

Introduzione alle Reti di Code

Ovvero: Le Reti di Code con le Quattro Operazioni

Moreno Marzolla
moreno.marzolla@pd.infn.it

INFN Sezione di Padova

29 maggio 2008



Indice

Introduzione

Notazione

Leggi Fondamentali

Analisi dei Limiti

Sistemi Aperti

Sistemi Chiusi

Mean Value Analysis

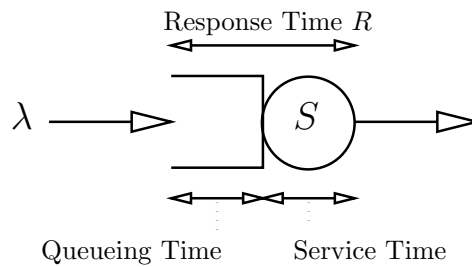
MVA per Reti Aperte

MVA per Reti Chiuse

Una Applicazione



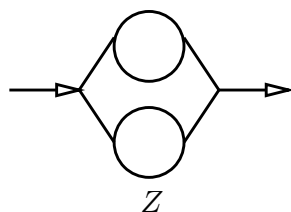
Centro di Servizio



- ▶ Nella sua forma più semplice, un centro di servizio (server) è rappresentato graficamente da un *server* con una *coda* associata
- ▶ Le richieste arrivano al centro di servizio con un certo tasso di arrivo λ . Se il server è occupato, vengono poste in coda
- ▶ Il server estrae le richieste dalla coda in base ad una opportuna politica di scheduling (es, FIFO), e processa le richieste con un tempo medio di servizio S
- ▶ Response Time = Service Time + Queueing Time



Delay Center



- ▶ delay center sono una forma particolare di centro di servizio: sono composti da un numero infinito di server identici
- ▶ Ogni richiesta in arrivo viene assegnata ad uno dei server liberi (ce ne sono sempre, essendo infiniti), quindi non si forma mai coda
- ▶ Le richieste spendono mediamente tempo Z in servizio, e poi proseguono
- ▶ Sono generalmente usati per modellare ritardi di quantità media Z



Reti di code

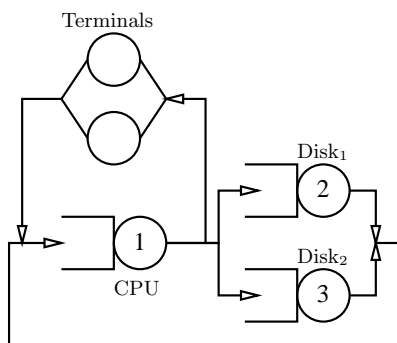
Una rete di code è un insieme di K centri di servizio interconnessi.

- ▶ La rete può essere aperta se ci sono arrivi di richieste dall'esterno del sistema
- ▶ La rete è chiusa se nel sistema circolano una popolazione fissa di N richieste.

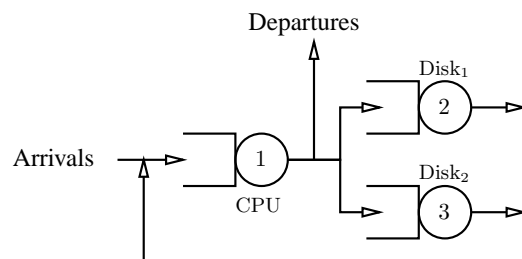


Reti di code

Esempio di rete chiusa

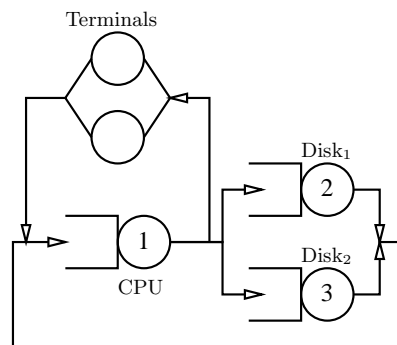


Esempio di rete aperta



Esempio: Central Server Model

Jeffrey P. Buzen: *Computational Algorithms for Closed Queueing Networks with Exponential Servers*. Comm. ACM 16(9): 527-531 (1973)



- ▶ Esiste una popolazione finita di N jobs
- ▶ Le richieste spendono un certo tempo tempo di attesa (think time) nei delay center rappresentati da *terminali*
- ▶ Il server centrale rappresenta la CPU
- ▶ Ulteriori server rappresentano periferiche di I/O



Analisi Operazionale

Definizione

Le ipotesi operazionalmente testabili (operationally testable) sono quelle che possono essere verificate mediante misure.

Esempio

È possibile verificare se in un intervallo di tempo T il numero di arrivi al centro di servizio i -esimo è uguale al numero di partenze; L'assunzione di bilanciamento del flusso (job flow balance) è operazionalmente testabile.

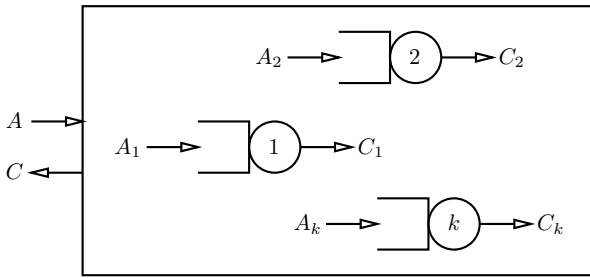
Esempio

È impossibile stabilire mediante misure se i tempi di servizio delle richieste formano una sequenza di variabili casuali indipendenti. Le assunzioni di indipendenza, sebbene comunemente usate nell'analisi statistica delle reti di code, non sono operazionalmente testabili.



Alcune Leggi Fondamentali

Osserviamo il server k -esimo per un certo tempo



T Durata intervallo di osservazione;

A, A_k Numero di arrivi

C, C_k Numero di completamenti

B_k Quantità di tempo in cui c'è almeno una richiesta ($B_k \leq T$).

Definizione

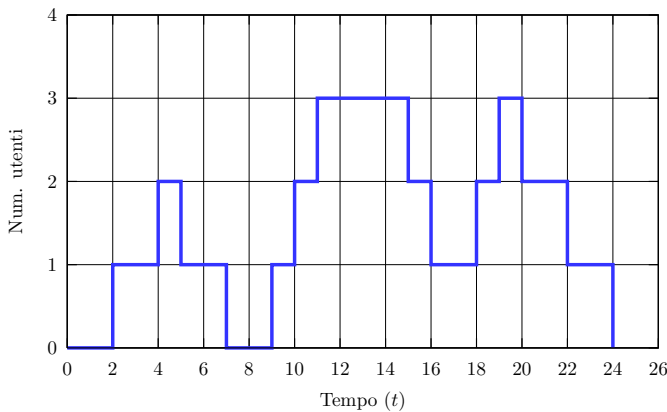
Tasso di Arrivo $\lambda_k \equiv A_k / T$ (1)

Throughput $X_k \equiv C_k / T$ (2)

Utilizzazione $U_k \equiv B_k / T$ (3)

Tempo medio di servizio $S_k \equiv B_k / C_k$ (4)

Esempio



Osserviamo il server k -esimo riportando sul grafico il numero di utenti *totali* (in coda o in servizio)

In questo caso abbiamo:

- ▶ Numero di arrivi $A_k = 7$
- ▶ Numero di completamenti $C_k = 7$
- ▶ $T = 26$

Da cui:

- ▶ Tasso di Arrivo: $\lambda_k = \frac{A_k}{T} = 7/26$
- ▶ Throughput: $X_k = \frac{C_k}{T} = 7/26$
- ▶ Utilizzazione: $U_k = \frac{B_k}{T} = 20/26$
- ▶ Tempo medio di servizio: $S_k = \frac{B_k}{C_k} = 20/7$

Alcune Leggi Fondamentali

Utilization Law

Da $X_k = C_k/T$ (2) e $S_k = B_k/C_k$ (4), considerando la definizione di utilizzazione, possiamo dedurre che:

$$S_k X_k = \frac{C_k}{T} \frac{B_k}{C_k} = \frac{B_k}{T} = U_k$$

da cui si ha:

Utilization Law

$$U_k = X_k S_k \quad (5)$$



Alcune Leggi Fondamentali

Legge di Little

La legge dell'utilizzazione è un caso particolare della legge di Little. Definiamo:

N Numero medio di richieste presenti nel sistema

R Tempo medio di risposta

Intuitivamente se il throughput del sistema è X richieste/secondo, e ciascuna richiesta rimane nel sistema mediamente R secondi, allora per ciascun secondo ci saranno esattamente XR richieste nel sistema.

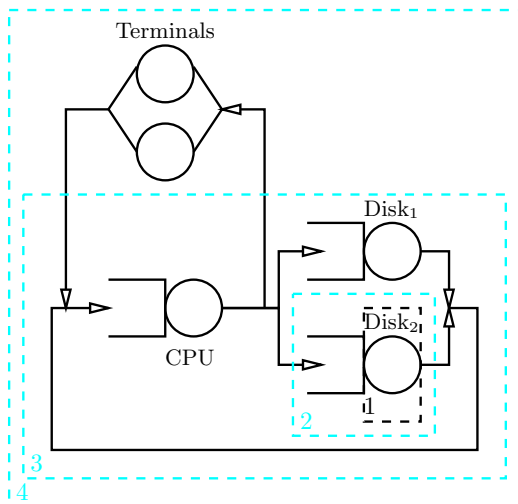
Legge di Little

$$N = XR \quad (6)$$

Le legge di Little si applica sia all'intero sistema, sia a parti di esso.



Esempio



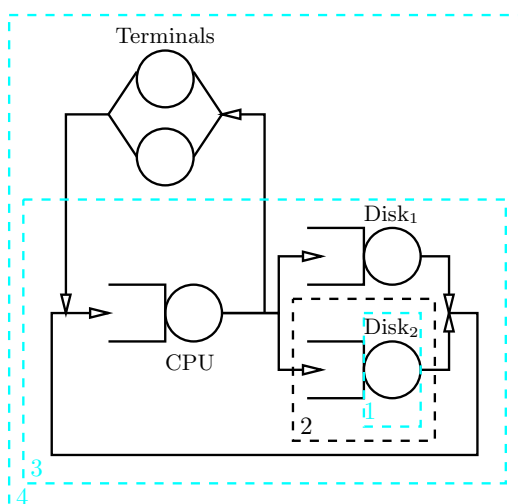
Si può applicare al singolo server $Disk_2$ (esclusa la coda) (Box 1).

- ▶ $N_{(1)}$ rappresenta l'utilizzazione di $Disk_2$
- ▶ $R_{(1)}$ rappresenta il tempo medio di servizio delle richieste;
- ▶ $X_{(1)}$ rappresenta il tasso a cui il server soddisfa le richieste.

Esempio

Supponiamo che il disco processi 40 richieste/secondo ($X_{(1)} = 40$) e mediamente ciascuna richiesta richieda 0.0225 secondi ($R_{(1)} = 0.0225$). Da (6) si ricava $N_{(1)} = 0.9$, ossia l'utilizzazione del disco è 90%

Esempio



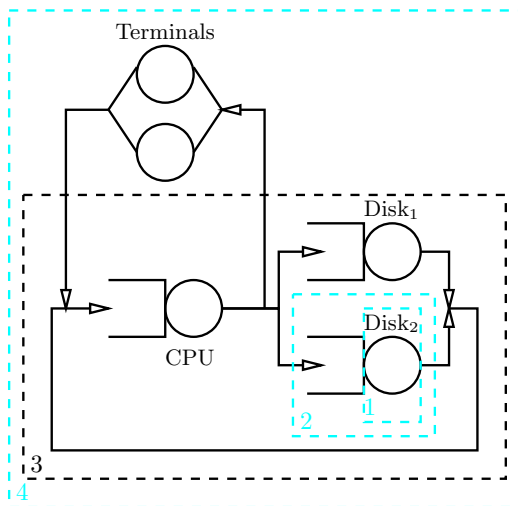
Includiamo la coda (Box 2).

- ▶ $N_{(2)}$ è il numero *totale* di utenti (includendo quello in servizio e quelli in coda);
- ▶ $R_{(2)}$ è il tempo speso da ciascuna richiesta nel sistema, includendo il tempo di attesa in coda e quello in servizio
- ▶ $X_{(2)}$ è il throughput

Esempio

Supponiamo che il disco stia servendo 40 richieste/secondo ($X_{(2)} = 40$) e il numero medio di richieste sia $N_{(2)} = 4$. Da (6) si ricava $R_{(2)} = 0.1$. Sapendo che $R_{(1)} = 0.0225$, si ricava che il tempo speso in coda è $R_{(2)} - R_{(1)} = 0.0775$.

Esempio



Consideriamo il sottosistema centrale (Box 3).

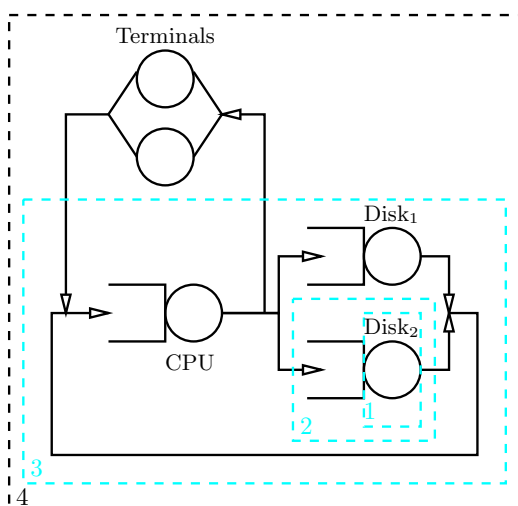
- ▶ $N_{(3)}$ è il numero totale di utenti nel sottosistema;
- ▶ $R_{(3)}$ è il tempo mediamente speso da ciascuna richiesta nel sottosistema;
- ▶ $X_{(3)}$ è il throughput del sottosistema.

Esempio

Supponiamo che il throughput sia $X_{(3)} = 0.5$ e il numero medio di richieste sia $N_{(3)} = 7.5$. Da (6) si ricava $R_{(3)} = 15$.



Esempio



Consideriamo ora il sistema completo (Box 4).

- ▶ $N_{(4)}$ è il numero *totale* di utenti nel sistema (costante, essendo un sistema chiuso);
- ▶ $R_{(4)}$ è il tempo totale di servizio + di attesa nel delay center (*think time*);
- ▶ $X_{(4)}$ è il tasso di passaggio dai terminali al sottosistema centrale.

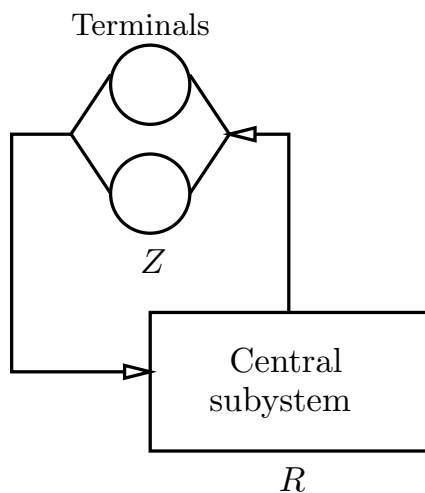
Esempio

Supponiamo che ci siano $N_{(4)} = 10$ utenti, e che il think time sia $Z = 15$. Il tempo speso nel sistema è $R_{(3)} = 15$. Poichè $R_{(4)} = R_{(3)} + Z$, possiamo scrivere $N_{(4)} = X_{(4)}(R_{(3)} + Z)$, da cui $X_{(4)} = 0.5$.



Alcune Leggi Fondamentali

Response Time Law



L'ultimo dei punti precedenti dà luogo ad una legge di tipo generale, applicabile a tutti i sistemi in cui siano presenti dei delay center. Sia:

- ▶ R il tempo di risposta del sottosistema privo dei delay center
- ▶ X il tasso di passaggio di richieste dai terminali al sottosistema centrale

Si ha allora:

Response Time Law

$$R = \frac{N}{X} - Z \quad (7)$$



Alcune Leggi Fondamentali

Forced Flow Law

Sia C_k il numero di richieste completate dal centro di servizio k -esimo

Definizione

$$\text{Visit Count} \quad V_k = C_k / C$$

Riscrivendolo come $C_k = V_k C$ e ricordando che $X_k = C_k / T$ (2), si ha:

Forced Flow Law

$$X_k = V_k X \quad (8)$$



Alcune Leggi Fondamentali

Service Demand

Abbiamo definito con S_k il tempo di servizio richiesto per ogni singola visita al centro di servizio k -esimo.

Ci tornerà comodo definire una ulteriore quantità D_k (domanda di servizio), che rappresenta il tempo di servizio totale richiesto al centro k -esimo.

Definizione

$$\text{Service Demand} \quad D_k = V_k S_k \quad (9)$$



Analisi dei Bound

Vedremo ora come calcolare dei limiti (bound) sui valori del throughput X e del tempo di risposta R di un sistema in funzione del tasso di arrivo delle richieste o del numero di utenti presenti.

L'analisi dei bound:

- ▶ Richiede pochissimo sforzo e può fornire informazioni utili in molte situazioni di modellazione dei sistemi.
- ▶ Aiuta a individuare l'effetto dei colli di bottiglia (*bottlenecks*) del sistema.
- ▶ Consente di confrontare rapidamente sistemi diversi.



Asymptotic Bounds vs Balanced System Bounds

- ▶ I Bound Asintotici consentono di ricavare dei limiti inferiori e superiori su X e R .
 - ▶ Vengono ricavati considerando i casi estremi di carico molto basso o molto alto
 - ▶ Si applicano solo se il tempo di servizio di una richiesta non dipende dal numero di richieste presenti nel sistema
- ▶ I Balanced System Bounds consentono di ricavare dei bound più stretti.
 - ▶ Vengono derivati considerando come casi limite dei sistemi in cui tutte le domande di servizio sono uguali
 - ▶ Ovviamente i bound si applicano a sistemi generali, in cui le domande di servizio sono arbitrarie.



Asymptotic Bound per Sistemi Aperti

Bound superiore su Throughput $X(\lambda)$

Consideriamo la legge dell'utilizzazione per il centro di servizio k -esimo (eq. 5):

$$U_k = X_k S_k$$

Denotando con X il throughput dell'intero sistema si ha $X_k = X V_k$ (eq. 8), che combinata con la precedente da:

$$U_k = X V_k S_k = X D_k$$

Dato che per definizione $U_k \leq 1$, si ha che deve valere $\forall k, X \leq 1/D_k$ ossia, posto $D_{max} = \max_k \{D_k\}$:

Bound su $X(\lambda)$

$$X(\lambda) \leq \frac{1}{D_{max}} \quad (10)$$



Asymptotic Bound per Sistemi Aperti

Bound sul Tempo di Risposta $R(\lambda)$

- ▶ Nel caso ottimo, in cui nel sistema ci sia una sola richiesta, il tempo di risposta R sarà uguale alla domanda totale di servizio $D = \sum_k D_k$:

Bound su $R(\lambda)$

$$R(\lambda) \geq D \quad (11)$$

- ▶ Nel caso pessimo non è possibile fornire alcun bound
 - ▶ Assumiamo che n utenti arrivino ogni n/λ unità di tempo (il tasso d'arrivo è $n\lambda/n = \lambda$);
 - ▶ Nel caso peggiore, l'ultimo utente può venire accodato dietro a tutti i precedenti, e quindi sperimentare un tempo di risposta arbitrariamente alto al crescere di n .



Asymptotic Bound per Sistemi Chiusi

Bound su Throughput $X(N)$

Sia N il numero di utenti nel sistema. Se N cresce, chiaramente aumenta l'utilizzazione dei centri di servizio, la quale comunque deve restare minore o uguale a uno

$$U_k = XD_k \leq 1 \quad \text{per ogni } k$$

Da questo si deriva:

$$X(N) \leq 1/D_{max} \quad (12)$$



Asymptotic Bound per Sistemi Chiusi

Bound su Throughput $X(N)$

Nel caso limite di $N = 1$ (una singola richiesta), il throughput del sistema sarebbe $X = 1/(D + Z)$ perchè ad ogni interazione la richiesta spende tempo $D = \sum D_k$ in servizio, e Z in attesa.

- Il throughput massimo si ha quando gli utenti non interferiscono l'un l'altro (cioè nessun utente trova altri utenti davanti a sé in coda ai centri di servizio)

$$X(N) \leq N/(D + Z) \quad (13)$$

- Il throughput minimo si ha quando invece ciascun utente trova nelle code davanti a sé gli altri $N - 1$ utenti. In questo caso $(N - 1)D$ tempo è speso in coda dietro agli altri $N - 1$ utenti, D tempo è speso in servizio e Z in attesa fuori dal sistema

$$X(N) \geq N/(ND + Z) \quad (14)$$



Asymptotic Bound per Sistemi Chiusi

Bound su Throughput $X(N)$

Combinando le Eq. 12, 13 e 14 si ottiene

Bound per $X(N)$

$$\frac{N}{ND + Z} \leq X(N) \leq \min \left(\frac{N}{D + Z}, \frac{1}{D_{max}} \right) \quad (15)$$



Asymptotic Bound su Sistemi Chiusi

Bound su Tempo di Risposta $R(N)$

Riscriviamo (15), ricordando che da (7) si ha $X(N) = N/(R(N) + Z)$

$$\frac{N}{ND + Z} \leq \frac{N}{R(N) + Z} \leq \min \left(\frac{1}{D_{max}}, \frac{N}{D + Z} \right)$$

Invertiamo tutti i membri

$$\max \left(D_{max}, \frac{D + Z}{N} \right) \leq \frac{R(N) + Z}{N} \leq \frac{ND + Z}{N}$$

Da cui si ottiene:

Bound per $R(N)$

$$\max(D, ND_{max} - Z) \leq R(N) \leq ND \quad (16)$$



Riepilogo Bound Asintotici

Sistemi Aperti

$$X(\lambda) \leq 1/D_{max}$$

$$D \leq R(\lambda)$$

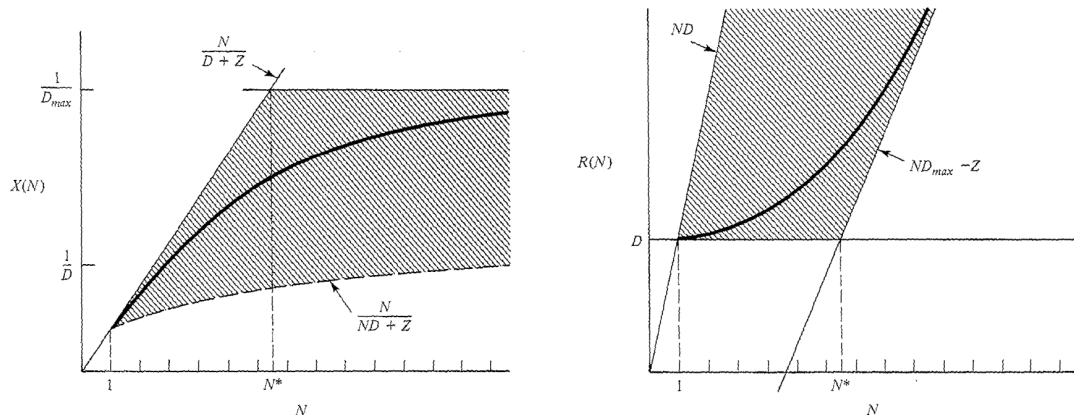
Sistemi Chiusi

$$\frac{N}{ND + Z} \leq X(N) \leq \min \left(\frac{N}{D + Z}, \frac{1}{D_{max}} \right)$$

$$\max(D, ND_{max} - Z) \leq R(N) \leq ND$$



Riepilogo Bound Asintotici



Fonte: Edward D. Lazowska, John Zahorjan, G. Scott Graham, Kenneth C. Sevcik, *Quantitative System Performance: Computer Systems Analysis using Queueing Network Models*, Prentice-Hall, p. 75



Calcolo dei bound asintotici

Implementazione in GNU Octave

ab_closed.m

```

## Throughput Asymptotic Bound
function [lower, upper] = ab_X_closed( N, D, Z )
    D_tot = sum(D);
    D_max = max(D);
    lower = N/(N*D_tot+Z);
    upper = min( N/(D_tot+Z), 1/D_max );
endfunction

## Response Time Asymptotic Bound
function [lower, upper] = ab_R_closed( N, D, Z )
    D_max = max(D);
    D_tot = sum(D);
    lower = max( D_tot, N*D_max-Z );
    upper = N*D_tot;
endfunction

```



Calcolo dei bound asintotici

Implementazione in GNU Octave

Esempio

```
source("ab_closed.m");

Z = 15;           # Think Time
D = [1.0, 2.0, 0.5]; # Domande di servizio
for N=1:5
    [l,u] = ab_R_closed(N, D, Z);
    printf("%02d_%.6f_%.6f\n", N, l, u);
endfor
```

Output:

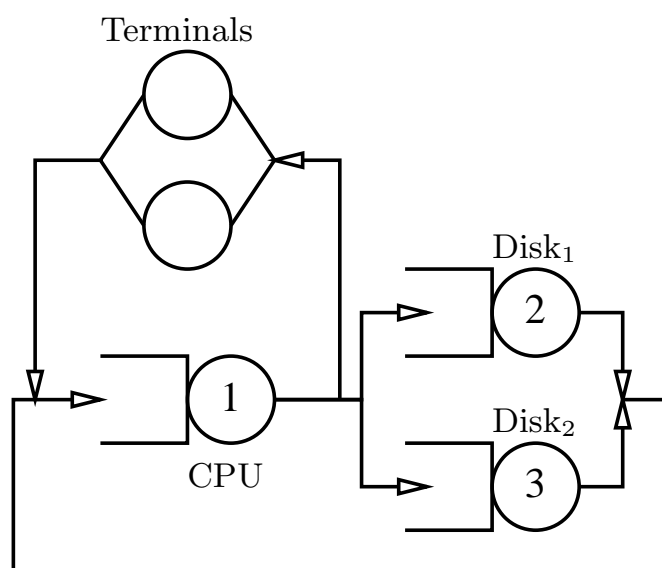
```
01  3.50  3.50
02  3.50  7.00
03  3.50 10.50
04  3.50 14.00
05  3.50 17.50
```



Esempio

Bound Asintotici

Riconsideriamo il nostro sistema a server centrale



Parametri Dati

- ▶ $D_1 = 2.0, D_2 = 0.5, D_3 = 3.0$
- ▶ $V_2 = 10, V_3 = 100$
- ▶ $S_2 = 0.05, S_3 = 0.03$
- ▶ $Z = 15s$



Esempio

Consideriamo quattro possibili scenari:

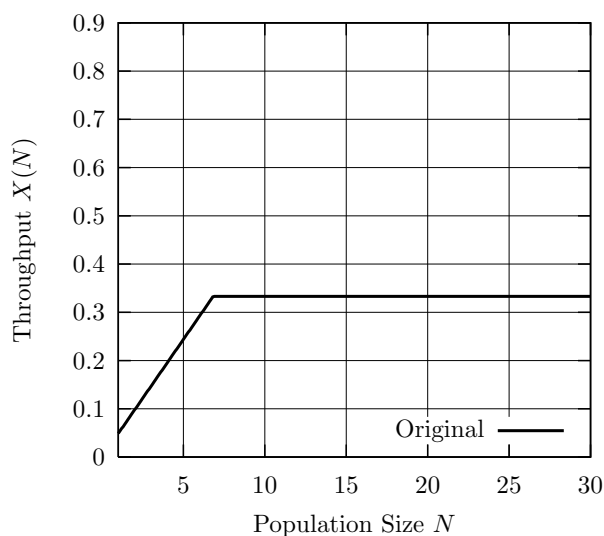
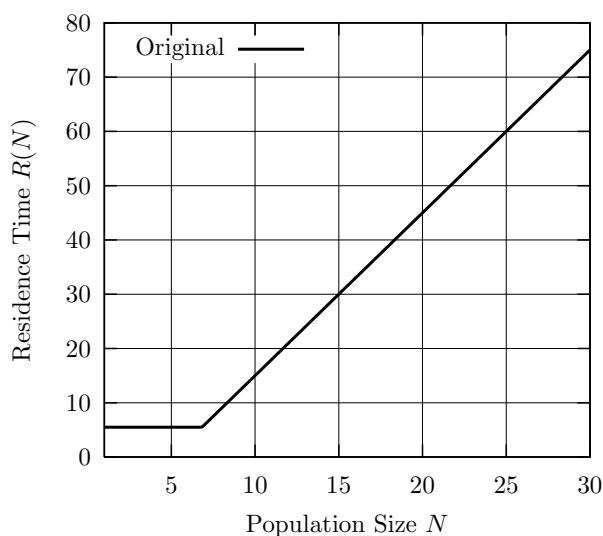
1. Rimpiazzare la CPU con una il doppio più veloce ($D_1 \leftarrow 1$);
2. Spostare alcuni file dal disco lento (centro serv. 2) a quello veloce (centro serv. 3) in modo da rendere uguali le domande di servizio tra i due;
3. Aggiungere un ulteriore disco (centro serv. 4, con $S_4 = 0.03$) che gestisca metà del carico del centro serv. 3;
4. Tutte e tre le alternative precedenti: CPU più veloce, nuovo disco con $S_4 = 0.03$, bilanciamento della domanda di servizio tra i tre dischi.



Esempio

Sistema Originale

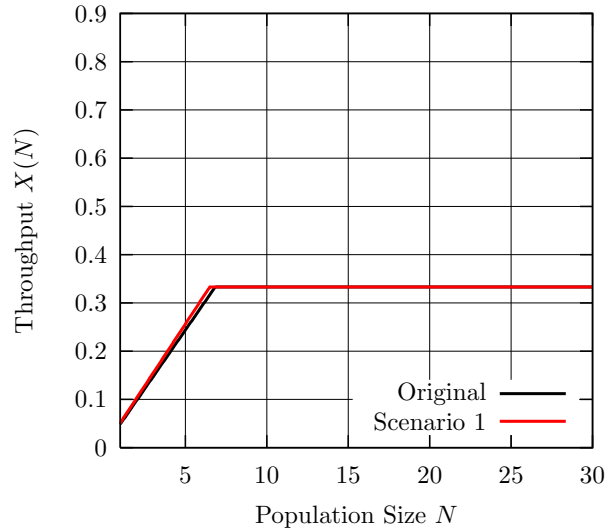
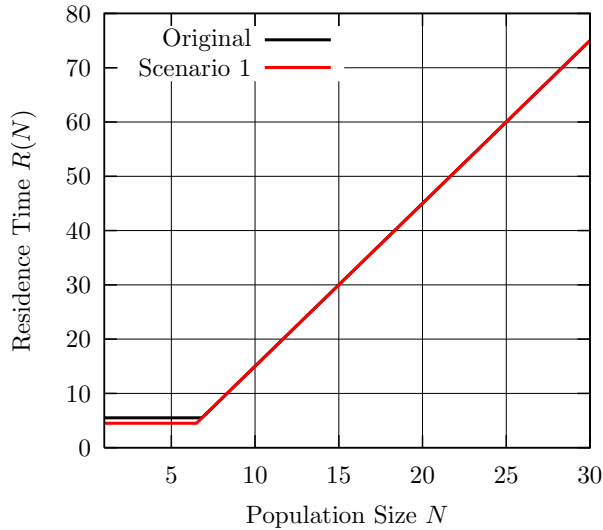
$$D_1 = 2.0, D_2 = 0.5, D_3 = 3.0$$



Esempio

Scenario 1: CPU più veloce

Rimpiazziamo la CPU (centro di servizio 1) con una veloce il doppio. Quindi D_1 si dimezza, da cui: $D_1 = 1.0, D_2 = 0.5, D_3 = 3.0$



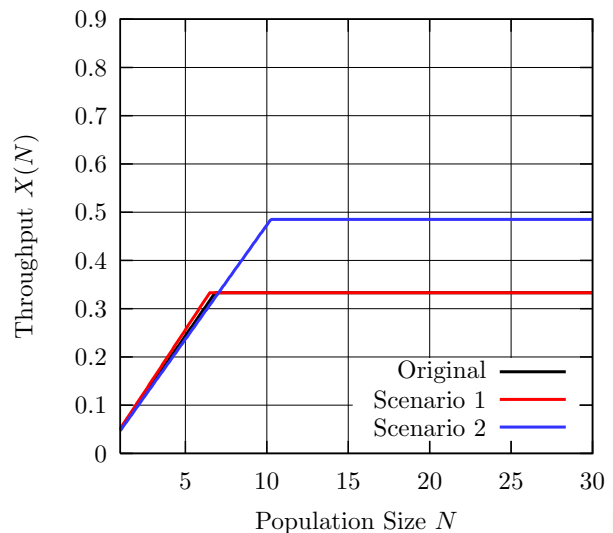
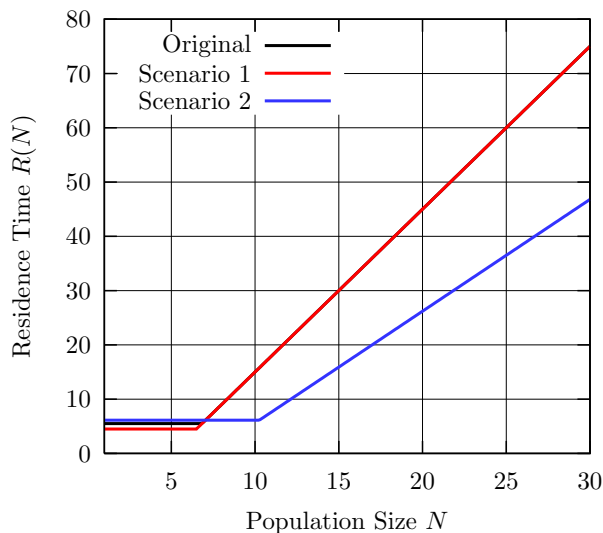
Esempio

Scenario 2: Bilanciamento carico tra i dischi

Spostiamo i dati dal disco lento a quello veloce, in modo da avere $D_2 = D_3$ (ricordiamo che $D_k = V_k S_k$). Si deve risolvere il sistema lineare:

$$\begin{cases} V_2 + V_3 = 110 & \text{Il numero di visite deve restare inalterato} \\ V_2 S_2 = V_3 S_3 & \text{Bilanciare le domande di servizio} \end{cases}$$

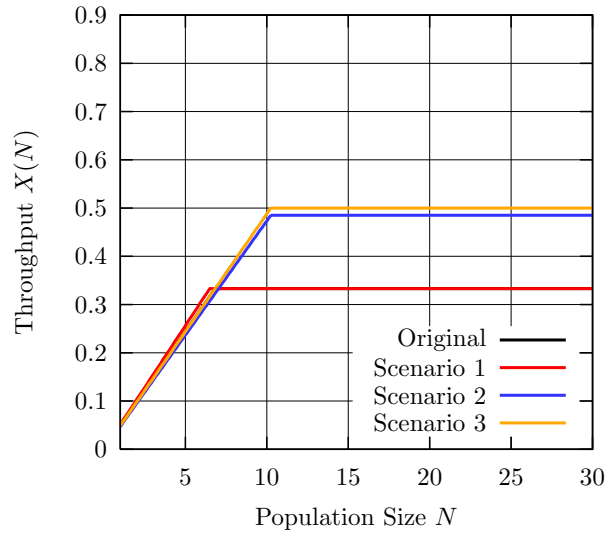
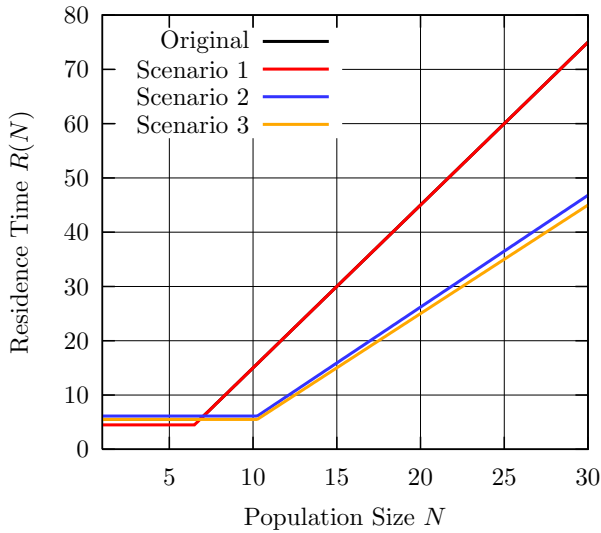
Da cui risulta $V_2 = 41, V_3 = 69$, ossia $D_2 = D_3 = 2.06$



Esempio

Scenario 3: Ulteriore disco veloce

Aggiungiamo un ulteriore disco “veloce” che gestisca metà del carico del disco esistente più carico. Avremo quindi $K = 4$ centri di servizio, con $D_1 = 2.0, D_2 = 0.5, D_3 = 1.5, D_4 = 1.5$



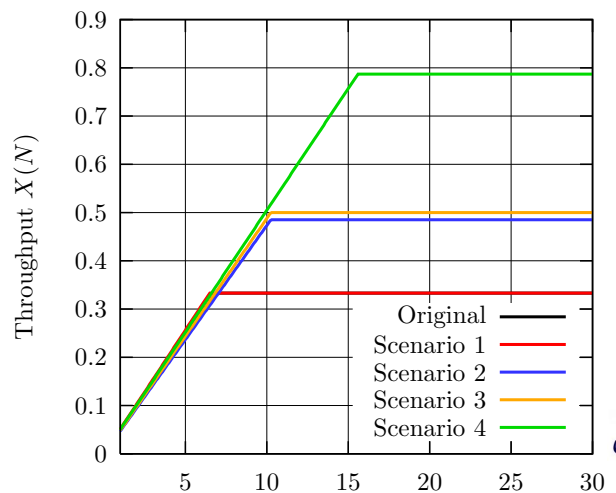
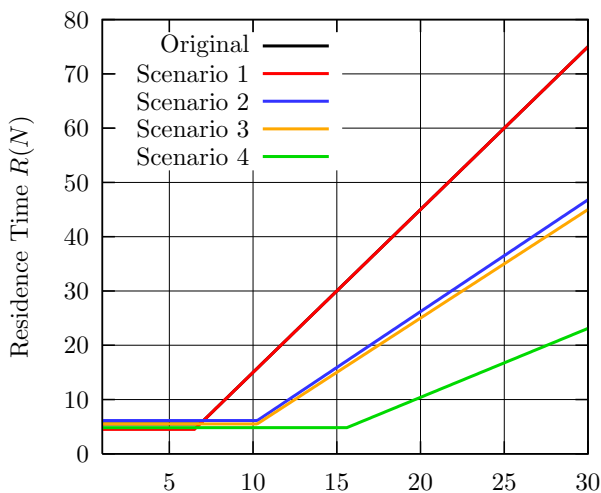
Esempio

Scenario 4: Tutte i miglioramenti contemporaneamente

In questo scenario usiamo una CPU più veloce ($D_1 \leftarrow 1.0$), aggiungiamo un disco veloce ($S_4 = 0.03$) e facciamo in modo da bilanciare il carico tra tutti i dischi. Similmente al caso 2 avremo:

$$\begin{cases} V_2 + V_3 + V_4 = 110 \\ V_2 S_2 = V_3 S_3 \\ V_3 S_3 = V_4 S_4 \end{cases}$$

Da cui si ottiene $D_2 = D_3 = D_4 = 1.27$.



Balanced System Bound

Bound per Sistemi Aperti

$$X(\lambda) \leq 1/D_{max}$$

$$\frac{D}{1 - \lambda D_{ave}} \leq R(\lambda) \leq \frac{D}{1 - \lambda D_{max}}$$

Bound per Sistemi Chiusi

$$\frac{N}{D + Z + \frac{(N-1)D_{max}}{1+Z/(ND)}} \leq X(N) \leq \min \left(\frac{1}{D_{max}}, \frac{N}{D + Z + \frac{(N-1)D_{ave}}{1+Z/D}} \right)$$

$$\max \left(ND_{max} - Z, D + \frac{(N-1)D_{ave}}{1 + Z/D} \right) \leq R(N) \leq D + \frac{(N-1)D_{max}}{1 + Z/(ND)}$$

Calcolo dei BSB

Implementazione in GNU Octave

bsb_closed.m

```

## Throughput Balanced System Bound
function [lower, upper] = bsb_X_closed( N, D, Z )
    D_max = max(D);
    D_tot = sum(D);
    D_ave = sum(D) / size(D,2);
    lower = N/(D_tot+Z+(N-1)*D_max)/(1+Z/(N*D_tot));
    upper = min(1/D_max, N/(D_tot+Z+(N-1)*D_ave)/(1+Z/D_tot));
endfunction

## Response Time Balanced System Bound
function [lower, upper] = bsb_R_closed( N, D, Z )
    D_max = max(D);
    D_tot = sum(D);
    D_ave = sum(D) / size(D,2);
    lower = max(N*D_max-Z, D_tot+(N-1)*D_ave)/(1+Z/D_tot);
    upper = D_tot + (N-1)*D_max/(1+Z/(N*D_tot));
endfunction

```

Confronto tra AB e BSB

Esempio

```

source("ab_closed.m");
source("bsb_closed.m");

D = [2, 0.5, 3];
Z = 15;

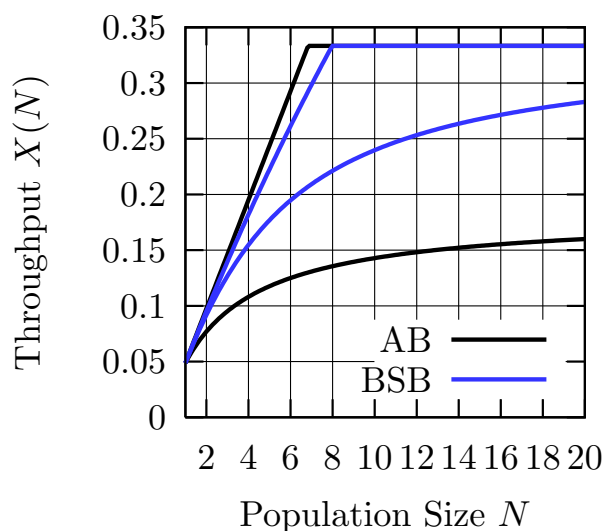
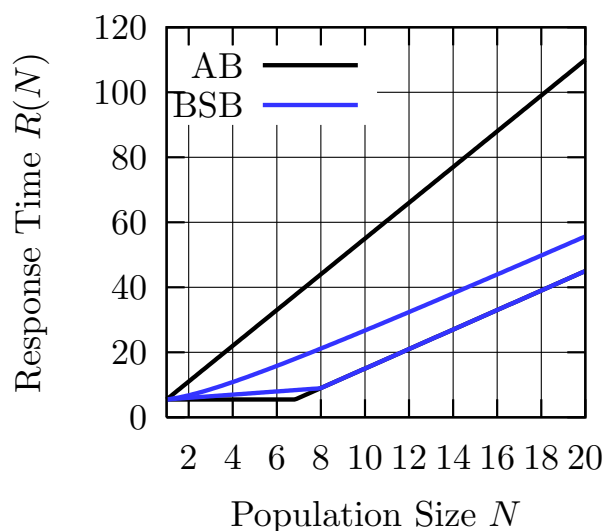
for N=1:20
    [ab_R_low , ab_R_high ] = ab_R_closed( N, D, Z );
    [ab_X_low , ab_X_high ] = ab_X_closed( N, D, Z );
    [bsb_R_low , bsb_R_high] = bsb_R_closed( N, D, Z );
    [bsb_X_low , bsb_X_high] = bsb_X_closed( N, D, Z );
    printf("%f_%f_%f_%f_%f_%f_%f_%f_%f\n", N,
        ab_R_low, ab_R_high, bsb_R_low, bsb_R_high,
        ab_X_low, ab_X_high, bsb_X_low, bsb_X_high
    );
endfor

```



Confronto tra AB e BSB

$$D_1 = 2, D_2 = 0.5, D_3 = 3, Z = 15$$



Mean Value Analysis (MVA)

Mean Value Analysis (MVA) è un algoritmo molto semplice per la risoluzione di reti aperte o chiuse.

Mediante MVA è possibile calcolare il valore esatto di:

- ▶ Utilizzazione U_i
- ▶ Tempo medio di risposta R_i
- ▶ Numero medio di utenti nel centro di servizio Q_i

da cui è poi facile ricavare le quantità di interesse rimanenti



Fondamenti Teorici

L'algoritmo MVA descritto in seguito funziona solo se applicato a reti che godono delle seguenti proprietà:

1. In ciascun centro di servizio il numero di arrivi deve uguagliare il numero di completamenti (Service Center Flow Balance)
2. Non devono avvenire due cambiamenti di stato nel sistema nello stesso istante (One-Step Behaviour)
3. La probabilità che una richiesta che completa servizio al centro i venga messa in coda al centro j è indipendente dalla lunghezza di qualsiasi coda (Routing Homogeneity)
4. Il tasso di completamento in un centro di servizio deve essere indipendente dal numero di job in coda, e dal numero di utenti nel sistema (Service Time Homogeneity)
5. Il numero di arrivi dall'esterno non deve dipendere dal numero o dal posizionamento delle eventuali richieste nel sistema (Homogeneous External Arrivals)



MVA per Reti Aperte

Parametri di Input

Consideriamo una generica rete aperta di cui siano noti i parametri:

- ▶ V_k numero di visite al centro k -esimo
- ▶ S_k tempo medio di servizio
- ▶ λ tasso di arrivo
 - ▶ Il tasso di arrivo massimo che il sistema può supportare senza diventare saturo è

$$\lambda_{sat} = 1/D_{max}$$

- ▶ Consideriamo pertanto solo valori $\lambda < \lambda_{sat}$



MVA per Reti Aperte

Descrizione dell'Algoritmo

- ▶ Throughput: In un sistema stabile, il tasso di arrivo λ deve essere uguale al throughput complessivo X . Per la legge del flusso forzato (8), avremo:

$$X_k(\lambda) = \lambda V_k$$

- ▶ Utilizzazione: Applichiamo la legge dell'utilizzazione (5) combinata con (9):

$$U_k(\lambda) = X_k(\lambda)S_k = \lambda V_k S_k = \lambda D_k$$



MVA per Reti Aperte

Descrizione dell'Algoritmo

► Tempo di Risposta:

- Nel caso di *Delay Center*, non c'è mai coda e il tempo di risposta è il tempo medio di servizio

$$R_k(\lambda) = S_k \quad (\text{delay centers})$$

- Nel caso di centro di servizio con coda, sia $A_k(\lambda)$ il numero medio di utenti in coda visti da un nuovo utente che arriva. Allora si ha:

$$\begin{aligned} R_k(\lambda) &= S_k + S_k A_k(\lambda) \\ &= S_k (1 + A_k(\lambda)) \end{aligned}$$

- Se valgono le ipotesi descritte in precedenza, allora si ha $A_k(\lambda) = Q_k(\lambda)$, da cui:

$$\begin{aligned} R_k(\lambda) &= S_k (1 + Q_k(\lambda)) \\ &= S_k (1 + X_k R_k(\lambda)) \quad \text{applicando (6)} \\ &= S_k + U_k(\lambda) R_k(\lambda) \quad \text{applicando (5)} \\ &= \frac{S_k}{1 - U_k(\lambda)} \quad (\text{queueing centers}) \end{aligned}$$



MVA per Reti Aperte

Descrizione dell'Algoritmo

- Numero utenti nel centro di servizio: da (6) si ha:

$$\begin{aligned} Q_k(\lambda) &= X_k(\lambda) R_k(\lambda) \\ &= \begin{cases} U_k(\lambda) & (\text{delay centers}) \\ \frac{U_k(\lambda)}{1 - U_k(\lambda)} & (\text{queueing centers}) \end{cases} \end{aligned}$$

- Tempo di risposta del sistema:

$$R(\lambda) = \sum_k V_k R_k(\lambda)$$

- Numero medio di utenti totali nel sistema:

$$Q(\lambda) = \sum_k Q_k(\lambda)$$



MVA per Reti Aperte

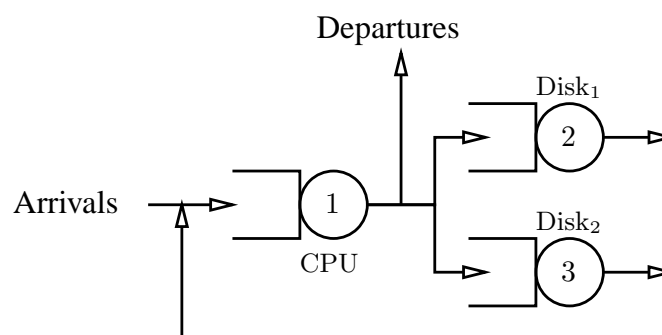
Implementazione in GNU Octave

mva.m

```
function [U,R,Q] = mva_open( lambda , V, S, delay )
K = size(S,2);
R = zeros(1,K);
Q = zeros(1,K);
D = V .* S;
U = lambda * D;
for k=1:K
    if ( delay(k) )
        R(k) = S(k);
        Q(k) = U(k);
    else
        R(k) = S(k)/(1-U(k));
        Q(k) = U(k)/(1-U(k));
    endif
endfor
endfunction
```



Esempio



Sono dati i seguenti parametri:

- ▶ $V_1 = 121, V_2 = 70, V_3 = 50$
- ▶ $S_1 = 0.005, S_2 = 0.030, S_3 = 0.027$



Esempio

```

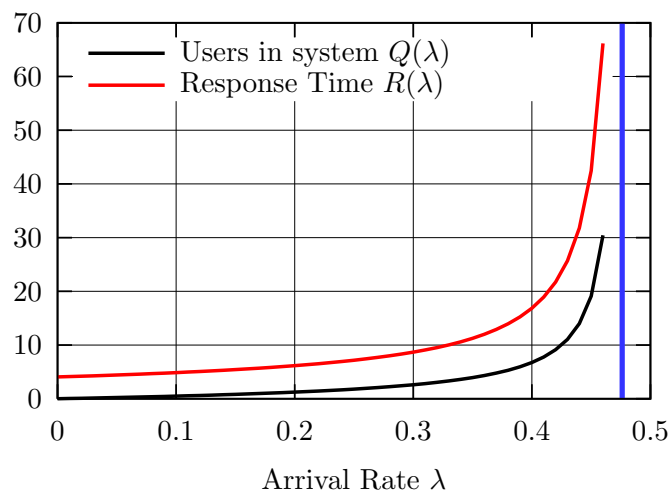
> source("mva_open.m");
# Definizione dei parametri
> V = [121, 70, 50];
> S = [0.005, 0.030, 0.027];
> lambda = 0.3;
# Calcoliamo le domande di servizio
> D = V .* S
D = 0.60500    2.10000    1.35000
# Calcoliamo il tasso di arrivo max
> lambda_sat = 1 / max(D)
lambda_sat = 0.47619
# Risolviamo il modello
> [U,R,Q] = mva_open( lambda, V, S, zeros(1,3) )
#   CPU      Disk1      Disk2
U = 0.18150    0.63000    0.40500
R = 0.0061087  0.0810811  0.0453782
Q = 0.22175    1.70270    0.68067
# Tempo di risposta complessivo
> R_tot = sum(R .* V)
R_tot = 8.6837
# Numero totale di utenti nel sistema
> Q_tot = lambda * R_tot
Q_tot = 2.6051
# ...si poteva calcolare anche come
> Q_tot =sum(Q)
Q_tot = 2.6051

```



Esempio

Consideriamo il sistema precedente, con $V_1 = 121$, $V_2 = 70$, $V_3 = 50$,
 $S_1 = 0.005$, $S_2 = 0.030$, $S_3 = 0.027$



$Q(\lambda)$ e $R(\lambda)$ tendono ad infinito se $\lambda \rightarrow \lambda_{sat}$



MVA per Reti Chiuse

Descrizione dell'Algoritmo

- ▶ Tempo di Risposta del sistema: Il tempo di risposta dell'intero sistema $R(N)$ è dato da

$$R(N) = \sum_k V_k R_k(N)$$

- ▶ Throughput: La legge di Little applicata all'intero sistema da

$$X(N) = \frac{N}{Z + R(N)}$$

- ▶ Numero utenti in coda: La legge di Little applicata al singolo centro di servizio k da

$$\begin{aligned} Q_k(N) &= X_k(N) R_k(N) \\ &= X(N) V_k R_k(N) \end{aligned}$$



MVA per Reti Chiuse

Descrizione dell'Algoritmo

Per il calcolo del tempo di risposta $R_k(N)$ vale ancora la relazione precedentemente individuata:

$$R_k(N) = \begin{cases} S_k & \text{(delay centers)} \\ S_k(1 + A_k(N)) & \text{(queueing centers)} \end{cases}$$

Nel caso di reti chiuse in cui valgono le proprietà precedentemente descritte, si ha $A_k(N) = Q_k(N - 1)$.

Quindi nel caso di reti chiuse con N utenti l'algoritmo MVA richiede di calcolare iterativamente $X(n)$, $R_k(n)$, $Q_k(n)$ per $n = 1 \dots N$.



MVA per Reti Chiuse

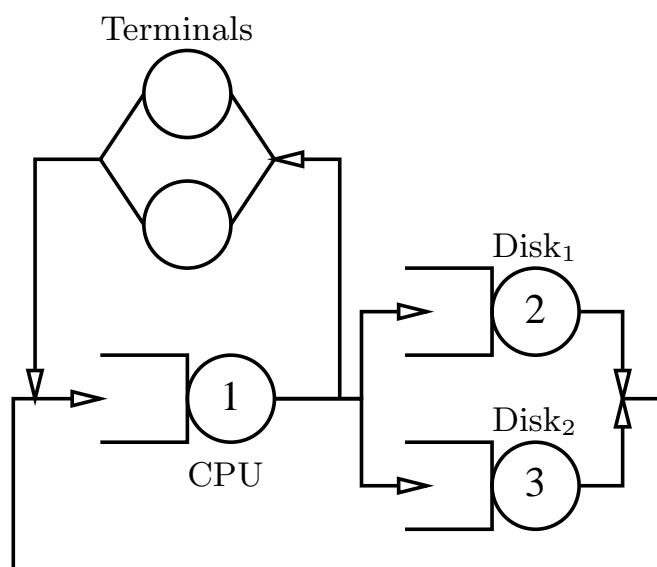
Implementazione in GNU Octave

mva.m

```
function [U,R,Q] = mva_closed( N, V, S, Z, delay )
    K = size(S,2);
    Q = zeros(1,K);
    R = zeros(1,K);
    D = V .* S;
    X = 0;
    for n=1:N
        for k=1:K
            if ( delay(k) )
                R(k) = S(k);
            else
                R(k) = S(k)*(1+Q(k));
            endif
        endfor
        R_tot = sum( R .* V ); # system response time
        X = n / (Z+R_tot);
        Q = X*(V .* R);
    endfor
    U = X * D;
endfunction
```



Esempio



Sono dati i seguenti parametri:

- ▶ $V_1 = 121, V_2 = 70, V_3 = 50$
- ▶ $S_1 = 0.005, S_2 = 0.030, S_3 = 0.027$
- ▶ $N = 3, Z = 15$



Esempio

```

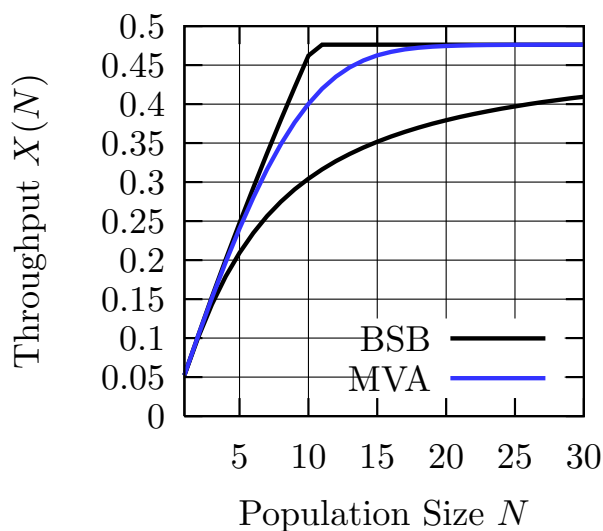
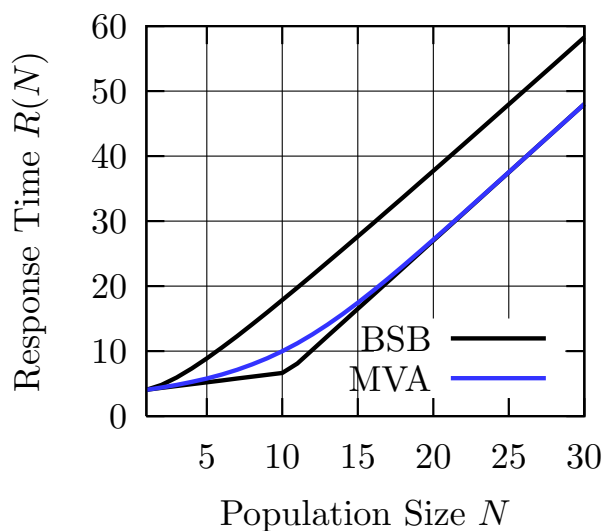
> source("mva_closed.m");
# Definizione dei parametri
> V = [121, 70, 50];
> S = [0.005, 0.030, 0.027];
> D = V .* S;
> N = 3;
> Z = 15;
# Risolviamo il modello
> [U,R,Q] = mva_closed( N, V, S, Z, zeros(1,3) );
#   CPU      Disk1      Disk2
U = 0.091667  0.318185  0.204547
R = 0.0053217 0.0372101 0.0310237
Q = 0.097566  0.394657  0.235030
# Tempo di risposta del sistema
> R_tot = sum( V .* R )
R_tot = 4.7998
# Throughput del sistema
> X = N/(Z+R_tot)
X = 0.15152
# Numero totale di utenti _in coda_ nel sistema
> Q_tot = N-X*Z
Q_tot = 0.72725

```



MVA vs Balanced System Bounds

Consideriamo il sistema chiuso visto in precedenza, facendo variare N . I grafici mostrano il confronto tra il risultato ottenuto dall'algoritmo MVA con i Balanced System Bound per sistemi chiusi (vedi p. 41)



BPEL e i Web Services

Cosa sono i Web Services?

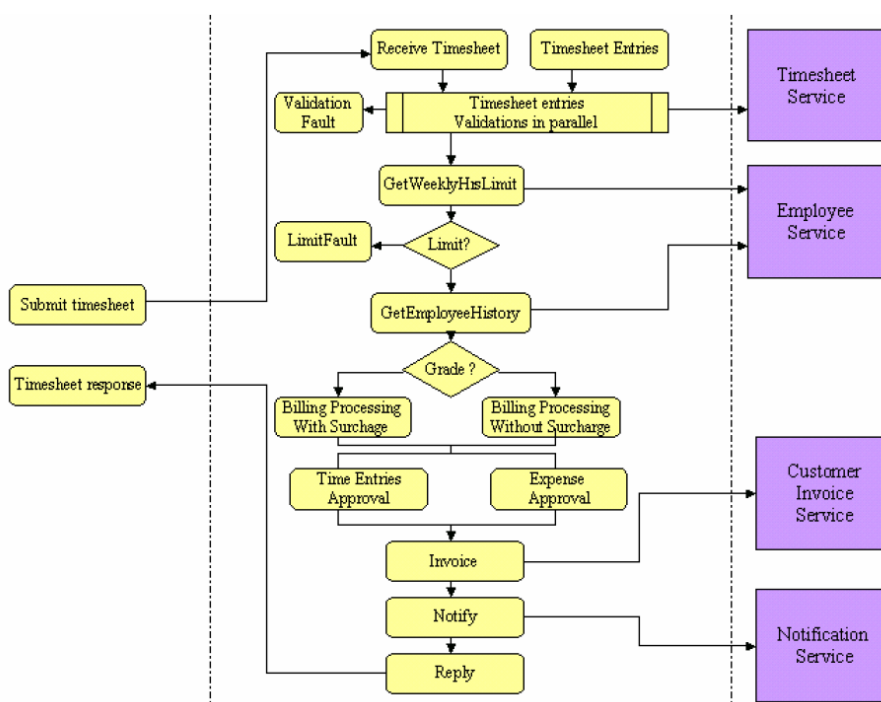
- ▶ I Web Services (WS) consentono la comunicazione tra programmi usando XML come “lingua franca”
- ▶ Lo standard dei Web Services definisce il formato del messaggio, l'interfaccia verso il quale il messaggio viene spedito, definisce meccanismi per pubblicare e rintracciare le interfacce Web Service
- ▶ Facilitano l'accesso ad applicazioni Internet

Cos'è BPEL?

- ▶ BPEL (Business Process Execution Language) è una specifica per descrivere dei *workflow* in notazione XML
- ▶ BPEL consente di rappresentare sequenze di interazioni complesse con WS diversi



Esempio



Fonts: <http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf>



Formulazione del problema

Problema

È possibile stimare in modo semplice le prestazioni di un workflow rappresentato tramite un BPEL?

Ci sono almeno due approcci possibili:

1. Approccio basato su misure: si misura direttamente il tempo di esecuzione del workflow
2. Approccio basato su modelli: si stima il tempo di esecuzione a partire da un opportuno *modello* del sistema che si vuole misurare



Scelta dell'approccio

L'alternativa basata su misure è quella in grado di fornire i risultati più realistici. Tuttavia presenta alcuni svantaggi:

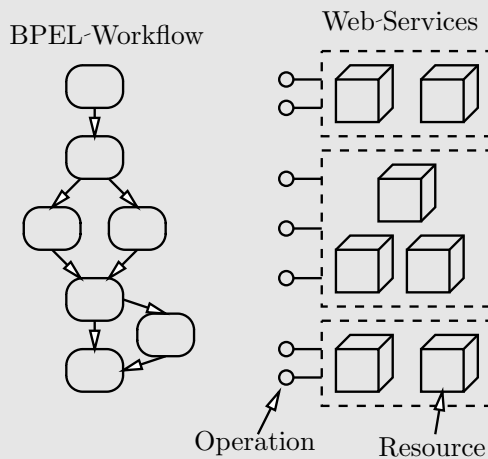
- ▶ Può essere applicata solo con il sistema già implementato (tutti i WS attivi e correttamente installati, il BPEL già definito ecc.)
- ▶ Risulta quindi impossibile (o troppo costoso) analizzare alternative diverse, o testare il comportamento del sistema in condizioni different (es., cosa succederebbe sostituendo i server su cui sono installati i WS con macchine più veloci?)

Vedremo come un semplice modello a reti di code possa dare risposta a molte domande che ci si pone in fase di pianificazione e messa in opera dei servizi.

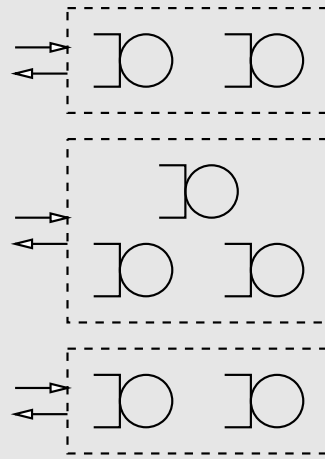


Modellare il sistema

Sistema da modellare



Modello a reti di code



Un semplice caso di studio

Analisi dati in ambiente Grid

Supponiamo di disporre dei seguenti tre Web Services:

1. Uno Storage Element, che fornisce funzionalità di memorizzazione dati
2. Un Computing Element che fornisce funzionalità di esecuzione di programmi
3. Un Analysis Element che fornisce funzionalità di elaborazione specifica dei dati (ad esempio, per il calcolo di fit, statistiche e altro).

Un semplice caso di studio

Analisi dati in ambiente Grid

Un utente esegue il seguente workflow sui servizi di cui sopra:

1. Autenticazione presso il Computing Element
2. Trasferimento di dati dallo Storage Element al Computing Element
3. Avvio del processamento dei dati presso il Computing Element
4. Trasferimento dei dati di output verso l'Analysis Element
5. Analisi dei dati di output di cui al punto precedente



Caso di studio

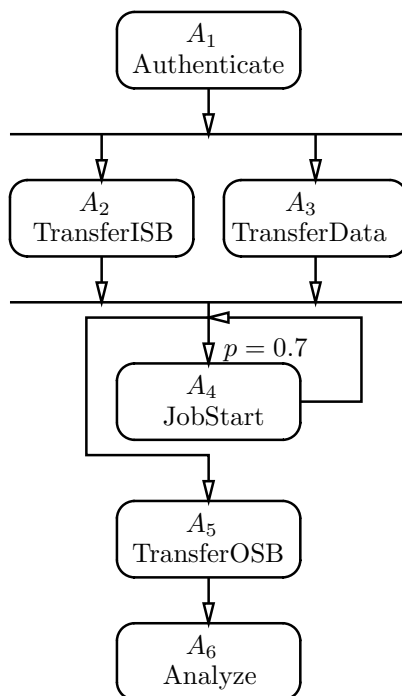
Descrizione del workflow

BPEL del workflow

```
<bpws:process>
  <perf:workload type="closed" thinktime="120"/>
  <bpws:import
    importType="http://schemas.xmlsoap.org/wsdl/"
    location="CaseStudy.wsdl">
  <bpws:sequence>
    <bpws:invoke operation="Authenticate"/>
    <bpws:flow>
      <bpws:invoke operation="TransferISB"/>
      <bpws:invoke operation="TransferData"/>
    </bpws:flow>
    <bpws:while>
      <bpws:condition prob="0.7"/>
      <bpws:invoke operation="JobStart"/>
    </bpws:while>
    <bpws:invoke operation="TransferOSB"/>
    <bpws:invoke operation="Analyze"/>
  </bpws:sequence>
</bpws:process>
```



Rappresentazione Grafica del BPEL



- ▶ Ciascuna azione è etichettata con il nome dell'operazione invocata
- ▶ Per ogni operazione, ci sarà scritto da qualche parte la domanda di servizio sulle risorse su cui viene compiuta
- ▶ L'operazione JobStart viene reiterata con probabilità $p = 0.7$



Caso di studio

WSDL del servizio Storage Element

```

<!-- Interface for Storage Element -->
<definitions>
  <portType name="DataFactory">
    <operation name="TransferData">
      <perf:PAdemand resource="DF:CPU" value="1"/>
      <perf:PAdemand resource="DF:Disk" value="120"/>
      <perf:PAdemand resource="Network" value="80"/>
    </operation>
  </portType>
</definitions>
  
```



Caso di studio

WSDL del servizio Computing Element

```

<!-- Interface for Computing Element -->
<definitions>
  <portType name="ComputingElement">
    <operation name="Authenticate">
      <perf:PAdemand resource="CE:CPU" value="10"/>
    </operation>
    <operation name="TransferISB">
      <perf:PAdemand resource="CE:CPU" value="2"/>
      <perf:PAdemand resource="CE:Disk" value="120"/>
      <perf:PAdemand resource="Network" value="10"/>
    </operation>
    <operation name="JobStart">
      <perf:PAdemand resource="CE:CPU" value="4"/>
    </operation>
    <operation name="TransferOSB">
      <perf:PAdemand resource="CE:CPU" value="1"/>
      <perf:PAdemand resource="CE:Disk" value="30"/>
      <perf:PAdemand resource="Network" value="80"/>
    </operation>
  </portType>
</definitions>

```

Caso di studio

WSDL del servizio Analysis Element

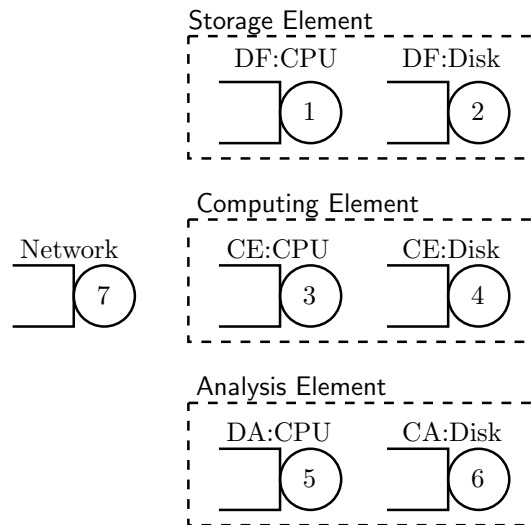
```

<!-- Interface for Analysis Element -->
<definitions>
  <portType name="DataAnalysis">
    <operation name="Analyze">
      <PAdemand resource="DA:CPU" value="100"/>
      <PAdemand resource="DA:Disk" value="30"/>
    </operation>
  </portType>
</definitions>

```

Caso di studio

Rappresentazione delle risorse

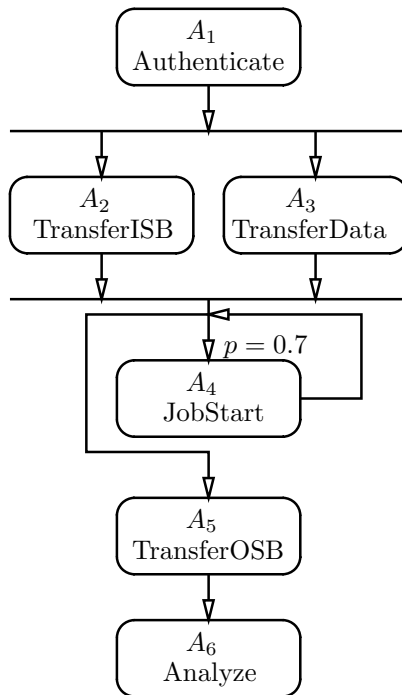


Analisi

- ▶ Per analizzare la rete di code, è come prima cosa necessario calcolare la domanda di servizio D_k ai centri di servizio, per $K = 1 \dots 7$.
- ▶ Per calcolare D_k dobbiamo come prima cosa sapere quante volte viene eseguita ciascuna azione del BPEL
- ▶ Da questo, e dalle annotazioni dei WSDL, possiamo risalire a quali risorse vengono visitate e calcolare la domanda di servizio totale

Analisi

Primo passo: Calcolo delle visite alle azioni



Risolviamo:

$$\left\{ \begin{array}{l} V_{A_2} = V_{A_1} \\ V_{A_3} = V_{A_1} \\ V_{A_4} = \frac{p}{1-p} V_{A_3} \\ V_{A_5} = V_{A_3} \\ V_{A_6} = V_{A_5} \\ V_{A_1} = 1 \end{array} \right.$$

Da cui si ha:

$$\begin{aligned} V_{A_1} = V_{A_2} = V_{A_3} = V_{A_5} = V_{A_6} &= 1. \\ V_{A_4} &= 2.3333 \end{aligned}$$



Analisi

Secondo passo: Calcolo delle visite alle risorse

Dalle visite alle singole azioni, possiamo ora risalire alle visite totali alle risorse.

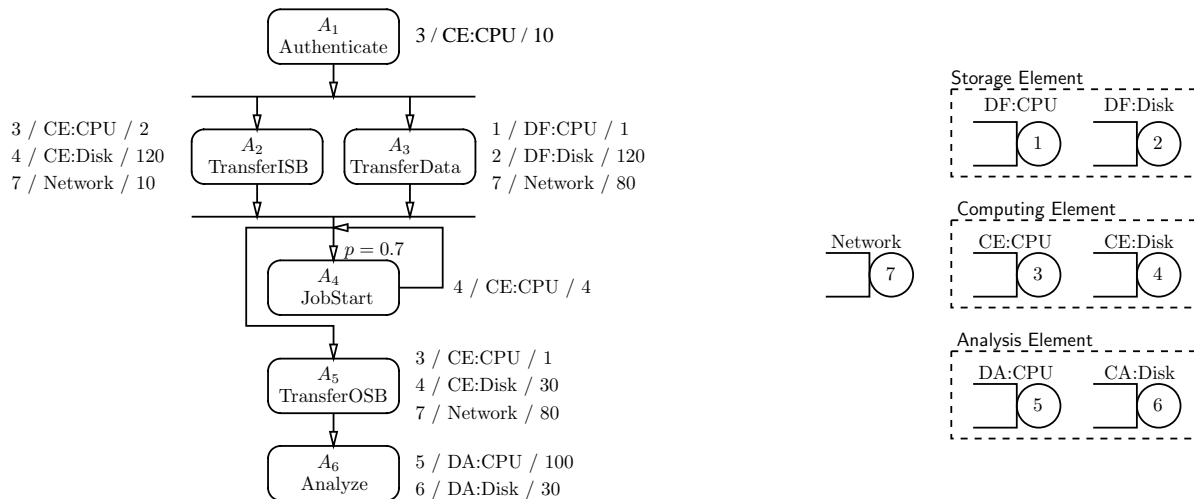
```

source("bsb_closed.m");
source("mva_closed.m");
## Visite alle singole azioni
V_A = [1 1 1 2.333 1 1];
## Visite ai centri di servizio
V = [
    V_A(3)                # DF:CPU
    V_A(3)                # DF:Disk
    V_A(1)+V_A(2)+V_A(4)+V_A(5) # CE:CPU
    V_A(2)+V_A(5)        # CE:Disk
    V_A(6)                # DA:CPU
    V_A(6)                # DA:Disk
    V_A(2)+V_A(3)+V_A(5) # Network
];
  
```



Analisi

Terzo passo: Calcolo delle Domande di Servizio



Esempio

$$D_7 = 10V_{A_2} + 80V_{A_3} + 80V_{A_5}$$



Analisi

```
## Domande di servizio
D = [
  V_A(3)*1           # DF:CPU
  V_A(3)*120        # DF:Disk
  V_A(1)*10+V_A(2)*2+V_A(4)*4+V_A(5)*1 # CE:CPU
  V_A(2)*120+V_A(5)*30 # CE:Disk
  V_A(6)*100        # DA:CPU
  V_A(6)*30        # DA:Disk
  V_A(2)*10+V_A(3)*80+V_A(5)*80 # Network
];
```

Dalla domanda di servizio risaliamo al tempo medio di servizio, che vale D_k/V_k per $k = 1 \dots 7$.

$S = D ./ V;$



Esempio

Quarto Passo: Valutazione delle Alternative

Le domande di servizio del sistema iniziale sono le seguenti:

DF:CPU	DF:Disk	CE:CPU	CE:Disk	DA:CPU	DA:Disk	Network
$D_1 = 1$	$D_2 = 120$	$D_3 = 22.3320$	$D_4 = 150$	$D_5 = 100$	$D_6 = 30$	$D_7 = 170$

Esaminiamo due possibili miglioramenti:

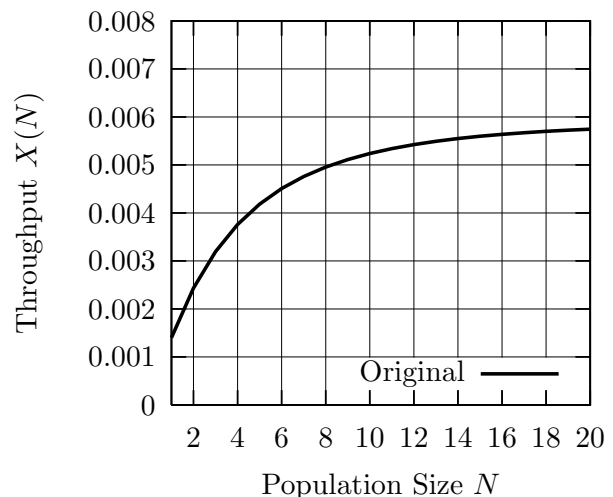
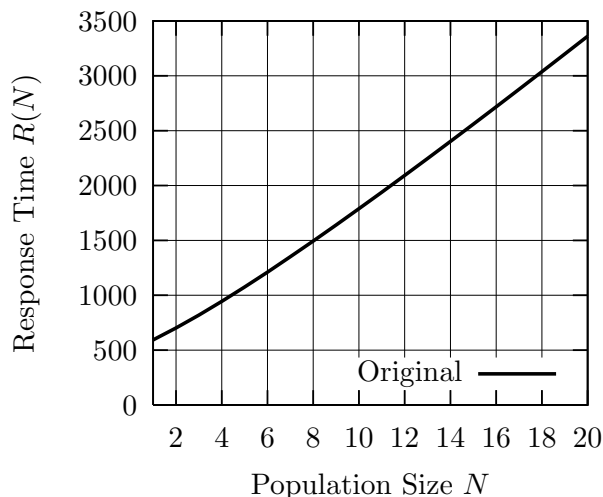
1. Sostituiamo la CPU del CE con una il doppio più veloce (quindi $D_3 \leftarrow 0.5D_3$);
2. Migliorare le prestazioni della rete del 30% (quindi $D_7 \leftarrow 0.7D_7$)
3. In aggiunta al punto precedente, sostituire CE:Disk con un disco il 20% più veloce (quindi $D_4 \leftarrow 0.8D_4$)



Risultati

Situazione iniziale

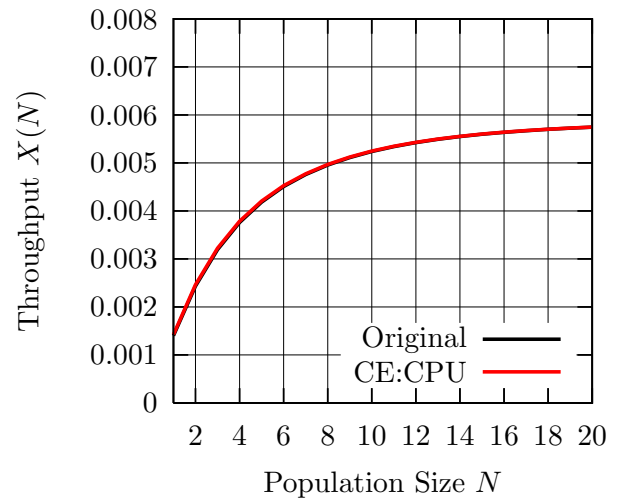
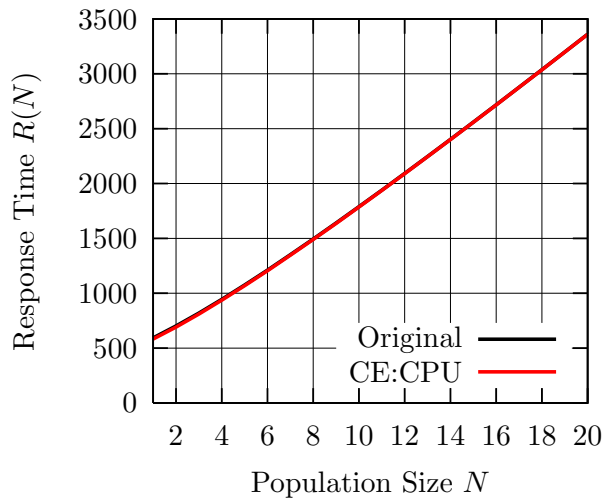
DF:CPU	DF:Disk	CE:CPU	CE:Disk	DA:CPU	DA:Disk	Network
$D_1 = 1$	$D_2 = 120$	$D_3 = 22.3320$	$D_4 = 150$	$D_5 = 100$	$D_6 = 30$	$D_7 = 170$



Risultati

Riduzione domanda di servizio su CE : CPU

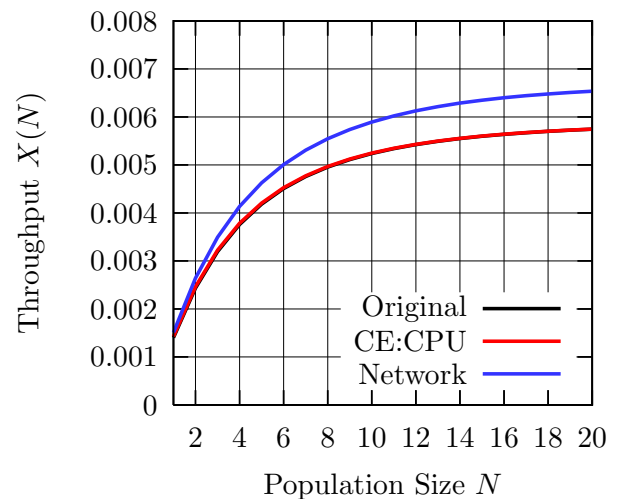
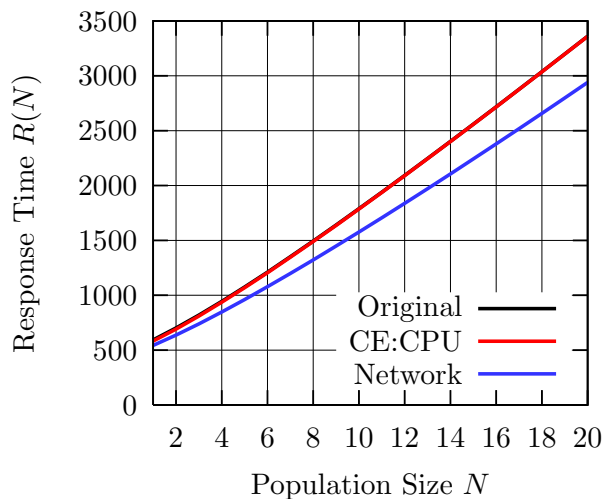
DF:CPU $D_1 = 1$	DF:Disk $D_2 = 120$	CE:CPU $D_3 = 11.1660$	CE:Disk $D_4 = 150$	DA:CPU $D_5 = 100$	DA:Disk $D_6 = 30$	Network $D_7 = 170$
---------------------	------------------------	---------------------------	------------------------	-----------------------	-----------------------	------------------------



Risultati

Riduzione domanda di servizio sulla rete

DF:CPU $D_1 = 1$	DF:Disk $D_2 = 120$	CE:CPU $D_3 = 22.3320$	CE:Disk $D_4 = 150$	DA:CPU $D_5 = 100$	DA:Disk $D_6 = 30$	Network $D_7 = 119$
---------------------	------------------------	---------------------------	------------------------	-----------------------	-----------------------	------------------------



Risultati

Miglioramento di CE:Disk e rete

DF:CPU $D_1 = 1$	DF:Disk $D_2 = 120$	CE:CPU $D_3 = 22.3320$	CE:Disk $D_4 = 120$	DA:CPU $D_5 = 100$	DA:Disk $D_6 = 30$	Network $D_7 = 119$
---------------------	------------------------	---------------------------	------------------------	-----------------------	-----------------------	------------------------

