Università degli Studi di Roma Tor Vergata

Facoltà di Ingegneria

Corso di Metriche e Modelli per Internet

Progetto A

Professore:
Salvatore Tucci
Emiliano Casalicchio

Studenti:
Ibrahim Khalili
Simone Notargiacomo
Lorenzo Tavernese

Anno Accademico 2007-2008

Indice

Capitolo 1

Introduzione

Il lavoro svolto per l'esame di Metriche e Modelli di Internet ha posto l'interesse nell' analisi del comportamento di un sistema web al variare di alcune sue componenti. Nello specifico sono state condotte due tipi di analisi, la prima da un punto di vista *simulativo* mentre la seconda da un punto di vista *analitico*. Entrambe fanno riferimento ad una rappresentazione matematica del sistema reale(modello) denominata *rete di code*(QN, Queueing Network). Una QN tiene conto della contesa per le risorse del sistema e delle code che si generano in attesa del servizio. La soluzione analitica consente di risolvere mediante l'uso di formule e algoritmi una rete di code, ottenendo in questo modo i valori prestazionali desiderati. Tali valori possono altresì essere ottenuti emulando il comportamento del sistema attraverso una simulazione. La simulazione è realizzata mediante un programma in grado di mimare il funzionamento del sistema attraverso flussi di transazioni tra le varie risorse. I valori di performance sono ricavati dalle statistiche collezionate dal programma per ogni coda relativa a tali risorse. Per poter generare un modello di performance è necessario innanzitutto stilare un modello che rappresenti il workload, che nel caso specifico, è stato generato a partire da diverse distribuzioni rappresentanti più valori, come ad esempio il numero di richieste per sessione o la dimensione della pagina html richiesta. Per rendere più chiara l'idea al lettore, si presentano di seguito i requisiti del progetto.

1.1 Requisiti del Progetto

Si consideri il cluster di Server Web in figura 1.1. Il sistema è composto da un Web Switch che riceve le richieste dagli utenti e le distribuisce ad un set di N Server Web. Il Web Switch è connesso alla rete internet mediante il link L1. I Web Server e il Web Switch sono tra di loro connessi mediante la LAN L2. Ogni server è caratterizzato da una CPU, da uno o più dischi ed una scheda di rete (Iw2). La CPU dei server ha un tasso di servizio pari a 150 richieste al secondo (indipendentemente dal tipo di richieste). Il disco ha le seguenti caratteristiche: Seek Time 8.5ms, Controller Time 0.2ms, Rotational Speed 7200 RPM, transfer rate 100MB/sec, block size 2048 byte. Il Web Switch è caratterizzato da una CPU e da due scheda di rete (Is1 e Is2), una che lo connette alla rete L1 ed una che lo connette alla rete L2. La rete L2 è una rete Ethernet 1000Mbit/sec e sia i Web Server che il Web Switch sono ad essa connessi mediante delle schede di rete con un transfer rate di 1000Mbit/sec.

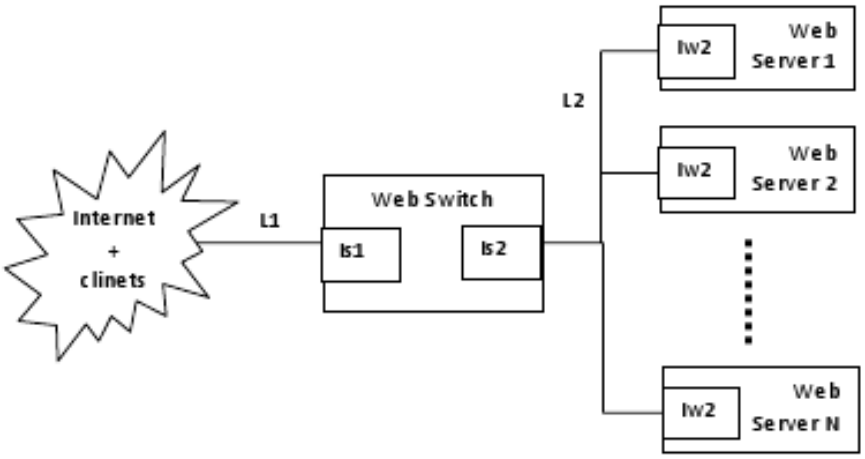

Figura 1.1: Schema

Si suppone che il carico del sistema è caratterizzato come segue:

Richieste per Sessione	Gaussiana Inversa	$X > 0; \mu = 3.86;$ $\lambda = 9.46$
User Think Time	Pareto	$X \geq k; \alpha = 1.4; k = 1$
Num. Oggetti per Richiesta	Pareto	$X \geq k; \alpha = 1.33; k = 2$
Dim. Pag. HTML	LogNormale-Pareto	$\mu = 7.63; \sigma = 1.001$ $X \geq k; \alpha = 1; k = 10240(byte)$
Embedded Objects	LogNormale	$X > 0; \mu = 8.215; \sigma = 1.46$

Tabella 1.1: specifiche

Si chiede agli studenti di studiare il comportamento del sistema utilizzando un modello di simulazione e, fatta l'opportuna caratterizzazione del carico, mediante la risoluzione di un modello analitico. Nello specifico si chiede di:

- Dimensionare opportunamente il sistema per far si che sia in grado di sopportare un carico di 150 utenti al secondo con un utilizzazione massima della risorsa collo di bottiglia compresa tra il 65 ed il 70%.
- Studiare le prestazioni del sistema al variare delle politiche di selezione dei server (ROUND ROBIN, RANDOM, LEAST LOADED)
- Valutare che impatto ha sulle prestazioni del sistema l'introduzione di
  1. una scheda di rete addizionale per i web server (per la gestione separata delle richieste in ingresso ed in uscita)
  2. un link in uscita, dimensionato opportunamente, per lo smistamento delle risposte (Vedi fig. 1.2)
- Valutare che impatto ha, sulle prestazioni del sistema, l'introduzione di un proxy server che offra un cache hit rate del 40%.

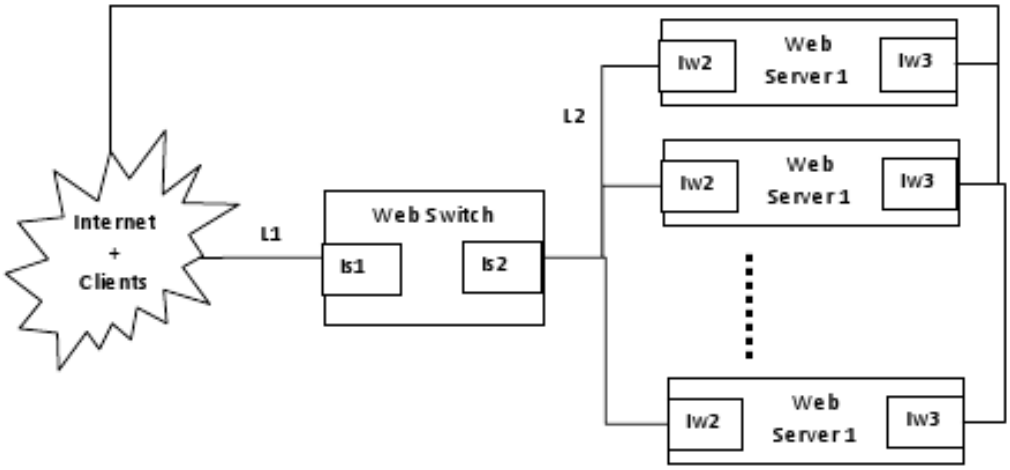

Figura 1.2: Schema

1.2 Approccio Adottato

Si presenta di seguito un breve sunto dei passi che hanno consentito di realizzare il progetto (NON MI PIACE PER NIENTE, DA CORREGGERE), e che verranno affrontati nei capitoli successivi.

1. Caratterizzazione del workload raggruppando i documenti in cluster in base alle loro dimensioni;
2. Scelta delle risorse da modellare come centri di servizio
3. Sviluppo della simulazione ed esecuzione di un set di simulazione per la stima dei parametri da modellare;
4. Studio del transiente
5. Esclusione del transiente dalla simulazione
6. Sviluppo ed esecuzione del modello analitico

Capitolo 2

Caratterizzazione del Workload

Il *workload* del sistema può essere visto come un insieme di tutti gli input che questo riceve durante un dato periodo di tempo. Un *modello di workload* è pertanto una rappresentazione che imita il comportamento del carico reale e risulta utile in quanto permette di modificare in modo semplice i parametri per osservare la risposta del sistema. Il carico di un sistema Web può essere caratterizzato a diversi livelli di astrazione. Il livello più elevato è detto di *business*, e si rivela utile per descrivere il carico in termini di peculiarità delle applicazioni principali; immediatamente sotto vi è il livello di *caratterizzazione funzionale*, il quale descrive i programmi, le richieste o le applicazioni che compongono il carico. L'ultimo livello di dettaglio, detto *orientato alle risorse*, consente di caratterizzare il consumo delle risorse del sistema dovuto al workload e poichè i due livelli più alti non consentono di catturare queste informazioni in modo quantitativo, si è adottato proprio quest'ultimo livello di dettaglio, in quanto è stato modellato il consumo delle risorse in termini di domanda di servizio e di intensità di carico (misurata in richieste HTTP/sec sottoposte al web server).

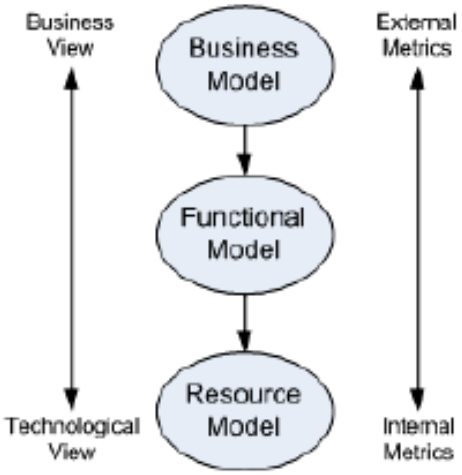

Figura 2.1: Schema

Questa metrica è ben diversa dal numero di utenti che accedono in media al sistema ogni secondo, o al numero di richieste di pagine HTML sottoposte in media al sistema; in quest'ultimo caso, infatti, non si terrebbe conto dei numerosi oggetti inclusi in una pagina HTML. Non essendo in possesso di un carico reale (ad esempio un log di sistema), non è stato possibile effettuare un'analisi del fattore di *burstiness* poichè non si conoscevano il numero medio di richieste HTTP ricevute dal sistema in un determinato intervallo di tempo, nè tanto meno effettuare un'analisi analitica. Di conseguenza, la prima operazione eseguita è stata quella di effettuare un partizionamento dei documenti sulla base dei valori presentati in tabella (tabella requisiti del progetto).

2.1 Partizionamento del Carico

I carichi reali possono presentare componenti molto eterogenee tra loro, in particolar modo per quel che riguarda la loro utilizzazione delle risorse del sistema; spesso la rappresentazione di un carico mediante una sola classe manca di accuratezza in questo senso, poiché un solo valore medio non è

molto significativo in caso di elevata varianza dei campioni considerati. La motivazione per cui è utile partizionare il carico è duplice: in primo luogo la rappresentatività del modello di carico è incrementata, ed inoltre è molto più elevato anche il potere predittivo del modello stesso. Le tecniche di partizione suddividono il carico in una serie di classi le cui popolazioni sono omogenee secondo un qualche criterio. Il più significativo nell ambito della valutazione delle prestazioni è senza dubbio quello dell utilizzazione delle risorse di sistema A questo punto è necessario partizionare le insieme delle richieste in classi di similarità. I cosiddetti algoritmi di clustering sono delle tecniche che consentono di trovare dei raggruppamenti, detti appunto cluster, nell insieme degli elementi considerati. Questi si basano sull'individuazione di alcuni centroidi, ovvero elementi medi dei cluster, e cercano di assegnare ciascun elemento al cluster più oppurtuno mediante il calcolo della minima distanza da uno dei centroide fissati. Per l'analisi del caso di studio si è deciso di utilizzare il k-means che rientra nella classe degli algoritmi di clustering non-gerarchici .

Inputs:
  $I = \{i_1, ..., i_k\}$ (Instances to be clustered)
  n (Number of clusters)
Outputs:
  $C = \{c_1, ..., c_n\}$ (cluster centroids)
  $m : I \rightarrow C$ (cluster membership)

procedure **KMeans**
  Set **C** to initial value (e.g. random selection of **I**)
  For each $i_j \in I$
    $m(i_j) = \underset{k \in \{1..n\}}{\operatorname{argmin}} distance(i_j, c_k)$
  End
  While m has changed
    For each $j \in \{1..n\}$
      Recompute i_j as the centroid of $\{i | m(i) = j\}$
    End
    For each $i_j \in P$
      $m(i_j) = \underset{k \in \{1..n\}}{\operatorname{argmin}} distance(i_j, c_k)$
    End
  End
return C
End

Figura 2.2: Schema

L'algoritmo K-Means è stato applicato a 10000000 documenti, generati tenendo conto delle varie distribuzioni e suddividendoli in tre classi. Si è deciso di scegliere i centroidi iniziali il più lontano possibile tra loro, prendendo come punti di riferimento i valori minimo, massimo e medio delle dimensioni dei file. I centroidi iniziali sono stati:

- centroide min 1.453304
- centroide max 715827882.666667
- centroide medio 357913940.606681

Si è scelto di far convergere l'algoritmo nel caso in cui la differenza di valori dei centroidi tra due iterazioni successive fosse minore di 10^{-10} ed i risultati sono stati i seguenti: Si può notare come i documenti seguano una distribuzione

Classe	Dimensione(byte)	Richieste
1	10281	9999995
2	279513744	4
3	715827882	1

Tabella 2.1: Risultati clustering

heavy-tailed, dove per heavy tailed si intende dire che vi è un' elevatissima presenza di documenti di piccole dimensioni e una piccola, ma non trascurabile, presenza di documenti di dimensioni maggiori anche di diversi ordini di grandezza. Questo fenomeno è catturato dall'espressione, detta *Power-Law*, qui riportata

$$P[X > x] = kx^{-\alpha}L(X)$$

in cui L(x) è una funzione che varia molto lentamente. Un caso particolare di questa, è la distribuzione di *Pareto* in cui la coda della distribuzione è esprimibile con la relazione :

$$P[X > x] = kx^{-\alpha}$$

Proprio per questa caratteristica una singola classe non può approssimare con sufficiente precisione il carico Web, dal momento che la grande variabilità degli elementi riduce il significato statistico delle misurazioni effettuate sui valori medi. Per ogni classe sono state inoltre ricavate le probabilità di arrivo di richieste HTTP, che torneranno utili nel momento in cui si utilizzerà il modello analitico:

Classe	Prob. arrivo richieste classe R
1	0.9999995
2	0.0000004
3	0.0000001

Tabella 2.2: specifiche2

Capitolo 3

Simulazione

Il modello simulativo è stato di fondamentale importanza in quanto ha permesso di ricavare i parametri mancanti del sistema. A tal proposito, in questa sezione verranno presentati:

- Introduzione a CSIM
- Rappresentazione del workload
- Rappresentazione delle risorse
- Dimensionamento del sistema
- Esecuzione del transiente
- Presentazione dei risultati

3.1 CSIM

Un *simulation engine* consiste di un insieme di oggetti e metodi usati per costruire un modello di simulazione. CSIM è un package *process-oriented* integrabile in programmi C/C++, che realizza un modello di simulazione discreto: *next event time advance* (event-driven). CSIM consente di modellare il sistema mediante l'utilizzo di speciali processi che interagiscono tra loro

utilizzando delle apposite strutture dati. Il programma mantiene un tempo simulato in modo da consentire lo studio del comportamento del sistema al variare del tempo. CSIM fornisce al programmatore una serie di oggetti, che definiscono le astrazioni di supporto alla costruzione e all'esecuzione del modello simulativo. Gli oggetti principali sono:

Processes: sono le entità attive che sfruttano le risorse disponibili, attendono gli eventi o collezionano statistiche.

Facilities: rappresentano quelle risorse (ad esempio serventi) che vengono utilizzate dai processi attivi.

Events: eventi usati al fine di mantenere una sincronizzazione tra i processi.

Mailboxes: usate per lo scambio di messaggi che realizza le comunicazioni tra i processi.

Data collection structures: apposite strutture dati in grado di collezionare le statistiche rilevate durante la simulazione. Nel caso di studio in esame si è fatto uso anche di boxes e meters, i primi per collezionare statistiche relative ad un sottoinsieme di risorse del sistema, i secondi per stimare il numero di passaggi in un determinato punto della rete.

Process classes: classi definite per caratterizzare i processi nell uso di risorse, nella gestione degli eventi e nella raccolta statistiche.

Stream: flussi di numeri casuali.

A titolo di esempio, si potrebbe dunque rappresentare un sistema di tipo Client/Server come una facility (il server) ed un processo CSIM (il client) che usa la facility. Data la natura del CSIM, brevemente riassunta finora, è evidente come questo offra ottime possibilità di tradurre un modello a reti di code (QN) in un modello simulativo.

3.2 Rappresentazione del workload

La rappresentazione del carico mediante le distribuzioni è stata facilitata in quanto CSIM offre delle funzioni per generare i valori sia secondo la distribuzione Lognormale che secondo la distribuzione Pareto. Per quanto concerne il numero di richieste per sessione, che seguono la distribuzione gaussiana inversa, si è utilizzato l'*algoritmo di Michael/Shucany/Haas*:

1. Si generi V da $N(0,1)$ (distribuzione normale) e si ponga y = v^2;
2. Si ponga $x_1 = \mu + \mu^2 * \frac{y}{2*\lambda} \breve{} \frac{\mu}{2*\lambda} * \sqrt{4 * \mu * \lambda * y + \mu^2 * y^2}$;
3. Si generi u da $U(0,1)$ (distribuzione uniforme);
4. Se $u \leq \frac{\mu}{mu+x_1}$ allora si ponga $x = x_1$, altrimenti si ponga $x = \mu^{\frac{2}{x_1}}$

Il workload cui è sottoposto il sistema per ogni utente è il seguente:

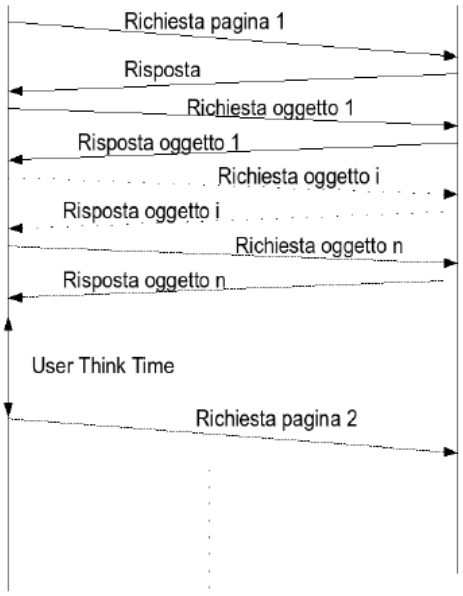

Figura 3.1: Carico

Il singolo utente, durante una sessione, richiede un numero imprecisato di pagine web intervallate da un think time. Ognuna di queste pagine web può

poi essere composta di oggetti embedded (nel caso in esame almeno due), come ad esempio fogli di stile, immagini e quant'altro. Di conseguenza il carico è stato generato seguendo quest'approccio:

1. Si generano il numero di richieste per sessione che l'utente sottoporrà al sistema
2. Si genera la dimensione della pagina da richiedere
3. Si richiede ed ottiene il documento
4. Si genera il numero di oggetti per la pagina richiesta
5. Per ogni oggetto si genera la dimensione e si avvia la richiesta
6. Si attende un tempo pari a User Think Time e se sono presenti altre richieste si torna al punto 2.

Si presenta di seguito il frammento di codice rappresentante quest'approccio:

```
//generazione delle sessioni e delle relative richieste
while(i<array_length) {
   session = session_request(mu_session, lambda_session);
   for(j=0; j < session; j++) {
        if( i < array_length) {
           array[i] = html_page_size(mu_html, sigma_html, alfa_html);
       }
   i++;
   objects = object_per_request(alfa_obj);
   for(k=0; k<objects; k++){
      if(i<array_length)
          array[i] = embedded_object_size(mu_emb, sigma_emb);
      i++;
    }
  }
}
```

3.3 Rappresentazione delle risorse

Come brevemente introdotto in precedenza, una facility modella in ambiente CSIM una risorsa del sistema, con la rispettiva coda. La libreria mette a disposizione diversi tipologie di facility, scegliendo le più adatte al modello:

- facility risorsa con una sola coda ed un solo servente, occupabile da un solo processo alla volta;
- facility set array di facility di primo tipo e quindi una risorsa con più code e più serventi (una coda per ogni servente) indipendenti tra loro.
- multi-server facility risorsa con una sola coda ma più serventi, occupabili tutti contemporaneamente;

Per realizzare la simulazione sono state utilizzate solo le prime due categorie. Si riporta il codice CSIM in cui avviene la dichiarazione delle suddette facility, che verranno analizzate in seguito:

```
FACILITY cpuWS[NUM_SERVER];
FACILITY diskWS[NUM_DISK*NUM_SERVER];
FACILITY L2;
FACILITY CPU_web_switch;
FACILITY inLink;
FACILITY outLink;
FACILITY LINK\_ADD;
FACILITY LS1;
FACILITY LS2;
FACILITY LW2[NUM_SERVER];
FACILITY LW3[NUM_SERVER];
```

Prima di presentare le domande di servizio per ogni FACILITY, si definiscono dei parametri della rete e alcune funzioni di supporto per consentire una maggiore comprensione:

`TCPOV` (overhead in byte introdotto dal TCP)	20
`IPOV` (overhead in byte introdotto da IP)	20
`FRAMEOV` (overhead in byte introdotto dal tipo di rete fisica)	18
`MSS` (massima dimensione in byte di un pacchetto TCP)	1460
`AVG_SIZE_HTTP_REQ` (dimensione media di una richiesta HTTP)	290

Tabella 3.1: specifiche3

```
int NDatagrams(double m) {
        int n;
        double f;
        f=m/(double)MSS;
        n=(int)ceil(f);
        return n;
}
```

`Ndatagrams` consente di determinare il numero di messaggi necessari per inviare un messaggio lungo m bytes.

```
int Overhead(double m) {
   return (NDatagrams(m)*(TCPOV+IPOV+FRAMEOV));
}
```

`Overhead` rappresenta l'overhead necessario per inviare un messaggio lungo m bytes.

```
double NetworkTime(double m, double bandwidth) {
        return (double)(8*(m+Overhead(m)))/(double)(1000*1000*bandwidth);
}
```

`NetworkTime` consente di calcolare il tempo (in secondi) necessario ad un messaggio lungo m bytes per transitare attraverso una rete caratterizzata da una banda `bandwidth` (in Mbps). Si presentano di seguito le domande di servizio relative ai componenti della rete (N.B. : non sono presenti le domande di servizio delle schede di rete LS2, LW2 e LW3, poiché si è assunto che il loro transfer rate sia pari alla banda delle reti a cui sono collegate).

```
double D_InLink() {
    return NetworkTime(AVG_SIZE_HTTP_REQ, INLINK_BANDWIDTH) +
    3*NetworkTime(0.0001, INLINK_BANDWIDTH);
}

double D_OutLink(double docSize) {
    return NetworkTime(docSize, OUTLINK_BANDWIDTH) + 2 * NetworkTime(0.0001,
    OUTLINK_BANDWIDTH);
}

double D_LAN(double docSize) {
   if(docSize == 0) {
       return NetworkTime(AVG_SIZE_HTTP_REQ, BANDWIDTH_L2);
   }
    return NetworkTime( docSize, BANDWIDTH_L2);
}
```

Per quanto concerne la domanda di servizio della LAN si è operata una distinzione tra fase di richieste (in cui `docSize = 0`) e fase di risposta

```
double D_Cpu(double service_rate) {
   return 1 / service_rate;
}
```

La domanda di servizio della CPU vale sia per il Web Server che per il Web Switch, infatti il Service_Rate dev'essere specificato come parametro di input.

```
double D_WSDisk(double doc_size) {
   double ret = (number_of_blocks(doc_size)) * ((DISK_SEEK_TIME/pow(10,3)) +
        ROTATIONAL_LATENCY + (CONTROLLER_TIME/pow(10,3)) + (BLOCK_SIZE/((
       double)DISK_TRANSFER_RATE*pow(10,6))));
   return ret;
}
```

La funzione `number_of_blocks` indica il numero di blocchi necessari da leggere per un intero documento di dimensione `doc_size` bytes ed è così definita:

```
int number_of_blocks(double docSize) {
   return ceil(docSize/BLOCK_SIZE);
}
```

I restanti parametri della domanda di servizio del disco sono:

SeekTime: secondi che impiega il braccio del disco a posizionarsi sul cilindro corretto

ControllerTime: secondi spesi dal controller del disco per processare una richiesta di I/O

TransferRate: tasso espresso in MB/sec con cui i dati vengono trasferiti dal/al disco

BlockSize: dimensione in bytes del blocco del disco

RotationalLatency: tempo medio in secondi affinchè il settore giusto capiti sotto la testina del braccio del disco

```
double D_linkAdd(double doc_size) {
 return NetworkTime(doc_size, BANDWIDTH_LINKADD) + 2 * NetworkTime(0.0001,
     BANDWIDTH_LINKADD);
}

double D_LS1in()
{
  return NetworkTime(AVG_SIZE_HTTP_REQ, LS1_TRANSFER_RATE) + 3*NetworkTime
      (0.0001, LS1_TRANSFER_RATE);
}

double D_LS1out(double doc_size)
{
  return NetworkTime(doc_size, LS1_TRANSFER_RATE) + 2 * NetworkTime(0.0001,
      LS1_TRANSFER_RATE);
}
```

3.4 Dimensionamento del Sistema

Rappresentato il workload e le risorse del sistema, è stato possibile dimensionare i parametri mancanti della rete, adottando un approccio di tipo empirico, iniziando a dimensionare i vari link per poi passare ai web server (CPU e Dischi). I parametri da dimensionare erano i seguenti:

- Banda dell'incoming link;
- Banda dell'outgoing link;
- CPU Web Swtich Service Rate;
- Numero di Web Server
- Numero di dischi per Web Server
- Banda del link Addizionale

Transfer rate delle schede di rete LS1, LS2, LW2, LW3. E' doveroso sottolineare che tutte le schede di rete sono state dimensionate tenendo conto delle reti a cui erano collegate e quindi queste presenteranno un transfer rate pari alla banda della LAN a cui sono accoppiate. Le simulazioni sono state condotte sulla configurazione standard del sistema in cui il server veniva scelto in modo random e i risultati sono stati mediati su 10 run diversi, poiché non è molto corretto attenersi agli esiti di una singola simulazione del sistema. I parametri risultanti sono stati:

Risorsa	Valore
Banda incoming Link	45 Mbps (T3)
Banda Outgoing Link	1000 Mbps
CPU Web Switch Service Rate	9600 richieste/secondo
Numero di Web Server	66
Numero di dischi per Web Server	12
Transfer Rate LS1	1000 Mbps
Transfer Rate LS2	1000 Mbps
Transfer Rate lw2	1000 Mbps
Transfer Rate LW3	622 Mbps (OC-12)
Banda Link Addizionale	622 Mbps (OC-12)

Tabella 3.2: specifiche4

3.5 Transiente

Un sistema web attraversa una fase iniziale durante la quale le code dei vari serventi cominciano a riempirsi, fino a raggiungere un livello di regime, in cui gli indici di prestazione del sistema assumono valori stabili. E' quindi importante non tenere conto delle statistiche relative al periodo iniziale (transiente o transitorio), in quanto queste potrebbero rischiare di falsare i risultati complessivi. La determinazione della lunghezza del transiente è stata ricavata tramite l'algoritmo di Welch descritto da Law e Kelton in. Esso consente di determinare il numero di osservazioni l (di un indice del sistema Y) che è necessario scartare dal numero totale di osservazioni m. Se l ed m sono scelti troppo piccoli la stima dell'indice potrebbe discostarsi dal caso stazionario; viceversa se l è scelto troppo grande la stima dell'indice rischia di possedere una varianza troppo grande. L'algoritmo proposto da Welch fa uso di una procedura grafica per determinare un valore di l tale che $E[Yi] \approx lim(i \rightarrow \inf)E[Y_i] = \nu peri > l$. Ciò è equivalente a determinare per quale valore di l la curva media del transiente si appiattisce fino a raggiungere il valore ν . In generale è difficile ricavare l tramite una singola replica, a causa della variabilità dell'indice Y. Per risolvere un tale problema Welch propone una procedura basata sull'utilizzo di n repliche della simulazione indipendenti tra loro. L'algoritmo è il seguente:

1. Fare n repliche della simulazione (dove $n \geq 5$), ognuna di lunghezza m (dove m sono le osservazioni fatte nella simulazione), porre Y_{ij} come la i-esima osservazione della j-esima replica ($j = 1, 2, \ldots, n; i = 1, 2, \ldots, m$)

2. Calcolare Y_i come media delle Y_{ij} di ogni replica

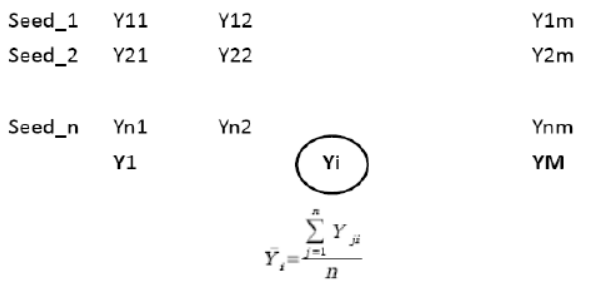

Figura 3.2: Carico

3. Calcolare il valore di Moving average $Y_i(w)$, dove w è la finestra ed è un intero positivo tale che $w \leq \frac{m}{2}$, come segue:

$$Y_w = \{\frac{\sum_{s=-w}^{w} Y_{i+s}}{2w+1} sei = w+1, \ldots, m-w$$

4. Graficare la media mobile e si sceglie l = i tale che la media mobile sembra convergere.

Per quanto riguarda il caso di studio in esame, sono state effettuate 3 prove utilizzando la configurazione standard (random), effettuando n=10 repliche della simulazione ognuna di lunghezza m = 500000, graficando il tempo di risposta al variare della finestra W.

Caso 1: n=10, m = 500000, W = 5000

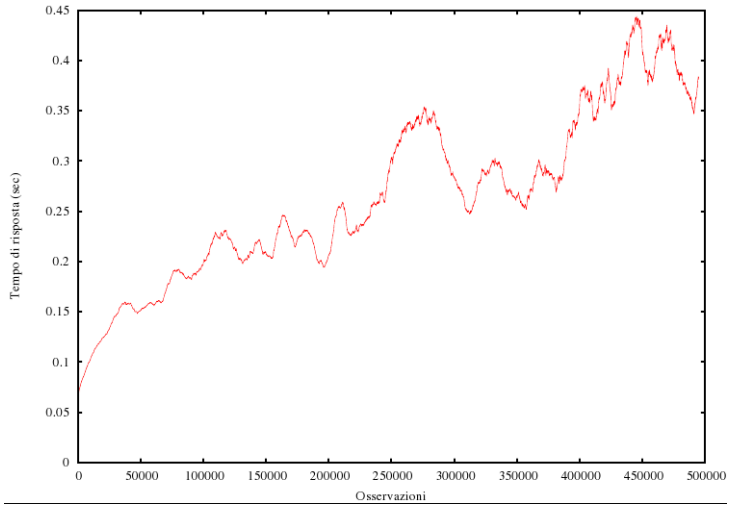

Figura 3.3: Grafico1

Utilizzando una finestra piccola rispetto al numero di osservazioni, si nota un'alta variabilità dei valori osservati e un trend monotono crescente all'aumentare del numero di osservazioni.

Caso2 : n=10, m=500000, W = 50000

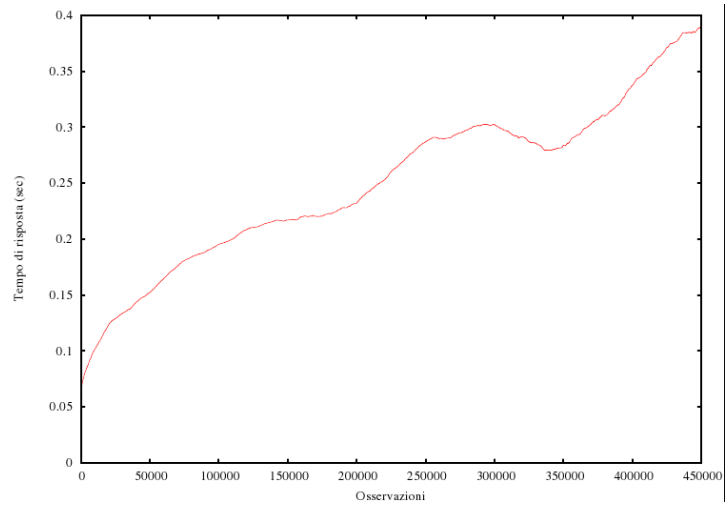

Figura 3.4: Grafico2

In questo caso è sempre presente l'andamento crescente del tempo di risposta, ma la variazione dei valori è molto minore rispetto al caso 1.

Caso3: n=10, m=500000, W = 100000

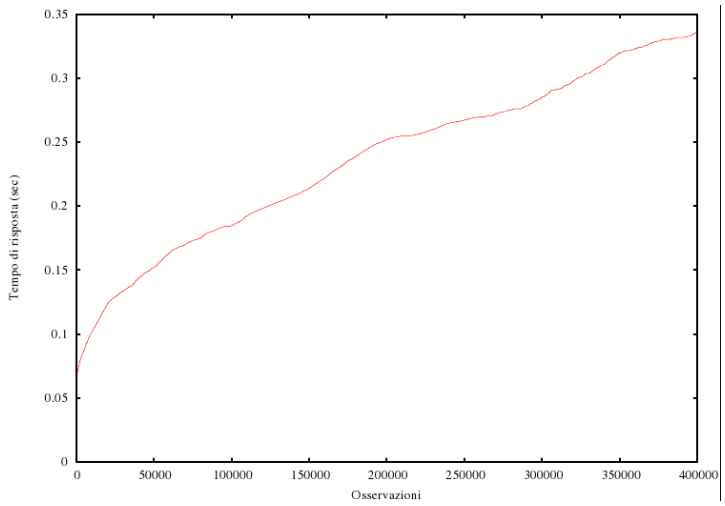

Figura 3.5: Grafico3

Anche in quest'ultimo caso, il tempo di risposta non sembra convergere, nonostante presenti un andamento piuttosto stabile. La mancata convergenza verso un valore potrebbe dipendere dalla distribuzione heavy tailed dei documenti, e dal fatto che i dischi potrebbero essere sottoposti a svolgere delle operazioni per file di centinaia di megabyte (si vedano i risultati del partizionamento del carico). Non essendo stato possibile determinare con precisione il valore l, si è deciso comunque di eliminare dalla simulazione le prime 100000 osservazioni della simulazione, pur avendo la consapevolezza che il tempo di risposta tende ad aumentare al numero di osservazioni.

3.6 Struttura della simulazione

Si presentano di seguito alcuni aspetti ritenuti importanti per comprendere al meglio la simulazione, in modo particolare la generazione di una sessione

e delle relative richieste (il processo web client), e il processo principale che si occupa di raccogliere le statistiche.

3.6.1 Webclient

Di seugito si riporta un frammento di codice relativo alla creazione di una sessione e relative richieste.

```
int session = session_request(mu_session, lambda_session);
..
for(i=0; i < session; i++) {
   html_page = html_page_size(mu_html, sigma_html, alfa_html);
  set_process_class(requestClasses[get_doc_class(html_page)]);
   if(variant == PROXY && stream_prob(p_hit_proxy) > 0.4) {
     web_client(html_page, variant, bool_transient, iter);
   }
   if(variant != PROXY) {
      web_client(html_page, variant, bool_transient, iter);
   }
   num_embedded_objects = object_per_request(alfa_obj);
   for(j=0; j < num_embedded_objects; j++) {
      emb_obj_size = embedded_object_size(mu_emb, sigma_emb);
      set_process_class(requestClasses[get_doc_class(emb_obj_size)]);
      if(variant == PROXY && stream_prob(p_hit_proxy) > 0.4) {
          web_client(emb_obj_size, variant, bool_transient, iter);
     }
     if(variant != PROXY) {
         web_client(emb_obj_size, variant, bool_transient, iter);
     }
  }
  hold(user_think_time(alfa_tt));
}
```

Per ogni richiesta della sessione viene invocata la funzione web_client specifica quali risorse vengono utilizzate e in che misura.

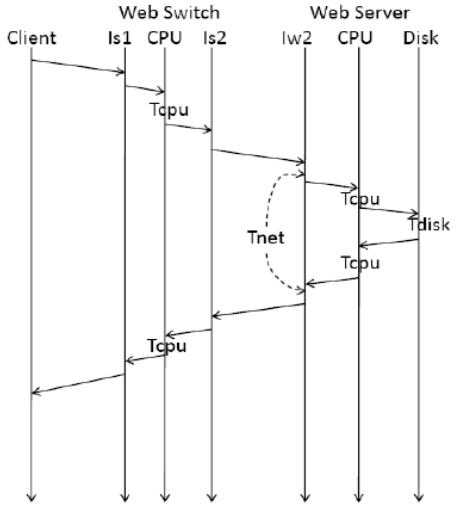

Figura 3.6: webclient

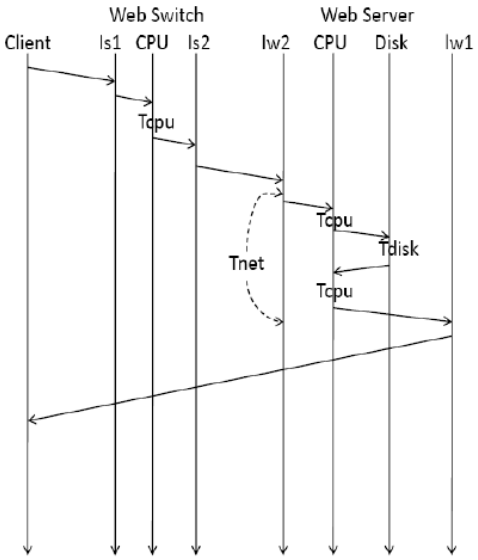

Figura 3.7: webclient2

Si presenta di seguito un frammento della funzione web_client per specificare come avvengono i cicli di richiesta risposta sopra rappresentati. In CSIM questi cicli sono rappresentati da un uso esclusivo delle risorse modellate:

```
use(inLink, D_InLink());
switch_start_time = enter_box(WebSwitch);
use(LS1, D_LS1in());
use(CPU_web_switch, D_Cpu(CPU_WEB_SWITCH_SERVICE_RATE));
 use(LS2, D_LAN(0)); //stessa banda della LAN, in richiesta doc_size=0
 exit_box(WebSwitch, switch_start_time);
 use(L2, D_LAN(0));
/* random */
if(variant == RANDOM || variant == LINK\_ADD || variant == PROXY) {
    tmp_server = current_server;
    current_server = csim_random_int(0, NUM_SERVER-1);
}
else if (variant == ROUND_ROBIN) { //round robin
    tmp_server = current_server;
    current_server = (tmp_server+1)%NUM_SERVER;
}
// least loaded
else if (variant == LEAST_LOADED) {
   tmp_server = get_least_loaded();
}
server_start_time = enter_box(WebServer);
use(LW2[tmp_server], D_LAN(0));
use(cpuWS[tmp_server], D_Cpu(CPU_SERVICE_RATE));
//selezione del disco round robin
tmp_disk = currentDisk[tmp_server];
currentDisk[tmp_server] = (tmp_disk+1)%NUM_DISK;
use(diskWS[tmp_server*NUM_DISK + tmp_disk], D_WSDisk(doc_size));
use(cpuWS[tmp_server], D_Cpu(CPU_SERVICE_RATE));
if(variant != LINK\_ADD) {
    use(LW2[tmp_server], D_LAN(doc_size));
}
else {
   use(LW3[tmp_server], D_linkAdd(doc_size));
}
exit_box(WebServer, server_start_time);
if(variant == LINK\_ADD) {
   use(LINK\_ADD, D_linkAdd(doc_size));
}
else {
  use(L2, D_LAN(doc_size));
  use(LS2, D_LAN(doc_size));
  use(CPU_web_switch, D_Cpu(CPU_WEB_SWITCH_SERVICE_RATE));
  use(LS1, D_LS1out(doc_size));
  use(outLink, D_OutLink(doc_size));
}
```

Dal frammento di codice sopra riportato si può notare come sia il web switch che i web server sono stati modellati attraverso dei BOXES (che servono per collezionare statistiche relative ad un sottoinsieme delle risorse di un sistema), inoltre si può notare come alcuni componenti nel caso sia presente il link addizionale siano molto meno utilizzati (come ad esempio il web switch o la LAN L2).

3.6.2 Il processo principale

Ogni client è rappresentato come appena discusso da un processo CSIM. Ogni processo viene creato da un processo principale secondo una particolare legge degli arrivi (in questo caso esponenziale). Il processo principale si occupa inoltre di reimpostare le statistiche raccolte una volta superato il periodo di transiente in maniera da escludere dalle considerazioni finali quelli che sono valori non particolarmente aderenti alla realtà perché ottenuti in condizioni del sistema non standard (carico minino, code vuote etc.). La simulazione procede, grazie al costrutto while e la condizione di convergenza, finché non vengono raccolti valori tali da ottenere dei risultati considerati rilevanti (con un certo grado di confidenza) Nel caso in esame si è scelto un livello di confidenza del 98% con un errore relativo di 0.005. Il codice che realizza tutto questo è di seguito riportato.

```
..
for(i=0; i<NUM_ITERATIONS; i++){
 ...
 while(state(converged)==NOT_OCC && num_osservazioni<500000) {
   hold(exponential(1/(double)ARRIVAL));
   web_session(client_id, variante, 0, -1);
   client_id++;
   //valori da scartare perche' facenti parte del transiente
   if(num_osservazioni>100000 &&(!reset)) {
        printf("Reset statistics %g\n", simtime());
        reset();
        reset=1;
        ...
   }
}
report_facilities();
```

```
report_table(rtime);
report_boxes();
meter_summary();
tabulate(resptime, table_mean(rtime));
statistics(i, variante);
rerun();
...
}
```

Dal frammento di codice sopra riportato si può notare il ciclo `for` e la funzione `rerun()` che consentono di eseguire più simulazioni e quindi di ottenere risultati più accurati; inoltre si può notare come siano stati scartati i valori relativi al transiente. E' opportuno sottolineare come alla condizione di convergenza sia stata aggiunta un'ulteriore condizione basata sul numero di osservazioni, poiché come visto nello studio del transiente, la convergenza potrebbe non verificarsi per tutte le iterazioni.

3.7 Presentazione dei risultati

Si presentano di seguito i risultati mediati su 10 iterazioni, per ogni componente del sistema preso in esame e per tutte le varianti analizzate. Le statistiche riportate sono l'utilizzazione, la lunghezza media delle coda e il tempo di risposta per ogni classe di richiesta.

3.7.1 Risultati Random

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	0.6673596	0.0000002	0.0000001	0.6673599
disco i-esimo:	0.2901815	0.0000000	0.0000000	0.2901815
inLink:	0.3070558	0.0000002	0.0000001	0.3070561
outLink:	0.2779962	0.0000000	0.0000000	0.2779962
cpu web switch:	0.6882190	0.0000003	0.0000001	0.6882193
LAN:	0.2774720	0.0000000	0.0000000	0.2774720
LS1:	0.2917997	0.0000000	0.0000000	0.2917997
LS2:	0.2774737	0.0000000	0.0000000	0.2774737
LW2: 0.0042041 0.0000000 0.0000000	0.0042041			

Tabella 3.3: Utilizzazioni

Come si può notare, il vincolo sull'utilizzazione richiesto nelle specifiche del progetto è stato rispettato, con la risorsa collo di bottiglia (CPU Web Switch) che presenta un'utilizzazione del 68,8%. Le altre risorse, ad eccezione della CPU del web server, presentano un'utilizzazione piuttosto basse.

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	1.5556346	0.0000003	0.0000001	1.5556350
disco i-esimo:	1.6398481	0.0000002	0.0000000	1.6398483
inLink:	0.8287492	0.0000002	0.0000001	0.8287495
outLink:	4.2762899	0.0000000	0.0000000	4.2762899
cpu web switch:	3.6788684	0.0000005	0.0000003	3.6788692
LAN:	1.6334769	0.0000001	0.0000000	1.6334770
LS1:	3.3884849	0.0000000	0.0000000	3.3884849
LS2:	2.7263059	0.0000002	0.0000001	2.7263062
LW2:	0.0044529	0.0000000	0.0000000	0.0044529

Tabella 3.4: Lunghezza code

Per quanto concerne la lunghezza della coda, si nota un accodamento delle risposte nel link di uscita, che presenta valori più alti persino del web switch. I tempi medi di risposta per la Classe1 evidenziano una lentezza generale dei

Tabella 3.5: Tempo medio di risposta

Centro	Classe1	Classe2	Classe3
cpu web server i-esimo:	0.0155025	0.0000349	0.0000133
disco i-esimo:	0.8416113	0.0000000	0.0000000
inLink:	0.0002501	0.0000286	0.0000122
outlink:	0.0012935	0.0000000	0.0000000
cpu web switch:	0.0005546	0.0000599	0.0000377
LAN:	0.0002467	0.0000079	0.0000003
LS1:	0.0005116	0.0000013	0.0000004
LS2:	0.0004116	0.0000223	0.0000111
LW2:	0.0000445	0.0000000	0.0000000

dischi nel soddisfare le richieste. Per le altre due classi non è possibile fare alcun tipo di assunzione, dato il limitato numero di osservazioni di Classi 2 e 3 a cui è sottoposto il sistema e dato il limite temporale della simulazione, che non permette alle richieste di queste due classi di essere soddisfatte (non mi piace molto la cosa del limitato numero di osservazioni). Risultati sicuramente più significativi per le Classi superiori saranno osservabili nel momento in cui verranno presentati i risultati del modello analitico.

3.7.2 Risultati Round Robin

Le utilizzazioni delle varie risorse adottando la politica round robin sono quasi identiche a quelle con politica random presentate precedentemente. Rispetto al caso presentato precedentemente, si nota un leggero aumento delle code dei dischi del web server e della cpu del web switch, mentre si ha una leggera diminuzione o valori analoghi per tutte le altre risorse. Anche per quanto

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	0.6747827	0.0000003	0.0000001	0.6747830
disco i-esimo:	0.2937349	0.0000000	0.0000000	0.2937349
inLink:	0.3104327	0.0000002	0.0000001	0.3104331
outLink:	0.2814336	0.0000000	0.0000000	0.2814336
cpu web switch:	0.6958719	0.0000003	0.0000001	0.6958723
LAN:	0.2809161	0.0000000	0.0000000	0.2809161
LS1:	0.2954094	0.0000000	0.0000000	0.2954094
LS2:	0.2809159	0.0000000	0.0000000	0.2809159
LW2: 0.0042562 0.0000000 0.0000000	0.0042562			

Tabella 3.6: Utilizzazioni

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	1.0056119	0.0000004	0.0000001	1.0056125
disco i-esimo:	1.6242344	0.0000000	0.0000002	1.6242346
inLink:	0.8558574	0.0000002	0.0000005	0.8558581
outLink:	4.4133256	0.0000000	0.0000000	4.4133256
cpu web switch:	3.9749297	0.0000004	0.0000003	3.9749304
LAN:	1.6945618	0.0000001	0.0000000	1.6945619
LS1:	3.4855153	0.0000006	0.0000000	3.4855159
LS2:	2.8162152	0.0000000	0.0000001	2.8162153
LW2:	0.0045441	0.0000000	0.0000000	0.0045441

Tabella 3.7: Lunghezza Code

Centro	Classe1	Classe2	Classe3
cpu web server i-esimo:	0.0099264	0.0000537	0.0000101
disco i-esimo:	0.5745555	0.0000000	0.0000000
inLink:	0.0002556	0.0000278	0.0000574
outlink:	0.0013225	0.0000000	0.0000000
cpu web switch:	0.0005932	0.0000531	0.0000403
LAN:	0.0002535	0.0000113	0.0000003
LS1:	0.0005215	0.0000742	0.0000004
LS2:	0.0004213	0.0000008	0.0000151
LW2:	0.0000449	0.0000000	0.0000000

Tabella 3.8: Tempo medio di risposta

riguarda i tempi medi di risposta, i risultati sono analoghi al caso precedente, fatta eccezione per i dischi del web server che presentano un leggero aumento dei tempi, dovuti all'aumento delle code.

3.7.3 Risultati Least Loaded

Adottando questa politica diminuiscono i tempi medi di servizio e di conseguenza si ha un aumento del carico del sistema. Per questo motivo è possibile notare un aumento delle utilizzazioni per tutte le risposte del sistema, che porta a superare la soglia del 70% sia per la CPU del web switch che per quella del Web Server. Come avviene per le utilizzazioni, anche la lunghezza media delle code aumenta per tutti i centri, ad eccezione dei dischi del web server, poiché non accade mai che un server particolarmente carico riceva altre richieste da far processare ai suoi dischi se prima le code non sono state svuotate I tempi di risposta sono più o meno analoghi a quelli presentati in precedenza, fatta eccezione per i dischi del web server, che, come ci si poteva aspettare, riducono il loro tempo medio di risposta.

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	0.7167869	0.0000002	0.0000001	0.7167871
disco i-esimo:	0.3128002	0.0000000	0.0000000	0.3128002
inLink:	0.3294591	0.0000002	0.0000001	0.3294594
outLink:	0.2996635	0.0000000	0.0000000	0.2996635
cpu web switch:	0.7392626	0.0000002	0.0000001	0.7392628
LAN:	0.2990815	0.0000000	0.0000000	0.2990815
LS1:	0.3144901	0.0000000	0.0000000	0.3144901
LS2:	0.2990815	0.0000000	0.0000000	0.2990815
LW2: 0.0045316 0.0000000 0.0000000	0.0045316			

Tabella 3.9: Utilizzazioni

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	3.9782364	0.0000012	0.0000006	3.9782382
disco i-esimo:	0.6957288	0.0000000	0.0000000	0.6957288
inLink:	1.0129555	0.0000002	0.0000001	1.0129558
outLink:	5.6197781	0.0000000	0.0000000	5.6197781
cpu web switch:	6.0216761	0.0000009	0.0000001	6.0216771
LAN:	2.0022744	0.0000002	0.0000000	2.0022746
LS1:	4.5046757	0.0000000	0.0000000	4.5046757
LS2:	3.6126354	0.0000000	0.0000000	3.6126354
LW2:	0.0051400	0.0000000	0.0000000	0.0051400

Tabella 3.10: Lunghezza Code

Centro	Classe1	Classe2	Classe3
cpu web server i-esimo:	0.0365676	0.0001307	0.0000703
disco i-esimo:	0.1738966	0.0000000	0.0000000
inLink:	0.0002853	0.0000227	0.0000093
outlink:	0.0015856	0.0000000	0.0000000
cpu web switch:	0.0008480	0.0000995	0.0000137
LAN:	0.0002824	0.0000231	0.0000003
LS1:	0.0006352	0.0000008	0.0000004
LS2:	0.0005094	0.0000005	0.0000003
LW2:	0.0000495	0.0000000	0.0000000

Tabella 3.11: Tempo medio di risposta

3.7.4 Risultati Proxy

L'introduzione del proxy, con un cache hit rate del 40%, porta ad un notevole calo delle utilizzazioni, riducendo a meno del 46% l'utilizzazione della risorsa collo di bottiglia del sistema (la CPU del web switch) e della CPU del Web Server. Anche le code tendono a diminuire notevolmente, come era logico aspettarsi. Analogo discorso vale anche per i tempi medi di risposta. Si ricorda che per l'esecuzione della simulazione con l'introduzione del proxy (e con il link addizionale, che sarà presentato nella prossima sezione), è stata adottata la politica random.

3.7.5 Risultati Link Addizionale

L'introduzione del link addizionale comporta una notevole diminuzione dell'utilizzazione della cpu del web switch e della maggior parte delle componenti di rete, come la LAN L2 e le schede di rete LS1, LS2 e LW2, poiché queste hanno il compito di gestire solamente le richieste, mentre le risposte vengono interamente smistate verso la scheda di rete LW3 e il link addizionale. Anche la lunghezza media delle code dello switch e delle componenti di rete tende

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	0.4382355	0.0000001	0.0000000	0.4382356
disco i-esimo:	0.1921759	0.0000000	0.0000000	0.1921759
inLink:	0.2014804	0.0000001	0.0000000	0.2014805
outLink:	0.1842500	0.0000000	0.0000000	0.1842500
cpu web switch:	0.4519287	0.0000001	0.0000000	0.4519288
LAN:	0.1838706	0.0000000	0.0000000	0.1838706
LS1:	0.1933170	0.0000000	0.0000000	0.1933170
LS2:	0.1838711	0.0000000	0.0000000	0.1838711
LW2: 0.0027859 0.0000000 0.0000000	0.0027859			

Tabella 3.12: Utilizzazioni

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	0.6329342	0.0000002	0.0000000	0.6329343
disco i-esimo:	0.9622401	0.0000000	0.0000000	0.9622401
inLink:	0.3370955	0.0000001	0.0000000	0.3370956
outLink:	2.1766564	0.0000000	0.0000000	2.1766564
cpu web switch:	0.9132859	0.0000002	0.0000000	0.9132861
LAN:	0.8057551	0.0000000	0.0000000	0.8057551
LS1:	1.7810221	0.0000057	0.0000000	1.7810278
LS2:	1.3518652	0.0000001	0.0000000	1.3518653
LW2:	0.0029656	0.0000000	0.0000000	0.0029656

Tabella 3.13: Lunghezza Code

Centro	Classe1	Classe2	Classe3
cpu web server i-esimo:	0.0096208	0.0000288	0.0000000
disco i-esimo:	1.0033886	0.0000000	0.0000000
inLink:	0.0001552	0.0000186	0.0000000
outlink:	0.0010055	0.0000000	0.0000000
cpu web switch:	0.0002102	0.0000291	0.0000000
LAN:	0.0001859	0.0000005	0.0000000
LS1:	0.0004110	0.0011045	0.0000000
LS2:	0.0003119	0.0000110	0.0000000
LW2:	0.0000451	0.0000000	0.0000000

Tabella 3.14: Tempo medio di risposta

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	0.6729036	0.0000003	0.0000001	0.6729039
disco i-esimo:	0.2926927	0.0000000	0.0000000	0.2926927
inLink:	0.3096163	0.0000002	0.0000001	0.3096167
outLink:	0.0000000	0.0000000	0.0000000	0.0000000
cpu web switch:	0.3475399	0.0000003	0.0000001	0.3475402
LAN:	0.0090708	0.0000000	0.0000000	0.0090708
LS1:	0.0139327	0.0000000	0.0000000	0.0139327
LS2:	0.0090708	0.0000000	0.0000000	0.0090708
LW2: 0.0001374 0.0000000 0.0000000	0.0001374			
LINK_ADD:	0.4508218	0.0000000	0.0000000	0.4508218
LW3:	0.0068306	0.0000000	0.0000000	0.0068306

Tabella 3.15: Utilizzazioni

Centro	Classe1	Classe2	Classe3	Totale
cpu web server i-esimo:	1.5404288	0.0000006	0.0000001	1.5404295
disco i-esimo:	1.6460958	0.0000006	0.0000000	1.6460963
inLink:	0.4752640	0.0000007	0.0000001	0.4752647
outLink:	0.0000000	0.0000000	0.0000000	0.0000000
cpu web switch:	0.4067200	0.0000004	0.0000001	0.4067205
LAN:	0.0090708	0.0000000	0.0000000	0.0090708
LS1:	0.0139327	0.0000000	0.0000000	0.0139327
LS2:	0.0090708	0.0000000	0.0000000	0.0090708
LW2:	0.0001374	0.0000000	0.0000000	0.0001374
LINK_ADD:	3.5259678	0.0000000	0.0000000	3.5259678
LW3:	0.0071490	0.0000000	0.0000000	0.0071490

Tabella 3.16: Lunghezza Code

a diminuire notevolmente, mentre restano immutate quelle dei web server. Quanto evidenziato in precedenza, risulta essere valido anche per il tempo medio di risposta.

Centro	Classe1	Classe2	Classe3
cpu web server i-esimo:	0.0152177	0.0000734	0.0000101
disco i-esimo:	0.8363018	0.0000000	0.0000000
inLink:	0.0001423	0.0000773	0.0000093
outlink:	0.0000000	0.0000000	0.0000000
cpu web switch:	0.0001219	0.0000428	0.0000104
LAN:	0.0000027	0.0000008	0.0000003
LS1:	0.0000042	0.0000013	0.0000004
LS2:	0.0000027	0.0000008	0.0000003
LW2:	0.0000027	0.0000000	0.0000000
LINK_ADD:	0.0010579	0.0000000	0.0000000
LW3:	0.0001419	0.0000000	0.0000000

Tabella 3.17: Tempo medio di risposta

Capitolo 4

Modello Analitico

Dal modello simulativo è stato possibile ricavare il tasso di arrivo di richeste HTTP, dato fondamentale per poter eseguire un modello analitico. Dalla configurazione standard si è ricavato il valore:

$$\lambda = 3308.79 \frac{richieste}{sec}$$

Le probabilità di arrivo di richieste di classe r, sono date dalla formula:

$$p_r = \frac{count_r}{observation(scrivereperbene)}$$

Di seguito si riportano nuovamente i valori già presentati nel capitolo 2

Classe	Dimensione(byte)	Richieste
1	10281	9999995
2	279513744	4
3	715827882	1

Tabella 4.1: Risultati clustering

Classe	Prob. arrivo richieste classe R
1	0.9999995
2	0.0000004
3	0.0000001

Tabella 4.2: specifiche2

Il tasso di richieste per la singola classe è dato dalla formula:

$$\lambda_r = \lambda * p_r$$

Formule indici di prestazione: (inserire le formule di utilizzazione, lunghezza coda e tempi di residenza tramite un editor, per ora sono scritte a cazzo). Utilizzazione:

$$U_{i,r} = \lambda_r * D_{i,r}$$

$$U_i(vettore) = \sum_{1}^{NUM\_CLASSES} U_{i,r}$$

Lunghezza coda:

$$n_{i,r} = \frac{U_{i,r}}{1 - U_{i,r}}$$

$$n_i(vettore) = \sum_{1}^{NUM\_CLASSES} n_{i,r}$$

Tempo di residenza:

$$R_{i,r} = \frac{D_{i,r}}{1 - U_{i,r}}$$

Per $D_{i,r}$ si intende la domanda di servizio della risorsa i-esima per richieste di classe r-esima. Per quanto concerne le domande di servizio, queste sono le stesse presentate nel modello simulativo, con la differenza che in questo caso, quando è necessario fornire la dimensione del documento, per la classe r bisogna usare il centroide r-esimo. I risultati ottenuti per le componenti del web server dovranno poi essere opportunamente mediati per il numero di server (nel caso delle schede di rete e delle CPU) e per il numero di server moltiplicato per il numero di dischi (nel caso dei dischi dei web server).

Questa assunzione è valida nel caso il carico sia equamente ripartito tra i server e i dischi.

4.1 Presentazione dei risultati

Si presentano di seguito i risultati ottenuti applicando il modello analitico, sia alla rete in configurazione standard che nelle sue varianti.

4.1.1 Risultati standard

Centro	Classe1	Classe2	Classe3	Totale
L2:	0.286753040	0.003004982	0.001923920	0.291681942
inLink:	0.307055735	0.000000123	0.000000031	0.307055889
outLink:	0.287494007	0.003077099	0.001970093	0.292541199
cpu web switch:	0.689330905	0.000000276	0.000000069	0.68933125
cpu web server:	0.668442090	0.000000267	0.000000067	0.668442424
disco web server:	0.323036672	0.002939248	0.001881830	0.32785775
LS1:	0.301311515	0.003077104	0.001970095	0.306358714
LS2:	0.286753040	0.003004982	0.001923920	0.291681942
LW2:	0.004344743	0.000045530	0.000029150	0.004419423

Tabella 4.3: Utilizzazioni

Dalla tabella presentata si può notare che, anche attraverso la risoluzione analitica, i vincoli sulle utilizzazioni sono rispettati e che sono molto simili a quelli presentati nel modello simulativo, se si escludono una differenza di circa il 3% per quanto riguarda l'utilizzazione dei dischi. Come era logico aspettarsi, data la natura della formula che rappresenta la lunghezza della coda, le risorse più utilizzate sono quelle che presentano i valori più alti. In questa tabella si notano immediatamente i valori spropositati dei tempi di residenza dei dischi del web server nel caso di richieste di Classe 2 e 3.

Centro	Classe1	Classe2	Classe3	Totale
L2:	0.402038923	0.003014039	0.001927629	0.406980591
inLink:	0.443117507	0.000000123	0.000000031	0.443117661
outLink:	0.403496967	0.003086597	0.001973982	0.408557546
cpu web switch:	2.218858963	0.000000276	0.000000069	2.218859308
cpu web server:	2.016064373	0.000000267	0.000000067	2.016064707
disco web server:	0.477184891	0.002947912	0.001885377	0.48201818
LS1:	0.431253014	0.003086602	0.001973984	0.4363136
LS2:	0.402038923	0.003014039	0.001927629	0.406980591
LW2:	0.004363702	0.000045532	0.000029151	0.004438385

Tabella 4.4: Lunghezza Code

Centro	Classe1	Classe2	Classe3
L2:	0.000117695	2.277294228	5.825778128
inLink:	0.000133921	0.000092800	0.000092800
outLink:	0.000121947	2.332118913	5.965873728
cpu web switch:	0.000670596	0.000208333	0.000208333
cpu web server:	0.040214192	0.013333337	0.013333334
disco web server:	0.114220190	1764.048526653	4512.885306683
LS1:	0.000130336	2.332123114	5.965877921
LS2:	0.000117695	2.277294228	5.825778128
LW2:	0.000084312	2.270554378	5.814739298

Tabella 4.5: Tempo di residenza

Questi valori sono dovuti al fatto che i file appartenenti a queste due classi sono dell'ordine di centinaia di megabyte e dunque le operazioni di lettura dal disco richiedono svariati minuti per essere completate (si ricorda al lettore che i centroidi di classe 2 e 3 hanno una dimensione rispettivamente di 279 MB e 715 MB).

4.1.2 Risultati proxy

Centro	Classe1	Classe2	Classe3	Totale
L2:	0.172051824	0.001802989	0.001154352	0.175009165
inLink:	0.184233441	0.000000074	0.000000018	0.184233533
outLink:	0.172496404	0.001846259	0.001182056	0.175524719
cpu web switch:	0.413598543	0.000000165	0.000000041	0.413598749
cpu web server:	0.401065254	0.000000160	0.000000040	0.401065454
disco web server:	0.193822003	0.001763549	0.001129098	0.19671465
LS1:	0.180786909	0.001846263	0.001182057	0.183815229
LS2:	0.172051824	0.001802989	0.001154352	0.175009165
LW2:	0.002606846	0.000027318	0.000017490	0.002651654

Tabella 4.6: Utilizzazioni

Le utilizzazioni, nel caso dell'introduzione del proxy server, tendono a diminuire notevolmente, in modo particolare se si considerano le CPU del Web Switch e dei Web Server. Anche in questo caso, i risultati sono molto simili a quelli ottenuti col modello simulativo. L'introduzione del proxy comporta anche un minore accodamento delle richieste. Si può notare infatti come ogni centro presenti delle code inferiori ad uno. L'introduzione del proxy server non risolve il problema della gestione di file di Classe 2 e 3, infatti rispetto al caso standard le differenze tra i tempi di residenza sono minime.

Centro	Classe1	Classe2	Classe3	Totale
L2:	0.207805064	0.001806246	0.001155686	0.210766996
inLink:	0.225840884	0.000000074	0.000000018	0.225840976
outLink:	0.208453963	0.001849674	0.001183455	0.211487092
cpu web switch:	0.705316364	0.000000165	0.000000041	0.70531657
cpu web server:	0.669630968	0.000000160	0.000000040	0.669631168
disco web server:	0.240420855	0.001766664	0.001130374	0.243317893
LS1:	0.220683618	0.001849678	0.001183456	0.223716752
LS2:	0.207805064	0.001806246	0.001155686	0.210766996
LW2:	0.002613659	0.000027319	0.000017490	0.002658468

Tabella 4.7: Lunghezza Code

Centro	Classe1	Classe2	Classe3
L2:	0.000101390	2.274551992	5.821289614
inLink:	0.000113758	0.000092800	0.000092800
outLink:	0.000105000	2.329243139	5.961166833
cpu web switch:	0.000355274	0.000208333	0.000208333
cpu web server:	0.022261746	0.013333335	0.013333334
disco web server:	0.095912913	1761.970872492	4509.484474463
LS1:	0.000111160	2.329247330	5.961171019
LS2:	0.000101390	2.274551992	5.821289614
LW2:	0.000084165	2.270513026	5.814671497

Tabella 4.8: Tempo di residenza

4.1.3 Risultati Link Addizionale

Centro	Classe1	Classe2	Classe3	Totale
L2:	0.008995768	0.000000004	0.000000001	0.008995773
inLink:	0.307055735	0.000000123	0.000000031	0,307055889
cpu web switch:	0.344665453	0.000000138	0.000000034	0.344665625
cpu web server:	0.668442090	0.000000267	0.000000067	0.668442424
disco web server:	0.323036672	0.002939248	0.001881830	0.32785775
linkAdd:	0.462209015	0.004947104	0.003167353	0.470323472
LS1:	0.013817508	0.000000006	0.000000001	0.013817515
LS2:	0.008995768	0.000000004	0.000000001	0.008995773
LW2:	0.000136300	0.000000000	0.000000000	0.0001363
LW3:	0.007003167	0.000074956	0.000047990	0.007126113

Tabella 4.9: Utilizzazioni

Come già evidenziato nel modello simulativo, l'introduzione del link addizionale permette di ridurre notevolmente le utilizzazioni del web switch e delle componenti di rete "interne" (LS1, LS2, L2, LW2). In questo caso la risorsa collo di bottiglia diventa la CPU del Web Server. Anche in quest'ultimo caso, le differenze rispetto alle utilizzazioni ottenuti col modello simulativo sono minime. Le code risultano essere tutte minori di uno, ad eccezione della CPU del Web Server che presenta un valore pari a 2 a causa dell'alta utilizzazione. Le considerazioni fatte sui tempi di residenza nei due casi precedenti sono valide anche in quest'ultimo caso.

Centro	Classe1	Classe2	Classe3	Totale
L2:	0.009077427	0.000000004	0.000000001	0.009077432
inLink:	0.443117507	0.000000123	0.000000031	0.443117661
cpu web switch:	0.525938170	0.000000138	0.000000034	0.525938342
cpu web server:	2.016064373	0.000000267	0.000000067	2.016064707
disco web server:	0.477184891	0.002947912	0.001885377	0.48201818
linkAdd:	0.859458465	0.004971700	0.003177417	0.867607582
LS1:	0.014011107	0.000000006	0.000000001	0.014011114
LS2:	0.009077427	0.000000004	0.000000001	0.009077432
LW2:	0.000136318	0.000000000	0.000000000	0.000136318
LW3:	0.007052557	0.000074962	0.000047992	0.007175511

Tabella 4.10: Lunghezza Code

Centro	Classe1	Classe2	Classe3
L2:	0.000002743	0.000002719	0.000002719
inLink:	0.000133921	0.000092800	0.000092800
cpu web switch:	0.000317904	0.000208333	0.000208333
cpu web server:	0.040214192	0.013333337	0.013333334
disco web server:	0.114220190	1764.048526653	4512.885306683
linkAdd:	0.000259750	3.756433553	9.602956788
LS1:	0.000004235	0.000004176	0.000004176
LS2:	0.000002743	0.000002719	0.000002719
LW2:	0.000002719	0.000002719	0.000002719
LW3:	0.000140677	3.738130279	9.573000246

Tabella 4.11: Tempo di residenza

Capitolo 5

Conclusioni

Sulla base dei risultati ottenuti e sui problemi riscontrati durante lo sviluppo, si vogliono effettuare alcune considerazioni. Per quanto riguarda il modello simulativo, l'adozione di una politica round robin piuttosto che random non comporta alcuna differenza significativa, mentre l'adozione di una politica di tipo least loaded comporta un aumento dell'utilizzazione (e quindi del carico sottoposto al sistema), mentre tendono a diminuire i tempi medi di risposta. L'introduzione di componenti addizionali (il proxy e il link addizionale) tendono a far diminuire le utilizzazioni e le code del sistema, ma non comportano una diminuzione dei tempi di risposta. Per quanto concerne il modello analitico invece, si nota come i valori delle utilizzazioni siano molto simili a quelli ottenuti col modello simulativo (le differenze maggiori sono intorno al 4% e solo nel caso dei dischi), mentre le lunghezze delle code risultano essere piuttosto diverse tra i due modelli (questo potrebbe essere dovuto al fatto che si assume che il carico sia equamente ripartito tra i server e di conseguenza tra i dischi). I tempi medi di residenza inoltre evidenziano come i dischi siano sottoposti a operazioni molto onerose in caso di richieste di Classe 2 e 3. Nel modello simulativo invece le richieste di Classe 2 e 3 sono molto rare e infatti i valori presentati per le richieste di Classe 1 sono sempre più alti rispetto alle restanti classi. Di conseguenza, il modello simulativo riflette meglio la distribuzione non uniforme delle richieste. Per

quanto concerne i problemi riscontrati durante lo sviluppo dell'applicativo, è doveroso sottolineare come l'utilizzo delle distribuzioni per la generazione delle pagine HTML e degli oggetti embedded abbia portato ad ottenere file di dimensioni di centinaia di megabyte, che hanno portato a tempi medi di residenza spropositati (si vedano i risultati nel modello analitico). Il problema di avere file di queste dimensioni si riflette sicuramente anche nel calcolo del transiente, che come si è visto non consente di trovare un valore tale per cui la media mobile sembra convergere, ma si assiste ad un andamento crescente dei tempi di risposta, come se il sistema fosse ancora in fase di carica. Questo problema si è poi ripercosso anche sul calcolo degli indici nel modello simulativo (e di conseguenza in quello analitico, visto che il fattore lambda è stato ricavato dalla simulazione), poiché sono state escluse solo le prime 100000 osservazioni dalla simulazione (numero non sufficiente per superare il transiente) e di conseguenza i valori ottenuti potrebbero non essere sufficientemente accurati. L'utilizzo di un carico reale, utilizzato nell'esercizio 2 d'esame, avrebbe sicuramente permesso di ottenere dei risultati più affidabili. Una possibile soluzione a questi problemi, potrebbe essere rappresentata dall'utilizzo di un valore di soglia che impedisce di ritenere valide delle richieste superiori a tali soglia, ma che tuttavia non impedisca di avere una distribuzione heavy-tailed (??? non mi piace molto come conclusione né tanto meno come soluzione).

Capitolo 6

Appendice

6.1 common.h

```
#ifndef _COMMON_H
#define _COMMON_H

#include <csim.h>

#define _ISOC99_SOURCE

#define BLOCK_SIZE 2048 //dimensione in byte
#define MBYTE 1048576  //equivalente di un mega espresso in byte
#define ARRIVAL 150 //richieste/sec  da parte degli utenti
#define CPU_SERVICE_RATE 150 //richieste/secondo tasso servizio CPU
#define SEEK_TIME_DISK 8.5 //millisecondi
#define CONTROLLER_TIME 0.2 //millisecondi
#define ROTATIONAL_SPEED 7200 //RPM
#define TRANSFER_RATE 100 //MB/sec
#define BANDWIDTH_L2 1024 // Mbps
#define CPU_WEB_SWITCH_SERVICE_RATE 9600 //richieste/secondo tasso servizio
    CPU switch
#define NUM_SERVER 66
#define NUM_DISK 12
#define BANDWIDTH_LINKADD 622 //Mbps
#define INLINK_BANDWIDTH 45 //Mbps
#define OUTLINK_BANDWIDTH 1000 //Mbps
#define LS1_TRANSFER_RATE 1000 //Mbps
#define P_HIT 0.4 //probabilita' che la risorsa richiesta sia conservata nel
     proxy
```

```
#define MAX_SERVERS 5000
#define MAX_PROCESSES 100000000
#define MAX_FACILITIES 5000
#define MAX_CLASSES 10
#define MAX_OBSERVATION 400000
#define NUM_CLASSES 3 //uguale a K

#define K 3//numero di cluster
#define NUM_ITERATIONS 10
#define SEED 3

//definizione delle varianti
#define RANDOM 0
#define ROUND_ROBIN 1
#define LEAST_LOADED 2
#define LINK_ADD 3
#define PROXY 4

//variabili CSIM
FACILITY cpuWS[NUM_SERVER];
FACILITY diskWS[NUM_DISK*NUM_SERVER];
BOX WebServer;
BOX WebSwitch;
FACILITY L2;
FACILITY CPU_web_switch;
FACILITY inLink;
FACILITY outLink;
FACILITY link_add;
FACILITY LS1;
FACILITY LS2;
FACILITY LW2[NUM_SERVER];
FACILITY LW3[NUM_SERVER];
TABLE wsrtime;
TABLE rtime;
METER lambda;
CLASS requestClasses[K];

#endif
```

6.2 cluster.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdarg.h>
#include<string.h>
#include<csim.h>
#include "gaussiana_inversa.h"
#include "client.h"

extern double mu_session;
extern double lambda_session;
extern double alfa_tt;
extern double alfa_obj;
extern double mu_html;
extern double sigma_html;
extern double alfa_html;
extern double mu_emb;
extern double sigma_emb;

extern STREAM sess_req_1;
extern STREAM sess_req_2;
extern STREAM user_tt;
extern STREAM object_req;
extern STREAM html_1;
extern STREAM html_2;
extern STREAM obj_size;

//Algoritmo di clustering K-Means
void cluster(double *array, int array_length, int k)
{
        int i= 0;
        int j = 0;
        int iter = 0;
        int counter = 0;
        int converged = 0;
        double distance = 0.0;
        double distance_temp = 0.0;
        double min = array[0];
        double max = array[0];
        double *centroids = (double*)malloc(sizeof(double)*k) ;
        double *old_centroids = (double*)malloc(sizeof(double)*k);
        int *num_data_per_cluster = (int*)malloc(sizeof(long)*k);
        double *total_data_per_cluster = (double*)malloc(sizeof(double)*k);
```

```
//calcolo del minimo e del massimo

for(i=1;i < array_length; i++) {
        if(array[i] > max)
                max = array[i];
        if(array[i] < min)
                min = array[i];
}
printf("k %d:\n", k);
//assegnazione dei primi due centroidi
centroids[0] = min;
printf("centroide min %lf\n", centroids[0]);
centroids[k-1] = max;
printf("centroide max %lf\n", centroids[k-1]);
//calcolo degli altri centroidi in caso k>2
/***
if(k>2) {
        for(i=1; i < k-1; i++) {
                centroids[i] = min+ i*(max-min)/k;
                printf("new %lf\n", centroids[i]);
        }
}
for(i=0; i < k; i++)
 printf("centroide[%d] = %lf\n", i, centroids[i]);
*****************************************/
////PROVA CLUSTER RANDOM
for(i=1; i < k-1; i++)
{
        int rand = csim_random_int(0,array_length);
        centroids[i] = array[rand];

}
/*** FINE PROVA CLUSTER RANDOM*/
while(!iter) {

        //assegnazione dei dati ad ogni cluster
        for(i=0; i < array_length; i++) {
                distance = fabs(centroids[0] - array[i]);

                for(j=1; j < k; j++) {

                        distance_temp = fabs(centroids[j]-array[i]);
                        if(distance > distance_temp) {
                                distance = distance_temp;
                                counter = j;
                                //printf("counter: %d\n", counter);
                        }
```

```
                        }

                        num_data_per_cluster[counter] += 1;
                        total_data_per_cluster[counter] += array[i];
                        distance = 0.0;
                        j = 0;
                        counter = 0;
                }
                //step per tenere memoria dell'iterazione precedente
                for(i=0; i < k; i++) {
                        old_centroids[i] = centroids[i];
                }
                printf("\n");
                //aggiornamento dei centroidi in seguito al primo step e
                    confronto con la precedente iterazione
                for(i=0; i < k; i++) {
                        centroids[i] = total_data_per_cluster[i] / ((double)
                            num_data_per_cluster[i]);
                        printf("i: %d,  centroide: %lf - tot_dati: %d\t", i,
                             centroids[i], num_data_per_cluster[i]);
                        if( fabs(centroids[i]-old_centroids[i]) < pow
                            (10,-10) ) {  //valore epsilon = 10^-10
                                printf("converged: %d\n", converged);
                                converged++;
                        }
                }
                //controllo se l'algoritmo converge
                if(converged == k) {
                        iter = 1;
                }
                converged = 0;
                memset(num_data_per_cluster,0,k*sizeof(double));
                memset(total_data_per_cluster,0,k*sizeof(double));
        }
}


//Simulazione per la generazione dei cluster di documenti.
void sim(int argc, char **argv)
{
        //creazione e reseeding degli stream
        sess_req_1 = create_stream();
        reseed(sess_req_1, SEED);
        sess_req_2 = create_stream();
        reseed(sess_req_2, SEED);
        user_tt = create_stream();
        reseed(user_tt, SEED);
```

```
object_req = create_stream();
reseed(object_req, SEED);
html_1 = create_stream();
reseed(html_1, SEED);
html_2 = create_stream();
reseed(html_2, SEED);
obj_size = create_stream();
reseed(obj_size, SEED);

create("prova");
int array_length= 10000000;
double *array = (double*)malloc(sizeof(double)*array_length);
double session;
int objects;
int i=0;
int j=0;
int k=0;
//generazione delle sessioni e delle relative richieste
while(i<array_length) {
        session = session_request(mu_session, lambda_session);
        for(j=0; j < session; j++) {
                if( i < array_length) {
                        array[i] = html_page_size(mu_html,
                            sigma_html, alfa_html);
                }
                i++;
                objects = object_per_request(alfa_obj);
                for(k=0; k<objects; k++){
                        if(i<array_length)
                                array[i] = embedded_object_size(
                                    mu_emb, sigma_emb);
                        i++;
                }
        }
}
//clustering dei documenti
cluster(array,array_length,5);

int m=0;
/*FILE *fd_file;
char *pathname = (char*)malloc(128);
sprintf(pathname, "docs");
fd_file = fopen(pathname, "w");
for(m=0;m<array_length;m++) {
        fprintf(fd_file, "%g\n", array[m]);
}
fclose(fd_file);
```

```
*/

}
```

6.3 gaussiana_inversa.c

```
#include "gaussiana_inversa.h"

//dichiarazione degli stream
STREAM sess_req_1;
STREAM sess_req_2;
STREAM user_tt;
STREAM object_req;
STREAM html_1;
STREAM html_2;
STREAM obj_size;
STREAM p_hit_proxy;

//calcolo della media per una distribuzione lognormale
double calc_mean_lognormal(double mu, double sigma)
{
        double sum = mu+(pow(sigma,2)/2);
        return exp(sum);
}

//calcolo della deviazione standard per una distribuzione lognormale
double calc_stddev_lognormal(double mu, double sigma)
{
        double sum = (exp(pow(sigma,2))-1)*exp(2*mu+pow(sigma,2));
        return sqrt(sum);
}

//algoritmo di Michael/Schucany/Haas per la generazione di valori tramite la
     gaussiana inversa
int session_request(double mu, double lambda)   //dimensionare come un
    intero???
{
        int x;
        double v = stream_normal(sess_req_1,0,1);
        double y = pow(v,2);
        double x1 = 0;
        x1 = mu + ((pow(mu,2)*y)/(2*lambda)) - ((mu/(2*lambda)) * sqrt(4*mu*
            lambda*y+pow(mu,2)*pow(y,2)));
        double u = stream_uniform(sess_req_2,0,1);
        double temp = mu/(mu+x1);
        if(u <= temp) {
                x = (int)round(x1);
        }
        else
                x = (int)round(pow(mu,2)/x1);
```

```
        return x;
}

//generazione di valori per il think time dell'utente (Pareto)
double user_think_time(double alfa)
{
        double x = 0.0;
        while(x < 1) {
                x = stream_pareto(user_tt, alfa);
        }
        return x;
}

//generazione del numero di oggetti per richiesta (Pareto)
int object_per_request(double alfa)
{
        int x = 0;
        while(x < 2) {
                x = (int)round(stream_pareto(object_req, alfa));
        }
        return x;
}

//generazione della dimensione della pagina html (Pareto e Lognormale)
double html_page_size(double mu, double sigma, double alfa)
{
        double x = 0.0;
        int k = 10240; //dimensionarlo in KB???
        x = stream_lognormal(html_1, calc_mean_lognormal(mu,sigma),
            calc_stddev_lognormal(mu,sigma));
        if(x<k)
                return x;
        else {
                x = stream_pareto(html_2, alfa);
                while(x < k) {
                        x = stream_pareto(html_2, alfa);
                }
        }
        return x;
}

//generazione della dimensione degli embedded object (Lognormale)
double embedded_object_size(double mu, double sigma)
{
        double x = 0.0;
        while(x <= 0.0) {
```

```
                x = stream_lognormal(obj_size, calc_mean_lognormal(mu,sigma)
                    ,calc_stddev_lognormal(mu,sigma));
        }
        return x;
}
```

6.4 service.c

```
#include "service.h"

/**
 * In questo file sono contenute tutte le domande di servizio per i
     componenti della rete.
 * Le domande di servizio delle schede di rete LS2, LW2 e LW3 non sono state
      implementate
 * poiche' si e' assunto che queste presentino un transfer rate pari alla
     banda delle LAN a
 * cui sono collegate.
 */

// Divide il numero di Byte m in ingresso in datagrammi
int NDatagrams(double m) {
        int n;
        double f;
        f=m/(double)MSS;          //MSS in Byte
        n=(int)ceil(f);
        return n;
}

// Calcola l'overhead della connessione TCP/IP e del livello DataLink
int Overhead(double m) {
        return (NDatagrams(m)*(TCPOV+IPOV+FRAMEOV)); //[Byte]
}

// Calcola il numero di blocchi da leggere per un documento
int number_of_blocks(double docSize) {
        return ceil(docSize/BLOCK_SIZE);
}

/**
 * Calcola il tempo necessario per trasferire m Byte (sec)
 * m in byte, bandwidth in bit
 */
double NetworkTime(double m, double bandwidth) {
        return (double)(8*(m+Overhead(m)))/(double)(1000*1000*bandwidth); //
            [sec]
}

// Domanda di servizio (sec) per l'incoming link
double D_InLink() {
        return NetworkTime(AVG_SIZE_HTTP_REQ, INLINK_BANDWIDTH) + 3*
            NetworkTime(0.0001, INLINK_BANDWIDTH);
}
```

```
//domanda di servizio per l'outgoing link
double D_OutLink(double docSize) {
        return NetworkTime(docSize, OUTLINK_BANDWIDTH) + 2 * NetworkTime
            (0.0001, OUTLINK_BANDWIDTH);
}

/**
 * Calcola la domanda di servizio sottomessa alla LAN che collega il Web
     Switch con i Web Server.
.* Il fattore moltiplicativo 1024 e' commentato perche' le richieste sono
    gia' espressi in byte
 */
double D_LAN(double docSize) {
        if(docSize == 0) {
                return NetworkTime(AVG_SIZE_HTTP_REQ, BANDWIDTH_L2);
        }

        return NetworkTime(/*1024 **/ docSize, BANDWIDTH_L2);

}

// Calcola la domanda di servizio sottomessa ad una CPU (sia Web Switch che
    web server) (sec),
double D_Cpu(double speed) {
        return 1 / speed;
}

// Calcola la domanda di servizio sottomessa ad un disco del Web Server (sec
    )
double D_WSDisk(double doc_size) {
        double ret = (number_of_blocks(doc_size)) * ((DISK_SEEK_TIME/pow
            (10,3)) + ROTATIONAL_LATENCY + (CONTROLLER_TIME/pow(10,3)) + (
            BLOCK_SIZE/((double)DISK_TRANSFER_RATE*pow(10,6))));
        return ret;
}

//calcola la domanda di servizio del link addizionale
double D_linkAdd(double doc_size) {
        return NetworkTime(doc_size, BANDWIDTH_LINKADD) + 2 * NetworkTime
            (0.0001, BANDWIDTH_LINKADD);
}

//calcola la domanda di servizio in entrata della scheda di rete LS1
double D_LS1in()
{
```

```
        return NetworkTime(AVG_SIZE_HTTP_REQ, LS1_TRANSFER_RATE) + 3*
            NetworkTime(0.0001, LS1_TRANSFER_RATE);
}

//calcola la domanda di servizio in uscita della scheda di rete LS1
double D_LS1out(double doc_size)
{
  return NetworkTime(doc_size, LS1_TRANSFER_RATE) + 2 * NetworkTime(0.0001,
      LS1_TRANSFER_RATE);
}
```

6.5 client.c

```
#include "client.h"

double mu_session = 3.86;
double lambda_session = 9.46;
double alfa_tt = 1.4;
double alfa_obj = 1.33;
double mu_html = 7.63;
double sigma_html = 1.001;
double alfa_html = 1;
double mu_emb = 8.215;
double sigma_emb = 1.46;

extern FACILITY cpuWS[NUM_SERVER];
extern FACILITY diskWS[NUM_DISK*NUM_SERVER];
extern BOX WebServer;
extern BOX WebSwitch;
extern FACILITY L2;
extern FACILITY CPU_web_switch;
extern FACILITY inLink;
extern FACILITY outLink;
extern FACILITY link_add;
extern FACILITY LS1;
extern FACILITY LS2;
extern FACILITY LW2[NUM_SERVER];
extern FACILITY LW3[NUM_SERVER];
extern TABLE rtime;
extern METER lambda;
extern CLASS requestClasses[K];

extern STREAM p_hit_proxy;

int currentDisk[NUM_SERVER];
int num_osservazioni;
int current_server;
double **observations;
int observed_sample;
int maxObservation;

double client_response_time;

// Ritorna l'indice del server meno utilizzato (serve per la least loaded)
int get_least_loaded()
{
        int i=0;
        int j=0;
```

```
        double disk_qlen = 0.0;
        double server_qlen = 0.0;
        double qlen_tmp = 0.0;
        int index = 0;
        for(i=0; i < NUM_DISK; i++)
                disk_qlen += qlength(diskWS[i]);

        server_qlen = qlength(cpuWS[0])+disk_qlen+qlength(LW2[0]);
        for(i=1; i<NUM_SERVER; i++) {
                disk_qlen = 0.0;
                for(j=0; j < NUM_DISK; j++) {
                        disk_qlen += qlength(diskWS[j+i*NUM_DISK]); //i*
                            NUM_DISK perche' per ogni server ci sono
                            NUM_DISK dischi
                }
                qlen_tmp = disk_qlen+qlength(cpuWS[i])+qlength(LW2[i]);
                if(server_qlen > qlen_tmp) {
                        server_qlen = qlen_tmp;
                        index = i;
                }
                qlen_tmp = 0.0;
        }
        return index;
}

//Gestisce l'intero flusso richiesta-risposta
int web_client(double doc_size, int variant, int bool_transient, int iter)
{
        double startTime;
        double server_start_time = 0.0;
        double switch_start_time = 0.0;
        int tmp_server;
        startTime = simtime();
        int tmp_disk = 0;
        //vedi pag. 131 user guide
        note_passage(lambda);
        use(inLink, D_InLink());

        switch_start_time = enter_box(WebSwitch);
        use(LS1, D_LS1in());
        use(CPU_web_switch, D_Cpu(CPU_WEB_SWITCH_SERVICE_RATE));
  use(LS2, D_LAN(0)); //stessa banda della LAN, in richiesta doc_size=0
  exit_box(WebSwitch, switch_start_time);

        use(L2, D_LAN(0));
```

```
/* random */
if(variant == RANDOM || variant == LINK_ADD || variant == PROXY) {
        tmp_server = current_server;
        current_server = csim_random_int(0, NUM_SERVER-1);
}
else if (variant == ROUND_ROBIN) { //round robin
        tmp_server = current_server;
        current_server = (tmp_server+1)%NUM_SERVER;
}
// least loaded
else if (variant == LEAST_LOADED) {
        tmp_server = get_least_loaded();
}

server_start_time = enter_box(WebServer);
use(LW2[tmp_server], D_LAN(0));
use(cpuWS[tmp_server], D_Cpu(CPU_SERVICE_RATE));


//selezione del disco round robin
tmp_disk = currentDisk[tmp_server];
currentDisk[tmp_server] = (tmp_disk+1)%NUM_DISK;
use(diskWS[tmp_server*NUM_DISK + tmp_disk], D_WSDisk(doc_size));
use(cpuWS[tmp_server], D_Cpu(CPU_SERVICE_RATE));
if(variant != LINK_ADD) {
        use(LW2[tmp_server], D_LAN(doc_size));
}
else {
        use(LW3[tmp_server], D_linkAdd(doc_size));
}
exit_box(WebServer, server_start_time);
if(variant == LINK_ADD) {
        use(link_add, D_linkAdd(doc_size));
}
else {
        use(L2, D_LAN(doc_size));
        use(LS2, D_LAN(doc_size));
        use(CPU_web_switch, D_Cpu(CPU_WEB_SWITCH_SERVICE_RATE));
        use(LS1, D_LS1out(doc_size));
        use(outLink, D_OutLink(doc_size));
}

client_response_time = simtime()-startTime;

tabulate(rtime, simtime()-startTime);
num_osservazioni++;
```

```
        //utilizzato per il transiente per tener conto del numero di
            osservazioni
        if(bool_transient == 1 && observed_sample<=maxObservation){
                observations[iter][observed_sample] = simtime()-startTime;
                observed_sample++;
        }

        return 0;
}

//Generazione di una sessione e delle relative richieste
void web_session(int cli_id, int variant, int bool_transient, int iter)
{
        char *prova = (char*)malloc(64); //il nome del processo
        sprintf(prova,"%d",cli_id);
        create(prova);
        double html_page, emb_obj_size;
        int num_embedded_objects;
        int session = session_request(mu_session, lambda_session);
        int i = 0;
        int j = 0;
        for(i=0; i < session; i++) {
                html_page = html_page_size(mu_html, sigma_html, alfa_html);

                set_process_class(requestClasses[get_doc_class(html_page)]);
                if(variant == PROXY && stream_prob(p_hit_proxy) > 0.4) {
                        web_client(html_page, variant, bool_transient, iter)
                            ;
                }
                if(variant != PROXY) {
                        web_client(html_page, variant, bool_transient, iter)
                            ;
                }
                num_embedded_objects = object_per_request(alfa_obj);
                for(j=0; j < num_embedded_objects; j++) {
                        emb_obj_size = embedded_object_size(mu_emb,
                            sigma_emb);
                        set_process_class(requestClasses[get_doc_class(
                            emb_obj_size)]);
                        if(variant == PROXY && stream_prob(p_hit_proxy) >
                            0.4) {
                                web_client(emb_obj_size, variant,
                                    bool_transient, iter);
                        }
                        if(variant != PROXY) {
                                web_client(emb_obj_size, variant,
                                    bool_transient, iter);
```

```
                        }
                }
                hold(user_think_time(alfa_tt));

        }

        csim_terminate();
}

//assegna una classe ad un documento, sulla base delle distanze da i
    centroidi
int get_doc_class(double doc_size)
{
        double distance[K];
        int i = 0;
        double min;
        int index = 0;
        distance[0] = fabs(10281-doc_size);
        distance[1] = fabs(279513744-doc_size);
        distance[2] = fabs(715827882-doc_size);
        /*distance[0] = fabs(10073-doc_size);
        distance[1] = fabs(17740962-doc_size);
        distance[2] = fabs(70926601-doc_size);
        distance[3] = fabs(279513744-doc_size);
        distance[4] = fabs(715827882-doc_size);
        */
        min = distance[0];
        for(i=1; i < K; i++) {
                if(distance[i] < min) {
                        min = distance[i];
                        index = i;
                }
        }
        return index;
}
```

6.6 transient.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<csim.h>
#include "gaussiana_inversa.h"
#include "client.h"
#include "common.h"

extern FACILITY cpuWS[NUM_SERVER];
extern FACILITY diskWS[NUM_DISK*NUM_SERVER];
extern BOX WebServer;
extern BOX WebSwitch;
extern FACILITY L2;
extern FACILITY CPU_web_switch;
extern FACILITY inLink;
extern FACILITY outLink;
extern FACILITY link_add;
extern FACILITY LS1;
extern FACILITY LS2;
extern FACILITY LW2[NUM_SERVER];
extern TABLE rtime;
extern METER lambda;
extern CLASS requestClasses[K];

extern STREAM sess_req_1;
extern STREAM sess_req_2;
extern STREAM user_tt;
extern STREAM object_req;
extern STREAM html_1;
extern STREAM html_2;
extern STREAM obj_size;
extern STREAM p_hit_proxy;

extern double client_response_time;
extern double **observations;
extern int observed_sample;
extern int maxObservation;

TABLE resptime;

int WELCH_N = 0, WELCH_M = 0, WELCH_W = 0;

char * output_file_name;

//Parsa i parametri passati da linea di comando
```

```
void parse_command_line(int argc, char *argv[]){

        if(argc != 5){
                printf("ERRORE!!!\nUtilizzo: ");
                printf("./transient <WELCH_N> <WELCH_M> <WELCH_W> <
                    file_output>\n");
                exit(1);
        }

        WELCH_N = atoi(argv[1]);
        WELCH_M = atoi(argv[2]);
        WELCH_W = atoi(argv[3]);
        output_file_name = argv[4];

        if(WELCH_W > WELCH_M/2){
                printf("ERRORE!!!\nIl valore della finestra W deve essere
                    minore della parte intera di M/2.\n\n");
                exit(1);
        }


        return;

}

//Simulazione del transiente
void sim(int argc, char *argv[]){

int clientID=0,
         cont=0,
         i,s,
         n_repl=0,
         m_repl=0;

         parse_command_line(argc, argv);

double intT,
                *averaged_process,
                sum=0.0,
                *moving_average;

FILE * mov_avg_fd;
        maxObservation = WELCH_M;
        observed_sample = 1;
        create("sim");
```

```
max_processes(MAX_PROCESSES);    //numero massimo dei processi in
    giro nella rete
max_facilities(MAX_FACILITIES);
max_servers(MAX_SERVERS);
inLink = facility("inLink");
outLink = facility("outLink");
LS1 = facility("LS1");
LS2 = facility("LS2");
CPU_web_switch = facility("CPU_web_switch");
L2 = facility("L2");
facility_set(cpuWS, "cpuWS", NUM_SERVER);
facility_set(diskWS, "diskWS", NUM_SERVER*NUM_DISK);
facility_set(LW2, "LW2", NUM_SERVER);
resptime = table("System Response Time"); // table intialization

char className[20];
WebSwitch = box("Web Switch");
WebServer = box("Web Server");

lambda = meter("Arrival Rate");
max_classes(MAX_CLASSES);        //viene assegnato il numero massimo
    di classi per evitare l'errore "TOO MANY PROCESSES CLASS"
for(cont=0; cont<NUM_CLASSES; cont++) {
        className[0] = '\0';
        sprintf(className, "Classe%d", cont);
        requestClasses[cont] = process_class(className);
}
// Inizializzazione degli stream (reseed simtime*i+num)
sess_req_1 = create_stream();
reseed(sess_req_1, (int)simtime()+i+1);
sess_req_2 = create_stream();
reseed(sess_req_2, (int)simtime()*2+i+1);
user_tt = create_stream();
reseed(user_tt, (int)simtime()*3+i+1);
object_req = create_stream();
reseed(object_req, (int)simtime()*4+i+1);
html_1 = create_stream();
reseed(html_1, (int)simtime()*5+i+1);
html_2 = create_stream();
reseed(html_2, (int)simtime()*6+i+1);
obj_size = create_stream();
reseed(obj_size, (int)simtime()*7+i+1);
p_hit_proxy = create_stream();
reseed(p_hit_proxy, (int)simtime()*8+i+1);

collect_class_facility_all();
```

```
rtime=table("Response Time");

averaged_process = malloc((WELCH_M +1)*sizeof(double));
moving_average=malloc((WELCH_M - WELCH_W +1)*sizeof(double));
observations = (double**)malloc(WELCH_N * sizeof(double*));
for(i=0; i<WELCH_N; i++){
        observations[i] = (double*)malloc(WELCH_M * sizeof(double));
}

//PASSO 1: vengono effettuate WELCH_N repliche di lunghezza WELCH_M
    ognuna. I risultati
//                          sono inseriti nella matrice sample_matrix.
//                Il tempo di risposta percepito dal client viene
    campionato ogni SAMPLE_TIME secondi
//            Il generatore di numeri casuali non viene mai azzerato

//ciclo di generazione delle richieste


printf("n_repl =%d, m_repl =%d\n", n_repl, m_repl);
while(n_repl < WELCH_N) {
        observed_sample = 1;
        while(observed_sample < WELCH_M) {

                intT=exponential(1/(double)(ARRIVAL));
                hold(intT);             //think time
                web_session(clientID, RANDOM, 1, n_repl);
                clientID++;
        }
        printf("Replication n %d terminated\n",n_repl);
        //ogni volta che viene terminata una replica e' necessario
        //riavviare le risorse
        wait(event_list_empty);
        reset();
        reseed(sess_req_1, (int)simtime()+i+1);
        reseed(sess_req_2, (int)simtime()*2+i+1);
        reseed(user_tt, (int)simtime()*3+i+1);
        reseed(object_req, (int)simtime()*4+i+1);

        reseed(html_1, (int)simtime()*5+i+1);
        reseed(html_2, (int)simtime()*6+i+1);
        reseed(obj_size, (int)simtime()*7+i+1);
        reseed(p_hit_proxy, (int)simtime()*8+i+1);
        n_repl++;
}
```

```
//PASSO 2: Viene riempito l'array averaged_process con la media
    ottenuta dividendo
//per WELCH_N, ovvero il numero di repliche effettuate, la somma di
    tutte le repliche
//per un dato indice appartenente all'insieme WELCH_M

for(m_repl = 0; m_repl < WELCH_M; m_repl++){

        for(n_repl=0;n_repl<WELCH_N; n_repl++) {
          sum += observations[n_repl][m_repl];
        }


        averaged_process[m_repl+1] = sum/(double)WELCH_N;

        sum = 0.0;
}

        //PASSO 3: Calcolo dell'array Moving Average

for(i = 1; i <= (WELCH_M - WELCH_W); i++){
        sum = 0.0;
        if(i <= WELCH_W){
                for(s = -(i-1); s <= (i-1); s++)
                        sum = sum + averaged_process[i+s];

                moving_average[i] = sum/(double)(2*i - 1);

        }
        else{
                for(s = -WELCH_W; s <= WELCH_W; s++)
                        sum = sum + averaged_process[i+s];

                moving_average[i] = sum/(double)(2*WELCH_W + 1);
        }

}


//PASSO 4: scrittura dell'array moving average su file che sara'
    utile poi per graficare
//          i valori e determinare il valore l (lunghezza del
    transiente).


mov_avg_fd = fopen(output_file_name, "w");
```

```
        for(i = 1; i <= (WELCH_M - WELCH_W); i++)

                fprintf(mov_avg_fd, "%d\t%g\n", i, moving_average[i]);


        fclose(mov_avg_fd);

        printf("Il file %s e' stato riempito con l'array Moving Average.\n\n
            ", output_file_name);

        free(averaged_process);
        free(moving_average);
        csim_terminate();

}
```

6.7 main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <csim.h>
#include "client.h"
#include "common.h"
#include "gaussiana_inversa.h"
#include "service.h"

#ifndef _ISOC99_SOURCE
#define _ISOC99_SOURCE
#endif

extern STREAM sess_req_1;
extern STREAM sess_req_2;
extern STREAM user_tt;
extern STREAM object_req;
extern STREAM html_1;
extern STREAM html_2;
extern STREAM obj_size;
extern STREAM p_hit_proxy;

extern FACILITY cpuWS[NUM_SERVER];
extern FACILITY diskWS[NUM_DISK*NUM_SERVER];
extern BOX WebServer;
extern BOX WebSwitch;
extern FACILITY L2;
extern FACILITY CPU_web_switch;
extern FACILITY inLink;
extern FACILITY outLink;
extern FACILITY link_add;
extern FACILITY LS1;
extern FACILITY LS2;
extern FACILITY LW2[NUM_SERVER];
extern FACILITY LW3[NUM_SERVER];
extern TABLE rtime;
extern METER lambda;
//decidere quante classi fare sulla base del clustering
extern CLASS requestClasses[K];

extern int num_osservazioni;

TABLE resptime;
double lambda_tmp;
double utilizzazione_L2[NUM_CLASSES], utilizzazione_inLink[NUM_CLASSES],
    utilizzazione_outLink[NUM_CLASSES],
```

```
utilizzazione_cpu_web_switch[NUM_CLASSES], utilizzazione_cpu_web_server[
    NUM_CLASSES], utilizzazione_disk[NUM_CLASSES], utilizzazione_link_add[
    NUM_CLASSES], utilizzazione_ls1[NUM_CLASSES], utilizzazione_ls2[
    NUM_CLASSES], utilizzazione_lw2[NUM_CLASSES], utilizzazione_lw3[
    NUM_CLASSES];
double qlen_L2[NUM_CLASSES], qlen_inLink[NUM_CLASSES], qlen_outLink[
    NUM_CLASSES], qlen_cpu_web_switch[NUM_CLASSES],
qlen_cpu_web_server[NUM_CLASSES], qlen_disk[NUM_CLASSES], qlen_link_add[
    NUM_CLASSES], qlen_ls1[NUM_CLASSES], qlen_ls2[NUM_CLASSES], qlen_lw2[
    NUM_CLASSES], qlen_lw3[NUM_CLASSES];
double rtime_L2[NUM_CLASSES], rtime_inLink[NUM_CLASSES], rtime_outLink[
    NUM_CLASSES], rtime_cpu_web_switch[NUM_CLASSES],
rtime_cpu_web_server[NUM_CLASSES], rtime_disk[NUM_CLASSES], rtime_link_add[
    NUM_CLASSES], rtime_ls1[NUM_CLASSES], rtime_ls2[NUM_CLASSES], rtime_lw2[
    NUM_CLASSES], rtime_lw3[NUM_CLASSES];

double tot_temp;
//Calcolo delle statistiche per ogni componente: utilizzazione, lunghezza
    delle code e tempo di risposta

void print_tot(FILE *fd)
{
        fprintf(fd, "&%.7f\\\\\n\\hline", tot_temp);
        tot_temp = 0;
}

void statistics(int iteration, int variant) {
        lambda_tmp += meter_rate(lambda);
        int i = 0;
        int j = 0;


        for(; i<NUM_CLASSES; i++) {
                utilizzazione_L2[i] += class_util(L2, requestClasses[i]);
                qlen_L2[i] += class_qlen(L2, requestClasses[i]);
                rtime_L2[i] += class_resp(L2, requestClasses[i]);

                utilizzazione_inLink[i] += class_util(inLink, requestClasses
                    [i]);
                qlen_inLink[i] += class_qlen(inLink, requestClasses[i]);
                rtime_inLink[i] += class_resp(inLink, requestClasses[i]);

                utilizzazione_outLink[i] += class_util(outLink,
                    requestClasses[i]);
                qlen_outLink[i] += class_qlen(outLink, requestClasses[i]);
                rtime_outLink[i] += class_resp(outLink, requestClasses[i]);
```

```
            utilizzazione_cpu_web_switch[i] += class_util(CPU_web_switch
                , requestClasses[i]);
            qlen_cpu_web_switch[i] += class_qlen(CPU_web_switch,
                requestClasses[i]);
            rtime_cpu_web_switch[i] += class_resp(CPU_web_switch,
                requestClasses[i]);

            utilizzazione_ls1[i] += class_util(LS1, requestClasses[i]);
            qlen_ls1[i] += class_qlen(LS1, requestClasses[i]);
            rtime_ls1[i] += class_resp(LS1, requestClasses[i]);


            utilizzazione_ls2[i] += class_util(LS2, requestClasses[i]);
            qlen_ls2[i] += class_qlen(LS2, requestClasses[i]);
            rtime_ls2[i] += class_resp(LS2, requestClasses[i]);

            if(variant == LINK_ADD) {
                    utilizzazione_link_add[i] += class_util(link_add,
                        requestClasses[i]);
                    qlen_link_add[i] += class_qlen(link_add,
                        requestClasses[i]);
                    rtime_link_add[i] += class_resp(link_add,
                        requestClasses[i]);
            }
    }

    double util_cpu_tmp[NUM_CLASSES] = {0.0};
    double qLen_cpu_tmp[NUM_CLASSES] = {0.0};
    double rtime_cpu_tmp[NUM_CLASSES] = {0.0};

    double util_disk_tmp[NUM_CLASSES] = {0.0};
    double qLen_disk_tmp[NUM_CLASSES] = {0.0};
    double rtime_disk_tmp[NUM_CLASSES] = {0.0};

    double util_lw2_tmp[NUM_CLASSES] = {0.0};
    double qLen_lw2_tmp[NUM_CLASSES] = {0.0};
    double rtime_lw2_tmp[NUM_CLASSES] = {0.0};

    double util_lw3_tmp[NUM_CLASSES] = {0.0};
    double qLen_lw3_tmp[NUM_CLASSES] = {0.0};
    double rtime_lw3_tmp[NUM_CLASSES] = {0.0};



    // Per ogni classe colleziono le statistiche di interesse (mediate
        sul numero dei server e dei dischi)
    for(j=0; j<NUM_CLASSES; j++) {
```

```
//calcolo metriche cpu web server
for(i=0; i<NUM_SERVER; i++){
        util_cpu_tmp[j] += class_util(cpuWS[i],
            requestClasses[j]);
        qLen_cpu_tmp[j] += class_qlen(cpuWS[i],
            requestClasses[j]);
        rtime_cpu_tmp[j] += class_resp(cpuWS[i],
            requestClasses[j]);

        util_lw2_tmp[j] += class_util(LW2[i], requestClasses
            [j]);
        qLen_lw2_tmp[j] += class_qlen(LW2[i], requestClasses
            [j]);
        rtime_lw2_tmp[j] += class_resp(LW2[i],
            requestClasses[j]);
        if(variant == LINK_ADD) {
                util_lw3_tmp[j] += class_util(LW3[i],
                    requestClasses[j]);
                qLen_lw3_tmp[j] += class_qlen(LW3[i],
                    requestClasses[j]);
                rtime_lw3_tmp[j] += class_resp(LW3[i],
                    requestClasses[j]);

        }
}
utilizzazione_cpu_web_server[j] += util_cpu_tmp[j]/(double)
    NUM_SERVER;
qlen_cpu_web_server[j] += qLen_cpu_tmp[j]/(double)NUM_SERVER
    ;
rtime_cpu_web_server[j] += rtime_cpu_tmp[j]/(double)
    NUM_SERVER;

utilizzazione_lw2[j] += util_lw2_tmp[j]/(double)NUM_SERVER;
qlen_lw2[j] += qLen_lw2_tmp[j]/(double)NUM_SERVER;
rtime_lw2[j] += rtime_lw2_tmp[j]/(double)NUM_SERVER;
if(variant == LINK_ADD) {
        utilizzazione_lw3[j] += util_lw3_tmp[j]/(double)
            NUM_SERVER;
        qlen_lw3[j] += qLen_lw3_tmp[j]/(double)NUM_SERVER;
        rtime_lw3[j] += rtime_lw3_tmp[j]/(double)NUM_SERVER;
}
// calcolo metriche disco
for(i=0; i<(NUM_SERVER*NUM_DISK); i++){
        util_disk_tmp[j] += class_util(diskWS[i],
            requestClasses[j]);
        qLen_disk_tmp[j] += class_qlen(diskWS[i],
            requestClasses[j]);
```

```
                rtime_disk_tmp[j] += class_resp(diskWS[i],
                    requestClasses[j]);
        }
        utilizzazione_disk[j] += util_disk_tmp[j]/(double)(
            NUM_SERVER*NUM_DISK);
        qlen_disk[j] += qLen_disk_tmp[j]/(double)(NUM_SERVER*
            NUM_DISK);
        rtime_disk[j] += rtime_disk_tmp[j]/(double)(NUM_SERVER*
            NUM_DISK);
}

FILE *fd_file;
char *pathname = "util_qlen_rtime";

//Stampa dei risultati su file
if(iteration==NUM_ITERATIONS-1) {
        fd_file = fopen(pathname, "w");
        fprintf(fd_file, "\\hline\nUtilizzazioni\n\\hline\n");
        fprintf(fd_file, "Centro &Classe1 &Classe2 &Classe3 &Totale
            \\\\\n\\hline\n\\hline");
        fprintf(fd_file, "\n cpu web server i-esimo: \t");
        for(j=0; j<NUM_CLASSES; j++)
        {
                fprintf(fd_file, "&%.7f\t",
                    utilizzazione_cpu_web_server[j]/(NUM_ITERATIONS)
                    );
                tot_temp += utilizzazione_cpu_web_server[j]/(
                    NUM_ITERATIONS);
        }
        print_tot(fd_file);

        fprintf(fd_file, "\n disco i-esimo: \t");
        for(j=0; j<NUM_CLASSES; j++)
        {
                fprintf(fd_file, "&%.7f\t", utilizzazione_disk[j]/(
                    NUM_ITERATIONS));
                tot_temp += utilizzazione_disk[j]/(NUM_ITERATIONS);
        }
        print_tot(fd_file);

        fprintf(fd_file, "\n inLink: \t");
        for(j=0; j<NUM_CLASSES; j++)
        {
                fprintf(fd_file, "&%.7f\t", utilizzazione_inLink[j
                    ]/(NUM_ITERATIONS));
                tot_temp += utilizzazione_inLink[j]/(NUM_ITERATIONS)
                    ;
```

```
}
print_tot(fd_file);

fprintf(fd_file, "\n outLink: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", utilizzazione_outLink[j
            ]/(NUM_ITERATIONS));
        tot_temp += utilizzazione_outLink[j]/(NUM_ITERATIONS
            );
}
print_tot(fd_file);

fprintf(fd_file, "\n cpu web switch: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t",
            utilizzazione_cpu_web_switch[j]/(NUM_ITERATIONS)
            );
        tot_temp += utilizzazione_cpu_web_switch[j]/(
            NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n LAN: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", utilizzazione_L2[j]/(
            NUM_ITERATIONS));
        tot_temp +=  utilizzazione_L2[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n LS1: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", utilizzazione_ls1[j]/(
            NUM_ITERATIONS));
        tot_temp += utilizzazione_ls1[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n LS2:\t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", utilizzazione_ls2[j]/(
            NUM_ITERATIONS));
```

```
        tot_temp += utilizzazione_ls2[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n LW2: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "%.7f\t", utilizzazione_lw2[j]/(
            NUM_ITERATIONS));
        tot_temp += utilizzazione_lw2[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

if(variant == LINK_ADD) {
        fprintf(fd_file, "\n LINK_ADD: \t");
        for(j=0; j<NUM_CLASSES; j++)
        {
                fprintf(fd_file, "&%.7f\t",
                    utilizzazione_link_add[j]/(
                    NUM_ITERATIONS));
                tot_temp += utilizzazione_link_add[j]/(
                    NUM_ITERATIONS);
        }
  print_tot(fd_file);

        fprintf(fd_file, "\n LW3: \t");
  for(j=0; j<NUM_CLASSES; j++)
  {
                fprintf(fd_file, "&%.7f\t",
                    utilizzazione_lw3[j]/(NUM_ITERATIONS));
                tot_temp += utilizzazione_lw3[j]/(
                    NUM_ITERATIONS);
        }
        print_tot(fd_file);

}

fprintf(fd_file, "\n\n\\hline\nLunghezza media delle code\n
    \\hline\n");
fprintf(fd_file, "Centro &Classe1 &Classe2 &Classe3 &Totale
    \\\\\n\\hline\n\\hline");
fprintf(fd_file, "\n cpu web server i-esimo: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", qlen_cpu_web_server[j]/(
            NUM_ITERATIONS));
        tot_temp += qlen_cpu_web_server[j]/(NUM_ITERATIONS);
```

```
}
print_tot(fd_file);

fprintf(fd_file, "\n disco i-esimo: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", qlen_disk[j]/(
            NUM_ITERATIONS));
        tot_temp += qlen_disk[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n inLink: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", qlen_inLink[j]/(
            NUM_ITERATIONS));
        tot_temp += qlen_inLink[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n outLink: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", qlen_outLink[j]/(
            NUM_ITERATIONS));
        tot_temp += qlen_outLink[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n cpu web switch: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", qlen_cpu_web_switch[j]/(
            NUM_ITERATIONS));
        tot_temp += qlen_cpu_web_switch[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n LAN: \t");
for(j=0; j<NUM_CLASSES; j++)
{
        fprintf(fd_file, "&%.7f\t", qlen_L2[j]/(
            NUM_ITERATIONS));
        tot_temp += qlen_L2[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);
```

```
                fprintf(fd_file, "\n LS1: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", qlen_ls1[j]/(
                            NUM_ITERATIONS));
                        tot_temp += qlen_ls1[j]/(NUM_ITERATIONS);
                }
                print_tot(fd_file);

                fprintf(fd_file, "\n LS2: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", qlen_ls2[j]/(
                            NUM_ITERATIONS));
                        tot_temp += qlen_ls2[j]/(NUM_ITERATIONS);
                }
                print_tot(fd_file);

                fprintf(fd_file, "\n LW2: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", qlen_lw2[j]/(
                            NUM_ITERATIONS));
    tot_temp += qlen_lw2[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

if(variant == LINK_ADD) {
                        fprintf(fd_file, "\n LINK_ADD: \t");
                        for(j=0; j<NUM_CLASSES; j++)
                        {
                                fprintf(fd_file, "&%.7f\t", qlen_link_add[j
                                    ]/(NUM_ITERATIONS));
                                tot_temp += qlen_link_add[j]/(NUM_ITERATIONS
                                    );
                        }
                        print_tot(fd_file);

                        fprintf(fd_file, "\n LW3: \t");
                for(j=0; j<NUM_CLASSES; j++)
                        {
                                fprintf(fd_file, "&%.7f\t", qlen_lw3[j]/(
                                    NUM_ITERATIONS));
                                tot_temp += qlen_lw3[j]/(NUM_ITERATIONS);
                        }
                        print_tot(fd_file);
```

```
                }

                fprintf(fd_file, "\n\n\\hline\nTempo Medio di Risposta\n\\
                    hline\n");
                fprintf(fd_file, "Centro &Classe1 &Classe2 &Classe3 &Totale
                    \\\\\n\\hline\n\\hline");
                fprintf(fd_file, "\n cpu web server i-esimo: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", rtime_cpu_web_server[j
                            ]/(NUM_ITERATIONS));
                        tot_temp += rtime_cpu_web_server[j]/(NUM_ITERATIONS)
                            ;
                }
                print_tot(fd_file);

                fprintf(fd_file, "\n disco i-esimo: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", rtime_disk[j]/(
                            NUM_ITERATIONS));
    tot_temp += rtime_disk[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

                fprintf(fd_file, "\n inLink: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", rtime_inLink[j]/(
                            NUM_ITERATIONS));
                        tot_temp += rtime_inLink[j]/(NUM_ITERATIONS);
                }
                print_tot(fd_file);

                fprintf(fd_file, "\n outlink: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", rtime_outLink[j]/(
                            NUM_ITERATIONS));
                        tot_temp += rtime_outLink[j]/(NUM_ITERATIONS);
                }
                print_tot(fd_file);

                fprintf(fd_file, "\n cpu web switch: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
```

```
                        fprintf(fd_file, "&%.7f\t", rtime_cpu_web_switch[j
                            ]/(NUM_ITERATIONS));
                        tot_temp += rtime_cpu_web_switch[j]/(NUM_ITERATIONS)
                            ;
                }
                print_tot(fd_file);

                fprintf(fd_file, "\n LAN: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", rtime_L2[j]/(
                            NUM_ITERATIONS));
                        tot_temp += rtime_L2[j]/(NUM_ITERATIONS);
                }
                print_tot(fd_file);

                fprintf(fd_file, "\n LS1: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", rtime_ls1[j]/(
                            NUM_ITERATIONS));
    tot_temp += rtime_ls1[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n LS2: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", rtime_ls2[j]/(
                            NUM_ITERATIONS));
    tot_temp += rtime_ls2[j]/(NUM_ITERATIONS);
}
print_tot(fd_file);

fprintf(fd_file, "\n LW2: \t");
                for(j=0; j<NUM_CLASSES; j++)
                {
                        fprintf(fd_file, "&%.7f\t", rtime_lw2[j]/(
                            NUM_ITERATIONS));
                        tot_temp += rtime_lw2[j]/(NUM_ITERATIONS);
}
                print_tot(fd_file);

if(variant == LINK_ADD) {
                        fprintf(fd_file, "\n LINK_ADD: \t");
                        for(j=0; j<NUM_CLASSES; j++)
                        {
```

```
                                fprintf(fd_file, "&%.7f\t", rtime_link_add[j
                                    ]/(NUM_ITERATIONS));
                                tot_temp += rtime_link_add[j]/(
                                    NUM_ITERATIONS);
                        }
                        print_tot(fd_file);

                        fprintf(fd_file, "\n LW3: \t");
                        for(j=0; j<NUM_CLASSES; j++)
                        {
                                fprintf(fd_file, "&%.7f\t", rtime_lw3[j]/(
                                    NUM_ITERATIONS));
          tot_temp += rtime_lw3[j]/(NUM_ITERATIONS);
        }
        print_tot(fd_file);
                }
                fprintf(fd_file, "\n\n Tasso medio di arrivi (lambda)
                                        : %g\n", lambda_tmp/(
                    NUM_ITERATIONS));

                fclose(fd_file);
          }
}

//trovare un modo elegante di passare la variante, per ora passo RANDOM
//Simulazione dell'intero modello.
void sim(int argc, char **argv) {
          printf("Starting simulation...\n");
          int i=0;
          int client_id;
          int variante = LEAST_LOADED;
          int reset;
          char filename[25];
          FILE *output;
          for(i=0; i<NUM_ITERATIONS; i++) {
                num_osservazioni = 0;
                reset = 0;
                create("simulation");
                max_processes(MAX_PROCESSES);
                max_facilities(MAX_FACILITIES);
                max_servers(MAX_SERVERS);
                max_classes(MAX_CLASSES);
                filename[0] = '\0';
                sprintf(filename, "Sim_prova_%d", i);
                output = fopen(filename, "w");
                set_output_file(output);
```

```
                // Inizializzazione degli stream (reseed simtime*i+num)
                sess_req_1 = create_stream();
                reseed(sess_req_1, (int)simtime()+i+1);
                sess_req_2 = create_stream();
                reseed(sess_req_2, (int)simtime()*2+i+1);
                user_tt = create_stream();
                reseed(user_tt, (int)simtime()*3+i+1);
                object_req = create_stream();
                reseed(object_req, (int)simtime()*4+i+1);
                html_1 = create_stream();
                reseed(html_1, (int)simtime()*5+i+1);
                html_2 = create_stream();
                reseed(html_2, (int)simtime()*6+i+1);
                obj_size = create_stream();
                reseed(obj_size, (int)simtime()*7+i+1);
                p_hit_proxy = create_stream();
                reseed(p_hit_proxy, (int)simtime()*8+i+1);

                // Inizializzazione delle facility
                inLink = facility("inLink");
                outLink = facility("outLink");


                LS1 = facility("LS1");
                LS2 = facility("LS2");
                CPU_web_switch = facility("CPU_web_switch");
                L2 = facility("L2");
                facility_set(cpuWS, "cpuWS", NUM_SERVER);
                facility_set(diskWS, "diskWS", NUM_SERVER*NUM_DISK);
                facility_set(LW2, "LW2", NUM_SERVER);

    if(variante == LINK_ADD) {
                 link_add = facility("link_add");
                 facility_set(LW3, "LW3", NUM_SERVER);
                }
                // Table initialization
                rtime = table("System Response Time");
                resptime = permanent_table("Tempo di risposta del sistema");
                char className[20];
                className[0] = '\0';

                WebServer = box("Web Server");
                WebSwitch = box("Web Switch");
                lambda = meter("Arrival Rate");

                int j=0;
                for(; j<NUM_CLASSES; j++){
```

```
                    className[0] = '\0';
                    sprintf(className, "Classe%d", j);
                    requestClasses[j] = process_class(className);
                }

                collect_class_facility_all();

                reseed(sess_req_1, (int)simtime()+i+1);
                reseed(sess_req_2, (int)simtime()*2+i+1);
                reseed(user_tt, (int)simtime()*3+i+1);
                reseed(object_req, (int)simtime()*4+i+1);
                reseed(html_1, (int)simtime()*5+i+1);
                reseed(html_2, (int)simtime()*6+i+1);
                reseed(obj_size, (int)simtime()*7+i+1);
                reseed(p_hit_proxy, (int)simtime()*8+i+1);

                while(state(converged)==NOT_OCC && num_osservazioni<500000)
                    {
                        hold(exponential(1/(double)ARRIVAL));
                        printf("num_osservazioni %d\n", num_osservazioni);
                        web_session(client_id, variante, 0, -1);
                        client_id++;
                        if(num_osservazioni>100000 &&(!reset)) {
                                printf("Reset statistics %g\n", simtime());
                                reset();
                                reset=1;
                                table_confidence(rtime);
                                table_run_length(rtime, 0.005, 0.98,
                                    10000.0);
                        }
                }
                printf("Fine generazione a %g - iterazione %d\n", simtime(),
                     i);
                report_facilities();

                report_table(rtime);

                report_boxes();

                meter_summary();
                tabulate(resptime, table_mean(rtime));
                statistics(i, variante);
                rerun();
                printf("End %i\n", i);
        }
        table_summary();
}
```

6.8 analytical.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <string.h>
#include <stdarg.h>
#include "common.h"
#include "service.h"
#define LAMBDA_SIM 3308.79 //prova
#define NUM_OBS 10000000

//Calcola la lunghezza della coda
double calc_queue_length(double utilization)
{
        return utilization/(1-utilization);
}

//Calcola il tempo di risposta
double calc_response_time(double service, double utilization)
{
        return service/(1 - utilization);
}

//Calcola e salva su un file gli indici di prestazione del sistema
int main(int argc, char **argv)
{
        FILE *fd_file;
        char *pathname = (char*)malloc(128);
        sprintf(pathname, "risultati modello analitico.txt");
        fd_file = fopen(pathname, "w");

        double utilizzazione_L2[NUM_CLASSES], utilizzazione_inLink[
            NUM_CLASSES], utilizzazione_outLink[NUM_CLASSES],
        utilizzazione_cpu_web_switch[NUM_CLASSES],
            utilizzazione_cpu_web_server[NUM_CLASSES],
            utilizzazione_disco_web_server[NUM_CLASSES],
            utilizzazione_link_add[NUM_CLASSES], utilizzazione_ls1[
            NUM_CLASSES], utilizzazione_ls2[NUM_CLASSES], utilizzazione_lw2[
            NUM_CLASSES], utilizzazione_lw3[NUM_CLASSES];

        double coda_L2, coda_inLink, coda_outLink, coda_cpu_web_switch,
            coda_cpu_web_server, coda_disco_web_server, coda_link_add,
            coda_ls1, coda_ls2, coda_lw2, coda_lw3;
```

```
        double tr_L2, tr_inLink, tr_outLink, tr_cpu_web_switch,
            tr_cpu_web_server, tr_disco_web_server, tr_link_add, tr_ls1,
            tr_ls2, tr_lw2, tr_lw3;

        //centroidi
        double doc_size[NUM_CLASSES];
        /*doc_size[0] = 10281;  DOC_SIZE PER K=3
        doc_size[1] = 279513744;
        doc_size[2] = 715827882;
        */
        doc_size[0] = 10073;
        doc_size[1] = 17740962;
        doc_size[2] = 70926601;
        doc_size[3] = 279513744;
        doc_size[4] = 715827882;

        //probabilita' che la risorsa non si trovi nel proxy
        double miss = 1- P_HIT;
        int i = 0;

        //tasso di richieste per ogni classe = probabilita' di classe *
            tasso di richieste totale
    double lambda[NUM_CLASSES];
        /*lambda[0] = ((double)9999995/(double)NUM_OBS)*LAMBDA_SIM;
        lambda[1] = ((double)4/(double)NUM_OBS)*LAMBDA_SIM;
        lambda[2] = ((double)1/(double)NUM_OBS)*LAMBDA_SIM;
        */

        lambda[0] = ((double)9999917/(double)NUM_OBS)*LAMBDA_SIM;
        lambda[1] = ((double)65/(double)NUM_OBS)*LAMBDA_SIM;
        lambda[2] = ((double)13/(double)NUM_OBS)*LAMBDA_SIM;
        lambda[3] = ((double)4/(double)NUM_OBS)*LAMBDA_SIM;
        lambda[4] = ((double)1/(double)NUM_OBS)*LAMBDA_SIM;

        fprintf(fd_file, "calcolo indici locali variante standard\n");

        //calcolo utilizzazioni
        for(i=0; i < NUM_CLASSES; i++) {
                fprintf(fd_file, "\nutilizzazioni classe %d\n",i);
                utilizzazione_L2[i] = lambda[i]*(D_LAN(doc_size[i])+D_LAN(0)
                    );
                fprintf(fd_file, "L2: %9.9f\n", utilizzazione_L2[i]);

                utilizzazione_inLink[i] = lambda[i]*D_InLink();
                fprintf(fd_file, "inLink: %9.9f\n", utilizzazione_inLink[i])
                    ;
```

```
            utilizzazione_outLink[i] = lambda[i]*D_OutLink(doc_size[i]);
            fprintf(fd_file, "outLink: %9.9f\n", utilizzazione_outLink[i
                ]);

            utilizzazione_cpu_web_switch[i] = lambda[i] * D_Cpu(
                CPU_WEB_SWITCH_SERVICE_RATE)* 2; //fattore due perche'
                processa sia richieste in entrata che in uscita (two-way
                )
            fprintf(fd_file, "cpu web switch: %9.9f\n",
                utilizzazione_cpu_web_switch[i]);

            utilizzazione_cpu_web_server[i] = (lambda[i] * D_Cpu(
                CPU_SERVICE_RATE) * 2)/(double) NUM_SERVER;
            fprintf(fd_file, "cpu web server: %9.9f\n",
                utilizzazione_cpu_web_server[i]);

            utilizzazione_disco_web_server[i] = (lambda[i] * D_WSDisk(
                doc_size[i]))/(NUM_SERVER*NUM_DISK);
            fprintf(fd_file, "disco web server: %9.9f\n",
                utilizzazione_disco_web_server[i]);

            utilizzazione_ls1[i] = lambda[i]*(D_LS1in()+D_LS1out(
                doc_size[i]));
            fprintf(fd_file, "LS1: %9.9f\n", utilizzazione_ls1[i]);
            utilizzazione_ls2[i] = lambda[i]*(D_LAN(doc_size[i])+D_LAN
                (0));
            fprintf(fd_file, "LS2: %9.9f\n", utilizzazione_ls2[i]);
            utilizzazione_lw2[i] = (lambda[i]*(D_LAN(doc_size[i])+D_LAN
                (0)))/((double)NUM_SERVER);
            fprintf(fd_file, "LW2: %9.9f\n", utilizzazione_lw2[i]);
    }

    //calcolo lunghezza code
    for(i=0; i < NUM_CLASSES; i++) {
            fprintf(fd_file, "\nlunghezze code classe %d\n",i);
            coda_L2 = calc_queue_length(utilizzazione_L2[i]);
            fprintf(fd_file, "L2: %9.9f\n", coda_L2);

            coda_inLink = calc_queue_length(utilizzazione_inLink[i]);
            fprintf(fd_file, "inLink: %9.9f\n", coda_inLink);

            coda_outLink = calc_queue_length(utilizzazione_outLink[i]);
            fprintf(fd_file, "outLink: %9.9f\n", coda_outLink);

            coda_cpu_web_switch = calc_queue_length(
                utilizzazione_cpu_web_switch[i]);
```

```
            fprintf(fd_file, "cpu web switch: %9.9f\n",
                coda_cpu_web_switch);

            coda_cpu_web_server = calc_queue_length(
                utilizzazione_cpu_web_server[i]);
            fprintf(fd_file, "cpu web server: %9.9f\n",
                coda_cpu_web_server);

            coda_disco_web_server = calc_queue_length(
                utilizzazione_disco_web_server[i]);
            fprintf(fd_file, "disco web server: %9.9f\n",
                coda_disco_web_server);

            coda_ls1 = calc_queue_length(utilizzazione_ls1[i]);
            fprintf(fd_file, "LS1: %9.9f\n", coda_ls1);

            coda_ls2 = calc_queue_length(utilizzazione_ls2[i]);
            fprintf(fd_file, "LS2: %9.9f\n", coda_ls2);

            coda_lw2 = calc_queue_length(utilizzazione_lw2[i]);
            fprintf(fd_file, "LW2: %9.9f\n", coda_lw2);
    }

    //tempi di residenza
    for(i=0; i < NUM_CLASSES; i++) {
            fprintf(fd_file, "\ntempi di residenza %d\n", i);
            tr_L2 = calc_response_time(D_LAN(doc_size[i]),
                utilizzazione_L2[i]);
            fprintf(fd_file, "L2: %9.9f\n", tr_L2);

            tr_inLink = calc_response_time(D_InLink(),
                utilizzazione_inLink[i]);
            fprintf(fd_file, "inLink: %9.9f\n", tr_inLink);

            tr_outLink = calc_response_time(D_OutLink(doc_size[i]),
                utilizzazione_outLink[i]);
            fprintf(fd_file, "outLink: %9.9f\n", tr_outLink);

            tr_cpu_web_switch = calc_response_time(D_Cpu(
                CPU_WEB_SWITCH_SERVICE_RATE)* 2,
                utilizzazione_cpu_web_switch[i]);
            fprintf(fd_file, "cpu web switch: %9.9f\n",
                tr_cpu_web_switch);

            tr_cpu_web_server = calc_response_time(D_Cpu(
                CPU_SERVICE_RATE) * 2, utilizzazione_cpu_web_server[i]);
```

```
            fprintf(fd_file, "cpu web server: %9.9f\n",
                tr_cpu_web_server);

            tr_disco_web_server = calc_response_time(D_WSDisk(doc_size[i
                ]), utilizzazione_disco_web_server[i]);
            fprintf(fd_file, "disco web server: %9.9f\n",
                tr_disco_web_server);

            tr_ls1 = calc_response_time((D_LS1in()+D_LS1out(doc_size[i])
                ),utilizzazione_ls1[i]);
            fprintf(fd_file, "LS1: %9.9f\n", tr_ls1);

            tr_ls2 = calc_response_time(D_LAN(doc_size[i]),
                utilizzazione_ls2[i]);
            fprintf(fd_file, "LS2: %9.9f\n", tr_ls2);

            tr_lw2 = calc_response_time(D_LAN(doc_size[i]),
                utilizzazione_lw2[i]);
            fprintf(fd_file, "LS2: %9.9f\n", tr_lw2);
    }

    //nel caso di inserimento del proxy bisogna moltiplicare tutte le
        utilizzazioni per 0.6, nel caso di link addizionale bisogna
        ricalcolare l'utilizzazione della cpu del web  switch (one way)
        e calcolare la nuova utilizzazione della L3.
    fprintf(fd_file, "\ncalcolo indici locali con link addizionale\n");

    //calcolo utilizzazioni link addizionale, cosa cambia nella LAN L2?
    for(i=0; i < NUM_CLASSES; i++) {
            fprintf(fd_file, "\nutilizzazioni classe %d\n",i);
            utilizzazione_L2[i] = lambda[i]*D_LAN(0);
            fprintf(fd_file, "L2: %9.9f\n", utilizzazione_L2[i]);

            utilizzazione_inLink[i] = lambda[i]*D_InLink();
            fprintf(fd_file, "inLink: %9.9f\n", utilizzazione_inLink[i])
                ;

            utilizzazione_cpu_web_switch[i] = lambda[i] * D_Cpu(
                CPU_WEB_SWITCH_SERVICE_RATE);//fattore due perche'
                processa sia richieste in entrata che in uscita (two-way
                )
            fprintf(fd_file, "cpu web switch: %9.9f\n",
                utilizzazione_cpu_web_switch[i]);

            utilizzazione_cpu_web_server[i] = (lambda[i] * D_Cpu(
                CPU_SERVICE_RATE) * 2)/(double) NUM_SERVER;
```

```
            fprintf(fd_file, "cpu web server: %9.9f\n",
                utilizzazione_cpu_web_server[i]);

            utilizzazione_disco_web_server[i] = (lambda[i] * D_WSDisk(
                doc_size[i]))/(NUM_SERVER*NUM_DISK);
            fprintf(fd_file, "disco web server: %9.9f\n",
                utilizzazione_disco_web_server[i]);

            utilizzazione_link_add[i] = lambda[i]*D_linkAdd(doc_size[i])
                ;
            fprintf(fd_file, "linkAdd: %9.9f\n", utilizzazione_link_add[
                i]);

            utilizzazione_ls1[i] = lambda[i]*(D_LS1in());
            fprintf(fd_file, "LS1: %9.9f\n", utilizzazione_ls1[i]);

            utilizzazione_ls2[i] = lambda[i]*D_LAN(0);
            fprintf(fd_file, "LS2: %9.9f\n", utilizzazione_ls2[i]);

            utilizzazione_lw2[i] = (lambda[i]*D_LAN(0))/((double)
                NUM_SERVER);
            fprintf(fd_file, "LW2: %9.9f\n", utilizzazione_lw2[i]);

            utilizzazione_lw3[i] = (lambda[i]*D_linkAdd(doc_size[i]))/(
                double)NUM_SERVER;
            fprintf(fd_file, "LW3: %9.9f\n", utilizzazione_lw3[i]);
    }
    //calcolo lunghezza code
    for(i=0; i < NUM_CLASSES; i++) {
            fprintf(fd_file, "\nlunghezze code classe %d\n",i);
            coda_L2 = calc_queue_length(utilizzazione_L2[i]);
            fprintf(fd_file, "L2: %9.9f\n", coda_L2);

            coda_inLink = calc_queue_length(utilizzazione_inLink[i]);
            fprintf(fd_file, "inLink: %9.9f\n", coda_inLink);

            coda_cpu_web_switch = calc_queue_length(
                utilizzazione_cpu_web_switch[i]);
            fprintf(fd_file, "cpu web switch: %9.9f\n",
                coda_cpu_web_switch);

            coda_cpu_web_server = calc_queue_length(
                utilizzazione_cpu_web_server[i]);
            fprintf(fd_file, "cpu web server: %9.9f\n",
                coda_cpu_web_server);
```

```
            coda_disco_web_server = calc_queue_length(
                utilizzazione_disco_web_server[i]);
            fprintf(fd_file, "disco web server: %9.9f\n",
                coda_disco_web_server);

            coda_link_add = calc_queue_length(utilizzazione_link_add[i])
                ;
            fprintf(fd_file, "linkAdd: %9.9f\n", coda_link_add);

            coda_ls1 = calc_queue_length(utilizzazione_ls1[i]);
            fprintf(fd_file, "LS1: %9.9f\n", coda_ls1);

            coda_ls2 = calc_queue_length(utilizzazione_ls2[i]);
            fprintf(fd_file, "LS2: %9.9f\n", coda_ls2);

            coda_lw2 = calc_queue_length(utilizzazione_lw2[i]);
            fprintf(fd_file, "LW2: %9.9f\n", coda_lw2);

            coda_lw3 = calc_queue_length(utilizzazione_lw3[i]);
            fprintf(fd_file, "LW3: %9.9f\n", coda_lw3);

    }

    //tempi di residenza
    for(i=0; i < NUM_CLASSES; i++) {
            fprintf(fd_file, "\ntempi di residenza %d\n", i);
            tr_L2 = calc_response_time(D_LAN(0), utilizzazione_L2[i]);
            fprintf(fd_file, "L2: %9.9f\n", tr_L2);

            tr_inLink = calc_response_time(D_InLink(),
                utilizzazione_inLink[i]);
            fprintf(fd_file, "inLink: %9.9f\n", tr_inLink);

            tr_cpu_web_switch = calc_response_time(D_Cpu(
                CPU_WEB_SWITCH_SERVICE_RATE)* 2,
                utilizzazione_cpu_web_switch[i]);
            fprintf(fd_file, "cpu web switch: %9.9f\n",
                tr_cpu_web_switch);

            tr_cpu_web_server = calc_response_time(D_Cpu(
                CPU_SERVICE_RATE) * 2, utilizzazione_cpu_web_server[i]);
            fprintf(fd_file, "cpu web server: %9.9f\n",
                tr_cpu_web_server);

            tr_disco_web_server = calc_response_time(D_WSDisk(doc_size[i
                ]), utilizzazione_disco_web_server[i]);
```

```
            fprintf(fd_file, "disco web server: %9.9f\n",
                tr_disco_web_server);

            tr_link_add = calc_response_time(D_linkAdd(doc_size[i]),
                utilizzazione_link_add[i]);
            fprintf(fd_file, "linkAdd: %9.9f\n", tr_link_add);

            tr_ls1 = calc_response_time(D_LS1in(), utilizzazione_ls1[i])
                ;
            fprintf(fd_file, "LS1: %9.9f\n", tr_ls1);

            tr_ls2 = calc_response_time(D_LAN(0), utilizzazione_ls2[i]);
            fprintf(fd_file, "LS2: %9.9f\n", tr_ls2);

            tr_lw2 = calc_response_time(D_LAN(0), utilizzazione_lw2[i]);
            fprintf(fd_file, "LW2: %9.9f\n", tr_lw2);

            tr_lw3 = calc_response_time(D_linkAdd(doc_size[i]),
                utilizzazione_lw3[i]);
            fprintf(fd_file, "LW3: %9.9f\n", tr_lw3);

    }

    fprintf(fd_file, "\ncalcolo indici locali con proxy\n");

    for(i=0; i < NUM_CLASSES; i++) {
            fprintf(fd_file, "\nutilizzazioni classe %d\n",i);
            utilizzazione_L2[i] = miss*lambda[i]*(D_LAN(doc_size[i])+
                D_LAN(0));
            fprintf(fd_file, "L2: %9.9f\n", utilizzazione_L2[i]);

            utilizzazione_inLink[i] = miss*lambda[i]*D_InLink();
            fprintf(fd_file, "inLink: %9.9f\n", utilizzazione_inLink[i])
                ;

            utilizzazione_outLink[i] = miss*lambda[i]*D_OutLink(doc_size
                [i]);
            fprintf(fd_file, "outLink: %9.9f\n", utilizzazione_outLink[i
                ]);

            utilizzazione_cpu_web_switch[i] = miss*lambda[i] * D_Cpu(
                CPU_WEB_SWITCH_SERVICE_RATE)* 2; //fattore due perche'
                processa sia richieste in entrata che in uscita (two-way
                )
            fprintf(fd_file, "cpu web switch: %9.9f\n",
                utilizzazione_cpu_web_switch[i]);
```

```
            utilizzazione_cpu_web_server[i] = miss*(lambda[i] * D_Cpu(
                CPU_SERVICE_RATE) * 2)/(double) NUM_SERVER;
            fprintf(fd_file, "cpu web server: %9.9f\n",
                utilizzazione_cpu_web_server[i]);

            utilizzazione_disco_web_server[i] = miss*(lambda[i] *
                D_WSDisk(doc_size[i]))/(NUM_SERVER*NUM_DISK);
            fprintf(fd_file, "disco web server: %9.9f\n",
                utilizzazione_disco_web_server[i]);

            utilizzazione_ls1[i] = miss*lambda[i]*(D_LS1in()+D_LS1out(
                doc_size[i]));
            fprintf(fd_file, "LS1: %9.9f\n", utilizzazione_ls1[i]);
            utilizzazione_ls2[i] = miss*lambda[i]*(D_LAN(doc_size[i])+
                D_LAN(0));
            fprintf(fd_file, "LS2: %9.9f\n", utilizzazione_ls2[i]);
            utilizzazione_lw2[i] = miss*(lambda[i]*(D_LAN(doc_size[i])+
                D_LAN(0)))/((double)NUM_SERVER);
            fprintf(fd_file, "LW2: %9.9f\n", utilizzazione_lw2[i]);
    }
    //calcolo lunghezza code
    for(i=0; i < NUM_CLASSES; i++) {
            fprintf(fd_file, "\nlunghezze code classe %d\n",i);
            coda_L2 = calc_queue_length(utilizzazione_L2[i]);
            fprintf(fd_file, "L2: %9.9f\n", coda_L2);

            coda_inLink = calc_queue_length(utilizzazione_inLink[i]);
            fprintf(fd_file, "inLink: %9.9f\n", coda_inLink);

            coda_outLink = calc_queue_length(utilizzazione_outLink[i]);
            fprintf(fd_file, "outLink: %9.9f\n", coda_outLink);

            coda_cpu_web_switch = calc_queue_length(
                utilizzazione_cpu_web_switch[i]);
            fprintf(fd_file, "cpu web switch: %9.9f\n",
                coda_cpu_web_switch);

            coda_cpu_web_server = calc_queue_length(
                utilizzazione_cpu_web_server[i]);
            fprintf(fd_file, "cpu web server: %9.9f\n",
                coda_cpu_web_server);

            coda_disco_web_server = calc_queue_length(
                utilizzazione_disco_web_server[i]);
            fprintf(fd_file, "disco web server: %9.9f\n",
                coda_disco_web_server);
```

```
        coda_ls1 = calc_queue_length(utilizzazione_ls1[i]);
        fprintf(fd_file, "LS1: %9.9f\n", coda_ls1);

        coda_ls2 = calc_queue_length(utilizzazione_ls2[i]);
        fprintf(fd_file, "LS2: %9.9f\n", coda_ls2);

        coda_lw2 = calc_queue_length(utilizzazione_lw2[i]);
        fprintf(fd_file, "LW2: %9.9f\n", coda_lw2);
}

//tempi di residenza
for(i=0; i < NUM_CLASSES; i++) {
        fprintf(fd_file, "\ntempi di residenza %d\n", i);
        tr_L2 = calc_response_time(D_LAN(doc_size[i]),
            utilizzazione_L2[i]);
        fprintf(fd_file, "L2: %9.9f\n", tr_L2);

        tr_inLink = calc_response_time(D_InLink(),
            utilizzazione_inLink[i]);
        fprintf(fd_file, "inLink: %9.9f\n", tr_inLink);

        tr_outLink = calc_response_time(D_OutLink(doc_size[i]),
            utilizzazione_outLink[i]);
        fprintf(fd_file, "outLink: %9.9f\n", tr_outLink);

        tr_cpu_web_switch = calc_response_time(D_Cpu(
            CPU_WEB_SWITCH_SERVICE_RATE)* 2,
            utilizzazione_cpu_web_switch[i]);
        fprintf(fd_file, "cpu web switch: %9.9f\n",
            tr_cpu_web_switch);

        tr_cpu_web_server = calc_response_time(D_Cpu(
            CPU_SERVICE_RATE) * 2, utilizzazione_cpu_web_server[i]);
        fprintf(fd_file, "cpu web server: %9.9f\n",
            tr_cpu_web_server);

        tr_disco_web_server = calc_response_time(D_WSDisk(doc_size[i
            ]), utilizzazione_disco_web_server[i]);
        fprintf(fd_file, "disco web server: %9.9f\n",
            tr_disco_web_server);

        tr_ls1 = calc_response_time((D_LS1in()+D_LS1out(doc_size[i])
            ),utilizzazione_ls1[i]);
        fprintf(fd_file, "LS1: %9.9f\n", tr_ls1);

        tr_ls2 = calc_response_time(D_LAN(doc_size[i]),
            utilizzazione_ls2[i]);
```

```
                fprintf(fd_file, "LS2: %9.9f\n", tr_ls2);

                tr_lw2 = calc_response_time(D_LAN(doc_size[i]),
                    utilizzazione_lw2[i]);
                fprintf(fd_file, "LS2: %9.9f\n", tr_lw2);
        }
        fclose(fd_file);
        return 0;
}

void sim(int argc, char **argv) {

}
```

Listings

Elenco delle tabelle

Elenco delle figure

Indice analitico