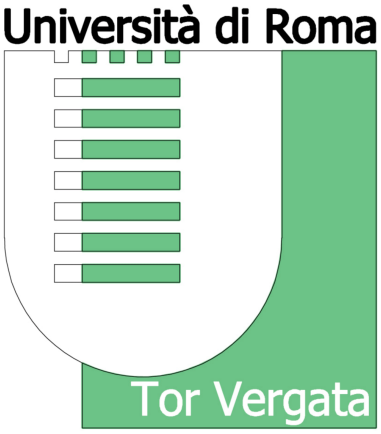
Modelli di Prestazioni di Sistemi e Reti

Simulatore di traffico in un sistema "multi-tier"

Simone Martucci - simone.martucci.91@gmail.com
Alessandro Valenti - alessandro.valenti1991@gmail.com



Indice

Capitolo 1

Generatore di Lehmer

Il progetto si pone di testare il noto generatore pseudo-casuale di numeri random di Lehmer utilizzando, a tale scopo, uno dei test di casualitá illustrati nel libro *Discrete-Event Simulation: A first course*. All'interno del progetto si é scelto di utilizzare la versione del generatore fornita come libreria C dallo stesso libro, é stato quindi creato un programma che si interfaccia con tali librerie, tramite chiamate alle API della libreria stessa, che serve a testare l'effettiva correttezza di tale implementazione. Al fine di comprendere al meglio i risultati ottenuti, verranno anche presentati dei grafici riassuntivi del test effettuato.

1.1 Funzionamento

Il *generatore di Lehmer* é un generatore basato su un algoritmo che dà origine ad una sequenza di numeri pseudo-casuali. É definito da due parametri:

- un modulo $\boldsymbol{m}$, che é un numero primo molto grande (in questo caso la libreria usa $2^{31} - 1$);
- un moltiplicatore $\boldsymbol{a}$ che rappresenta un numero intero compreso tra 1 ed $m - 1$.

La sequenza numerica pseudo-random viene generata tramite la formula :

$$x_{i+1} = ax_i mod m$$

Questa sequenza parte da un numero x_0 detto **seed**, anch'esso scelto tra 1 ed $m - 1$. Non tutte le combinazioni di a ed m peró sono ottimali per

realizzare una sequenza di numeri che garantiscano una buona randomicità. Per verificare dunque se un seed e un moltiplicatore garantiscono un buon livello di randomicità esistono dei test empirici. Nel paragrafo successivo tale generatore verrà sottoposto ad uno di questi.

1.2 Test degli estremi

Il test scelto per effettuare la verifica sul generatore, con parametri:

$$(a, m) = (48271, 2^{31} - 1)$$

é conosciuto come "*Test degli estremi*". Per la simulazione di tale test si può riassumere il processo in tre passi:

- Generazione di un campione di valori con chiamate ripetute al generatore.
- Computazione di un test statistico la cui distribuzione (pdf o funzione di densità di probabilità) é nota su variabili random uniformi in (0,1) indipendenti e identicamente distribuiti.
- Valutare la verosimiglianza del valore computato del test statistico con la relativa distribuzione teorica da cui é stato assunto adottando una metrica basata sulla distanza lineare.

Questo test si basa sulla seguente considerazione:

Teorema Se U_0, U_1,...,U_{d-1} é una sequenza di variabili *Uniform(0,1)* e se

$$\mathrm{R} = \max U_0, U_1, ..., U_{d-1}$$

allora la variabile U = R^d é una *Uniform(0,1)* [1].

In pratica tale test verifica che la variabilità delle altezze dell'istogramma prodotto dai numeri pseudo-casuali generati é sufficientemente piccola da poter concludere che questi numeri appartengano ad una popolazione *Uniform(0,1)*. Questa operazione é fondamentale in quanto tutte le altre distribuzioni di probabilità contenute all'interno della libreria utilizzata vengono generate a partire dalla *Uniform(0,1)*.

[1] Attenzione: il teorema afferma che la variabile U é una Uniform(0,1), mentre la variabile R non lo é.

1.3 Algoritmo

L'algoritmo del test degli estremi effettua un raggruppamento (*batching*) dei valori estratti dal generatore in gruppi di uguale lunghezza (determinato dal parametro d), trovando il massimo di ogni batch, elevando tale massimo all d-esima potenza e conteggiando tutti i massimi generati in un array. Tale funzione viene applicata per ogni stream del generatore di Lehmer. Tutto ciò é illustrato di seguito:

```
for(stream = 0; stream < 256L; stream++) {
    long o[K];
    memset(o, 0, K * sizeof(long));
    for(i = 0; i < N; i++) {
        double r = Random();
        for(j = 1; j < D; j++) {
            u = Random();
            if(u > r) r = u;
        }
        u = exp(D * log(r));
        x = u * K;
        o[x] ++;
    }
}
```

Per ogni stream, si determina quindi la variabile chi-quadro v tramite questa funzione:

```
    for(i = 0; i < K; i++) {
    v += (square(o[i] - e_x));
    }
    v /= e_x;
```

I valori critici v_1^* e v_2^* vengono calcolati utilizzando la funzione inversa *idfChisquare(long n, double u)* fornita dalla libreria rvms.c del libro di testo. Bisogna precisare che per il calcolo di tali variabili statistiche é stato scelto un livello di confidenza con parametro $\alpha = 0.05$, mentre i parametri N e K sono rispettivamente N = 100000 e K = N/20 = 5000. Successivamente si confronta la statistica chi-quadro v, determinata al passo precedente, con i valori critici v_1^* e v_2^* , per ogni stream (in totale sono 256 variabili chi-quadro v). Se $v < v_1^*$ o $v > v_2^*$ il test fallisce (per quello stream) con probabilitá $1 - \alpha$.

1.4 Test e conclusioni

Il grafico risultante di questo test empirico é illustrato di seguito:

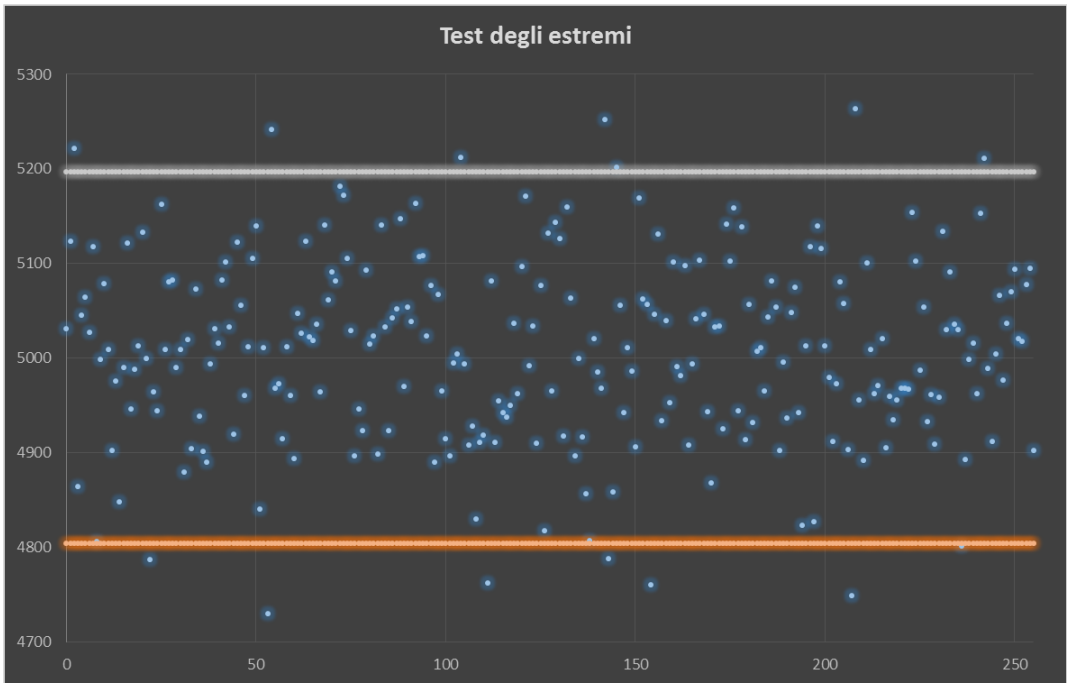


Figura 1.1: Test degli estremi

I valori critici sono visualizzati come linee orizzontali: quella inferiore rappresenta $v_1^* = 4804.92$ mentre quella superiore $v_2^* = 5196.86$.

Dalla simulazione effettuata si é notato che il numero di test statistici $v > v_2^*$ sono stati esattamente 7, come il numero di test $v < v_1^*$.

Di seguito é riportato l'output del programma:

```
Number of failed: 14
Number of passed: 242
Total number of tests: 256
```

Figura 1.2: Risultati

Considerando il numero totale di test falliti (*upper* e *lower bound*) pari a 14, si nota che non ci si discosta molto rispetto al valore atteso approssimato; infatti in 256 test con un livello di confidenza del 95% il valore atteso circa 256 * 0.05 = 13 fallimenti. Questo valore può essere una indicazione della bontà del generatore di Lehmer implementato.

Capitolo 2

Obiettivi

L'obiettivo del progetto é quello di modellare, pianificare e sviluppare un simulatore di traffico web che rispetti le specifiche di consegna.

Il sistema reale presenta le seguenti caratteristiche:

- un flusso di utenti che si connettono al sistema sotto forma di sessioni;
- un numero di richieste di cui si compone ogni sessione;
- un front-end server;
- un back-end server con un database.

Il simulatore verrà utilizzato per analizzare il comportamento stazionario relativo ad alcuni indici prestazionali quali il tempo di risposta del sistema, il throughput e la percentuale di sessioni abortite e rifiutate. Il sistema dispone infatti anche di un meccanismo di *gestione del sovraccarico* basato sul monitoraggio in tempo reale dell'utilizzazione del front-end.

Capitolo 3

Modello Concettuale

Il modello sviluppato é composto da un ramo principale (comprensivo di Front Server, Back-End Server e relative code) e da una componente di retroazione che si compone di un centro di Client, all'interno del quale gli utenti passano un certo tempo a pensare prima di effettuare la richiesta successiva.

Al sopraggiungere di una nuova sessione, questa verrà processata immediatamente dal Front Server, nel caso in cui la sua coda sia vuota, altrimenti verrà posta in attesa del proprio turno con un conseguente ritardo. Una volta che il Front Server ha elaborato tale sessione, quest'ultima verrà processata all'interno di un Back-End Server, se la coda di quest'ultimo é vuota, altrimenti potrebbe subire un certo ritardo dovuto all'attesa del proprio turno. Al termine di tale servizio la sessione uscirà dal sistema, nel caso in cui abbia completato tutte le richieste che la componevano, altrimenti verrà reindirizzata nuovamente all'ingresso del sistema attraverso una ramo di feedback.

In tale ramo la sessione permane in un centro di Client in cui l'utente può spendere del tempo per pensare alla sua richiesta successiva. Dopo tale attesa la sessione tenta di rientrare nel sistema per ricevere un ulteriore servizio, andandosi a posizionare alla fine della coda FIFO del Front Server, inoltre tale servizio é senza prelazione ed é conservativo.

Il sistema verrà regolamentato attraverso un meccanismo di *overload management* che permetterà di limitare i tempi di esecuzione della simulazione, nel caso della distribuzione peggiore. Questo sistema di gestione del carico consiste nel monitorare l'utilizzazione del Front Server, infatti appena raggiunto l' 85%, il sistema rigetterà tutte le richieste (sia le nuove in arrivo sia quelle già presenti nel sistema), fino a che l'utilizzazione non raggiunga una soglia inferiore al 75%. Tale tipologia di simulazione garantisce una notevole

semplicità di gestione dell'intero sistema attraverso la facilità di avanzamento del tempo ed il controllo delle diverse tipologie di eventi che occorrono durante le varie esecuzioni.

3.1 Variabili di Stato

Il sistema é descritto completamente dalle seguenti variabili di stato:

- $queue\_length_{fs}$= numero di richieste nella coda del Front Server
- $queue\_length_{bes}$= numero di richieste nella coda del Back-End Server
- $client\_counter$= numero di client attivi in un dato istante di tempo

3.2 Modello delle specifiche

Nello sviluppo del modello delle specifiche, l'attenzione é stata rivolta alla definizione dei modelli di input da utilizzare nel modello di simulazione. Tali modelli sono stati definiti in base alle specifiche fornite nel seguente modo:

- $\lambda_{sessioni} = 35\frac{richieste}{s}$ (distribuito esponenzialmente)
- $Dimensione_{sessioni} \sim Equilikely(5, 35)$
- $E[Z] = 7s$ (distribuito esponenzialmente)
- $E[D]_{front-end} = 0.00456s$ (distribuito esponenzialmente)
- $E[D]_{back-end} = 0.00117s$ (distribuito esponenzialmente)

3.3 Eventi

Dopo aver prodotto un modello delle specifiche sono stati identificati gli eventi generati dal sistema, durante la simulazione:

- **NewSession**: una nuova sessione entra nel sistema. Ovvero verrà inserita nel Front Server o nella sua coda qualora quest'ultimo fosse già occupato;
- **FS_Completion**: il Front Server evade una richiesta e la invia al Back-End Server;

- **BES_Completion**: il Back-End Server evade una richiesta. La sessione corrispondente può dunque essere completata del tutto e quindi uscire dal sistema oppure migrare verso il centro Client nel caso in cui il suo numero di richieste sia non nullo.

- **Client_Completion**: dopo aver passato un certo tempo in fase di *Thinking*, una sessione lascia il centro Client e rientra nel sistema attraverso il ramo di retroazione, quindi é direttamente inserita nella coda del Front Server.

Capitolo 4

Modello Progettuale

4.1 Architettura

Il diagramma del sistema da implementare può essere rappresentato semplicemente come in figura 4.1

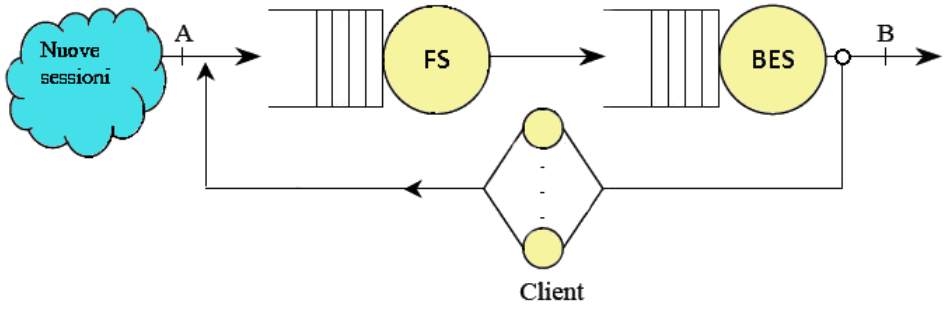


Figura 4.1: Rappresentazione grafica dell'architettura del sistema.

All'inizio della simulazione il primo evento che si verifica é sempre l'arrivo di una nuova sessione (evento NewSession).

I client del centro delle sessioni attive, all'interno del ramo di retroazione sono paralleli ed infiniti, definendo un infinity server.

I nuovi eventi in arrivo verso il sistema verranno posti in coda al Front Server, nel caso in cui non possano essere serviti. Come é visibile dalla figura precedente, la sessione una volta servita dal Front Server verrà posta in coda

verso il Back-End Server finché non verrà servita da quest'ultimo. Infine le sessioni saranno completate oppure poste nella zona di "*Thinking*" finché non verranno reinserite nella coda del Front-End in attesa di essere serviti, passando attraverso un ramo di feedback.

4.2 Clock di simulazione e schedulazione di Eventi

Nella fase implementativa si tiene conto dell'avanzamento del tempo per mezzo della variabile *current_time*. Il meccanismo di avanzamento del tempo scelto é il *Next-Event Time Advance*. Questa scelta garantisce che gli eventi occorrano nella sequenza corretta, ovvero vengono processati in ordine crescente rispetto al tempo di schedulazione. Si utilizza, inoltre, il flag di *arrivals* per regolare l'accettazione delle nuove sessioni, necessaria per il meccanismo di overload management: se impostata a zero vengono inibiti i nuovi arrivi[1], altrimenti si procede normalmente con la simulazione.

4.3 Event List

Per la gestione degli eventi si utilizza una lista collegata di strutture *Event*, come quella mostrata in figura, salvate in ordine crescente rispetto al tempo. Ogni nodo contiene il tempo di occorrenza e la sua tipologia. Un gestore di eventi é utilizzato per il demultiplexing di tale lista facendo uso di una funzione pop() per ottenere il *Next-Event* da processare.

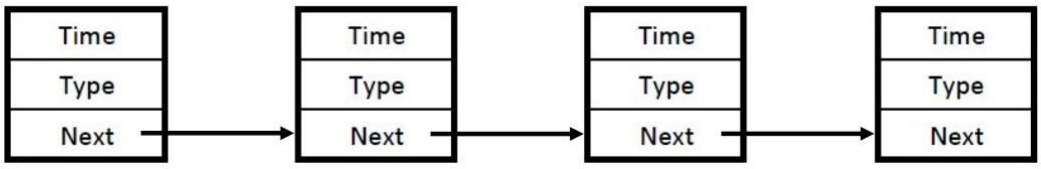


Figura 4.2: Struttura lista eventi

[1]Ovvero viene eseguito il drop della sessione in entrata o l'abort se la sessione é già nel sistema

4.4 Arrival Queue

Al fine di ottenere informazioni riguardo i tempi di attesa che le sessioni sperimentano durante la loro permanenza nel sistema, si utilizzano delle strutture dati atte a registrare tali informazioni ed utilizzate negli algoritmi del calcolo delle medie. Tali strutture, denominate *ArrivalQueue*, immagazzinano i tempi di arrivo delle sessioni nelle sottosezioni del sistema (ovvero quando una sessione entra nel *Front Server* o nella sua coda, quando entra nel *Back-End Server*, e così via). Ad ogni completamento sperimentato da una sessione viene utilizzata la coda relativa e si calcola, per differenza, il tempo effettivo che la sessione ha passato in quella parte di sistema. Tutto ciò é possibile grazie all'ipotesi che l'ordine di arrivo, all'interno delle code del sistema, é sempre `preservato`. Infatti la prima sessione ad entrare nella coda del Front Server, ad esempio, sarà la prima a lasciarlo.

4.5 Request Queue

Dal momento che é impossibile identificare una singola richiesta utilizzando la Next-Event Simulation, il problema di preservare l'informazione riguardante il numero di richieste attive di cui si compone una sessione viene risolto con la struttura dati *Request Queue*.

Ad ogni richiesta completata si decrementa il contatore delle richieste relative a quella sessione in modo da propagare tale informazione a tutte le richieste future. Quando il contatore arriva a zero la sessione viene completata del tutto e di conseguenza abbandona il sistema.

4.6 Client Order List

Per preservare una Next-Event Simulation priva di contaminazioni derivanti dall' aggiunta di dati identificanti le sessione o le richieste all'interno degli eventi di base, la Client Order List permette di conservare le informazioni relative all'ordine di arrivo e di uscita degli utenti all'interno del centro Client. Attraverso questa informazione é possibile gestire il corretto ordine degli elementi della Request Queue nonostante siano condizionati da una mancanza di determinismo circa l'ordine di completamento degli utenti durante il loro periodo di Think Time.

4.7 Personalizzazione del modello

In base alle scelte effettuate dall'utente nella fase di *setup* é possibile decidere di avviare la simulazione:

- senza **Overload Management**
- con **Overload Management**

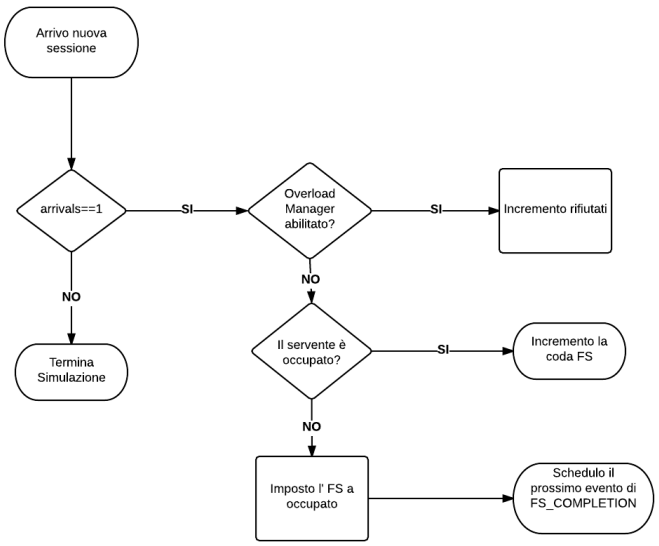
É inoltre possibile scegliere quale distribuzione utilizzare tra:

- *Esponenziale*
- *10-Erlang*
- *Iperesponenziale*

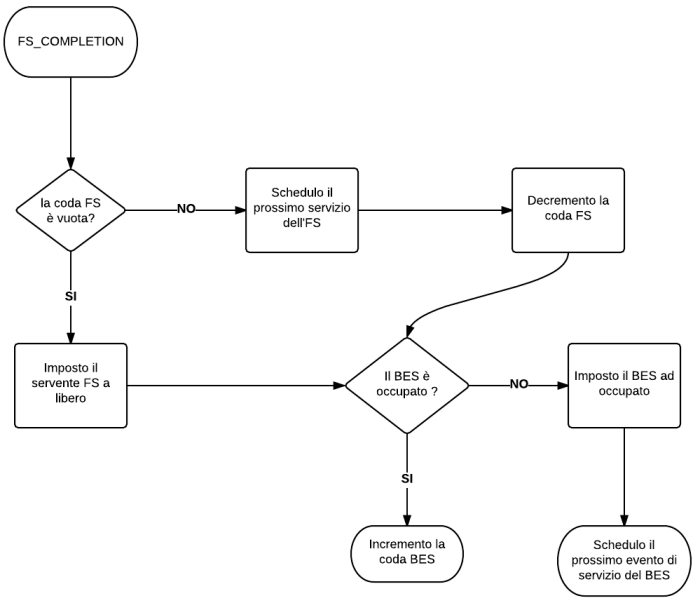
Successivamente l'utente può scegliere se effettuare un test con il metodo dei batch means oppure un'unica lunga simulazione. Se viene scelto il primo metodo verrà chiesto di inserire i parametri riguardanti il *numero di batch* e la *grandezza del batch* stesso; altrimenti sarà necessario inserire altri dati, quali, l'istante finale della simulazione, ossia lo *STOP*, e ogni quanto tempo campionare i dati durante il "run", ovvero lo *STEP*.

4.8 Algoritmi di Gestioni eventi

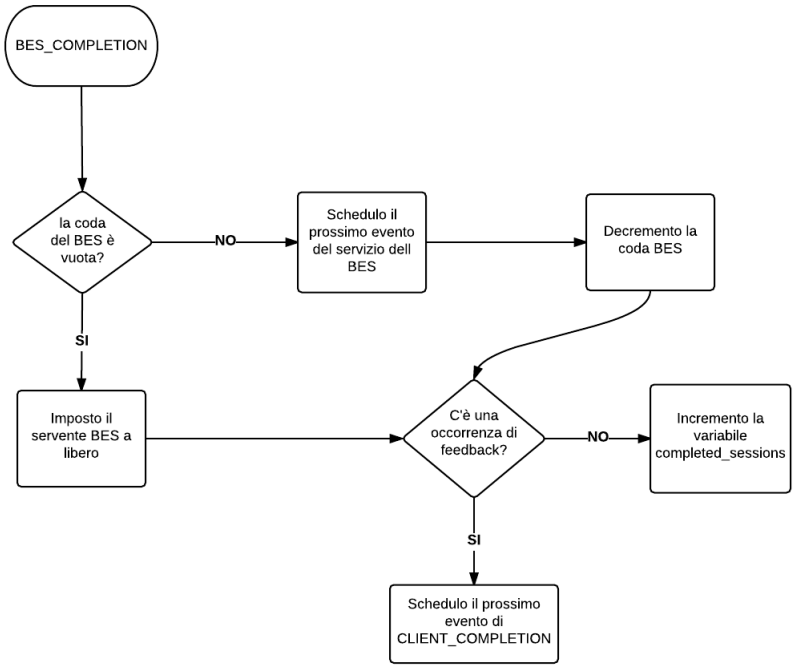
4.8.1 NewSession



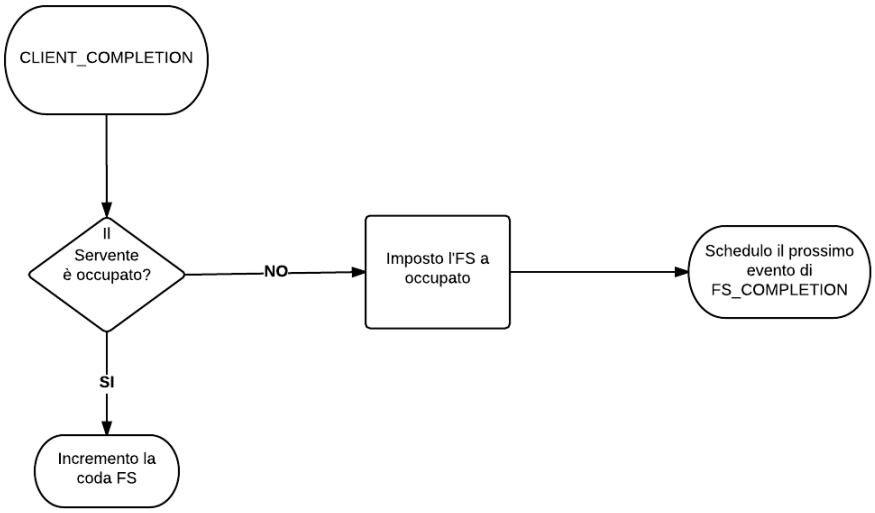
4.8.2 FS Completion



4.8.3 BES Completion



4.8.4 Client Completion



4.9 Indici di prestazioni

Il simulatore qui utilizzato genera un insieme di statistiche che permettono di ricavare informazioni utili per comprendere il comportamento del sistema. Gli indici di prestazione calcolati sono:

- **Useful Troughput**: indica il totale delle sessioni completate nell'unità di tempo. Tale indice viene calcolato attraverso il rapporto tra le sessioni totali processate dal sistema e l'intervallo di tempo necessario per ultimare questo compito. Il troughput basato sulle sessioni é un ottimo indice per valutare il numero di utenti serviti dal sistema in un intervallo dato, risultando una misura sensibile per l'utente finale.

- **Tempo di Risposta del Sistema**: viene inteso come la somma del tempo di risposta: Tempo in coda + Tempo di Servizio. In pratica il tempo di risposta é inteso come il tempo che intercorre tra l'istante in cui una richiesta entra nel Front Server e l'uscita della stessa dal Back-End Server.

- **Drop Ratio**: misura il rapporto tra il totale delle sessioni rifiutate dal sistema ed il numero di sessioni totali che tentano di entrare nel sistema (accettate + rifiutate).

- **Abort Ratio**: é il rapporto tra le richieste abortite ed il totale di richieste processate dal sistema. Questo é dovuto al fatto che una richiesta può rientrare nel sistema, tuttavia in condizioni di saturazione, tale richiesta non è in grado di poter rientrare all'interno del Front Server, quindi viene appunto abortita.

Capitolo 5

Modello Computazionale

Il programma realizzato é composto da un file eseguibile (simulatore.c) e di alcuni file dove sono contenute funzioni di appoggio. (*event_list*, *arrival_queue*, *autocorrelation*, *client_req*, *req_queue*). Il software sviluppato é codificato con il linguaggio *C*.

5.1 Simulatore.c

Questo applicativo é il cuore del simulatore implementato. Tale programma sfrutta un'interfaccia testuale per interrogare l'utente circa la configurazione da adottare per la simulazione da eseguire. É possibile selezionare diverse opzioni:

- Scegliere la distribuzione da testare;
- Scegliere se attivare o meno il meccanismo di gestione dell'overload;
- Scegliere il seed per la generazione di numeri random;
- Scegliere il tipo di simulazione e i parametri associati;
- Scegliere se visualizzare lo stato della simulazione live.

I risultati prodotti dalla simulazione vengono trascritti su un file di tipo "*.csv*" in modo da dare all'utente una chiara ed equilibrata visione dei dati ottenuti.

5.2 Event List

La lista di eventi é costituita da strutture di tipo *Event*, formate da un campo di tipo *double*, che rappresenta il tempo di occorrenza dell'evento, un *_EVENT_TYPE* che identifica il tipo di evento ed un puntatore *next* alla struttura seguente. Per merito della funzione **add_event()** é possibile aggiungere eventi alla lista. Verranno inseriti seguendo un ordinamento crescente rispetto alla variabile time. Durante l'inserimento dei dati viene effettuato un controllo sulla consistenza degli stessi, cioé si controlla, con una funzione **event_check()**, che il tempo sia un valore positivo e che il tipo di evento appartenga all'insieme degli eventi noti (es: `nuova sessione`, `fs completition`, ...). Gli eventi vengono estratti dalla struttura attraverso la funzione **pop_event()**, che preleva l'elemento in testa alla lista, restituendolo alla funzione chiamante.

5.3 Arrival Queue

Definisce una struttura dati, utilizzata per modellare le code sia del Front Server che del Back-End Server. Ogni coda associata ad un centro si compone di un tempo di arrivo e un puntatore all'elemento successivo. Le funzioni messe a disposizione per questa struttura dati sono: l'**arrival_add()** che aggiunge un nuovo elemento posizionandolo in fondo alla coda, l'**arrival_pop()** che permette l'estrazione dell'elemento in testa e l'**arrival_print()** che stampa lo stato della coda.

5.4 Request Queue

Il sistema all'arrivo di una nuova sessione genera in maniera random, con distribuzione esponenziale, un numero di richieste comprese tra 5 e 35. Per evitare di memorizzare tale informazione all'interno della sessione stessa é stata creata, questa struttura dati adibita a raccogliere tali parametri. Non appena la sessione arriva nel primo centro, ossia il Front Server, il dato contenente il numero di richieste associate, viene inserito in questa coda tramite la funzione **enqueue_req()**. Quando invece la sessione esce dal Back-End Server é necessario l'utilizzo della funzione di **dequeue_req()** poichè questa informazione non verr più memorizzata in questa lista, ma verrà salvata in una diversa struttura, nel caso la sessione non sia completata. Per verificarne la consistenza é stata implementata la funzione **print_req()**.

5.5 Client request

Questa struttura dati ci viene in aiuto nel momento in cui la sessione esce dal Back-End Server per entrare nel centro di client. In questo caso infatti non vengono più utilizzate le strutture precedentemente descritte, quali *Request Queue* e *Arrival Queue*, per modellare il comportamento del sistema, poichè durante la fase di "thinking" l'ordine di entrata delle sessioni non corrisponde a quello di uscita. Di conseguenza ´ stata creata questa struttura, che al suo interno contiene sia il tempo di arrivo della sessione nel centro, sia il numero di richieste rimanenti al suo completamento, per garantire una corretta corrispondenza tra la sessione ed il suo numero di richieste. Le funzioni di cui dispone questa struttura, sono: **add_client_req()**, **pop_ClientReq()** e **print_client_req()**, esattamente identiche a quelle delle strutture precedenti.

5.6 File Manager

La parte relativa al file manager gestisce tutto il flusso di dati da trascrivere su un file di output. Composto da tre funzioni:

- **get_date()**: permette di ottenere l'orario e la data correnti, da salvare sul file desiderato.
- **open_file()**: utilizzata per la creazione e l'apertura del file da salvare. Il nome del file viene creato sfruttando la funzione **get_date()**. Quindi il file contiene la data corrente della creazione più il tipo di distribuzione scelta durante la fase di setting.
- **close_file()**: adottata per chiudere il file una volta terminata la scrittura su di esso.

5.7 Utils

Il file **utils.h** contiene delle funzioni utilizzate per la pulizia della console e alcune funzioni per la manipolazione dei dati ottenuti al termine della simulazione.

Capitolo 6

Verifica

La fase di verifica é molto importante poiché consente di dimostrare la consistenza del programma creato con il modello delle specifiche.

In primis, é stata utilizzata una funzione di stampa per verificare il corretto flusso delle sessioni all'interno dell'intero sistema. Tale funzione, la `print_system_state()`, stampa le statistiche più importanti del sistema in tempo reale su standard output.

In secundis, si é notato e verificato che le liste contenenti le sessioni e le richieste si riempiono e si svuotano in modo corretto. Anche in questi casi sono state necessarie funzioni di stampa per verificare che l'aggiornamento fosse adeguato.

In terzis, i vincoli sullo stato del sistema e sull'entrata in azione dell'overload manager, e sul calcolo delle medie sono tutti soddisfatti.

Il simulatore parte con il numero di sessioni nullo e tutte le variabili di stato e di supporto sono inizializzate opportunamente. Il sistema consente di avere una stampa aggiornata di tutti i parametri rilevanti, come throughput e tempo medio di risposta.

Il numero di sessioni rifiutate e abortite cresce in maniera consistente con il meccanismo di controllo delle ammissioni.

Infine, come ultima verifica, é stato dimostrato che superato il numero massimo di sessioni per batch, che l'utente ha inserito manualmente, il sistema passa a simulare il batch successivo, e una volta raggiunto il numero massimo di batch, impostati sempre dall'utente, il programma termina correttamente la propria esecuzione.

Capitolo 7

Validazione

7.1 Modello Analitico

Al fine di prevedere, in linea di massima, i risultati del simulatore, viene elaborato un modello analitico semplificato per poter studiare il sistema preso in esame. Il sistema implementato presenta numerosi vincoli ed una complessità insita nelle specifiche, a tal proposito viene proposto un modello volontariamente e lievemente differente dal caso reale.

Il sistema esaminato risulta essere interattivo, in quanto avviene uno scambio di richieste tra Client e Server, con un comportamento a rete aperta (le nuove sessioni possono entrare nel sistema qualora questo non fosse saturo), a tal proposito si é deciso di presentare due modelli differenti tra loro, uno a rete aperta ed uno a rete chiusa. Questi permettono di descrivere in dettaglio la maggior parte degli elementi che costituiscono il sistema stesso.

7.2 Modello semplificato a rete aperta

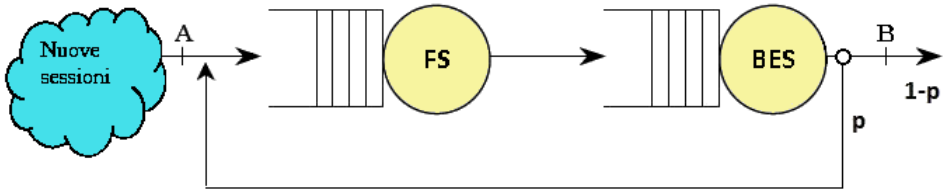


Figura 7.1: Modello a rete aperta

In questo primo modello si ha una rete aperta di Jackson con un tasso di sessioni in arrivo pari a λ . Si é deciso così, per questioni di semplificazione del modello, di non inserire il centro di Client, in quanto risultavano di difficile analisi in un modello a rete aperta come quello di Jackson.

$$\lambda = 35 sessioni/s$$

$$\begin{cases} y_1 = \lambda + p y_2 \\ y_2 = y_1 \end{cases} \rightarrow \begin{cases} y_1(1-p) = \lambda \\ y_2 = y_1 \end{cases} \rightarrow \begin{cases} y_1 = \frac{\lambda}{(1-p)} \\ y_2 = \frac{\lambda}{(1-p)} \end{cases}$$

$$y_1 = y_2 = \frac{\lambda}{(1-p)}; p = \frac{19}{20} \rightarrow y_1 = y_2 = 35 \times 20 = 700 \text{ richieste/s}$$

$$y_1 \gg \mu_{FS};\ y_2 = \mu_{FS} \text{ poichè } \mu_{BES} \gg y_1$$

Viene quindi calcolato il throughput, ovvero il numero di sessioni che escono dal sistema al secondo:

$$y_2 = \frac{\lambda}{(1-p)} \rightarrow \lambda = y_2(1-p) = \frac{\mu_{FS}}{20} \approx \frac{219}{20} \text{ richieste/s} = X_{sessioni}$$

Per quanto concerne l'indice riguardante la percentuale di sessioni rifiutate dal sistema, si trova:

$$dropped = \frac{\#sessioniaccettate}{\#totalearrivi} \rightarrow \frac{35 - X_{sessioni}}{35} = 1 - \frac{2}{7} \approx 0.7 \approx 70\%$$

Capitolo 8

Progettazione Esperimenti e Simulazioni

Tutte le simulazione sono state effettuate su diversi computer, sia Pc desktop che notebook, sfruttando la portabilità del software in modo tale da generare i dati sia su sistemi operativi *Windows* che *Linux*. La simulazione viene testata con parametri inseriti dall'utente manualmente, permettendo una facile impostazione del programma. Sono state sfruttate macchine aventi processori multi-core al fine di eseguire più esecuzioni in parallelo.
Gli esperimenti eseguiti sono stati i seguenti:

1. Simulazione del sistema quando il Front Server ha distribuzione esponenziale;
2. Simulazione del sistema quando il Front Server ha distribuzione 10-Erlang;
3. Simulazione del sistema quando il Front Server ha distribuzione Iperesponenziale;
4. Simulazione del sistema nel caso peggiore tra i precedenti con overload management attivo.

Gli esperimenti sono stati basati sui seguenti parametri nel caso del sistema instabile:

- ***Seed*** : indica il seed scelto dall'utente nella fase di setup;
- ***STOP*** : indica il termine ultimo della simulazione.

- ***Threshold*** : indica se l'overload management é attivo.

Per quanto riguarda invece il sistema in "*steady state*" si é scelto di usare il meccanismo dei batch means, e quindi i parametri utilizzati sono stati:

- ***Seed*** : indica il seed scelto dall'utente nella fase di setup;
- ***Batch size*** : indica quante sessioni compongono un batch;
- ***Batch number*** : indica quanti batch devono essere eseguiti in totale;
- ***Threshold*** : indica se l'overload management é attivo.

Per ogni simulazione, dello stesso tipo, si é scelto di impostare la grandezze caratteristiche con lo stesso valore in modo da poter effettuare un miglior confronto tra i diversi casi. Al termine di ogni *simulazione* i dati ottenuti dalle metriche considerate vengono salvati su un file "*csv*".

Capitolo 9

Analisi dei Risultati

I dati relativi agli indici di prestazione di interesse (tempi di risposta e throughput) misurati nelle precedenti fasi di test sono stati in seguito aggregati e visualizzati in forma di grafici. Infatti, in base al tipo di test effettuato, tali grafici si suddividono in due categorie: i grafici di throughput e tempi di risposta del sistema in stato di instabilità (front server sovraccarico) e quelli relativi al sistema in condizione di stazionarietà (front server non sovraccarico), ovvero quelli gestiti attraverso il meccanismo di overload management.

9.1 Sistema senza Overload Management

9.1.1 Response Time

In questo scenario l'utilizzazione del front-end server é praticamente uguale a 1, perciò non riesce a completare tutte le richieste entranti, con l'inevitabile situazione di veder aumentare indefinitamente la lunghezza della sua coda. Di conseguenza il tempo di risposta del sistema tende a divergere all'aumentare del tempo di simulazione. Il grafico illustrato di seguito mostra tale scenario:

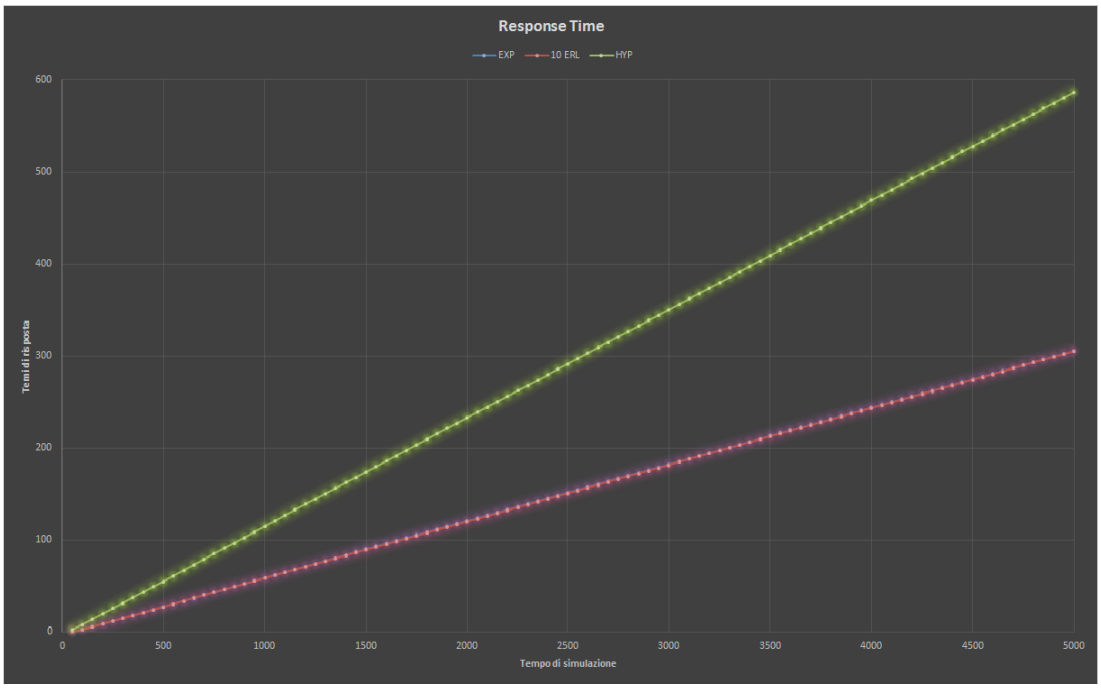


Figura 9.1: Tempi di risposta del sistema instabile

Come si intuisce dal grafico sopra riportato, i tempi di risposta si contraddistinguono in base al tipo di distribuzione dei tempi di servizio del front-end server (iperesponenziale, 10-erlang ed esponenziale), e in base all'andamento delle curve dei tempi di risposta, risulta evidente come la distribuzione iperesponenziale dei tempi di servizio del front-end risulta peggiore che nel caso esponenziale e 10-erlang, dove i tempi rimangono uguali.

Questa sostanziale differenza è dovuta al fatto che, nel caso iperesponenziale, avendo preimpostato la probabilit **p=0.1**, la varianza dei tempi di servizio delle richieste risulta essere molto elevato con la conseguenza di rallentare notevolmente il front-end server. Di conseguenza la curva dei tempi di risposta della iper-esponenziale diverge più rapidamente rispetto alle controparti 10-erlang ed esponenziale. Tuttavia il grafico mostra anche un aspetto insolito: la curva del tempo di risposta della 10-erlang coincide praticamente con la curva dell'esponenziale anche avendo impostato un parametro K=10, mentre ci si aspettava, al contrario, un miglioramento dei tempi di risposta rispetto alla curva della esponenziale. Probabilmente la scelta del parametro K pari a 10 é insufficiente a garantire un miglioramento significativo.

9.1.2 Autocorrelazione

L'evidente divergenza dei tempi di risposta del sistema é evidenziata anche dalla forte correlazione presente dai tempi di risposta delle singole richieste presenti nel sistema. Il grafico seguente illustra tale correlazione sulle tre distribuzioni:

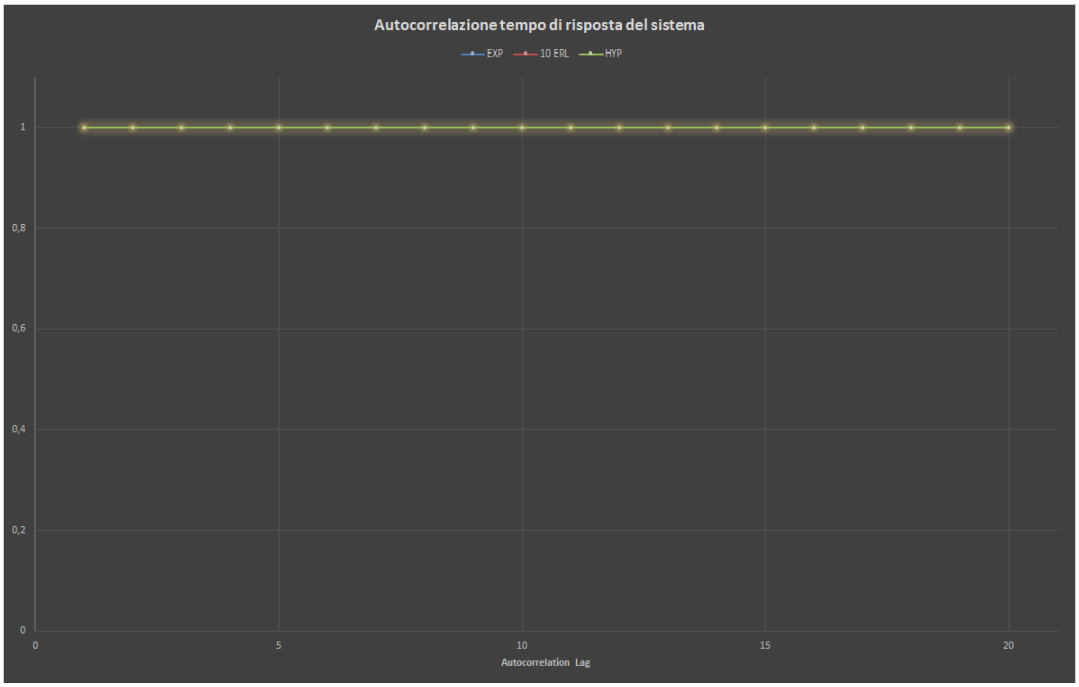


Figura 9.2: Autocorrelazione dei tempi di risposta

Dal grafico non si evince bene ma i valori dell'autocorrelazione relativi alla distribuzione iperesponenziale sono leggermente inferiori rispetto agli altri due casi (nell'ordine di 10^{-2}) ma si assestano tutti nell'intorno di 0.9.

9.1.3 Useful Throughtput

Il grafico successivo mostra l'andamento dello useful throughput i cui valori sono stati misurati dagli stessi test usati per il tempo di risposta del sistema. Anche in questo caso si distinguono le diverse distribuzioni dei tempi di servizio del front-end server:

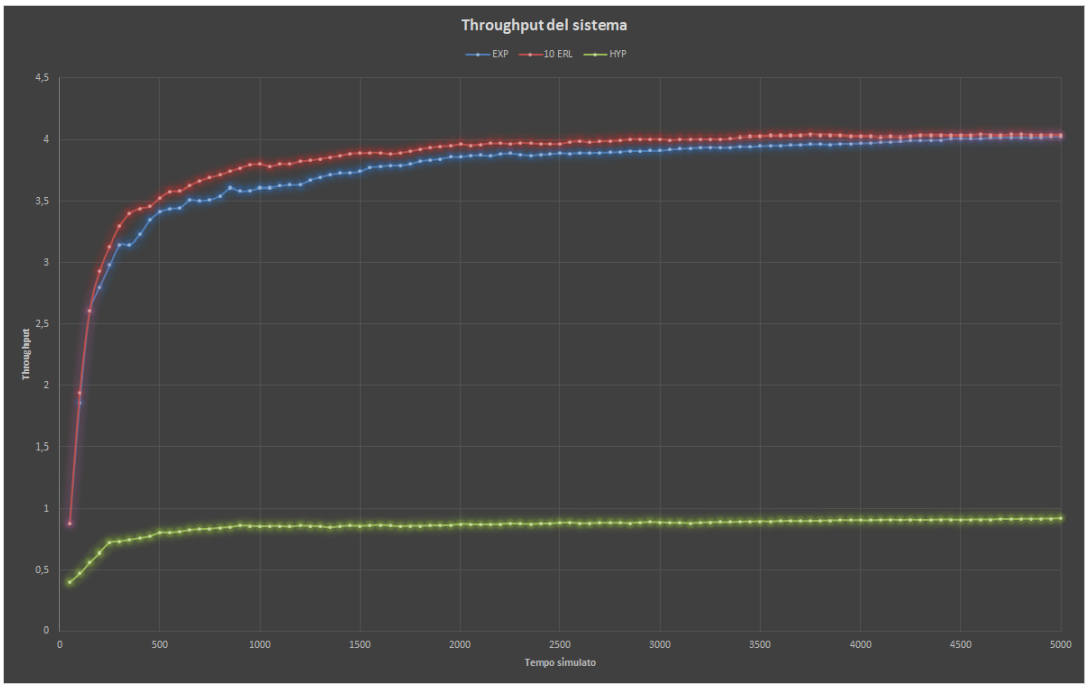


Figura 9.3: Throughput del sistema instabile

Anche in questo caso si nota la notevole differenza dello useful throughput del caso iperesponenziale da quelli esponenziali e 10-erlang, i quali coincidono per valori di run simulativi molto alti. Lo useful throughput é l'unico indice di prestazione che presenta un andamento stazionario all'aumentare del tempo di simulazione. Si può notare dal grafico, infatti, che tale valore di stazionarietà é all'incirca pari a 4 sessioni completate per unità di tempo.

Di seguito sono stati riportati gli intervalli di confidenza stimati per ogni distribuzione.

- ***Esponenziale*** : $[3,66109876; 3,83534225]$
- ***10 Erlang*** : $[3,75592814; 3,92657294]$
- ***Iperesponenziale*** : $[0,84149673; 0,87406350]$

Infine viene riportato l'istogramma relativo allo useful throughput dato che l'unico indice con comportamento steady-state:

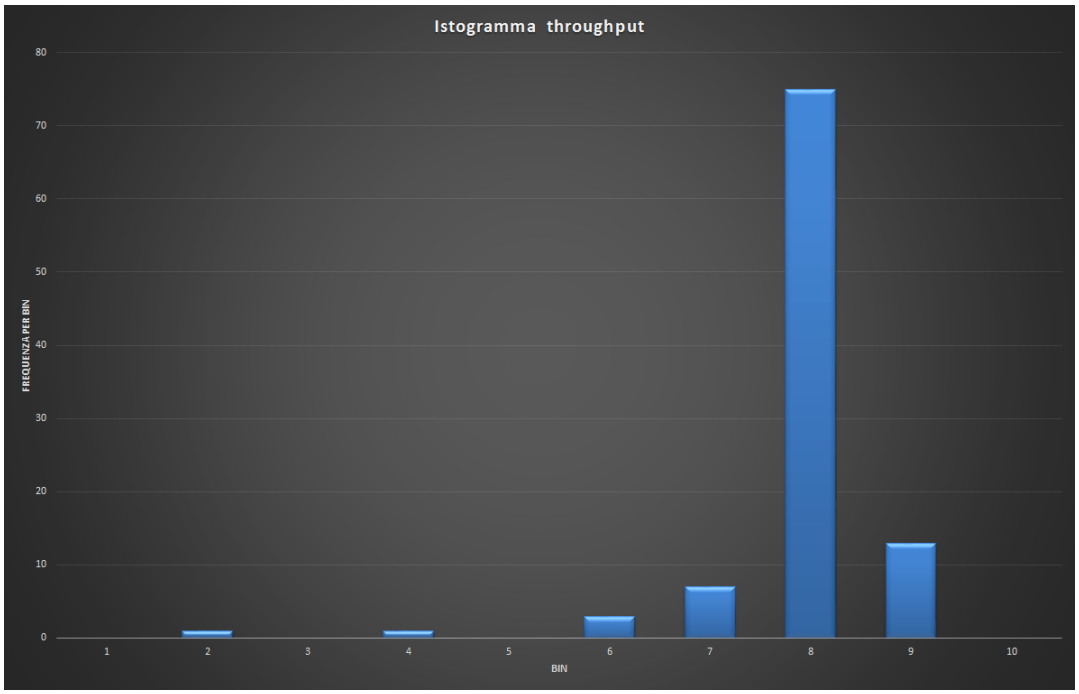


Figura 9.4: Istogramma del throughtput

9.2 Sistema con Overload Management

Dai risultati illustrati nel caso di sistema senza il meccanismo di *overload management* la distribuzione con i tempi di servizio "peggiore", ossia con i tempi di risposta maggiori, é risultata quella iperesponenziale. Per questa distribuzione, dunque, sono stati effettuati test con la gestione del sovraccarico. Neanche quando viene applicato l'overload management nel front-end server, gli indici di prestazione di interesse assumono un comportamento stazionario per run simulativi abbastanza lunghi. Ciò avviene perché anche se in questa situazione, il front server ha un'utilizzazione media più bassa di 1, e quindi non arriva mai a saturazione, nel momento in cui supera l' 85% del carico viene impedito l'accesso a tutte le sessioni, anche quelle già presenti nel sistema, e di conseguenza la sua coda si svuota, ma nel momento in cui vengono riabilitati gli arrivi la coda si riempie di nuovo, con l'effetto che tutte le metriche analizzate avranno oscillazioni non indifferenti rispetto al tempo.

9.2.1 Response Time

Di seguito viene riportato il grafico sul tempo di risposta del sistema con distribuzione dei tempi di servizio iperesponenziale del front-end server:

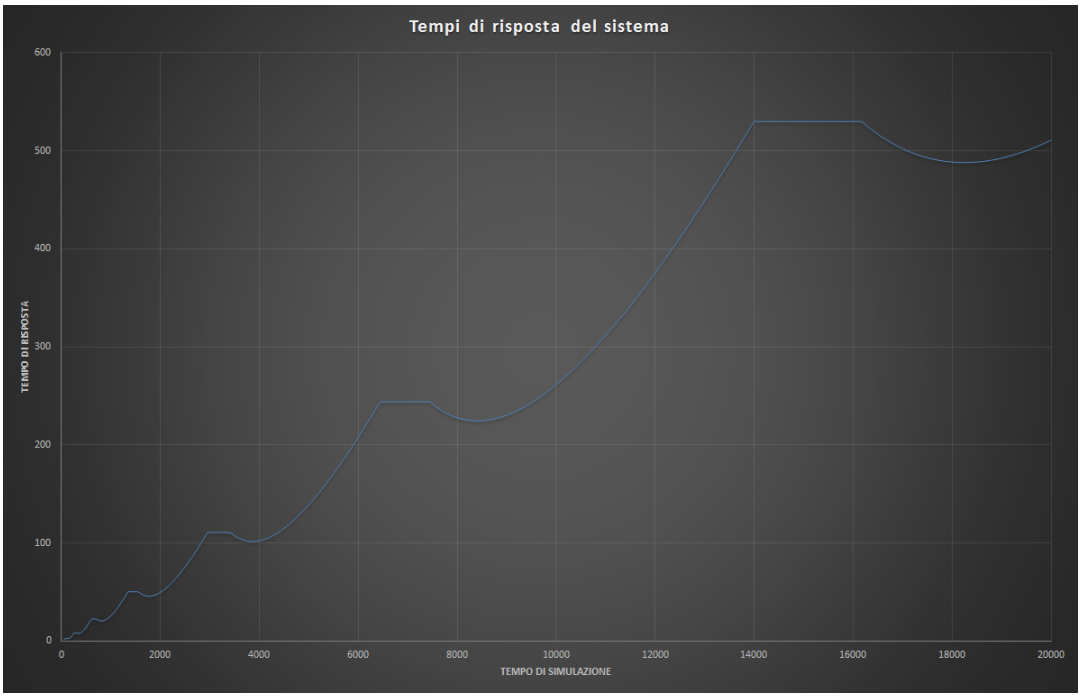


Figura 9.5: Tempo di risposta del sistema con distribuzione iperesponenziale e overload management attivo

Dal grafico riportato sopra, si nota ciò che era stato detto in precedenza, ovvero il response time non si stabilizza, ma tende a decrescere nei momenti in cui il sistema blocca gli accessi ed a crescere di nuovo quando vengono nuovamente abilitati.

9.2.2 Useful Throughtput

Il grafico successivo illustra l'andamento dello useful throughput del sistema con i parametri sopra citati:

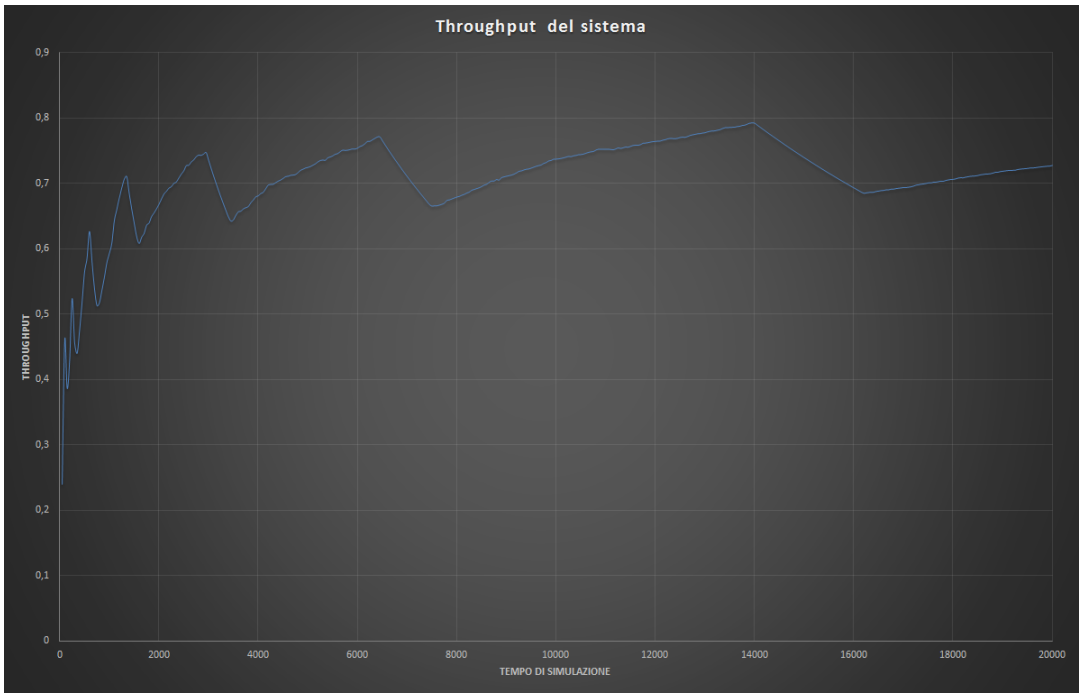


Figura 9.6: Throughput del sistema con distribuzione iperesponenziale e overload management attivo

Come è possibile notare, nell'analisi del throughput avviene un comportamento conforme con quello osservato per i tempi di risposta, anche se in questo caso a poco a poco le oscillazioni si vanno ad appiattire.

9.2.3 Drop e Aborted Ratio

Infine vengono riportati gli andamenti degli indici *drop ratio* e *aborted ratio*:

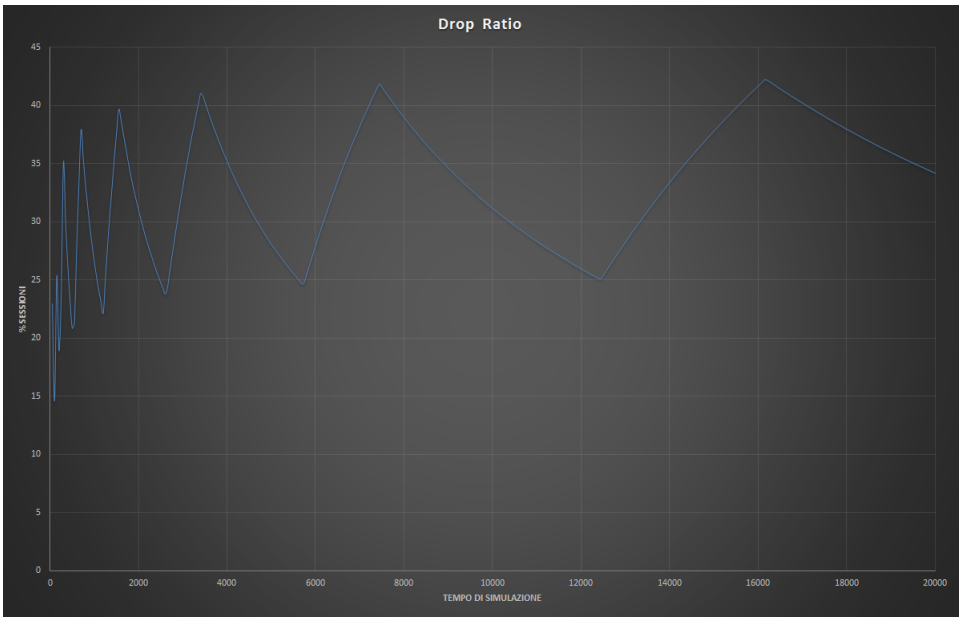


Figura 9.7: Drop Ratio

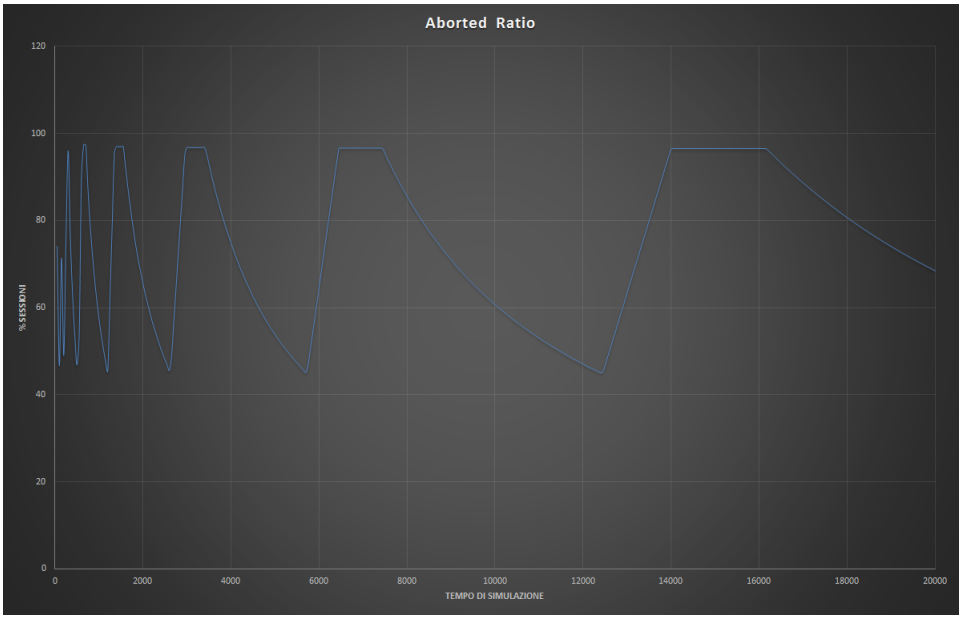


Figura 9.8: Aborted Ratio

Anche in questi due grafici notiamo come la percentuale di sessioni rifiutate e abortite oscilli in maniera molto ampia ed evidente.

9.3 Batch means e intervalli di confidenza

Non avendo raggiunto la stazionarietà neanche attraverso il meccanismo di overload management attivo, si é scelto di eseguire gli stessi test sfruttando il metodo dei batch means. Questo metodo consiste nell'effettuare un unico lungo run e partizionare i dati raccolti, relative alle statistiche di interesse in partizioni (batch) di uguale lunghezza, calcolare le medie di ciascun batch, e successivamente calcolare la media delle medie dei batch, ricavando anche la deviazione standard relativa per poter costruire gli intervalli di confidenza per la media. Particolare attenzione é stata prestata nella scelta dei parametri (b,k) necessari per determinare l'ampiezza e il numero dei batch. Seguendo le linee guida del libro si é scelto di impostare un k=64 fisso, mentre b stato determinato dal rapporto b=n/k. Il valore calcolato di b risultato ottimale anche controllando l'andamento della funzione di autocorrelazione (tendente a zero):

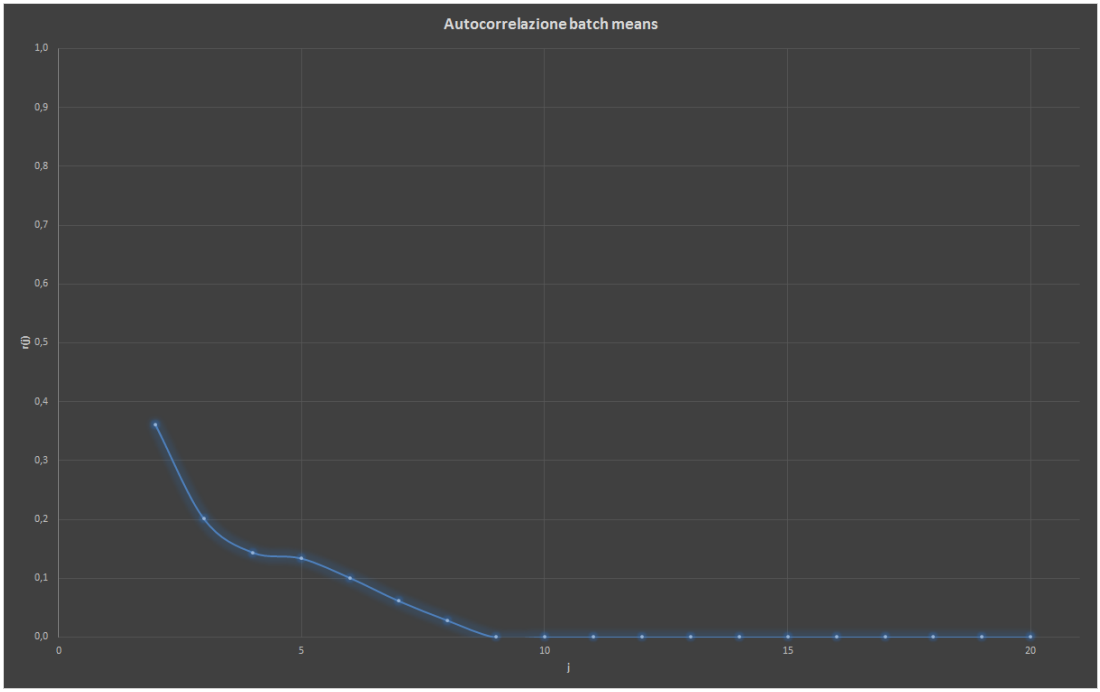


Figura 9.9: Autocorrelazione batch means

Nell'analisi effettuata in questo caso si può osservare, dal grafico successivo, come il tempo di risposta del sistema si stabilizzi e di conseguenza é risultato possibile calcolare un intervallo di confidenza al 95%.

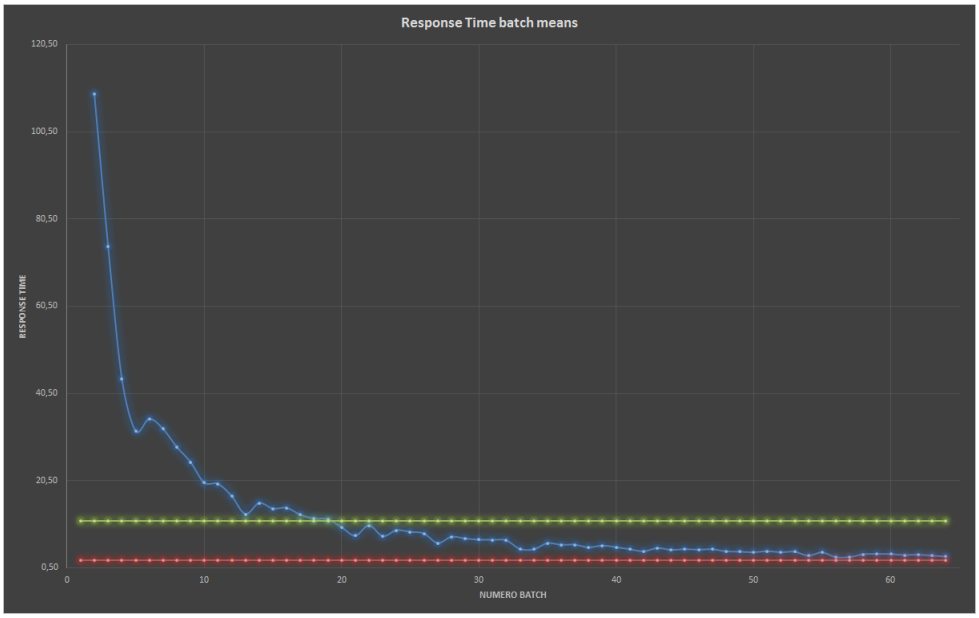


Figura 9.10: Response time nel caso dei batch means

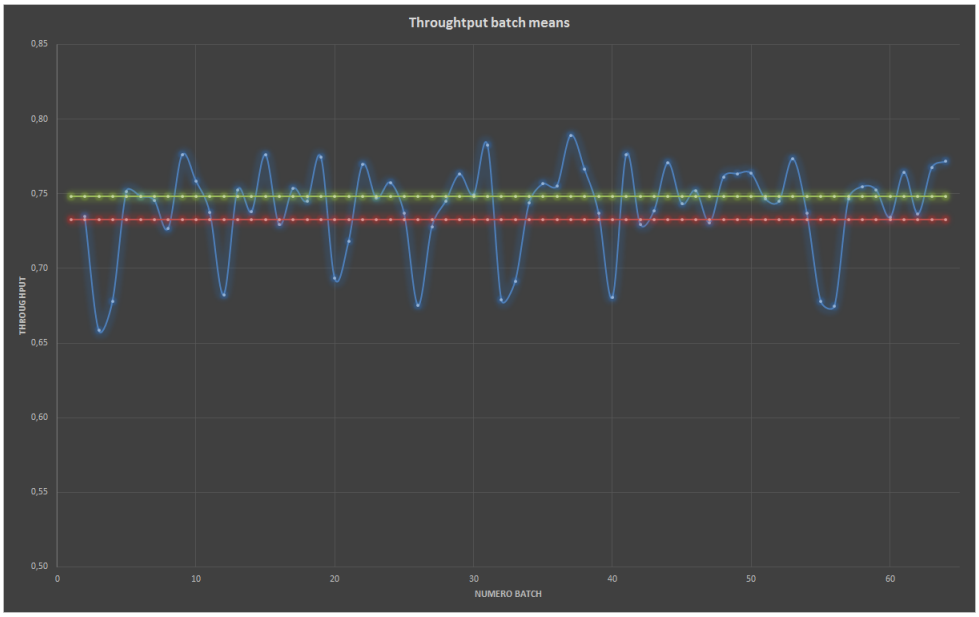


Figura 9.11: Throughput nel caso dei batch means

Nei primi batch possiamo notare che il sistema ha tempi di risposta molto alti, che vanno a diminuire nei batch successivi fino ad andarsi ad assestare,

questo comportamento é dovuto al fatto che il sistema all'inizio é in una fase transitoria, e quindi ancora deve stabilizzarsi.

La stessa analisi é stata effettuata anche per il throughtput, che però risulta oscillare in maniera più o meno evidente; probabilmente tale comportamento é dovuto al meccanismo di overload management che nel momento di attivazione permette al sistema di "scaricarsi".

9.4 Conclusioni

In questo progetto é stata effettuata l'analisi delle prestazioni di un sistema che emula uno scenario di traffico web reale. Infatti oltre a simulare un comportamento stocastico riguardante sia i tempi di interarrivo delle richieste effettuate dai client che i loro tempi di processamento nei server e sia la loro fase di thinking. Inoltre fissati i parametri del sistema e sfruttando le specifiche riportate ed applicando il meccanismo di overload management, il sistema presenta un comportamento simile ai server web reali.

Si é analizzato quindi le prestazioni di tale sistema al variare delle distribuzioni dei tempi di servizio, dell'applicazione del meccanismo di overload management valutando l'andamento degli indici di prestazione di interesse quali lo useful throughput e i tempi di risposta del sistema, oltre a valutare il drop ratio e l'abort ratio. A tale scopo é stato utilizzato un simulatore next-event il quale integra un meccanismo di avanzamento del tempo simulato basato sull'occorrenza degli eventi schedulati in apposite strutture dati.

Appendice A

Codice

A.1 Test degli estremi

A.1.1 test.c

```
/**
   Progetto MPSR
   Test degli estremi
**/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "rngs.h"
#include "rvms.h"

#define SEED 48271
#define N 100000L
//#define K 5000L
#define K N/20
#define ALPHA 0.05f
#define D 7

#define square(x) (x)*(x)

FILE *openFileForWrite(char *filePath) {
  if(filePath == NULL)
    return NULL;
  return fopen(filePath, "w");
}

void closeFile(FILE *f) {
  if(f != NULL) {
    fclose(f);
  }
}

void writeDataToFile(FILE *f, double v, double v1, double v2, long stream ) {
```

```
  if(f == NULL)
    return;
  if(fprintf(f, "%ld;%.6f;%.6f;%.6f\n", stream, v, v1, v2) < 0) {
    fprintf(stderr, "Failed to write data to file\n");
  }
}

int checkIfTestFailed(double v, double v1, double v2) {
  return v < v1 || v > v2;
}

int main(int argc, char **argv) {
  char *filename = "result.csv";
  long failed = 0L, passed = 0L;
  if(argc > 1) {
    filename = argv[1]; //nome del file
  }

  FILE *dataOut = openFileForWrite(filename);
  if(dataOut == NULL) {
    perror("Data out open");
    exit(EXIT_FAILURE);
  }

  double v1_star = idfChisquare(K - 1, ALPHA/2.0);
  double v2_star = idfChisquare(K - 1, 1.0 - (ALPHA/2.0));

  long stream = 0;
  for(; stream < 256L; stream++) {

    //inizializzazione
    long o[K];

    double v = 0.0;
    double e_x = (double) N / (double) (K);

    //imposta tutti gli intervalli dell'istogramma a 0
    memset(o, 0, K * sizeof(long));

    PlantSeeds(SEED);
    SelectStream(stream);

    //popolo il campione
    long i = 0;
    for(; i < N; i++) {
      double r = Random();
            double u;
            int j;
            for(j = 1; j < D; j++) {
                u = Random();
                if(u > r) r = u;
            }
            u = exp(D * log(r));
      long x = u * K;
      o[x] ++;
    }

    //calcolo la statistica del test
    for(i = 0; i < K; i++) {
      v += (square(o[i] - e_x));
    }
    v /= e_x;
```

```
    writeDataToFile(dataOut, v, v1_star, v2_star, stream);

    //risultati completi
    printf("===Uniformity Test (X~U(0,1))===\n");
    printf("stream ........... = %ld\n", stream);
    printf("seed ............. = %ld\n", SEED);
    printf("alpha ............ = %f\n", ALPHA);
    printf("n ................ = %ld\n", N);
    printf("k bins ........... = %ld\n", K);
    printf("n/k .............. = %f\n", e_x);
    printf("v1* .............. = %f\n", v1_star);
    printf("v2* .............. = %f\n", v2_star);
    printf("v ................ = %f\n", v);
    printf("result ........... = ");

    if(checkIfTestFailed(v, v1_star, v2_star)) {
      printf("FAILED\n");
      failed++;
    } else {
      printf("PASSED\n");
      passed++;
    }
  }

  printf("Number of failed: %ld\n", failed);
  printf("Number of passed: %ld\n", passed);
  printf("Total number of tests: %ld\n", failed+passed);

  closeFile(dataOut);
  return 0;
}
```

A.2 Intervalli di Confidenza

A.2.1 interval_calculator.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Intervalli di confidenza
 */

#define ALPHA 0.05

static double xm_res_time = 0.0;
static double v_res_time = 0.0;
static double n_res_time = 0.0;
static double incr_res_time = 0.0;

static double xm_t_sess = 0.0;
static double v_t_sess = 0.0;
static double n_t_sess = 0.0;
static double incr_t_sess = 0.0;

void set_ic_res_data(double value) {
```

```
    double d = value - xm_res_time;
    n_res_time++;
    v_res_time = v_res_time + d*d*(n_res_time-1)/n_res_time;
    xm_res_time = xm_res_time + d/n_res_time;
}

void set_ic_t_data(double value) {
    double d = value - xm_t_sess;
    n_t_sess++;
    v_t_sess = v_t_sess + d*d*(n_t_sess-1)/n_t_sess;
    xm_t_sess = xm_t_sess + d/n_t_sess;
}

void compute_ic_res() {
    double s = sqrt(v_res_time/n_res_time);
    //Calcolo il critical value
    double t = idfStudent((int) n_res_time-1, 1.0 - (ALPHA/2.0));
    incr_res_time = (t * s) / sqrt((double)(n_res_time-1));
}

void compute_ic_t() {
    double s = sqrt(v_t_sess/n_t_sess);
    //Calcolo il critical value
    double t = idfStudent((int) n_t_sess-1, 1.0 - (ALPHA/2.0));
    incr_t_sess = (t * s) / sqrt((double)(n_t_sess-1));
}

void print_ic_on_file(FILE *g) {
    compute_ic_t();
    compute_ic_res();
    fprintf(g, "\n");
    fprintf(g, "Response time IC = [ %6.8f ; %6.8f ]\n", (xm_res_time - incr_res_time),
         (xm_res_time + incr_res_time));
    fprintf(g, "Throughput IC = [ %6.8f ; %6.8f ]\n", (xm_t_sess - incr_t_sess), (xm_t_sess
         + incr_t_sess));
}
```

A.3 Autocorrelazione

A.3.1 autocorrelation.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Calcolo Autocorrelazione
 */

#define K_CORR 20
#define SIZE_CORR (K_CORR + 1)

static double sum_corr = 0.0;
static double hold_corr[SIZE_CORR];
static double cosum_corr[SIZE_CORR] = {0.0};
static double mean_corr = 0.0;
static long p_corr = 0, j_corr = 0;
```

```
void set_autocorr_data(long point, double average) {
    sum_corr += average;
    if(point < SIZE_CORR) {
        hold_corr[point] = average;
    }
    else {
        for (j_corr = 0; j_corr < SIZE_CORR; j_corr++) {
            cosum_corr[j_corr] += hold_corr[p_corr] * hold_corr[(p_corr + j_corr) %
                SIZE_CORR];
        }
        hold_corr[p_corr] = average;
        p_corr = (p_corr + 1) % SIZE_CORR;
    }
}

void compute_autocorr() {
    int i = 0;
    while (i < SIZE_CORR) {      /* empty the circular array */
        for (j_corr = 0; j_corr < SIZE_CORR; j_corr++) {
            cosum_corr[j_corr] += hold_corr[p_corr] * hold_corr[(p_corr + j_corr) %
                SIZE_CORR];
        }
        hold_corr[p_corr] = 0.0;
        p_corr = (p_corr + 1) % SIZE_CORR;
        i++;
    }
    mean_corr = sum_corr / (batch_num-1);
    for (j_corr = 0; j_corr <= K_CORR; j_corr++) {
        cosum_corr[j_corr] = (cosum_corr[j_corr] / (batch_num - 1 - j_corr)) - (mean_corr *
            mean_corr);
    }
}

void print_autocorr_on_file(FILE *g) {
    fprintf(g, "\n");
    fprintf(g, "Autocorrelation values:\nJ\tVALUE\n");
    for(j_corr = 1; j_corr < SIZE_CORR; j_corr++) {
        fprintf(g, "%3ld\t%6.8f\n", j_corr, cosum_corr[j_corr] / cosum_corr[0]);
    }
}
```

A.4 Simulatore

A.4.1 simulatore.c

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - main function
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <string.h>
```

```c
#include "rng.c"
#include "rvms.c"
#include "global_variables.h"
#include "file_manager.c"
#include "generate_random_value.h"
#include "event_manager.c"
#include "utils.h"
#include "user_signal.c"

void initialize() {
    busy_FS = 0;
    busy_BES = 0;
    FS_counter = 0;
    BES_counter = 0;
    client_counter = 0;
    queue_length_FS = 0;
    queue_length_BES = 0;
    active_client = 0;
    average_res_FS = 0.0;
    average_res_BES = 0.0;
    average_res_client = 0.0;
    FS_utilization = 0.0;
    FS_average_utilization = 0.0;
    completed_sessions = 0;
    opened_sessions = 0;
    requests = 0;
    dropped = 0;
    aborted = 0;
    threshold_exceeded = 0;
    throughput_sessions = 0.0;
    throughput_requests = 0.0;
    arrivals = 1;
    current_time = START;
    prev_time = START;
    ev_list = NULL;
    add_event(&ev_list, GetArrival(START), NEW_SESSION);
    prev_batch_time_completition = START;
    __throughput_sessions = 0.0;
    __throughput_requests = 0.0;
    __FS_utilization = 0.0;
    __FS_average_utilization = 0.0;
    __opened_sessions = 0;
    __completed_sessions = 0;
    __requests = 0;
    __dropped = 0;
    __aborted = 0;
}

void reset() {
    average_res_FS = 0.0;
    average_res_BES = 0.0;
    average_res_client = 0.0;
    FS_utilization = 0.0;
    FS_average_utilization = 0.0;
    completed_sessions = 0;
    opened_sessions = 0;
    requests = 0;
    dropped = 0;
    aborted = 0;
    throughput_sessions = 0.0;
    throughput_requests = 0.0;
```

```
    prev_batch_time_completition = current_time;
}

void compute_statistics() {
    __throughput_sessions += throughput_sessions;
    __throughput_requests += throughput_requests;
    __FS_utilization += FS_utilization;
    __FS_average_utilization += FS_average_utilization;
    __opened_sessions += opened_sessions;
    __completed_sessions += completed_sessions;
    __requests += requests;
    __dropped += dropped;
    __aborted += aborted;
}

void begin_simulation_batch(FILE *graphic) {

    int i = 0;
    initialize();
    for(current_batch = 1L; current_batch <= batch_num; current_batch++) {
        reset();
        while(completed_sessions < batch_size) {
            // fare tutto
            Event *current = pop_event(&ev_list);
            current_time = current->time;
            FS_utilization += (current_time - prev_time) * (busy_FS);
            FS_average_utilization = FS_utilization /
                 (current_time-prev_batch_time_completition);
            throughput_requests = ((double) requests)/
                 (current_time-prev_batch_time_completition);
            throughput_sessions = ((double)
                 completed_sessions)/(current_time-prev_batch_time_completition);

            manage_event(current);
            prev_time = current_time;

            if(visual_flag == 'Y' || visual_flag == 'y') {
                if(i%300 == 0) {
                    clear_console();
                    // Stampo
                    print_system_state(current->type);
                    i=0;
                    usleep(10000);
                }
                i++;
            }
        }
        print_system_state_on_file(graphic);
        set_ic_t_data(throughput_sessions);
        set_ic_res_data(average_res_FS+average_res_BES);
        printf("\nSimulation completed! (Batch completed: %ld)\n", current_batch);
        compute_statistics();
    }
    // stampa stato finale
    print_final_state_batch(graphic);
    compute_autocorr();
    print_autocorr_on_file(graphic);
    print_ic_on_file(graphic);
}

void begin_simulation_run(FILE *graphic) {
    int i = 0;
```

```
    initialize();
    CURRENT_STOP += STEP;
    // settare tutte le variabili
    while(ev_list != NULL) {
        Event *current = pop_event(&ev_list);
        current_time = current->time;
        // Aggiorno tutte le variabili
        FS_utilization += (current_time - prev_time) * (busy_FS);
        FS_average_utilization = FS_utilization / current_time;
        throughput_requests = ((double) requests)/current_time;
        throughput_sessions = ((double) completed_sessions)/current_time;

        manage_event(current);
        prev_time = current_time;

        if(current_time >= CURRENT_STOP) {
            print_system_state_on_file(graphic);
            set_ic_t_data(throughput_sessions);
            set_ic_res_data(average_res_FS+average_res_BES);
            printf("\nSimulation completed! (STEP: %ld)\n", CURRENT_STOP);
            compute_statistics();
            if(CURRENT_STOP == STOP)
                break;
            if(CURRENT_STOP < STOP)
                CURRENT_STOP += STEP;
        }

        if(visual_flag == 'Y' || visual_flag == 'y') {
            if(i%300 == 0) {
                clear_console();
                // Stampo
                print_system_state(current->type);
                i=0;
                usleep(10000);
            }
            i++;
        }
    }
    // stampa stato finale
    print_final_state_run(graphic);
    compute_autocorr();
    print_autocorr_on_file(graphic);
    print_ic_on_file(graphic);
}


int main (int argc, char *argv[]) {

    int choice;
    long long int SEED = 0;
    char t_flag;
    FILE *graphic;

    if(argc != 1) {
        fprintf(stderr, "Usage: %s\n", argv[0]);
        return EXIT_FAILURE;
    }
    add_signals();

    clear_console();
    printf("\nMulti-tier system simulator\n");
    printf("-------------------------------------------------------------------------------------\n");
```

```
printf("This system is composed by\n");
printf(" - Front Server\n - Back-end Server\n");
printf("-----------------------------------------------------------------------------------------\n\n");
printf("Choose if Front Server service time has to be:\n");
printf(" 1 - Exponetially distribuited\n");
printf(" 2 - Distributed as a 10-Erlang\n");
printf(" 3 - Distributed as Hyperexponential with p=0.1\n");
printf("Your choice: ");
scanf("%d", &choice);
getchar();
printf("-----------------------------------------------------------------------------------------\n\n");

switch(choice) {
    case 1:
        type = FE_EXP;
        break;
    case 2:
        type = FE_ERL;
        break;
    case 3:
        type = FE_HYP;
        break;
    default:
        printf("Errore. Nessuna distribuzione per il frontend selezionata.\n");
        return EXIT_FAILURE;
}

printf("Choose if threshold should be active:\n");
printf("Your choice: ");
scanf("%s", &t_flag);
getchar();
if(t_flag == 'Y' || t_flag == 'y') {
    printf("Threshold enabled\n");
    threshold_flag = 1;
}
else {
    printf("Threshold disabled\n");
    threshold_flag = 0;
}
printf("-----------------------------------------------------------------------------------------\n\n");

printf("Please select a SEED: \n");
printf(" 1 - 615425336\n");
printf(" 2 - 37524306\n");
printf(" 3 - 123456789\n");
printf(" 4 - Insert another SEED\n");
printf("Your choice: ");
scanf("%d", &choice);
getchar();

switch(choice) {
    case 1:
        SEED = 615425336;
        break;
    case 2:
        SEED = 37524306;
        break;
    case 3:
        SEED = 123456789;
        break;
    case 4:
        printf("Enter your SEED: ");
```

```
        scanf("%lld", &SEED);
        getchar();
        break;
    default:
        printf("Error. No distribution selected.\n");
        return EXIT_FAILURE;
}

printf("Chosen SEED is: %lld\n", SEED);
printf("----------------------------------------------------------------------------------------\n\n");

PutSeed(SEED);

printf("Choose simulation type:\n");
printf("1 - Long run simulation\n");
printf("2 - Batch simulation \n");
scanf("%d", &choice);
getchar();

switch (choice) {
    case 1:
        SIM_TYPE = 0;
        break;
    case 2:
        SIM_TYPE = 1;
        break;
    default:
        printf("Error. No simulation type selected.\n");
        return EXIT_FAILURE;
}

if (SIM_TYPE) {
    printf("Insert simulation parameters:\n");
    printf("Batch size (b): ");
    scanf("%ld", &batch_size);
    getchar();
    printf("Number of batches (k): ");
    scanf("%ld", &batch_num);
    getchar();

    printf("The inserted parameters are: %ld %ld\n", batch_size, batch_num);
    printf("----------------------------------------------------------------------------------------\n\n");

    printf("Would you like to see the \"system state\" during the simulation?\n");
    printf("Choose [Y/N]:");
    scanf("%c", &visual_flag);
    getchar();
    if(visual_flag == 'Y' || visual_flag == 'y')
        printf("You choose to displat system state\n");
    else
        printf("You choose NOT to displat system state\n");
    printf("----------------------------------------------------------------------------------------\n\n");

    printf("Press ANY key to start simulation\n");
    getchar();

    //Aprire i file per la simulazione e iniziare a scriverci dentro
    graphic = open_file();
    print_initial_settings(graphic, SEED, batch_size, batch_num);

    begin_simulation_batch(graphic);
```

```
        //chiudere i file
        close_file(graphic);
    }
    else {
        printf("Insert simulation parameters:\n");
        printf("Stop time: ");
        scanf("%ld", &STOP);
        getchar();
        printf("Step length: ");
        scanf("%ld", &STEP);
        getchar();

        printf("The inserted parameters are: %ld %ld\n", STOP, STEP);
        printf("-------------------------------------------------------------------------------------------\n\n");
        if(STOP < STEP)
            STOP = STEP;
        printf("Would you like to see the \"system state\" during the simulation?\n");
        printf("Choose [Y/N]:");
        scanf("%c", &visual_flag);
        getchar();
        if(visual_flag == 'Y' || visual_flag == 'y')
            printf("You choose to displat system state\n");
        else
            printf("You choose NOT to displat system state\n");
        printf("-------------------------------------------------------------------------------------------\n\n");

        printf("Press ANY key to start simulation\n");
        getchar();

        //Aprire i file per la simulazione e iniziare a scriverci dentro
        graphic = open_file();
        print_initial_settings(graphic, SEED, STOP, STEP);

        begin_simulation_run(graphic);

        //chiudere i file
        close_file(graphic);
    }

    return EXIT_SUCCESS;
}
```

A.4.2 arrival_queue.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Coda di arrivi
 */

struct ArrivalNode{
    double time;
    struct ArrivalNode* next;
};

typedef struct ArrivalNode ArrivalNode;

void arrival_add(ArrivalNode** list, double t){
```

```
    if(list == NULL) {
        printf("Errore: la lista  NULL\n");
        return;
    }
    ArrivalNode* temp = (ArrivalNode*) malloc(sizeof(ArrivalNode));
    temp->time = t;
    temp->next = NULL;
    if(*list == NULL){
        *list = temp;
        return;
    }

    ArrivalNode* curr = *list;
    while(curr->next != NULL){
        curr = curr->next;
    }
    curr->next = temp;
}

double arrival_pop(ArrivalNode** list){
    if(list == NULL || *list ==NULL) {
        printf("ArrivalQueue is empty\n");
        return -1.0;
    }
    ArrivalNode* curr = *list;
    *list = (*list)->next;

    double t = curr->time;
    free(curr);

    return t;
}

void arrival_print(ArrivalNode** list){
    if(list == NULL || *list ==NULL) {
        printf("ArrivalQueue is empty\n");
        return;
    }
    ArrivalNode *curr=*list;
    printf("[+] Arrival Queue:\n");
    while(curr != NULL){
        printf("[Time: %6.4f] -- ",curr->time);
    curr = curr->next;
    }
}
```

A.4.3 client_req.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Coda delle richieste del client
 */

struct ClientReq {
  double time; // time of Client completion
  unsigned int req; // number of remaining requests
  struct ClientReq *next;
```

```
};

typedef struct ClientReq ClientReq;

void print_client_req(ClientReq **list) {
  if(*list == NULL) {
    printf("ClientOrderList is empty\n");
    return;
  }
  ClientReq *curr = *list;
  printf("[+] ClientOrderList contains:\n");
  while(curr != NULL) {
    printf("[Time: %6.4f, Rem_Req: %d]\n", curr->time, curr->req);
    curr = curr->next;
  }
}

void add_client_req(ClientReq **list, double x, unsigned int r) {
    if(list == NULL) {
        printf("NULL Pointer to ClientOrderList\n");
        return;
    }

  ClientReq *new_ClientReq = (ClientReq *)malloc(sizeof(ClientReq));
  new_ClientReq->time = x;
  new_ClientReq->req = r;
  new_ClientReq->next = NULL;
  if(*list == NULL) { // empty list case
  //  printf("[T] Empty List\n");
    *list = new_ClientReq;
    return;
  }
  if((*list)->time >= x) { // first element replacement case
  //  printf("[T] First element replacement\n");
    new_ClientReq->next = *list;
    *list = new_ClientReq;
  }
    //  printf("[T] Default add case\n");
    ClientReq *curr = *list;
    while(curr->time < x) {
    //  printf("\n[T] curr->time: %d", curr->time);
        if(curr->next == NULL || curr->next->time >= x) {
            new_ClientReq->next = curr->next;
            curr->next = new_ClientReq;
            return;
        }
        else curr = curr->next;
    }
}

// Return the first element and remove it
ClientReq* pop_ClientReq(ClientReq** head) {
    if(head == NULL || *head == NULL) return NULL;
    ClientReq* temp = *head;
    *head = (*head)->next;
    return temp;
}
```

A.4.4 event_list.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Tipologia di Eventi
 */

#define MIN_TYPE NEW_SESSION
#define MAX_TYPE CL_COMPL

enum _EVENT_TYPE {
    NEW_SESSION,
    FS_COMPL,
    BES_COMPL,
    CL_COMPL,
    SYSTEM_PRINT
};

typedef enum _EVENT_TYPE EVENT_TYPE;

struct Event {
  double time;
  EVENT_TYPE type;
  struct Event *next;
};

typedef struct Event Event;

int event_check(double t, EVENT_TYPE type) {
    return t>=0.0 && (type >= MIN_TYPE && type <= MAX_TYPE);
}

void printlist(Event **list) {
  if(*list == NULL) {
    printf("List is empty\n");
    return;
  }
  Event *curr = *list;
  printf("[+] List contains:\n");
  while(curr != NULL) {
    printf("[Type: %d, Time: %6.4f)\n", curr->type, curr->time);
    curr = curr->next;
  }
}

void add_event(Event **list, double x, EVENT_TYPE typ) {
    if(list == NULL) {
        printf("NULL Pointer to list\n");
        return;
    }
    if(!event_check(x, typ)) {
        printf("[+] Event check failed. Please review your data.\n");
        return;
    }
  Event *new_Event = (Event *)malloc(sizeof(Event));
  new_Event->time = x;
  new_Event->type = typ;
  new_Event->next = NULL;
  if(*list == NULL) { // empty list case
  //  printf("[T] Empty List\n");
```

```
    *list = new_Event;
    return;
  }
  if((*list)->time >= x) { // first element replacement case
  //  printf("[T] First element replacement\n");
    new_Event->next = *list;
    *list = new_Event;
  }
    //  printf("[T] Default add case\n");
    Event *curr = *list;
    while(curr->time < x) {
    //  printf("\n[T] curr->time: %d", curr->time);
        if(curr->next == NULL || curr->next->time >= x) {
            new_Event->next = curr->next;
            curr->next = new_Event;
            return;
        }
        else curr = curr->next;
    }
}

// Return the first element and remove it
Event* pop_event(Event** head) {
    if(head == NULL || *head == NULL) return NULL;
    Event* temp = *head;
    *head = (*head)->next;
    return temp;
}

char* event_translator(EVENT_TYPE t){
  switch(t){
    case NEW_SESSION:
            return "New session";
    break;
    case FS_COMPL:
            return "Front server completition";
    break;
    case BES_COMPL:
            return "Back-end server completition";
    break;
    case CL_COMPL:
            return "Client completition";
    break;
        case SYSTEM_PRINT:
            return "User forced to print output";
    break;
    default:
            return "Error event";
    break;
  }
  return "Wrong code!\n";
}
```

A.4.5 event_manager.c

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
```

```
 *      Simulatore web - Processamento degli eventi
 */

long FS_counter, BES_counter, client_counter;
int queue_length_FS, queue_length_BES, busy_FS, busy_BES, active_client;
double average_res_FS, average_res_BES, average_res_client;
int threshold_exceeded;

// NewSession Management
void NewSession(Event* ev) {

    // If current time is not beyond STOP, a NewSession is scheduled
    if(arrivals) {
        double new_time = GetArrival(ev->time); // compute new session arrival time
        add_event(&ev_list, new_time, NEW_SESSION); // create NewSession event and schedule
             it
    }

    if(threshold_flag) {
        if(threshold_exceeded) {
            if(FS_average_utilization <= THRESHOLD_MIN)
                threshold_exceeded = 0;
            else {
                dropped++;
                return;
            }
        }
        // la iperesponenziale  la peggiore implemento quindi il Threshold all' 85%
        else if(FS_average_utilization >= THRESHOLD_MAX) {
            threshold_exceeded = 1;
            dropped++;
            return;
        }
    }
    opened_sessions++;
    enqueue_req(&req_queue, GetRequests());
    arrival_add(&arrival_queue_FS, ev->time);
    if(busy_FS) queue_length_FS++; // if server is busy, add 1 to its queue
    else {
        busy_FS = 1;
        switch(type) {
            case FE_EXP:
                add_event(&ev_list, GetExponentialServiceFS(ev->time), FS_COMPL); // create
                     FS_COMPL event and schedule it
                break;
            case FE_ERL:
                add_event(&ev_list, GetErlangServiceFS(ev->time), FS_COMPL); // create
                     FS_COMPL event and schedule it
                break;
            case FE_HYP:
                add_event(&ev_list, GetHyperexpServiceFS(ev->time), FS_COMPL); // create
                     FS_COMPL event and schedule it
                break;
        }
    }
}

// FS Completion Management
void FS_Completion(Event* ev) {

    // calculate average residence time in FS center. (Welford + destructive pop)
    FS_counter++;
```

```
    double res_time_FS = ev->time - arrival_pop(&arrival_queue_FS); // i-th session's
        residence time
    average_res_FS = average_res_FS + (res_time_FS - average_res_FS)/FS_counter;
    arrival_add(&arrival_queue_BES, ev->time); // save entrance time into BE center

    // Exiting from FS
    if(queue_length_FS > 0) {
        queue_length_FS--;
        switch(type) {
            case FE_EXP:
                add_event(&ev_list, GetExponentialServiceFS(ev->time), FS_COMPL); // create
                    FS_COMPL event and schedule it
                break;
            case FE_ERL:
                add_event(&ev_list, GetErlangServiceFS(ev->time), FS_COMPL); // create
                    FS_COMPL event and schedule it
                break;
            case FE_HYP:
                add_event(&ev_list, GetHyperexpServiceFS(ev->time), FS_COMPL); // create
                    FS_COMPL event and schedule it
                break;
        }
    }
    else busy_FS = 0;

    // Entering in BES
    if(busy_BES) queue_length_BES++;
    else {
        busy_BES = 1;
        add_event(&ev_list, GetServiceBES(ev->time), BES_COMPL);
    }
}

// BES Completion Management
void BES_Completion(Event* ev) {

    // calculate average residence time in BES center. (Welford + destructive pop)
    BES_counter++;
    double res_time_BES = ev->time - arrival_pop(&arrival_queue_BES); // i-th session's
        residence time
    average_res_BES = average_res_BES + (res_time_BES - average_res_BES)/BES_counter;
    // Exiting from BES
    if(queue_length_BES > 0) {
        queue_length_BES--;
        add_event(&ev_list, GetServiceBES(ev->time), BES_COMPL);
    }
    else busy_BES = 0;

    //Entering Client
    unsigned int this_requests = dequeue_req(&req_queue);
    if(this_requests <= 0) {
        // session is over. It will move out of the system
        set_autocorr_data(__completed_sessions+completed_sessions,
            average_res_FS+average_res_BES);
        completed_sessions++;
    }
    else {
        // This session still has some requests to be executed. Let's go to the Clients
        requests++;
        client_counter++;
        active_client++; // increase the number of active clients
```

```
        // Calculate average residence time in Clients center
        double temp = GetServiceClient(ev->time);

        double time_res_client = temp - ev->time;
        add_event(&ev_list, temp, CL_COMPL);

        average_res_client = average_res_client + (time_res_client -
             average_res_client)/client_counter;

        // Add this time and this session's remaining requests into ClientOrderList
        add_client_req(&client_req_list, temp, this_requests-1);
    }
}

// Client Completion Management
void Client_Completion(Event* ev) {
    active_client--; // decrease number of Clients
    // Insert the updated value of remaining requests (from THIS session) into the ReqQueue
    ClientReq* coming_back_session = pop_ClientReq(&client_req_list);

    // Gestire l'abort delle sessioni
    if(threshold_flag) {
        if(threshold_exceeded) {
            if(FS_average_utilization <= THRESHOLD_MIN)
                threshold_exceeded = 0;
            else {
                aborted++;
                //sessions++; // FIXME
                free(coming_back_session);
                return;
            }
        }
        else if(FS_average_utilization >= THRESHOLD_MAX) {
            threshold_exceeded = 1;
            aborted++;
            //sessions++; // FIXME
            free(coming_back_session);
            return;
        }
    }

    enqueue_req(&req_queue, coming_back_session->req);
    free(coming_back_session);

    if(busy_FS == 0) {
        busy_FS = 1;
        switch(type) {
            case FE_EXP:
                add_event(&ev_list, GetExponentialServiceFS(ev->time), FS_COMPL); // create
                     FS_COMPL event and schedule it
                break;
            case FE_ERL:
                add_event(&ev_list, GetErlangServiceFS(ev->time), FS_COMPL); // create
                     FS_COMPL event and schedule it
                break;
            case FE_HYP:
                add_event(&ev_list, GetHyperexpServiceFS(ev->time), FS_COMPL); // create
                     FS_COMPL event and schedule it
                break;
        }
    }
    else queue_length_FS++;
```

```
    arrival_add(&arrival_queue_FS, ev->time);
}

void manage_event(Event *e) {
    switch(e->type) {
        case NEW_SESSION:
            NewSession(e);
            break;
        case FS_COMPL:
            FS_Completion(e);
            break;
        case BES_COMPL:
            BES_Completion(e);
            break;
        case CL_COMPL:
            Client_Completion(e);
            break;
        default:
            printf("Wrong event!\n");
    }
    free(e);
}
```

A.4.6 file_manager.c

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Gestisce tutto il flusso di dati da scrivere su file
 */

#define MAX_DATETIME 150
extern SIMULATION_TYPES type;

char* get_date() {
    time_t td = time(NULL);
  struct tm *now = NULL;
  char * buffer = (char *)malloc(sizeof(char)*(MAX_DATETIME+1));
  buffer[MAX_DATETIME] = '\0';
  now = localtime(&td);   /* Get time and date structure */
  strftime (buffer, MAX_DATETIME, "%Y%m%d-%H%M%S",now);
  return buffer;
}

FILE *open_file() {
    char graphic_name[60], *g = graphic_name, *graphic_ext = ".csv";
    char *date = get_date();
    strcpy(graphic_name, date);
    g+=strlen(date);
    switch (type) {
        case FE_EXP:
            strcat(g,"-FE_EXP");
            g+=strlen("-FE_EXP");
            break;
        case FE_ERL:
            strcat(g,"-FE_ERL");
            g+=strlen("-FE_ERL");
            break;
```

```
        case FE_HYP:
            strcat(g,"-FE_HYP");
            g+=strlen("-FE_HYP");
            break;
        default:
            printf("Error. No known simulation type.\n");
    }
    if(threshold_flag) {
        strcat(g,"_OM");
        g+=strlen("_OM");
    }
    if(SIM_TYPE) {
        strcat(g,"_batch");
        g+=strlen("_batch");
    } else {
        strcat(g,"_longrun");
        g+=strlen("_longrun");
    }
    strcat(g, graphic_ext);
    g[strlen(graphic_ext)] = 0;
    free(date);
    return fopen(graphic_name, "a");
}

void close_file(FILE *g) {
    fclose(g);
}
```

A.4.7 generate_random_value.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - genera i valori random per arrivi e servizi
 */

#define MIN_REQ 5.0
#define MAX_REQ 35.0
#define ARRIVAL_TIME 1.0/35.0  // 1/lambda
#define THINK_TIME 7.0         // E[Z]
#define FS_COMPL_TIME 0.00456  // E[D]_front tempo di servizio (quello che spende nella
     palla)
#define BES_COMPL_TIME 0.00117 // E[D]_back tempo di servizio (quello che spende nella
     palla)
#define K_ERLANG 10            // Parametro per la distribuzione della 10 Erlang
#define P_HYP 0.1              // Parametro per la sitribuzione iperesponenziale

double Exponential(double m) {
    return (-m * log(1.0 - Random()));
}

double Erlang(long n, double b) {
    long  i;
    double x = 0.0;
    for (i = 0; i < n; i++)
        x += Exponential(b);
    return (x);
}
```

```
double Hyperexponential() {
    return Exponential(2*P_HYP*FS_COMPL_TIME) + Exponential(2*(1-P_HYP)*FS_COMPL_TIME);
}

long Equilikely(long a, long b) {
    return (a + (long) (Random() * (b - a + 1)));
}

double GetArrival(double prev_time) {
    return prev_time + Exponential(ARRIVAL_TIME);
}

double GetExponentialServiceFS(double prev_time) {
    return prev_time + Exponential(FS_COMPL_TIME);
}

double GetErlangServiceFS(double prev_time) {
    return prev_time + Erlang(K_ERLANG, FS_COMPL_TIME/K_ERLANG);
}

double GetHyperexpServiceFS(double prev_time) {
    return prev_time + Hyperexponential();
}

double GetServiceBES(double prev_time) {
    return prev_time + Exponential(BES_COMPL_TIME);
}

double GetServiceClient(double prev_time) {
    return prev_time + Exponential(THINK_TIME);
}

unsigned int GetRequests() {
    return (unsigned int)Equilikely(MIN_REQ, MAX_REQ);
}
```

A.4.8 global_variables.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Variabili Globali e include
 */

#define THRESHOLD_MAX     0.85f         // Threshold massimo dopo il quale droppo
#define THRESHOLD_MIN     0.75f         // Se scendo qua sotto riprendo
#define START             0.0f          // il tempo di inizio

double throughput_sessions, throughput_requests, current_time, prev_time, FS_utilization,
     FS_average_utilization;
double __throughput_sessions, __throughput_requests, __FS_utilization,
     __FS_average_utilization, prev_batch_time_completition;
int arrivals, threshold_flag;
long opened_sessions, completed_sessions, requests, dropped, aborted;
long __opened_sessions, __completed_sessions, __requests, __dropped, __aborted;
char visual_flag;
```

```
long batch_size;          // numero di job prima di finire un batch
long batch_num;           // numero di batch prima di terminare una simulazione
long current_batch;       // a quale batch sono arrivato

long STOP;                // a che tempo devo fermare la simulazione
long STEP;                // ogni quanto prendere i tempi
long CURRENT_STOP;

int SIM_TYPE;             // il tipo di simulazione

#include "simulation_type.h"
#include "event_list.h"
#include "arrival_queue.h"
#include "req_queue.h"
#include "client_req.h"
#include "autocorrelation.h"
#include "interval_calculator.h"

// Definizione del tipo di simulazione
SIMULATION_TYPES type;
// Code
Event *ev_list;
Request* req_queue;        // Request Queue - it stores information regarding Requests
    amount
ClientReq* client_req_list; // Matrix with Client completion time and session requests
ArrivalNode* arrival_queue_FS; // Arrival times queue at FS
ArrivalNode* arrival_queue_BES; // Arrival times queue at BES
```

A.4.9 req_queue.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Coda delle richieste
 */


struct Request{
    unsigned int n;
    struct Request* next;
};

typedef struct Request Request;

void enqueue_req(Request** list, unsigned int t){
    if(list == NULL) {
        printf("ReqQueue is NULL\n");
        return;
    }
    Request* temp = (Request*) malloc(sizeof(Request));
    temp->n = t;
    temp->next = NULL;
    if(*list == NULL){
        *list = temp;
        return;
    }

    Request* curr = *list;
```

```
    while(curr->next != NULL){
        curr = curr->next;
    }
    curr->next = temp;
}

unsigned int dequeue_req(Request** list){
    if(list == NULL || *list ==NULL) {
        printf("ReqQueue is empty\n");
        return 0.0;
    }
    Request* curr = *list;
    *list = (*list)->next;

    unsigned int t = curr->n;
    free(curr);

    return t;
}

void print_req(Request** list){
    if(list == NULL || *list ==NULL) {
        printf("ReqQueue is empty\n");
        return;
    }
    Request *curr=*list;
    printf("[+] ReqQueue:\n");
    while(curr != NULL){
        printf("[n: %d] -- ",curr->n);
    curr = curr->next;
    }
    printf("\n");
}
```

A.4.10 rng.c

```
/* ------------------------------------------------------------------------
 * This is an ANSI C library for random number generation. The use of this
 * library is recommended as a replacement for the ANSI C rand() and srand()
 * functions, particularly in simulation applications where the statistical
 * 'goodness' of the random number generator is important.
 *
 * The generator used in this library is a so-called 'Lehmer random number
 * generator' which returns a pseudo-random number uniformly distributed
 * between 0.0 and 1.0. The period is (m - 1) where m = 2,147,483,647 and
 * the smallest and largest possible values are (1 / m) and 1 - (1 / m)
 * respectively. For more details see:
 *
 *      "Random Number Generators: Good Ones Are Hard To Find"
 *                Steve Park and Keith Miller
 *             Communications of the ACM, October 1988
 *
 * Note that as of 7-11-90 the multiplier used in this library has changed
 * from the previous "minimal standard" 16807 to a new value of 48271. To
 * use this library in its old (16807) form change the constants MULTIPLIER
 * and CHECK as indicated in the comments.
 *
 * Name             : rng.c (Random Number Generation - Single Stream)
```

```
 * Authors         : Steve Park & Dave Geyer
 * Language        : ANSI C
 * Latest Revision : 09-11-98
 * ------------------------------------------------------------------------
 */

#include <stdio.h>
#include <time.h>
#include "rng.h"

#define MODULUS  2147483647L /* DON'T CHANGE THIS VALUE             */
#define MULTIPLIER 48271L   /* use 16807 for the "minimal standard"  */
#define CHECK    399268537L /* use 1043616065 for the "minimal standard" */
#define DEFAULT  123456789L /* initial seed, use 0 < DEFAULT < MODULUS */

static long seed = DEFAULT; /* seed is the state of the generator    */


   double Random(void)
/* ------------------------------------------------------------------------
 * Random is a Lehmer generator that returns a pseudo-random real number
 * uniformly distributed between 0.0 and 1.0. The period is (m - 1)
 * where m = 2,147,483,647 amd the smallest and largest possible values
 * are (1 / m) and 1 - (1 / m) respectively.
 * ------------------------------------------------------------------------
 */
{
  const long Q = MODULUS / MULTIPLIER;
  const long R = MODULUS % MULTIPLIER;
        long t;

  t = MULTIPLIER * (seed % Q) - R * (seed / Q);
  if (t > 0)
    seed = t;
  else
    seed = t + MODULUS;
  return ((double) seed / MODULUS);
}


   void PutSeed(long x)
/* ------------------------------------------------------------------------
 * Use this (optional) procedure to initialize or reset the state of
 * the random number generator according to the following conventions:
 *    if x > 0 then x is the initial seed (unless too large)
 *    if x < 0 then the initial seed is obtained from the system clock
 *    if x = 0 then the initial seed is to be supplied interactively
 * ------------------------------------------------------------------------
 */
{
  char ok = 0;

  if (x > 0L)
    x = x % MODULUS;                      /* correct if x is too large */
  if (x < 0L)
    x = ((unsigned long) time((time_t *) NULL)) % MODULUS;
  if (x == 0L)
    while (!ok) {
      printf("\nEnter a positive integer seed (9 digits or less) >> ");
      scanf("%ld", &x);
      ok = (0L < x) && (x < MODULUS);
      if (!ok)
```

```
        printf("\nInput out of range ... try again\n");
    }
  seed = x;
}


   void GetSeed(long *x)
/* ------------------------------------------------------------------
 * Use this (optional) procedure to get the current state of the random
 * number generator.
 * ------------------------------------------------------------------
 */
{
  *x = seed;
}
```

A.4.11 rng.h

```
/* ------------------------------------------------------------------------
 * Name           : rng.h (header file for the library file rng.c)
 * Author         : Steve Park & Dave Geyer
 * Language       : ANSI C
 * Latest Revision : 09-11-98
 * ------------------------------------------------------------------------
 */

#if !defined( _RNG_ )
#define _RNG_

double Random(void);
void   GetSeed(long *x);
void   PutSeed(long x);
void   TestRandom(void);

#endif
```

A.4.12 rvms.c

```
/* ------------------------------------------------------------------------
 * This is an ANSI C library that can be used to evaluate the probability
 * density functions (pdf's), cumulative distribution functions (cdf's), and
 * inverse distribution functions (idf's) for a variety of discrete and
 * continuous random variables.
 *
 * The following notational conventions are used
 *               x : possible value of the random variable
 *               u : real variable (probability) between 0.0 and 1.0
 *  a, b, n, p, m, s : distribution-specific parameters
 *
 * There are pdf's, cdf's and idf's for 6 discrete random variables
 *
 *     Random Variable  Range (x) Mean        Variance
 *
```

```
 *      Bernoulli(p)      0..1      p           p*(1-p)
 *      Binomial(n, p)    0..n      n*p         n*p*(1-p)
 *      Equilikely(a, b)  a..b      (a+b)/2     ((b-a+1)*(b-a+1)-1)/12
 *      Geometric(p)      0...      p/(1-p)     p/((1-p)*(1-p))
 *      Pascal(n, p)      0...      n*p/(1-p)   n*p/((1-p)*(1-p))
 *      Poisson(m)        0...      m           m
 *
 * and for 7 continuous random variables
 *
 *      Uniform(a, b)     a < x < b (a+b)/2     (b-a)*(b-a)/12
 *      Exponential(m)    x > 0     m           m*m
 *      Erlang(n, b)      x > 0     n*b         n*b*b
 *      Normal(m, s)      all x     m           s*s
 *      Lognormal(a, b)   x > 0        see below
 *      Chisquare(n)      x > 0     n           2*n
 *      Student(n)        all x     0  (n > 1)  n/(n-2)  (n > 2)
 *
 * For the Lognormal(a, b), the mean and variance are
 *
 *                     mean = Exp(a + 0.5*b*b)
 *                 variance = (Exp(b*b) - 1)*Exp(2*a + b*b)
 *
 * Name           : rvms.c (Random Variable ModelS)
 * Author         : Steve Park & Dave Geyer
 * Language       : ANSI C
 * Latest Revision : 11-22-97
 * -----------------------------------------------------------------------
 */

#include <math.h>
#include "rvms.h"

#define TINY   1.0e-10
#define SQRT2PI 2.506628274631          /* sqrt(2 * pi) */

static double pdfStandard(double x);
static double cdfStandard(double x);
static double idfStandard(double u);
static double LogGamma(double a);
static double LogBeta(double a, double b);
static double InGamma(double a, double b);
static double InBeta(double a, double b, double x);


   double pdfBernoulli(double p, long x)
/* ======================================
 * NOTE: use 0.0 < p < 1.0 and 0 <= x <= 1
 * ======================================
 */
{
   return ((x == 0) ? 1.0 - p : p);
}

   double cdfBernoulli(double p, long x)
/* ======================================
 * NOTE: use 0.0 < p < 1.0 and 0 <= x <= 1
 * ======================================
 */
{
   return ((x == 0) ? 1.0 - p : 1.0);
}
```

```
   long idfBernoulli(double p, double u)
/* ========================================
 * NOTE: use 0.0 < p < 1.0 and 0.0 < u < 1.0
 * ========================================
 */
{
   return ((u < 1.0 - p) ? 0 : 1);
}

   double pdfEquilikely(long a, long b, long x)
/* ============================================
 * NOTE: use a <= x <= b
 * ============================================
 */
{   if(x){}
   return (1.0 / (b - a + 1.0));
}

   double cdfEquilikely(long a, long b, long x)
/* ============================================
 * NOTE: use a <= x <= b
 * ============================================
 */
{
   return ((x - a + 1.0) / (b - a + 1.0));
}

   long idfEquilikely(long a, long b, double u)
/* ============================================
 * NOTE: use a <= b and 0.0 < u < 1.0
 * ============================================
 */
{
   return (a + (long) (u * (b - a + 1)));
}

   double pdfBinomial(long n, double p, long x)
/* ============================================
 * NOTE: use 0 <= x <= n and 0.0 < p < 1.0
 * ============================================
 */
{
   double s, t;

   s = LogChoose(n, x);
   t = x * log(p) + (n - x) * log(1.0 - p);
   return (exp(s + t));
}

   double cdfBinomial(long n, double p, long x)
/* ============================================
 * NOTE: use 0 <= x <= n and 0.0 < p < 1.0
 * ============================================
 */
{
   if (x < n)
     return (1.0 - InBeta(x + 1, n - x, p));
   else
     return (1.0);
}

   long idfBinomial(long n, double p, double u)
```

```
/* ================================================
 * NOTE: use 0 <= n, 0.0 < p < 1.0 and 0.0 < u < 1.0
 * ================================================
 */
{
   long x = (long) (n * p);          /* start searching at the mean */

   if (cdfBinomial(n, p, x) <= u)
     while (cdfBinomial(n, p, x) <= u)
       x++;
   else if (cdfBinomial(n, p, 0) <= u)
     while (cdfBinomial(n, p, x - 1) > u)
       x--;
   else
     x = 0;
   return (x);
}

   double pdfGeometric(double p, long x)
/* ====================================
 * NOTE: use 0.0 < p < 1.0 and x >= 0
 * ====================================
 */
{
   return ((1.0 - p) * exp(x * log(p)));
}

   double cdfGeometric(double p, long x)
/* ====================================
 * NOTE: use 0.0 < p < 1.0 and x >= 0
 * ====================================
 */
{
   return (1.0 - exp((x + 1) * log(p)));
}

   long idfGeometric(double p, double u)
/* ========================================
 * NOTE: use 0.0 < p < 1.0 and 0.0 < u < 1.0
 * ========================================
 */
{
   return ((long) (log(1.0 - u) / log(p)));
}

   double pdfPascal(long n, double p, long x)
/* ==========================================
 * NOTE: use n >= 1, 0.0 < p < 1.0, and x >= 0
 * ==========================================
 */
{
   double s, t;

   s = LogChoose(n + x - 1, x);
   t = x * log(p) + n * log(1.0 - p);
   return (exp(s + t));
}

   double cdfPascal(long n, double p, long x)
/* ==========================================
 * NOTE: use n >= 1, 0.0 < p < 1.0, and x >= 0
 * ==========================================
```

```
 */
{
  return (1.0 - InBeta(x + 1, n, p));
}

  long idfPascal(long n, double p, double u)
/* ================================================
 * NOTE: use n >= 1, 0.0 < p < 1.0, and 0.0 < u < 1.0
 * ================================================
 */
{
  long x = (long) (n * p / (1.0 - p)); /* start searching at the mean */

  if (cdfPascal(n, p, x) <= u)
    while (cdfPascal(n, p, x) <= u)
      x++;
  else if (cdfPascal(n, p, 0) <= u)
    while (cdfPascal(n, p, x - 1) > u)
      x--;
  else
    x = 0;
  return (x);
}

  double pdfPoisson(double m, long x)
/* ==================================
 * NOTE: use m > 0 and x >= 0
 * ==================================
 */
{
  double t;

  t = - m + x * log(m) - LogFactorial(x);
  return (exp(t));
}

  double cdfPoisson(double m, long x)
/* ==================================
 * NOTE: use m > 0 and x >= 0
 * ==================================
 */
{
  return (1.0 - InGamma(x + 1, m));
}

  long idfPoisson(double m, double u)
/* ==================================
 * NOTE: use m > 0 and 0.0 < u < 1.0
 * ==================================
 */
{
  long x = (long) m;                /* start searching at the mean */

  if (cdfPoisson(m, x) <= u)
    while (cdfPoisson(m, x) <= u)
      x++;
  else if (cdfPoisson(m, 0) <= u)
    while (cdfPoisson(m, x - 1) > u)
      x--;
  else
    x = 0;
  return (x);
```

```
}

   double pdfUniform(double a, double b, double x)
/* ============================================
 * NOTE: use a < x < b
 * ============================================
 */
{
    if(x) {}
   return (1.0 / (b - a));
}

   double cdfUniform(double a, double b, double x)
/* ============================================
 * NOTE: use a < x < b
 * ============================================
 */
{
   return ((x - a) / (b - a));
}

   double idfUniform(double a, double b, double u)
/* ============================================
 * NOTE: use a < b and 0.0 < u < 1.0
 * ============================================
 */
{
   return (a + (b - a) * u);
}

   double pdfExponential(double m, double x)
/* ========================================
 * NOTE: use m > 0 and x > 0
 * ========================================
 */
{
   return ((1.0 / m) * exp(- x / m));
}

   double cdfExponential(double m, double x)
/* ========================================
 * NOTE: use m > 0 and x > 0
 * ========================================
 */
{
   return (1.0 - exp(- x / m));
}

   double idfExponential(double m, double u)
/* ========================================
 * NOTE: use m > 0 and 0.0 < u < 1.0
 * ========================================
 */
{
   return (- m * log(1.0 - u));
}

   double pdfErlang(long n, double b, double x)
/* ============================================
 * NOTE: use n >= 1, b > 0, and x > 0
 * ============================================
 */
```

```
{
   double t;

   t = (n - 1) * log(x / b) - (x / b) - log(b) - LogGamma(n);
   return (exp(t));
}

   double cdfErlang(long n, double b, double x)
/* ============================================
 * NOTE: use n >= 1, b > 0, and x > 0
 * ============================================
 */
{
   return (InGamma(n, x / b));
}

   double idfErlang(long n, double b, double u)
/* ============================================
 * NOTE: use n >= 1, b > 0 and 0.0 < u < 1.0
 * ============================================
 */
{
   double t, x = n * b;               /* initialize to the mean, then */

   do {                               /* use Newton-Raphson iteration */
     t = x;
     x = t + (u - cdfErlang(n, b, t)) / pdfErlang(n, b, t);
     if (x <= 0.0)
       x = 0.5 * t;
   } while (fabs(x - t) >= TINY);
   return (x);
}

   static double pdfStandard(double x)
/* ==================================
 * NOTE: x can be any value
 * ==================================
 */
{
   return (exp(- 0.5 * x * x) / SQRT2PI);
}

   static double cdfStandard(double x)
/* ==================================
 * NOTE: x can be any value
 * ==================================
 */
{
   double t;

   t = InGamma(0.5, 0.5 * x * x);
   if (x < 0.0)
     return (0.5 * (1.0 - t));
   else
     return (0.5 * (1.0 + t));
}

   static double idfStandard(double u)
/* ==================================
 * NOTE: 0.0 < u < 1.0
 * ==================================
 */
```

```
{
   double t, x = 0.0;                /* initialize to the mean, then */

   do {                              /* use Newton-Raphson iteration */
     t = x;
     x = t + (u - cdfStandard(t)) / pdfStandard(t);
   } while (fabs(x - t) >= TINY);
   return (x);
}

   double pdfNormal(double m, double s, double x)
/* ============================================
 * NOTE: x and m can be any value, but s > 0.0
 * ============================================
 */
{
   double t = (x - m) / s;

   return (pdfStandard(t) / s);
}

   double cdfNormal(double m, double s, double x)
/* ============================================
 * NOTE: x and m can be any value, but s > 0.0
 * ============================================
 */
{
   double t = (x - m) / s;

   return (cdfStandard(t));
}

   double idfNormal(double m, double s, double u)
/* ====================================================
 * NOTE: m can be any value, but s > 0.0 and 0.0 < u < 1.0
 * ====================================================
 */
{
   return (m + s * idfStandard(u));
}

   double pdfLognormal(double a, double b, double x)
/* ================================================
 * NOTE: a can have any value, but b > 0.0 and x > 0.0
 * ================================================
 */
{
   double t = (log(x) - a) / b;

   return (pdfStandard(t) / (b * x));
}

   double cdfLognormal(double a, double b, double x)
/* ================================================
 * NOTE: a can have any value, but b > 0.0 and x > 0.0
 * ================================================
 */
{
   double t = (log(x) - a) / b;

   return (cdfStandard(t));
}
```

```
   double idfLognormal(double a, double b, double u)
/* ============================================================
 * NOTE: a can have any value, but b > 0.0 and 0.0 < u < 1.0
 * ============================================================
 */
{
   double t;

   t = a + b * idfStandard(u);
   return (exp(t));
}

   double pdfChisquare(long n, double x)
/* ====================================
 * NOTE: use n >= 1 and x > 0.0
 * ====================================
 */
{
   double t, s = n / 2.0;

   t = (s - 1.0) * log(x / 2.0) - (x / 2.0) - log(2.0) - LogGamma(s);
   return (exp(t));
}

   double cdfChisquare(long n, double x)
/* ====================================
 * NOTE: use n >= 1 and x > 0.0
 * ====================================
 */
{
   return (InGamma(n / 2.0, x / 2));
}

   double idfChisquare(long n, double u)
/* ====================================
 * NOTE: use n >= 1 and 0.0 < u < 1.0
 * ====================================
 */
{
   double t, x = n;                    /* initialize to the mean, then */

   do {                                /* use Newton-Raphson iteration */
     t = x;
     x = t + (u - cdfChisquare(n, t)) / pdfChisquare(n, t);
     if (x <= 0.0)
       x = 0.5 * t;
   } while (fabs(x - t) >= TINY);
   return (x);
}

   double pdfStudent(long n, double x)
/* ===================================
 * NOTE: use n >= 1 and x > 0.0
 * ===================================
 */
{
   double s, t;

   s = -0.5 * (n + 1) * log(1.0 + ((x * x) / (double) n));
   t = -LogBeta(0.5, n / 2.0);
   return (exp(s + t) / sqrt((double) n));
```

```
}

   double cdfStudent(long n, double x)
/* ==================================
 * NOTE: use n >= 1 and x > 0.0
 * ==================================
 */
{
   double s, t;

   t = (x * x) / (n + x * x);
   s = InBeta(0.5, n / 2.0, t);
   if (x >= 0.0)
     return (0.5 * (1.0 + s));
   else
     return (0.5 * (1.0 - s));
}

   double idfStudent(long n, double u)
/* ==================================
 * NOTE: use n >= 1 and 0.0 < u < 1.0
 * ==================================
 */
{
   double t, x = 0.0;                  /* initialize to the mean, then */

   do {                                /* use Newton-Raphson iteration */
     t = x;
     x = t + (u - cdfStudent(n, t)) / pdfStudent(n, t);
   } while (fabs(x - t) >= TINY);
   return (x);
}

/* ====================================================================
 * The six functions that follow are a 'special function' mini-library
 * used to support the evaluation of pdf, cdf and idf functions.
 * ====================================================================
 */

   static double LogGamma(double a)
/* ================================================================================
 * LogGamma returns the natural log of the gamma function.
 * NOTE: use a > 0.0
 *
 * The algorithm used to evaluate the natural log of the gamma function is
 * based on an approximation by C. Lanczos, SIAM J. Numerical Analysis, B,
 * vol 1, 1964. The constants have been selected to yield a relative error
 * which is less than 2.0e-10 for all positive values of the parameter a.
 * ================================================================================
 */
{
   double s[6], sum, temp;
   int    i;

   s[0] = 76.180091729406 / a;
   s[1] = -86.505320327112 / (a + 1.0);
   s[2] = 24.014098222230 / (a + 2.0);
   s[3] = -1.231739516140 / (a + 3.0);
   s[4] =  0.001208580030 / (a + 4.0);
   s[5] = -0.000005363820 / (a + 5.0);
   sum =  1.000000000178;
   for (i = 0; i < 6; i++)
```

```
    sum += s[i];
  temp = (a - 0.5) * log(a + 4.5) - (a + 4.5) + log(SQRT2PI * sum);
  return (temp);
}

  double LogFactorial(long n)
/* ================================================================
 * LogFactorial(n) returns the natural log of n!
 * NOTE: use n >= 0
 *
 * The algorithm used to evaluate the natural log of n! is based on a
 * simple equation which relates the gamma and factorial functions.
 * ================================================================
 */
{
  return (LogGamma(n + 1));
}

  static double LogBeta(double a, double b)
/* ================================================================
 * LogBeta returns the natural log of the beta function.
 * NOTE: use a > 0.0 and b > 0.0
 *
 * The algorithm used to evaluate the natural log of the beta function is
 * based on a simple equation which relates the gamma and beta functions.
 *
 */
{
  return (LogGamma(a) + LogGamma(b) - LogGamma(a + b));
}

  double LogChoose(long n, long m)
/* ================================================================
 * LogChoose returns the natural log of the binomial coefficient C(n,m).
 * NOTE: use 0 <= m <= n
 *
 * The algorithm used to evaluate the natural log of a binomial coefficient
 * is based on a simple equation which relates the beta function to a
 * binomial coefficient.
 * ================================================================
 */
{
  if (m > 0)
    return (-LogBeta(m, n - m + 1) - log(m));
  else
    return (0.0);
}

  static double InGamma(double a, double x)
/* ================================================================
 * Evaluates the incomplete gamma function.
 * NOTE: use a > 0.0 and x >= 0.0
 *
 * The algorithm used to evaluate the incomplete gamma function is based on
 * Algorithm AS 32, J. Applied Statistics, 1970, by G. P. Bhattacharjee.
 * See also equations 6.5.29 and 6.5.31 in the Handbook of Mathematical
 * Functions, Abramowitz and Stegum (editors). The absolute error is less
 * than 1e-10 for all non-negative values of x.
 * ================================================================
 */
{
  double t, sum, term, factor, f, g, c[2], p[3], q[3];
```

```
  long  n;

  if (x > 0.0)
    factor = exp(-x + a * log(x) - LogGamma(a));
  else
    factor = 0.0;
  if (x < a + 1.0) {             /* evaluate as an infinite series - */
    t    = a;                    /* A & S equation 6.5.29        */
    term = 1.0 / a;
    sum  = term;
    while (term >= TINY * sum) {  /* sum until 'term' is small */
      t++;
      term *= x / t;
      sum  += term;
    }
    return (factor * sum);
  }
  else {                         /* evaluate as a continued fraction - */
    p[0] = 0.0;                  /* A & S eqn 6.5.31 with the extended */
    q[0] = 1.0;                  /* pattern 2-a, 2, 3-a, 3, 4-a, 4,... */
    p[1] = 1.0;                  /* - see also A & S sec 3.10, eqn (3) */
    q[1] = x;
    f    = p[1] / q[1];
    n    = 0;
    do {                         /* recursively generate the continued */
      g  = f;                    /* fraction 'f' until two consecutive */
      n++;                       /* values are small               */
      if ((n % 2) > 0) {
        c[0] = ((double) (n + 1) / 2) - a;
        c[1] = 1.0;
      }
      else {
        c[0] = (double) n / 2;
        c[1] = x;
      }
      p[2] = c[1] * p[1] + c[0] * p[0];
      q[2] = c[1] * q[1] + c[0] * q[0];
      if (q[2] != 0.0) {         /* rescale to avoid overflow */
        p[0] = p[1] / q[2];
        q[0] = q[1] / q[2];
        p[1] = p[2] / q[2];
        q[1] = 1.0;
        f    = p[1];
      }
    } while ((fabs(f - g) >= TINY) || (q[1] != 1.0));
    return (1.0 - factor * f);
  }
}

  static double InBeta(double a, double b, double x)
/* ============================================================================
 * Evaluates the incomplete beta function.
 * NOTE: use a > 0.0, b > 0.0 and 0.0 <= x <= 1.0
 *
 * The algorithm used to evaluate the incomplete beta function is based on
 * equation 26.5.8 in the Handbook of Mathematical Functions, Abramowitz
 * and Stegum (editors). The absolute error is less than 1e-10 for all x
 * between 0 and 1.
 * ============================================================================
 */
{
  double t, factor, f, g, c, p[3], q[3];
```

```
  int    swap;
  long   n;

  if (x > (a + 1.0) / (a + b + 1.0)) { /* to accelerate convergence */
    swap = 1;                        /* complement x and swap a & b */
    x    = 1.0 - x;
    t    = a;
    a    = b;
    b    = t;
  }
  else                               /* do nothing */
    swap = 0;
  if (x > 0)
    factor = exp(a * log(x) + b * log(1.0 - x) - LogBeta(a,b)) / a;
  else
    factor = 0.0;
  p[0] = 0.0;
  q[0] = 1.0;
  p[1] = 1.0;
  q[1] = 1.0;
  f    = p[1] / q[1];
  n    = 0;
  do {                              /* recursively generate the continued */
    g = f;                          /* fraction 'f' until two consecutive */
    n++;                            /* values are small                   */
    if ((n % 2) > 0) {
      t = (double) (n - 1) / 2;
      c = -(a + t) * (a + b + t) * x / ((a + n - 1.0) * (a + n));
    }
    else {
      t = (double) n / 2;
      c = t * (b - t) * x / ((a + n - 1.0) * (a + n));
    }
    p[2] = p[1] + c * p[0];
    q[2] = q[1] + c * q[0];
    if (q[2] != 0.0) {              /* rescale to avoid overflow */
      p[0] = p[1] / q[2];
      q[0] = q[1] / q[2];
      p[1] = p[2] / q[2];
      q[1] = 1.0;
      f    = p[1];
    }
  } while ((fabs(f - g) >= TINY) || (q[1] != 1.0));
  if (swap)
    return (1.0 - factor * f);
  else
    return (factor * f);
}
```

A.4.13 rvms.h

```
/* ------------------------------------------------------------
 * Name          : rvms.h (header file for the library rvms.c)
 * Author        : Steve Park & Dave Geyer
 * Language      : ANSI C
 * Latest Revision : 11-02-96
 * ------------------------------------------------------------
 */
```

```
#if !defined( _RVMS_ )
#define _RVMS_

double LogFactorial(long n);
double LogChoose(long n, long m);

double pdfBernoulli(double p, long x);
double cdfBernoulli(double p, long x);
long   idfBernoulli(double p, double u);

double pdfEquilikely(long a, long b, long x);
double cdfEquilikely(long a, long b, long x);
long   idfEquilikely(long a, long b, double u);

double pdfBinomial(long n, double p, long x);
double cdfBinomial(long n, double p, long x);
long   idfBinomial(long n, double p, double u);

double pdfGeometric(double p, long x);
double cdfGeometric(double p, long x);
long   idfGeometric(double p, double u);

double pdfPascal(long n, double p, long x);
double cdfPascal(long n, double p, long x);
long   idfPascal(long n, double p, double u);

double pdfPoisson(double m, long x);
double cdfPoisson(double m, long x);
long   idfPoisson(double m, double u);

double pdfUniform(double a, double b, double x);
double cdfUniform(double a, double b, double x);
double idfUniform(double a, double b, double u);

double pdfExponential(double m, double x);
double cdfExponential(double m, double x);
double idfExponential(double m, double u);

double pdfErlang(long n, double b, double x);
double cdfErlang(long n, double b, double x);
double idfErlang(long n, double b, double u);

double pdfNormal(double m, double s, double x);
double cdfNormal(double m, double s, double x);
double idfNormal(double m, double s, double u);

double pdfLognormal(double a, double b, double x);
double cdfLognormal(double a, double b, double x);
double idfLognormal(double a, double b, double u);

double pdfChisquare(long n, double x);
double cdfChisquare(long n, double x);
double idfChisquare(long n, double u);

double pdfStudent(long n, double x);
double cdfStudent(long n, double x);
double idfStudent(long n, double u);

#endif
```

A.4.14 simulation_type.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Struttura dati per la simulazione
 */

enum _SIMULATION_TYPES {
    FE_EXP,
    FE_ERL,
    FE_HYP
};

typedef enum _SIMULATION_TYPES SIMULATION_TYPES;

char* simulation_traslator(SIMULATION_TYPES t){
  switch(t){
    case FE_EXP:
            return "Exponential";
    break;
    case FE_ERL:
            return "Erlang";
    break;
    case FE_HYP:
            return "Hyperexponential";
    break;
    default:
            return "Error event";
    break;
  }
  return "Wrong code!\n";
}
```

A.4.15 user_signal.c

```
#include <signal.h>

void sigprint() {
    clear_console();
    print_system_state(SYSTEM_PRINT);
}

void sigauto() {
    clear_console();
    print_system_state(SYSTEM_PRINT);
    exit(EXIT_SUCCESS);
}

void add_signals() {
    signal(SIGUSR1, sigprint);
    signal(SIGUSR2, sigauto);
}
```

A.4.16 utils.h

```
/**
 *      Progetto MPSR anno 14/15.
 *      Authori: S. Martucci, A. Valenti
 *      Simulatore web - Utility varie per la stampa
 */
void clear_console() {
#ifdef __WIN32
    system("cls");
#else
    // Assume POSIX
    char str[] = {0x1b, 0x5b, 0x48, 0x1b, 0x5b, 0x4a, '\0'};
    printf("%s", str);;
#endif
}

void print_initial_settings(FILE *g, long long int seed, long bs, long bn) {
    fprintf(g, "%s\t%lld\t\t%s\t%s\n", "SEED", seed, "FS distribution",
         simulation_traslator(type));
    if(SIM_TYPE == 2) {
        fprintf(g, "%s\t%ld\t\t%s\t%ld\n", "Batch Size", bs, "Total Batch", bn);
    } else {
        fprintf(g, "%s\t%ld\t\t%s\t%ld\n", "Stop time", bs, "Step", bn);
    }
    fprintf(g, "%s\t%s\t\n\n", "Threshold", (threshold_flag) ? "YES : 85%" : "NONE");
    fprintf(g, "%s\t%s\t%s\t%s\t", (SIM_TYPE) ? "Current batch" : "Current Stop", "Util
         FS", "Sessions", "Requests");
    fprintf(g, "%s\t%s\t%s\t%s\t", "Average Response Time", "Throughput (Sessions)",
         "Throughput (Requests)", "Average Request per Session");
    fprintf(g, "%s\t%s\t%s\t%s\t%s\t%s\t\n\n", "# Completed", "% Completed", "# Dropped",
         "% Dropped", "# Aborted", "% Aborted");
    fflush(g);
}

void print_system_state_on_file(FILE *g) {
    fprintf(g, "%ld\t%6.16f\t%ld\t%ld\t", (SIM_TYPE) ? current_batch : CURRENT_STOP,
         FS_average_utilization, opened_sessions, requests);
    fprintf(g, "%6.8f\t%6.8f\t%6.8f\t%6.8f\t", average_res_FS + average_res_BES,
         throughput_sessions, throughput_requests, (double) requests/opened_sessions);
    fprintf(g, "%ld\t%6.8f\t%ld\t%6.8f\t%ld\t%6.8f\t\t\n", completed_sessions, ((double)
         completed_sessions/opened_sessions)*100.0, dropped,
         ((double)dropped/(dropped+opened_sessions))*100.0, aborted,
         ((double)aborted/opened_sessions)*100.0);
    fflush(g);
}

void print_final_state_batch(FILE *g) {
    fprintf(g, "\n");
    fprintf(g, "%s\t%6.16f\t%ld\t%ld\t", "Final Statistics",
         __FS_average_utilization/batch_num, __opened_sessions/batch_num,
         __requests/batch_num);
    fprintf(g, "%6.8f\t%6.8f\t%6.8f\t%6.8f\t", average_res_FS + average_res_BES,
         __throughput_sessions, __throughput_requests,
         (double)__requests/__opened_sessions);
    fprintf(g, "%ld\t%6.8f\t%ld\t%6.8f\t%ld\t%6.8f\t\t\n", __completed_sessions,
         ((double)__completed_sessions/__opened_sessions)*100.0, __dropped,
         ((double)__dropped/(__dropped+__opened_sessions))*100.0, __aborted,
         ((double)__aborted/__opened_sessions)*100.0);
    fprintf(g, "%s\t%6.8f\n", "ELAPSED TIME:", current_time);
    fflush(g);
```

```
}

void print_final_state_run(FILE *g) {
    fprintf(g, "\n");
    fprintf(g, "%s\t%6.16f\t%ld\t%ld\t", "Final Statistics", __FS_average_utilization/STOP,
         __opened_sessions/STOP, __requests/STOP);
    fprintf(g, "%6.8f\t%6.8f\t%6.8f\t%6.8f\t", average_res_FS + average_res_BES,
         __throughput_sessions, __throughput_requests,
         (double)__requests/__opened_sessions);
    fprintf(g, "%ld\t%6.8f\t%ld\t%6.8f\t%ld\t%6.8f\t\t\n", __completed_sessions,
         ((double)__completed_sessions/__opened_sessions)*100.0, __dropped,
         ((double)__dropped/(__dropped+__opened_sessions))*100.0, __aborted,
         ((double)__aborted/__opened_sessions)*100.0);
    fprintf(g, "%s\t%6.8f\n", "ELAPSED TIME:", current_time);
    fflush(g);
}

void print_system_state(EVENT_TYPE t) {
    printf("::::::::::::::::::::::::::::: Timer :::::::::::::::::::::::::::::\n");
    if(SIM_TYPE)
        printf("Current batch.............: %ld of %ld\n", current_batch, batch_num);
    else
        printf("Current stop..............: %ld of %ld\n", CURRENT_STOP, STOP);
    printf("Current job...............: %ld of %ld\n", completed_sessions, batch_size);
    printf("Current time..............: %6.8f\n", current_time);
    printf("\n");
    printf("::::::::::::::::::::::: Front Server Info :::::::::::::::::::::::\n");
    printf("Choosen distribution......: %s\n", simulation_traslator(type));
    printf("Queue length..............: %d\n", queue_length_FS);
    printf("Active....................: %s\n", (busy_FS) ? "YES" : "NO");
    printf("Average response time.....: %6.8f\n", average_res_FS);
    printf("Average utilization.......: %6.8f%%\n", FS_average_utilization*100);
    printf("\n");
    printf("::::::::::::::::::::::: Back-end Server Info ::::::::::::::::::::::\n");
    printf("Queue length..............: %d\n", queue_length_BES);
    printf("Active....................: %s\n", (busy_BES) ? "YES" : "NO");
    printf("Average response time.....: %6.8f\n", average_res_BES);
    printf("\n");
    printf("::::::::::::::::::::::::::: Client Info ::::::::::::::::::::::::::\n");
    printf("Number of active client...: %d\n", active_client);
    printf("Average response time.....: %6.8f\n", average_res_client);
    printf("\n");
    printf("::::::::::::::::::::::::::: Generic Info ::::::::::::::::::::::::::\n");
    printf("Event type................: %s\n", event_translator(t));
    printf("Opened sessions...........: %ld\n", opened_sessions);
    printf("Completed sessions........: %ld\n", completed_sessions);
    printf("Completed requests........: %ld\n", requests);
    printf("Requests per session......: %6.8f\n", (double) requests/opened_sessions);
    printf("Sessions throughput.......: %6.8f\n", throughput_sessions);
    printf("Requests throughput.......: %6.8f\n", throughput_requests);
    printf("Connection dropped........: %ld\n", dropped);
    printf("Connection aborted........: %ld\n", aborted);
    printf("Threshold.................: %s\n", threshold_flag ? "YES : 85%" : "NONE");
    printf("\n");
}
```