0){
        bujpartial=setBuj(node,source,k,pi);
        for(j=1;j<Ui[node].length;j++)
            buj+=setBuj(node,source,j,pi);
        }else bujpartial=0;
        pb[node][source][k]=(pi-bujpartial)/(1-buj);
```

```
    }

//calcola la variabile buj secondo la formula 9.21
private double setBuj(int node, int source, int k, double piGreco){

double temp;
temp=(mu[node][source]*setOmega(node,k-1))/setOmega(node,k);
return temp*piGreco;
}

//calcola Omega seconda secondo la formula 9.18
private double setOmega(int node, int k){

    int n=k;
    int U;
    boolean reset;
    int array[]= new int[n];
    int i,j,m=0,l;
    double sum=0,prod=1;
    if(Ui[node].length<1 || k==0)
        return 1;
    if(Ui[node][0]==0)
        m=1;
    U=(Ui[node].length)-m;
    for(i=0;i<n;i++)
        array[i]=i+1;
    if(k==0 || U<=0 || n==0)
        return 1;
    while(array[0]<=(U-(n-1))) {

        for(l=0;l<n;l++) {
            prod*=mu[node][array[l]-1+m];
            }
        sum+=prod;
        prod=1;

        reset=false;

        for(i=n-1;i>=0;i--){
            if(reset){
                array[i]++;
                if(array[i]<=(U-(n-(i+1)))){
                    for(j=1;(i+j)<=(n-1);j++)
                        array[i+j]=array[i]+j;
                    reset=false;
                }
            }else if(i==n-1){
                array[i]++;
                if(array[i]>(U-(n-(i+1))))
                    reset=true;
            }
        }
    }
return sum;
```

```
    }

//calcola la probabilita' che in node ci siano n utenti della catena chain
    private double piGreco(int node, int chain, int n){

        double rho,rhon=1,rhon2=1,murTot=0,muTot=0,mdrTot=0,ris;
        int i;
        for(i=0;i<Ui[node].length;i++){
            murTot+=mur[node][i][chain];
        }
        rho = murTot/mdr[node][chain];
        if(!(rho>0 || rho<0))
            return 0;

        for(i=0;i<n;i++)
            rhon*=rho;
        ris=rhon*(1-rho);
        return rhon*(1-rho);
    }
//calcola la probabilita' che in node ci siano n utenti
    private double piGrecoB(int node, int n){

        int i,k,j;
        double sum,prod,p0,lambda,mi;

        //calcolo la prob che il nodo sia vuoto
        sum=0;
        for(i=0;i<(qnetwork.getCapacity(node)+(Ui[node].length-1));i++){
            prod=1;
            for(k=0;k<=i;k++){
                if(k<qnetwork.getCapacity(node)){
                    lambda=0;
                    for(j=0;j<R;j++)
                        lambda=lambda+mrstar[node][j];
                }else
                    lambda=(k*setOmega(node,(k-qnetwork.getCapacity(node)+1
))/setOmega(node,k-qnetwork.getCapacity(node)));
                mi=0;
                for(j=0;j<R;j++)
                    mi+=mdr[node][j];
                prod*=lambda/mi;
            }
            sum+=prod;
        }
        sum++;

        p0=1/sum;

        prod=1;
        for(k=1;k<n;k++){
            if(k<=qnetwork.getCapacity(node) || (Ui[node].length==1 && Ui[
node][0]==0)){
                lambda=0;
                for(j=0;j<R;j++)
                    lambda+=mrstar[node][j];
```

```
            }else
                lambda=(k*setOmega(node,k))/setOmega(node,k-1);
            mi=0;
            for(j=0;j<R;j++)
                mi+=mdr[node][j];
            prod*=lambda/mi;
        }
        return p0*prod;
    }

    //imposto la variabile mur[node][source][chain] secondo la formula 10.3
    private double setMur(int node, int source, int chain){

        int nodeSource=0,k;
        double temp1=0,temp2=0;
        if(source>=Ui[node].length)
        return 0;
        if(Ui[node][source]==0)
        mur[node][source][chain]=qnetwork.getArriveRate(node,chain);
        else{
        nodeSource = Ui[node][source]-1;
        setMrstar(nodeSource,chain);
        if(qnetwork.getProb(chain,nodeSource+1,node+1)==0)
            return 0;
        for(k=0;k<Ui[node].length-1;k++){
        temp1 += pb[node][source][k] * ((k+1)/mdr[node][chain]);
        }
        temp2 = ps[nodeSource][chain]/mrstar[nodeSource][chain];
        temp2 += 1/mdr[nodeSource][chain];
        temp2 -= qnetwork.getProb(chain,nodeSource+1,node+1)*temp1;
        mur[node][source][chain]=qnetwork.getProb(chain,nodeSource+1,node+1
)*(1/temp2);
        }
        return mur[node][source][chain];
    }

    //calcola mrstar[node][chain] secondo la formula 10.3
    private void setMrstar(int node, int chain){
        mrstar[node][chain]=0;
        for(int i=0;i<Ui[node].length;i++){
            if(mur[node][i][chain]>0 || mur[node][i][chain]<0)
            mrstar[node][chain]+=mur[node][i][chain];
            }
    }

    //calcola mdr[node][chain] secondo la formula 10.4
    private void setMdr(int node, int chain){

        int i,m,k,t=0,h,temp1=0,temp2=0;
        for(i=0;i<Di[node].length;i++){
            if(Di[node][i]!=0){
            m = Di[node][i]-1;
            //node e' la t-esima sorgente di m
            for(h=0;h<Ui[m].length;h++){
                if(Ui[m][h]-1==node)
```

```
                    t=h;
                }
            for(k=0;k<Ui[m].length-1;k++){
                temp1 += pb[m][t][k] * ((k+1)/mdr[m][chain]);
                }
            temp2 += qnetwork.getProb(chain,node+1,m+1)*temp1;
            }
        }
        mdr[node][chain]=1/((1/qnetwork.getServiceRate(node,chain))+temp2);
    }
}
```

```
package netAnalyser;

//gli oggetti di questa classe contengono i risultati dell'analisi della
//rete fatta con Analyser

class Result {
//mu[i][j] contiene il tasso d'arrivo dal nodo j alla subnetwork composta
//dal nodo i
    private double mu[][];
//mu[i][j][r] e' il tasso d'arrivo dal nodo j al nodo i degli utenti della
//catena r
    private double mur[][][];
//mdr[i][r] e' il tasso di servizio del nodo i per la catena r
    private double mdr[][];
//throughput entrante al nodo i per ogni sorgente j
    private double xuj[][];
//throughput uscente dal nodo i per ogni catena r
    private double xdr[][];

     public Result(double mu[][], double mur[][][], double mdr[][], double
xuj[][], double xdr[][])
     {
        this.mu=mu;
        this.mur=mur;
        this.mdr=mdr;
        this.xuj=xuj;
        this.xdr=xdr;
     }

     public double [][] getMu(){
            return mu;
     }

     public double [][][] getMur(){
            return mur;
     }

     public double [][] getMdr(){
            return mdr;
     }

     public double [][] getXuj(){
            return xuj;
     }

     public double [][] getXdr(){
            return xdr;
     }
}
```

Bibliografia

[AV89] Akyildiz I.F. e H. Von Brand, *Exact solutions for open, closed, and mixed queueing networks with rejection blocking*, Theoretical Computer Science , vol.64 (1989) pagg. 203-219

[Ba99] Balsamo S., *Appunti per il corso di Sistemi di Elaborazione dell'Informazione: valutazione delle prestazioni di sistema*, Url: http://www.dsi.unive.it/∼balsamo/dispense.html

[BCMP75] Baskett, M.K. Chandy, R.R. Muntz e F.G. Palacios, *Open, closed and mixed networks of queues with different classes of costumers*, J. ACM (1975)

[BPO01] S. Balsamo, V. de Nitto Personé, R. Onvural, *Analysis of queueing networks with blocking*, Kluwer's International Series (2001)

[CG86] Conway A.E. e N.D. Georganas, *RECAL - A New Efficient Algorithm for the Exact Analysis of Multiple-Chain Closed Queuing Networks*, Journal of the ACM, Vol. 33 (1986) 768-791

[CSL89] Conway A. E., De Souza E. Silva e Lavenberg S.S. *Mean value analysis by chian of product form queueing networks*, IEEE Trans on Comp Vol. 38 (1989) 434-442

[DSL89] De Souza E. Silva e Lavenberg S.S., *Calculating joint queue length distributions in product form queueing networks*, J. ACM, Vol. 36 (1989) 194-207

[GN67] Gordon W.J. e G.F. Newell, *Cyclic queueing systems with restricted queues*, Operation Research, Vol. 15 (1967) 286-302

[HBAK86] Hoyme K.P., S.C. Bruell, P.V. Afshari e R.Y. Kain, *A Tree-Structured Mean Value Analysis*, ACM Transactions on computer systems (TOCS), Vol. 4 (1986) 178-185

[JAC63] Jackson J.R., *Jobshop-like queueing systems*, Mgmt. Sci, 10, (1963)

[Ka92] Kant, *Introduction to Computer System Perfomance Evaluation*, McGraw-Hill (1992)

[KA95] Kouvatsos D. e I.U. Awan, *Arbitrary closed queueing networks with blocking and multiple job classes*, Proc. Third International Workshop on Queueing Networks with Finite Capacity, Bradford, UK (1995)

[KX89] Kouvatsos D. e N.P. Xenios, *Maximum Entropy Analysis of General Queueing Networks with Blocking*, First International Work-

shop on Queueing Networks whith Blocking,(Perros and Altiok Eds), Elsevier Science Publishers North Holland (1989)

[LBDF98] Lee H.S., A. Bouhcouch, Y. Dallery and Y. Frein, *Performance evaluation of open queueing networks with arbitrary configuration and finite buffer*, Annals of Operation Research, Vol. 78 (1998) 181-206

[LK00] Law Averill M. e W. D. Kelton, *Simulation modeling and analysis*, McGraw-Hill (2000)

[LL83] Lam Simon S. e Y. Luke Lien, *A Tree Convolution Algorithm for the Solution of Queueing Networks*, Communications of the ACM (1983) 203-215

[LP90] Lee H.S. e S.M. Pollock, *Approximate analysis of open acyclic exponential queueing networks with blocking*, Operations Research Vol.38 (1990) 1123-1134

[R91] Raj Jane, *The art of computer systems performance analysis*, John Wiley and Sons Inc. (1991)

[RK75] Reiser M. e H. Kobayashi, *Queueing networks with multiple closed chians: Theory and computational algorithms*, IBM J. Res. Dev. (1975) 283-294