UNIVERSITÀ DEGLI STUDI DI ROMA
TOR VERGATA
FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

A.A. 2015/2016

Performance Modeling Of Computer Systems And Networkig

Simulatore di traffico in un sistema Multi-Tier

DOCENTE

Vittoria De Nitto Persone'

STUDENTI

0234240 Agostini Stefano
0229621 Belli Gabriele
0233502 Salomè Paolo

"Se c'è un modo di fare meglio, trovalo"

Thomas Alva Edison

Indice

Capitolo 1

Introduzione

1.1 Panoramica

Lo scopo del progetto assegnatoci è quello di analizzare le prestazioni di un'architettura three-tier che fornisce un servizio web.
Un'applicazione realizzata con questa architettura è suddivisa in tre componenti:

- web server,
- application server,
- database di back-end.

Solitamente il web server e l'application server risiedono nello stesso server fisico che è chiamato server di front-end.
L'accesso da parte di un utente al servizio web costituisce l'inizio di una sessione, che può essere formata da più richieste. Nel momento in cui viene generata una sessione, le richieste del client transitano più volte tra front-end server e beck-end server prima di ritornare al client.
Quando la richiesta torna al client, che l'ha generata, quest'ultimo attende un tempo chiamato think time, prima di generare una nuova richiesta relativa alla sessione corrente. La simulazione del sistema reale è stata condotta attraverso i seguenti passaggi:

1. Definizione degli obiettivi;
2. Costruzione del modello concettuale;
3. Costruzione delle specifiche del modello concettuale;
4. Modello computazionale;
5. Verifica;
6. Validazione;
7. Progettazione degli esperimenti;

8. Organizzazione degli esperimenti;
9. Analisi dell'output della simulazione;
10. Fase decisionale;
11. Documentazione dei risultati.

Termineremo la trattazione mostrando il codice prodotto.
Buona lettura.

Capitolo 2

Simulazione

2.1 Obiettivi

L'obiettivo del progetto è la realizzazione di un simulatore ad eventi di un sistema che fornisce un generico servizio web.
Dove il sistema è composto da:

- un Front Server (FS) che racchiude in se le funzionalità offerte sia dal web server che dall'application server;
- un Back End Server (BE) che offre invece le funzionalità di interfaccia al database.

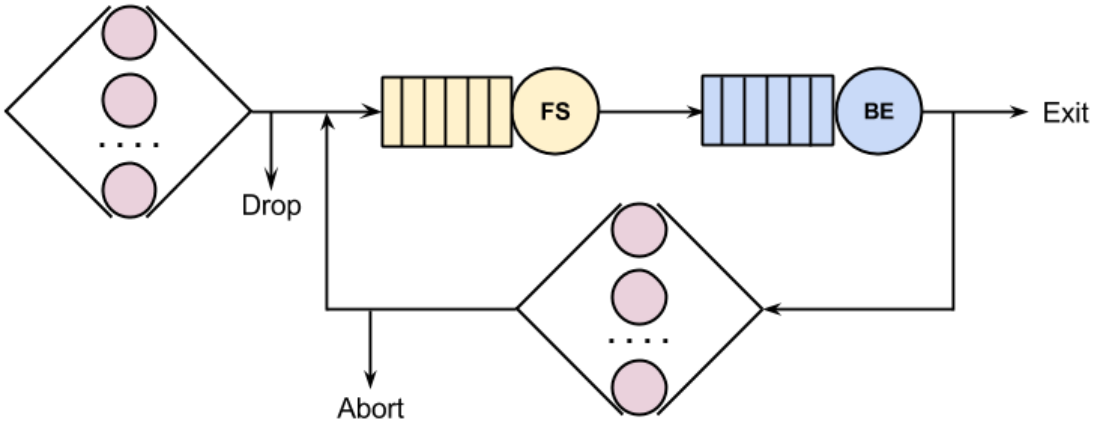
Tra le varie richieste emergeva quella di applicare al web server un meccanismo di retroregolazione in grado di tenere conto dei risultati del sistema per modificare le caratteristiche del sistema stesso, ovvero l'implementazione di un sistema di overload management, cioè un controllore che si occupa di filtrare le sessioni in ingresso al fine di offrire un servizio stabile ed accettabile per tutte le sessioni che il sistema può accettare.
Oltre a quanto su evidenziato veniva chiesto lo studio del sistema in condizioni di stazionarietà, della valutazione del tempo di risposta del sistema, del throughput utile, del numero di drop e abort e infine l'analisi della correlazione lineare tra il delay e la wait del Back End.

2.2 Modello concettuale

Il sistema modellato è 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: a tal proposito, per garantire il rientro nel sistema, si utilizza una coda a priorità, di capacità infinita. Il sistema può essere monitorato tramite il meccanismo di overload management(OMM): se esso è attivo e l'utilizzazione osservata supera 85% si attivano i meccanismi di abort e drop fintanto che l'utilizzazione non scende sotto il 75%.

All'arrivo di una nuova sessione, il meccanismo di OMM se attivo esegue tutti i controlli di ammissibilità e, qualora il sistema fosse sufficientemente libero, permette al nuovo utente di entrare. Tale sessione viene processata dal Front Server, dopo un'eventuale attesa nella sua coda, e successivamente entra nel Back-End Server, nuovamente dopo un possibile ritardo di coda. Al termine del servizio la sessione esce dal sistema nel caso in cui abbia completato tutte le richieste che la componevano, oppure rientra nel sistema attraverso un 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 ma, in caso di sistema saturo, rischia di essere abortita dal meccanismo di abort.
Quindi per evitare di saturare il sistema si utilizzano due meccanismi:

- **drop**: se l'utilizzazione del Front-end ha superato 85% e ancora non scende sotto la soglia del 75%, il sistema non accetta nessuna nuova connessione in ingresso;
- **abort**: se l'utilizzazione del Front-end ha superato 85% e ancora non scende sotto la soglia del 75%, il sistema rifiuta le nuove richieste appartenenti alle sessioni attive;



Nella schematizzazione su evidenziata si è scelto di replicare i nodi che rappresentano gli utenti per dare risalto alla suddivisione tra drop delle nuove sessioni e abort delle richieste appartenenti alle sessioni già attive.
Il sistema è modellato attraverso l'utilizzo della Next-Event Time Advance Simulation.
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

2.2.1 Variabili di stato

Il sistema è descritto completamente dalle seguenti variabili di stato:

- numero di richieste in servizio al tempo t nel front-end e nel back-end

- numero di richieste nelle code di front-end server e nel back end server al tempo t
- il numero di sessioni attive nel nodo di think.

2.3 Modello delle specifiche

Nello sviluppo del modello delle specifiche, innanzitutto l'attenzione è 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:

- tempo di servizio del front-end, distribuito esponenzialmente con media 0,00456 s;
- tempo di servizio del back-end, distribuito esponenzialmente con media 0,00117 s;
- tempo di interarrivo delle nuove sessioni, distribuito esponenzialmente di parametro 35 sessioni/s;
- lunghezza delle sessioni, uniforme discreta di parametri a=5, b=35;
- tempo di thinking, distribuito esponenzialmente di parametro 7 s.

Per le simulazioni viene utilizzato il modello next-event. In questo modello le variabili del sistema avanzano in maniera discreta secondo una sequenza di eventi programmata di volta in volta durante l'esecuzione.
Tale approccio assicura una generalizzazione del modello. Sarà quindi possibile aggiornare, modificare e complicare il modello in maniera estremamente veloce. Il modello ad eventi successivi è composto da cinque entità fondamentali che saranno illustrate nelle sezioni seguenti.

2.3.1 Design del simulatore next-event

Nella costruzione del modello di simulazione next-event sono stati eseguiti principalmente 3 step:

- costruzione delle variabili di stato, che forniscono una descrizione completa del sistema;
- identificazione degli eventi;
- costruzione di un insieme di algoritmi che definiscono i cambiamenti di stato che devono essere eseguiti all'occorrenza di un dato evento.

Di seguito illustriamo gli elementi sopra evidenziati che sono essenziali nel modello di simulazione next-event.

Stato del sistema

Per il calcolo delle metriche è stato necessario monitorare le seguenti variabili:

- tempo attuale di simulazione;
- tempo del prossimo evento;

- numero di sessioni arrivate nel sistema;
- numero di sessioni dropped dal sistema;
- numero di sessioni aborted dal sistema;
- numero di sessioni completate;
- numero di sessioni presenti nel Front End Server;
- numero di sessioni presenti nel Back End Server;
- numero di sessioni presenti nel nodo di think;
- numero di sessioni transitate nel Front End Server;
- numero di sessioni transitate nel Back End Server;
- numero di sessioni transitate nel nodo di think;

Eventi

Gli eventi che caratterizzano il sistema sono quattro:

- arrivo di una nuova sessione: un utente sta richiedendo al sistema di poter iniziare una nuova sessione. Se il meccanismo OMM non è attivo oppure non lo è quello di DROP la nuova sessione sarà accettata e accodata nel nodo di Front End.Qualora il meccanismo di DROP sia attivo, invece, per poter accettare una nuova sessione bisognerà attendere che l'utilizzazione sia inferiore al 75%.
- fine servizio del Front End Server: la sessione esce dal nodo di Front End e viene accodata al nodo di Back End;
- fine servizio del Back End Server: la sessione esce dal nodo di Back End e se ha completato tutte le sue richieste esce dal sistema altrimenti passa al nodo di think;
- fine tempo di think: il comportamento del sistema durante questo evento cambia se il meccanismo di OMM è attivo o meno. Se non è attivo la sessione esce dal nodo di think e viene accodata al nodo di Front End. Se è attivo il comportamento del sistema dipende dal fatto che il meccanismo di abort sia attivo o meno: nel primo caso la sessione viene scartata altrimenti no.

Oltre ai quattro eventi che descrivono il sistema considerato è stato deciso di generare un altro tipo di evento, detto sampling, che viene innescato ad intervalli regolari senza alterare lo stato del sistema, ma che può essere utilizzato per monitorarne lo stato.

2.3.2 Calcolo delle principali statistiche sulle variabili di stato

Per quanto riguarda il calcolo delle medie campionarie, varianze campionarie e deviazioni standard campionarie come il tempo medio di risposta del sistema e i tempi di attesa nelle singole code viene usato l'algoritmo one-pass di Wellford.

2.3.3 Indici di prestazione

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 Throughput: indica il numero di sessioni completate dal sistema in un'unità di tempo.
- Tempo di risposta del sistema: indica il tempo che intercorre tra l'istante in cui una richiesta entra nel front-server e l'istante in cui la stessa esce dal back-end server
- Aborted ratio: indica la percentuale di richieste rifiutate dal sistema rispetto al totale delle richieste.
- Drop ratio: indica la percentuale delle sessioni rifiutate dal sistema rispetto al totale delle sessioni

2.3.4 Durata delle sperimentazioni

Si è deciso di analizzare le metriche tramite un approccio finite-horizon statistics, che permette di misurare nel transitorio, per una lunghezza di tempo finita, la metriche di interesse.

2.4 Modello computazionale

Il modello è stato implementato mediante diverse strutture dati.

Struttura node.

```
1  struct node
2  {
3      struct node* next;
4
5      long     length;
6
7      double arrival_FS;
8      double ended_FS;
9      double ended;
10
11     double sessionStart;
12     double sessionEnded;
13
14     double endThinkTime;
15 };
```

La struttura Node rappresenta un nodo della lista ossia una sessione in coda nel centro. In questa struttura vengono memorizzati il numero di richieste rimanenti per completare la sessione, il tempo di arrivo della richiesta al Front Node, il tempo di uscita della richiesta dal Back End Server, il tempo in cui la sessione è entrata nel sistema, il tempo in cui la sessione ha completato le sue richieste. Il campo endThinkTime è utilizzato nel solo centro che rappresenta il think time degli utenti. Tale centro, a differenza dei restanti due, non ha coda per cui è necessario generare il tempo di think

per ogni sessione nello stesso momento in cui si aggiunge il nodo al centro e memorizzare il tempo assoluto al quale l'evento di fine think scatta per ognuna delle sessioni.

Struttura list.

```
1 struct list
2 {
3     struct node* head;
4     struct node* tail;
5     int size;
6 };
```

La struttura *list* rappresentata una lista contenente la coda dei centri che compongono il sistema.

Struttura server.

```
1 struct server
2 {
3     struct node* internal_node;
4     struct list* fifo;
5 };
```

Con la struttura *server* viene rappresentato uno dei centri che compongono il sistema memorizzando per ognuno di essi la richiesta in lavorazione e la lista di richieste che dovranno essere portate a termine.

Struttura area.

```
1 struct area
2 {
3     double x;
4     double q;
5     double l;
6 };
```

La struttura *area* tiene traccia del tempo cumulativo trascorso nell'intero nodo, in coda e in servizio da parte di tutte le sessioni che attraversano il nodo.

Struttura clock.

```
1 struct clock
2 {
3     double current;
4     double next;
5 };
```

La struttura *clock* modella il tempo di simulazione memorizzando i tempi assoluti relativi all'evento corrente e al prossimo evento.

Struttura calendar.

```
1 struct calendar
2 {
3     double* events_times;
4 };
```

É necessario usare nella simulazione ad eventi successivi un meccanismo di avanzamento del tempo per garantire che gli eventi scorrano in ordine corretto e che il clock di simulazione non torni mai indietro, per questo motivo è stata utilizzata la struttura *calendar*.

Struttura Metrics.

```
1  struct Metrics{
2      double sys_resp;
3      double sys_thr;
4
5      double fs_resp;
6      double fs_thr;
7      double fs_util;
8      double fs_pop;
9
10     double be_resp;
11     double be_delay;
12     double be_thr;
13     double be_util;
14
15     double th_pop;
16     double drop_ratio;
17     double abort_ratio;
18
19     double total_time;
20 };
```

Per analizzare il sistema sono state calcolate le seguenti metriche presenti nella struttura su evidenziata:

- il tempo di risposta e il throughput del sistema;
- il numero di sessioni presenti, tempo di risposta, il throughput e l'utilizzazione del front-end;
- il tempo di risposta,il ritardo medio, il throughput e l'utilizzazione del back-end;
- la popolazione, il numero delle sessioni droppate e il numero delle richieste abortite del nodo di think;
- il tempo totale di esecuzione della simulazione.

2.4.1 Gestione eventi

Gli eventi che il sistema gestisce sono:

- NEW_SESSION_INDEX : questo evento rappresenta la generazione di una nuova sessione. Il primo controllo che viene effettuato è che tale generazione non sia avvenuta ad un tempo superiore allo STOP_SIMULATION. In tal caso la nuova sessione verrebbe ignorata e impostato ad INFINITO il prossimo tempo di generazione, assicurandosi in tale maniera che nessun altro evento di nuova generazione sia creato. Se il tempo è antecedente allo STOP_SIMULATION verrà aggiornata la variabile total_generated_sessions che tiene conto delle sessioni generate dal

sistema. Se il meccanismo di drop è attivo il sistema è troppo saturo per accettare nuove sessione e, viene quindi aggiornato il contatore delle sessioni rifiutate,total_dropped_sessions. In caso contrario verrà aggiornato il contatore del numero di sessioni attive nel nodo di Front End, xfs. Vengono generate le informazioni relative alla nuova sessione (come la sua lunghezza, e il tempo in cui è entrata nel sistema), e la sessione viene accodata nella lista relativa al nodo di Front End. Se tale sessione è l'unica presente nella coda del Front End viene inoltre generato un evento di completamento per tale nodo (EXIT_FS_INDEX);

- EXIT_FS_INDEX: questo evento rappresenta il completamento del servizio per una sessione da parte del nodo di Front End. Viene incrementato il contattore delle sessioni che hanno transitato per tale nodo ,num_ended_req_fs. La sessione viene eliminata dalla lista rappresentante il nodo di Front End e accodata nella lista relativa al nodo di Back End. Se tale sessione è l'unica presente nella coda del Back End viene inoltre generato un evento di completamento per tale nodo (EXIT_BE_INDEX);

- EXIT_BE_INDEX : questo evento rappresenta il completamento del servizio per una sessione da parte del nodo di Back End. Viene incrementato il contattore delle sessioni che hanno transitato per tale nodo, num_entered_be_serv e decrementato il numero di richieste della sessione. La sessione viene eliminata dalla lista delle sessioni attive sul nodo di Back End e se sono presenti in lista altre sessioni viene generato un nuovo evento di completamento per il nodo di Back End. Viene aggiornata la variabile che tiene conto di tutti i tempi di risposta del sistema e del Back end. Se la sessione ha completato tutte le sue richieste la sessione esce dal sistema, viene incrementato il contatore del numero di richieste che hanno terminato e aggiornate le variabili che tengono conto del tempo totale passato da una richiesta all'interno del sistema. Se invece la sessione avesse altre richieste da completare viene generato un evento di fine think, aggiunta la sessione al nodo di think.

- EXIT_TH_INDEX: questo evento rappresenta il completamento del servizio per una sessione da parte del nodo di think. Se il meccanismo di abort è attivo la sessione viene eliminata dal sistema (abort) e aggiornato il contattore degli abort e dei drop. Se invece il meccanismo di abort è disattivato la sessione viene accodata al nodo di Front End, aggiornando le variabili che tengono conto del tempo in cui la richiesta viene aggiunta al nodo di Front End, necessarie in seguito per calcolare il tempo di risposta del sistema. Viene incrementato il contatore del numero di sessioni attive presenti nel nodo di Front End e se la sessione in esame è l'unica presente nel nodo di Front-End viene inoltre generato un evento di completamento per tale nodo.

Come detto in precedenza il sistema gestisce inoltre l'evento di sampling che non altera in alcun modo lo stato del sistema ma è risultato molto utile per poter monitorare lo stato del sistema ad intervalli regolari.

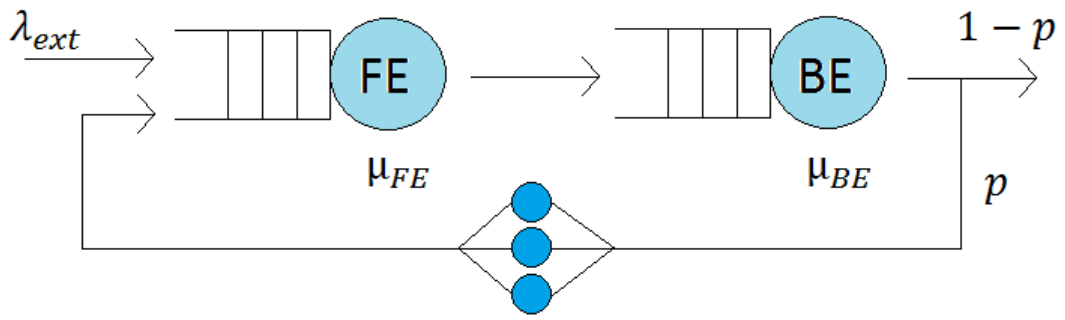
2.5 Verifica

La fase di verifica consente di dimostrare l'effettiva consistenza del programma con il modello delle specifiche.
Si è utilizzata la scrittura sul file per verificare il corretto flusso delle sessioni all'interno del sistema.
Si è notato e verificato inoltre che la lista contenente la sessione si riempia e si svuoti in modo corretto.
I vincoli sullo stato del sistema, sui cambi di fase sono tutti soddisfatti.
Come da specifiche, il simulatore parte per ogni singolo run da e termina in uno stato di quiete: il numero delle sessioni è nullo e tutte le variabili, di stato e di supporto, utilizzate tornano ai valori di partenza.
Inoltre il numero delle sessioni rifiutate cresce consistentemente con la presenza del meccanismo OMM e con il cambiamento del flusso in entrata.
Infine, come ultima verifica, è stato dimostrato che, superato il tempo di STOP_SIMULATION , nessuna nuova sessione fosse accettata dal sistema e, successivamente a tale istante, si assisteva allo svuotamento del sistema in modo consistente.
Non sono stati riscontrati errori in fase di compilazione ed esecuzione.

2.6 Validazione

Per esaminare la consistenza del simulatore con il sistema analizzato, sono state effettuate delle considerazioni basate su un modello analitico semplificato a rete aperta (rete aperta di Jackson).

2.6.1 Analisi senza meccanismo OMM attivo



Mediamente una sessione ha (5+35)/2 = 20 richieste (essendo la lunghezza della sessione distribuita come un'uniforme discreta tra 5 e 35).
Quindi possiamo considerare p = 19/20 e di conseguenza 1-p = 1/20.
Dati:

- λ_{ext} = 35 sessioni/s
- $\mu_{\text{FE}} \simeq$ = 219,3 rich/s

- $\mu_{\text{BE}} \simeq = 851{,}7$ rich/s

Risolviamo la rete aperta tramite Jackson:

$$\begin{cases} \lambda_{\text{FE}} = \lambda_{\text{ext}} + \dfrac{19}{20}\lambda_{\text{BE}} \\ \lambda_{\text{BE}} = \lambda_{\text{FE}} \end{cases} \tag{2.6.1}$$

Dalla soluzione notiamo che:

$$\left\{ \lambda_{\text{FE}} = 700 \frac{rich}{s} > \mu_{\text{FE}} = 219,3 \frac{rich}{s} \right. \tag{2.6.2}$$

Non vi è stabilità in quanto λ è maggiore di μ.
Il throughput del FE si attesta a 219 rich/s e la coda del FE cresce illimitatamente.

2.6.2 Analisi con meccanismo OMM attivo

L'utilizzazione del Front end è da considerare nel range (0,75;0,85), in media quindi $p_{\text{FE}} = 0.8$. Servendoci della Utilization Law:

$$\left\{ X_{\text{FE}} = p_{\text{FE}} * \mu_{\text{FE}} \right. \tag{2.6.3}$$

Troviamo che il throughput del front end server è pari a 175,44 rich/s che è uguale al throughput del back end server.
$X_{\text{FE}} = 175{,}44$ rich/s $= X_{\text{BE}}$ è in linea con i risultati sperimentali di tale metrica mediata tra 400 run di durata 10000 s ognuno.
Infatti $X_{\text{FE}} = X_{\text{BE}}$ si attesta ad un valore di 174,6 rich/s circa dopo 10000 s di simulazione.
Nel modello di analisi sopra descritto (modello teorico), sotto queste condizioni, il sistema raggiunge la stabilità. Tuttavia nelle simulazioni l'utilizzazione è stata misurata in modo cumulativo, non rendendo il meccanismo OMM sensibile alla rilevazione di picchi di carico.Ciò ha portato ad una crescita altalenante e infinita, seppure rallentata, dei tempi medi di attesa in coda e quindi ad una crescita lenta ma infinita della coda del FE, come si evince dai grafici mostrati nel seguito della trattazione.

2.7 Analisi dei risultati

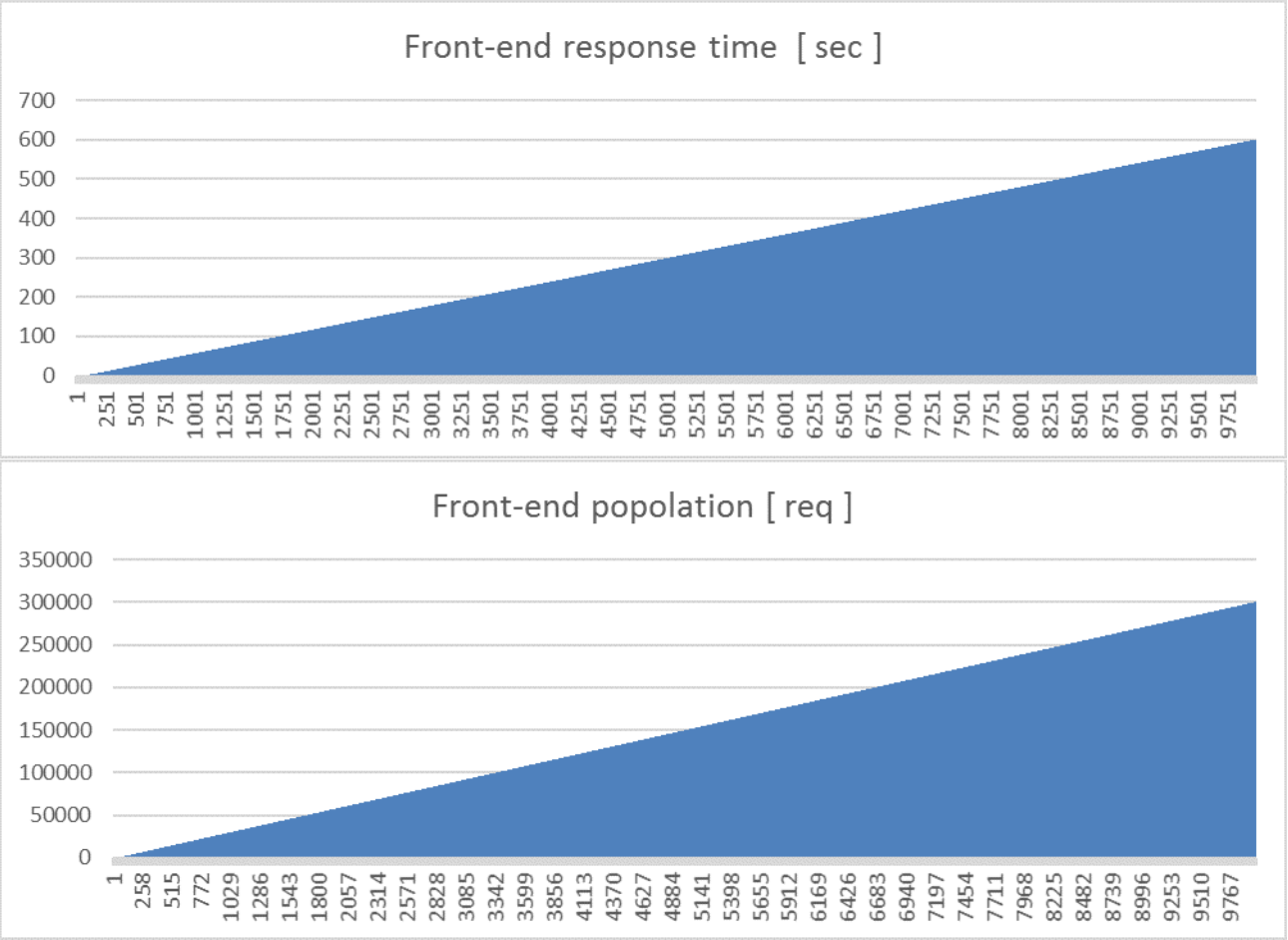
Per poter analizzare il comportamento del sistema sono state eseguite diverse simulazioni della stessa durata dalle quali sono stati prelevati ed analizzati i valori medi. In particolare per analizzare il comportamento del sistema quando il meccanismo di Abort/Drop è attivo sono stati eseguite 400 simulazioni tagliate a 10000 secondi. La larghezza dell'intervallo di confidenza dipende dal quantile della t-student, dalla varianza campionaria e dalla radice di n. Dato che il quantile destro di una t-student a code equiprobabili di ordine 0,975 e 399 gradi di libertà vale 1,96, abbiamo utilizzato un n tale da avere la larghezza dell'intervallo di confidenza pari ad un decimo della deviazione standard e per tanto abbiamo scelto n=400. Con questo approccio sono state prelevate le informazioni relative alle seguenti metriche:

- System response time
- System throughput
- Front end response time
- Front end throughput
- Back end response time
- Back end delay
- Back end throughput
- Abort ratio
- Drop ratio

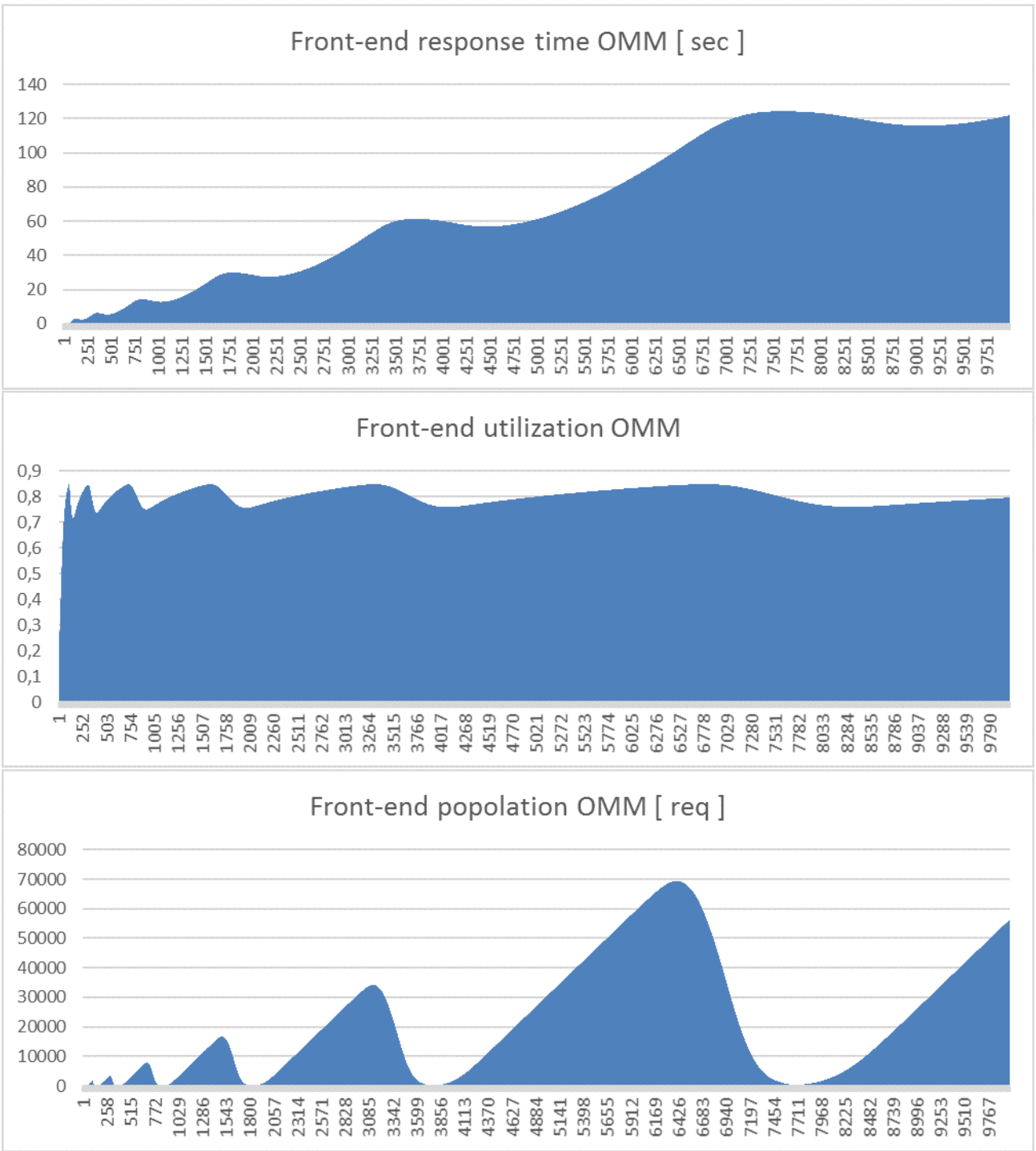
Oltre a queste metriche ne sono state tracciate delle altre per avere una visione maggiore del comportamento del sistema quali la popolazione nel front end e nel nodo di think. È stato adottato lo stesso approccio per analizzare il comportamento del sistema senza meccanismo di blocco (abort/drop ratio perdono importanza in quanto costantemente nulli).
Di tutti i dati raccolti nelle simulazioni sono stati calcolati i valori medi, le varianze e gli intervalli di confidenza e sono stati raccolti in delle tabelle riassuntive che vedremo dopo.
È da sottolineare che gli intervalli di confidenza perdono di significato per le metriche System response time e Front end response time in quanto il sistema non è stabile e non si attesta attorno ad un valore sia con che senza meccanismo di blocco, mentre per le metriche System throughput, Front end throughput e Back end throughput vale lo stesso discorso ma solo se è attivo il meccanismo di blocco.

2.7.1 Front end

Qui effettueremo un confronto delle metriche relative al front end prelevate dai diversi modelli (quello con e quello senza meccanismo di blocco). Come si vede dai grafici sottostanti, nel caso del sistema senza meccanismo di blocco non c'è modo di fermare la tendenza all'infinito del tempo di risposta derivante soprattutto dall'incremento della popolazione in coda.

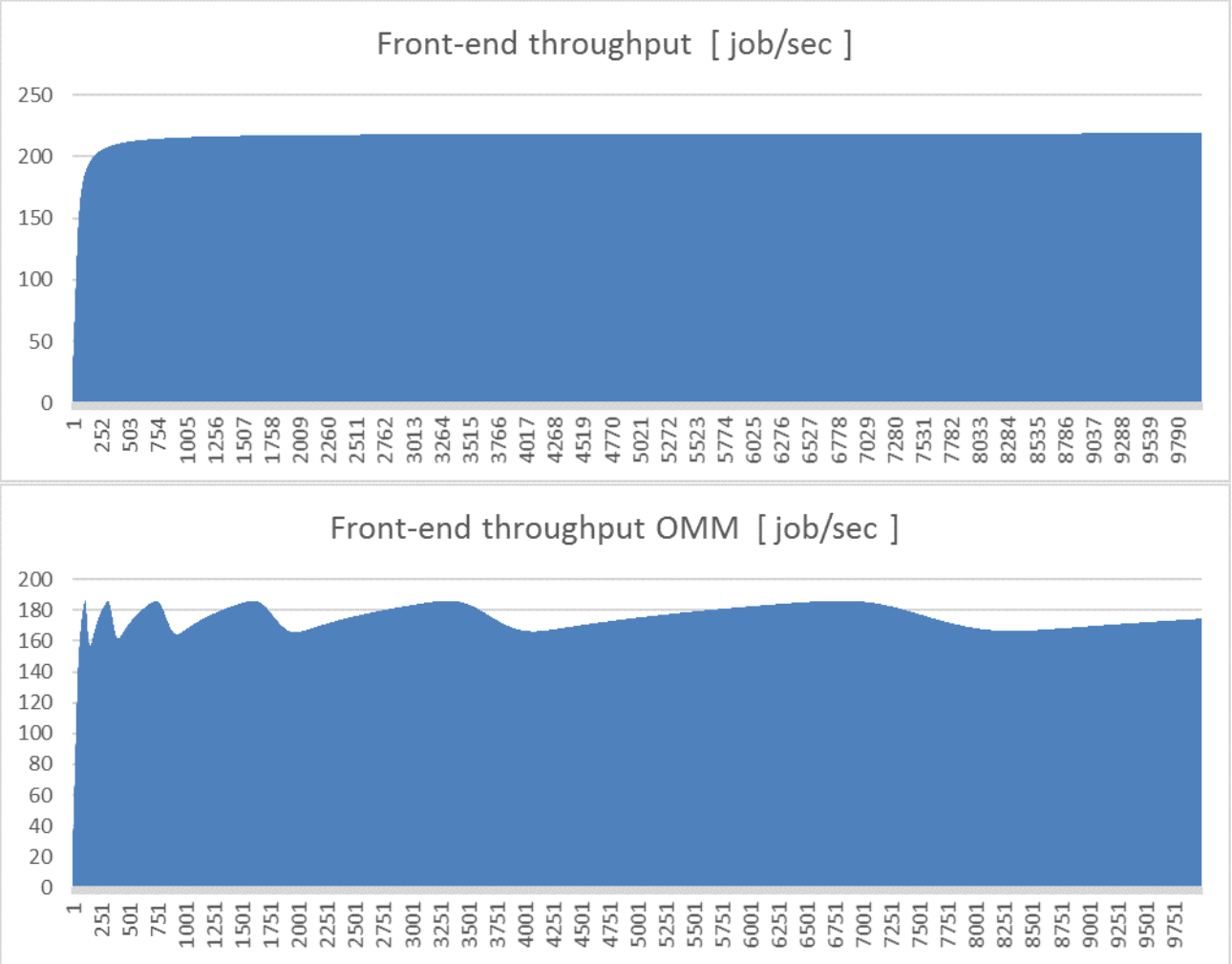


Terminata la trattazione del sistema con meccanismo di blocco non attivo passiamo quindi ad analizzare le metriche relative al front end dei diversi modelli quando l'OMM è attivo.
Nel caso del sistema con meccanismo di blocco questo incremento è solo rallentato in quanto dipende anche dal campionamento dell' utilizzazione del front end dalla quale dipende il meccanismo stesso.



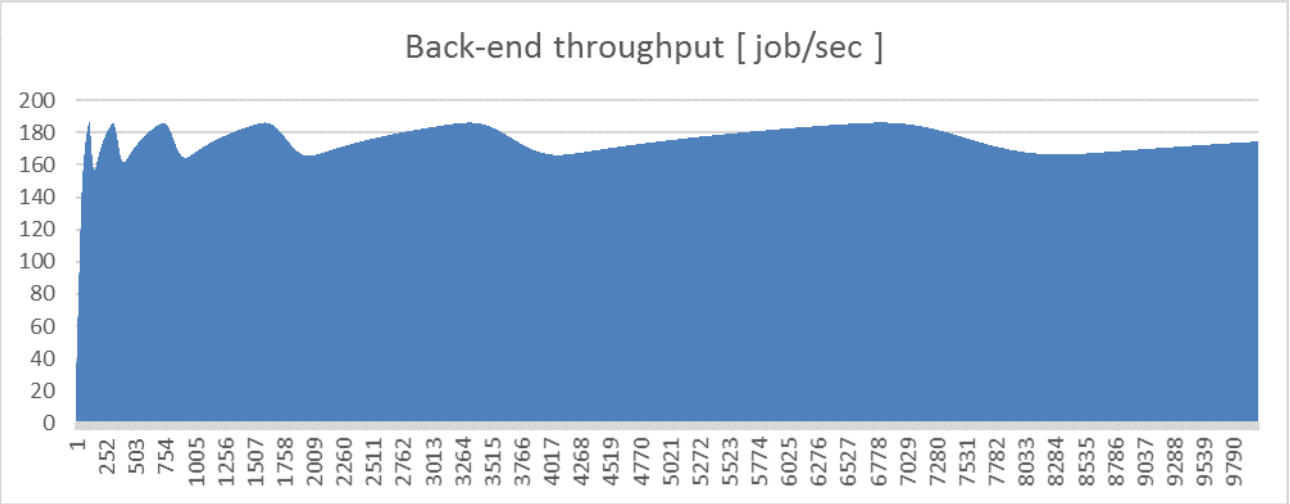
L'andamento del grafico nel caso in cui il sistema adotti il meccanismo di blocco è altalenante a causa del meccanismo (in quanto blocca tutte le richieste se l'utilizzazione supera la soglia di 85% e si disattiva se scende sotto al 75%), ma come si nota ha delle fasi di salita e di discesa sempre più lente all'aumentare del tempo: questo accade per il fatto che l'utilizzazione è stata calcolata in modo

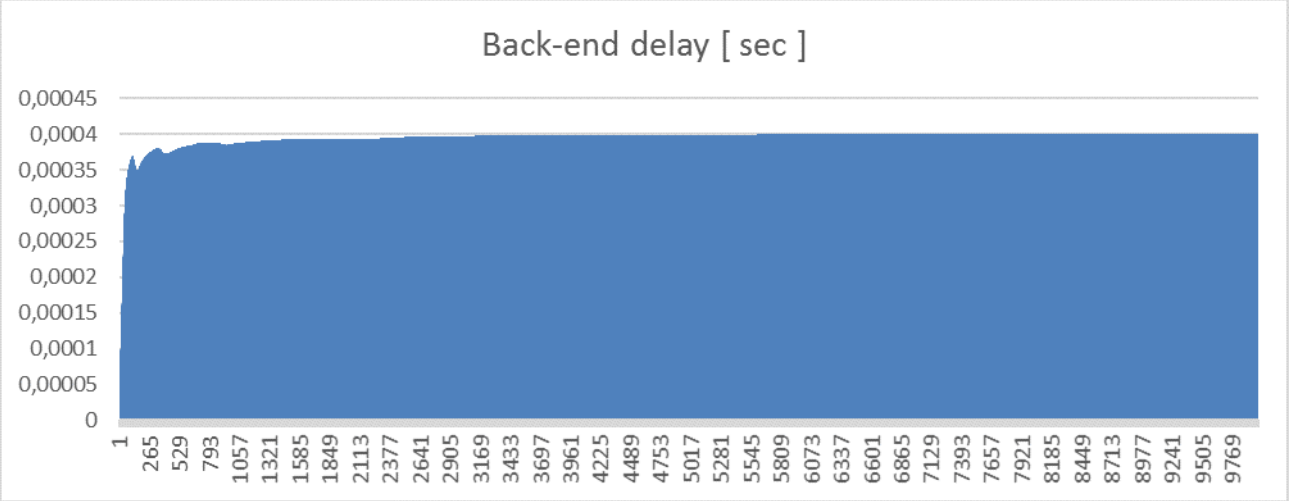
cumulativo e conseguentemente i cambiamenti dell'utilizzazione sono attenuati proporzionalmente al tempo di simulazione corrente.
Per quanto riguarda il throughput del front end nel caso in cui il sistema non adotti il meccanismo di blocco questo sarà abbastanza costante, mentre per il sistema con meccanismo avrà un andamento altalenante per i motivi detti prima.

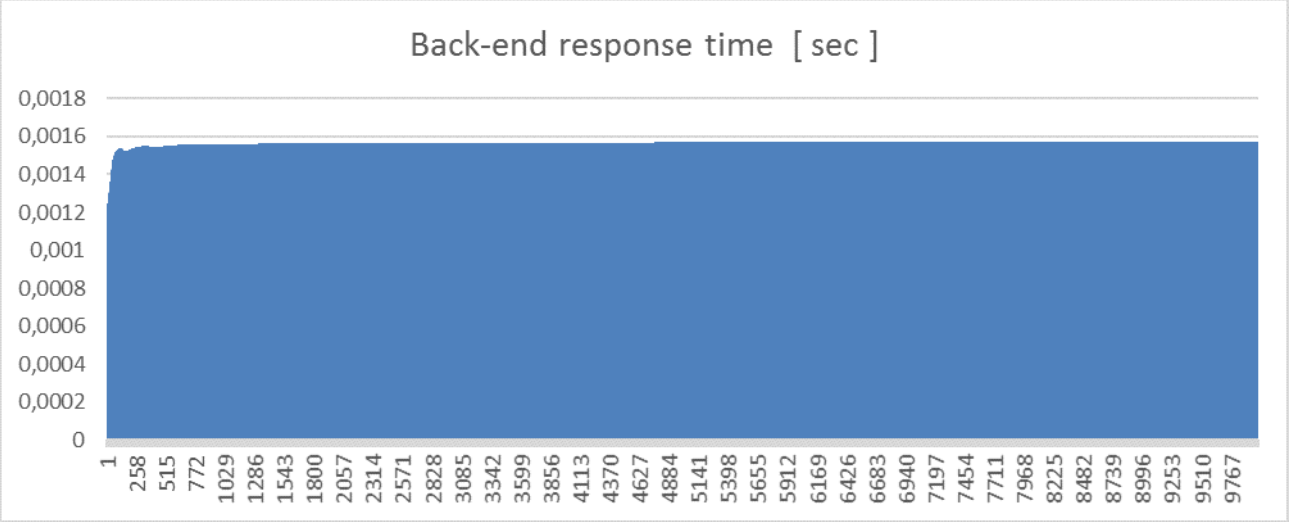


2.7.2 Back end

Il back end a differenza del front end per quanto riguarda il response time ed il delay non subisce alcun cambiamento dall'utilizzo o meno del meccanismo di blocco. Ma per quanto riguarda il throughput, nel sistema che adotta il meccanismo di blocco, si nota l'andamento altalenante visto e descritto nell'analisi del front-end ed inoltre nel caso del sistema con meccanismo di blocco il throughput non è massimo.


Back-end throughput [job/sec]
200
180
160
140
120
100
80
60
40
20
0
1
252
503
754
1005
1256
1507
1758
2009
2260
2511
2762
3013
3264
3515
3766
4017
4268
4519
4770
5021
5272
5523
5774
6025
6276
6527
6778
7029
7280
7531
7782
8033
8284
8535
8786
9037
9288
9539
9790

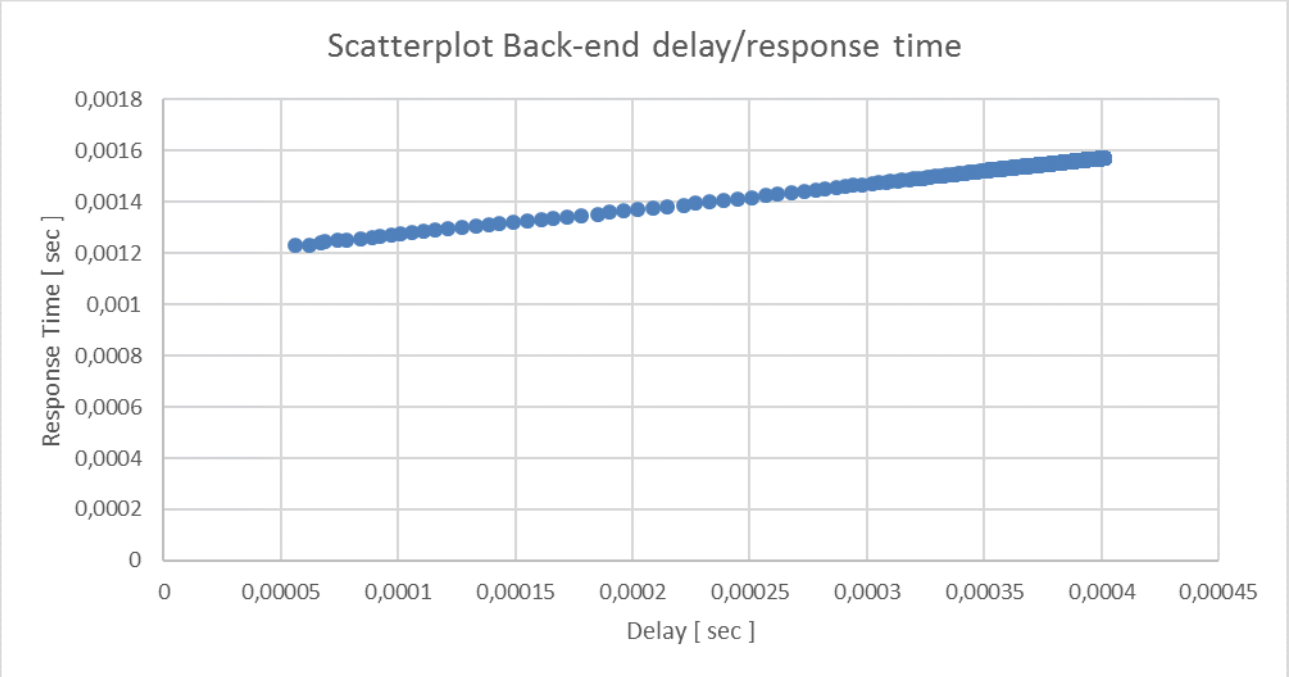

Back-end delay [sec]
0,00045
0,0004
0,00035
0,0003
0,00025
0,0002
0,00015
0,0001
0,00005
0
1
265
529
793
1057
1321
1585
1849
2113
2377
2641
2905
3169
3433
3697
3961
4225
4489
4753
5017
5281
5545
5809
6073
6337
6601
6865
7129
7393
7657
7921
8185
8449
8713
8977
9241
9505
9769


Back-end response time [sec]
0,0018
0,0016
0,0014
0,0012
0,001
0,0008
0,0006
0,0004
0,0002
0
1
258
515
772
1029
1286
1543
1800
2057
2314
2571
2828
3085
3342
3599
3856
4113
4370
4627
4884
5141
5398
5655
5912
6169
6426
6683
6940
7197
7454
7711
7968
8225
8482
8739
8996
9253
9510
9767

Correlazione delay-response time

Il grado di correlazione fra due variabili viene espresso mediante i cosiddetti indici di correlazione. Questi assumono valori compresi in [- 1, + 1], un indice di correlazione pari a zero indica un'assenza di correlazione. Due variabili indipendenti hanno sicuramente un indice di correlazione pari a 0, ma al contrario un valore pari a 0 non implica necessariamente che le due variabili siano indipendenti. Dai dati raccolti è possibile calcolare il grado di correlazione tra il delay ed in response time del back end. Il calcolo è stato effettuato a posteriori utilizzando il coefficiente di correlazione di Pearson-Bravais il quale è calcolato come rapporto tra la covarianza delle due variabili ed il prodotto delle loro deviazioni standard.

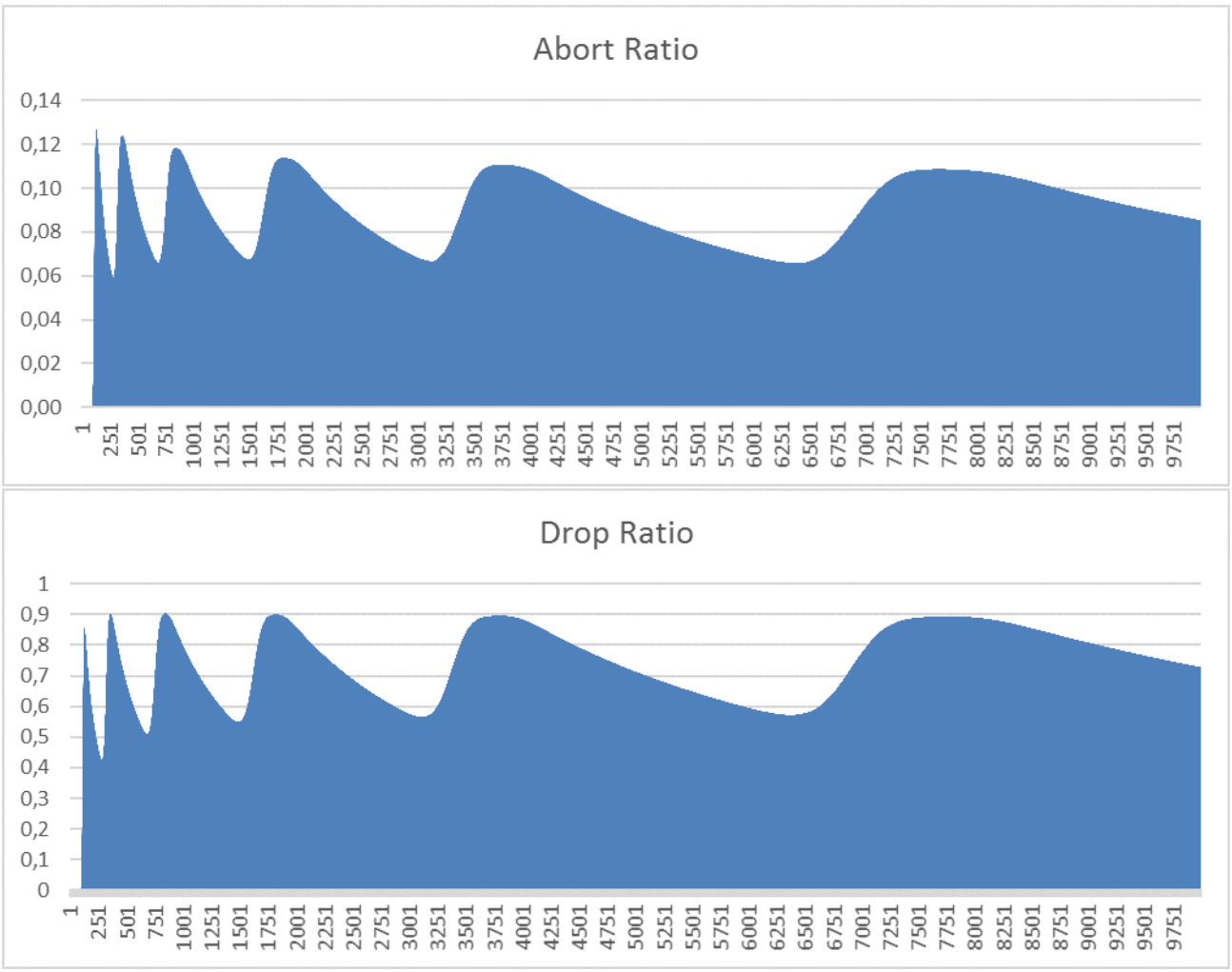
$$-1 \leq \rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i - \mu_x)^2}\sqrt{\sum_{i=1}^{n}(y_i - \mu_y)^2}} \leq +1$$



Dai calcoli effettuati il coefficiente è pari a 0,999808, il che implica forte correlazione tra il delay ed il response time del back end.

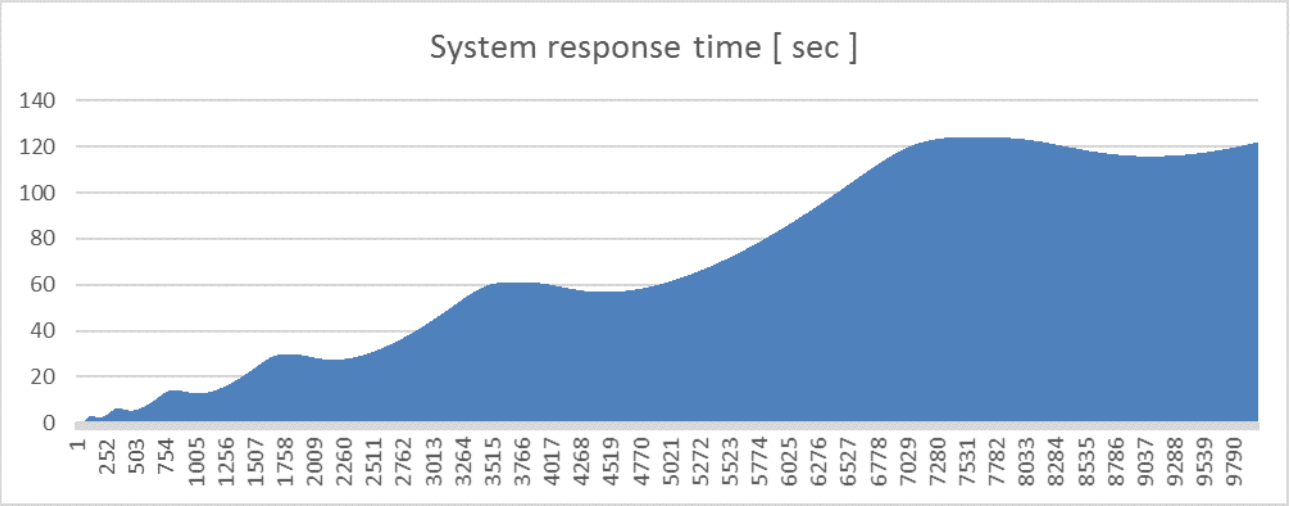
2.7.3 Abort e drop ratio

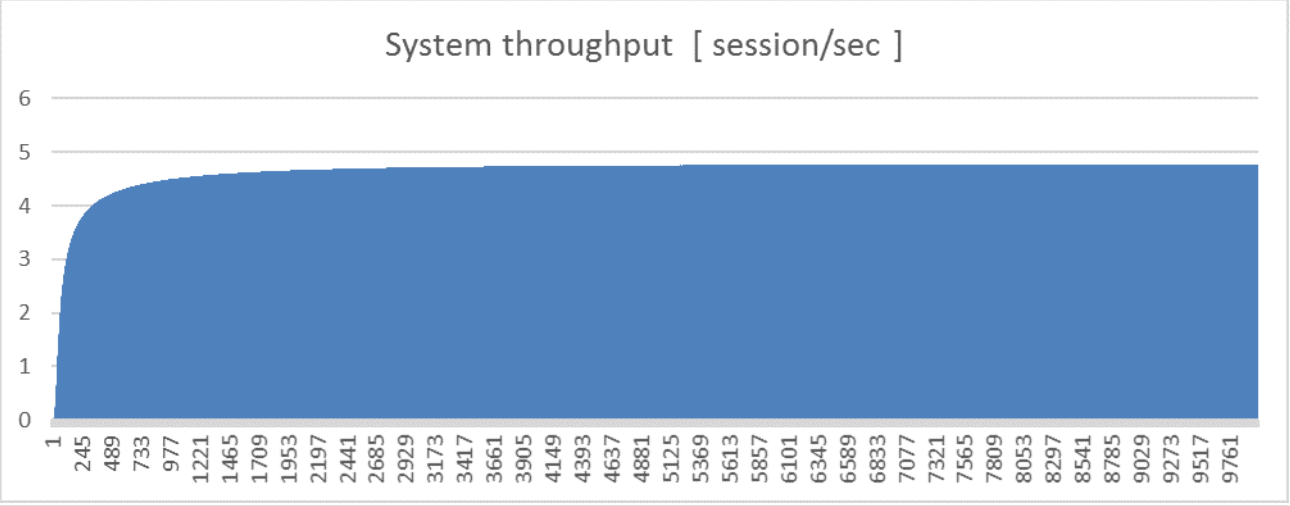
Con l'introduzione del meccanismo di blocco tutte le nuove richieste di sessioni sono bloccate e tutte le richieste provenienti da sessioni già attive vengono abortite. Nella simulazione abbiamo tenuto traccia del drop ratio e dell'abort ratio, il primo è la percentuale delle nuove richieste di connessione rigettate rispetto la totalità degli arrivi, mentre il secondo è la percentuale delle richieste rifiutate rispetto al totale delle richieste delle sessioni attive.

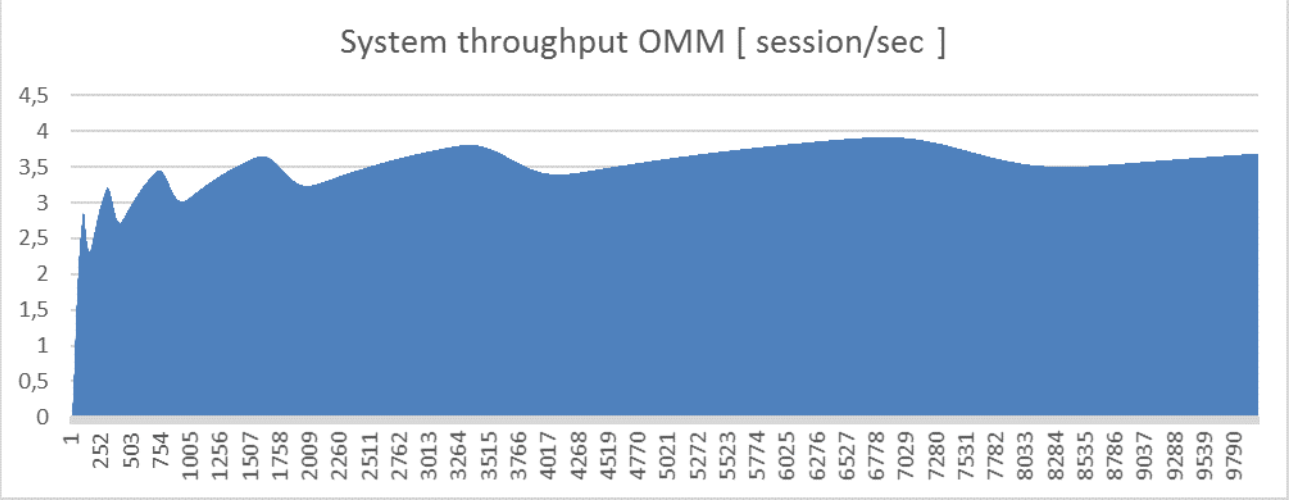


2.7.4 System

Per quanto riguarda il comportamento dell'intero sistema abbiamo registrato i dati relativi al throughput e al tempo di risposta totale. Il tempo di risposta segue lo stesso andamento del front end in quanto è influenzato sia dal front end che dal back end e la componente del secondo è trascurabile rispetto quella del primo.


System response time [sec]
140
120
100
80
60
40
20
0
1
252
503
754
1005
1256
1507
1758
2009
2260
2511
2762
3013
3264
3515
3766
4017
4268
4519
4770
5021
5272
5523
5774
6025
6276
6527
6778
7029
7280
7531
7782
8033
8284
8535
8786
9037
9288
9539
9790


System throughput [session/sec]
6
5
4
3
2
1
0
1
245
489
733
977
1221
1465
1709
1953
2197
2441
2685
2929
3173
3417
3661
3905
4149
4393
4637
4881
5125
5369
5613
5857
6101
6345
6589
6833
7077
7321
7565
7809
8053
8297
8541
8785
9029
9273
9517
9761


System throughput OMM [session/sec]
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
1
252
503
754
1005
1256
1507
1758
2009
2260
2511
2762
3013
3264
3515
3766
4017
4268
4519
4770
5021
5272
5523
5774
6025
6276
6527
6778
7029
7280
7531
7782
8033
8284
8535
8786
9037
9288
9539
9790

2.8 Conclusioni

L'introduzione di un meccanismo di blocco causa dei vantaggi in termini di response time ma conseguentemente comporta un'alta percentuale di connessioni rifiutate (drop ratio 75% circa) ed una percentuale notevole di richieste rigettate (abort ratio 8% circa). C'è da dire però che questi dati derivano dal calcolo comulativo dell'utilizzazione, e che quindi con un calcolo intervallare non è da escludere il raggiungimento di prestazioni migliori in termini di stazionarietà del sistema; ma bisogna fare attenzione alla larghezza degli intervalli:

- con un intervallo troppo stretto c'è rischio di stare troppo frequentemente con il meccanismo attivo e quindi di scartare tantissime connessioni ma si avrebbe una diminuzione dei tempi di attesa,
- con intervalli troppo larghi si tende alla situazione attuale e quindi ad attese infinite ma con percentuale di richieste rigettate più bassa.

Qui di seguito sono mostrate delle tabelle riassuntive di tutte le simulazioni riguardante media, varianza e larghezza dell'intervallo di confidenza relative alle metriche di principale interesse.

Tabella 2.1: **TABELLA RIASSUNTIVA DELLE METRICHE DEL SISTEMA CON OMM ALL'ISTANTE 10000 sec**

SYSTEM					
RESPONSE TIME			**USEFUL THROUGHPUT**		
MEAN	VAR	CONF.INTERVAL	MEAN	VAR	CONF.INTERVAL
122,1798	1,445578	0,118184	3,69119	0,00165	0,003993
ABORT RATIO			**DROP RATIO**		
MEAN	VAR	CONF.INTERVAL	MEAN	VAR	CONF.INTERVAL
0,085226	0,000013	0,00035	0,729227	0,00073	0,002655

FRONT END					
RESPONSE TIME			**THROUGHPUT**		
MEAN	VAR	CONF.INTERVAL	MEAN	VAR	CONF.INTERVAL
122,1782	1,445576	0,118184	174,5794	2,70406	0,161639

BACK END					
RESPONSE TIME			**THROUGHPUT**		
MEAN	VAR	CONF.INTERVAL	MEAN	VAR	CONF.INTERVAL
0,001571	0	0	174,5793	2,70406601	0,161639
DELAY					
MEAN		VAR		CONF.INTERVAL	
0,000401		0		0	

Tabella 2.2: **TABELLA RIASSUNTIVA DELLE METRICHE DEL SISTEMA SENZA OMM ALL'ISTANTE 10000 sec**

SYSTEM					
RESPONSE TIME			**USEFUL THROUGHPUT**		
MEAN	VAR	CONF.INTERVAL	MEAN	VAR	CONF.INTERVAL
601,1137	2,512203	0,155799	4,777278	0,000452	0,002089

FRONT END					
RESPONSE TIME			**THROUGHPUT**		
MEAN	VAR	CONF.INTERVAL	MEAN	VAR	CONF.INTERVAL
601,1122	2,512205	0,155799	218,9553	0,261108	0,0155883

BACK END					
RESPONSE TIME			**THROUGHPUT**		
MEAN	VAR	CONF.INTERVAL	MEAN	VAR	CONF.INTERVAL
0,001574	0	0	218,9953	0,026109011	0,015883
DELAY					
MEAN		VAR		CONF.INTERVAL	
0,000404		0		0	

Capitolo 3

Codice

3.1 rngs.h

```
/* ------------------------------------------------------------------------
 * Name             : rngs.h  (header file for the library file rngs.c)
 * Author           : Steve Park & Dave Geyer
 * Language         : ANSI C
 * Latest Revision  : 09-22-98
 * ------------------------------------------------------------------------
 */

#if !defined( _RNGS_ )
#define _RNGS_

double Random(void);
void   PlantSeeds(long x);
void   GetSeed(long *x);
void   PutSeed(long x);
void   SelectStream(int index);
void   TestRandom(void);

#endif
```

3.2 rvgs.h

```
/* ------------------------------------------------------------------------
 * Name             : rvgs.h (header file for the library rvgs.c)
 * Author           : Steve Park & Dave Geyer
 * Language         : ANSI C
 * Latest Revision  : 11-03-96
 * ------------------------------------------------------------------------
 */

#if !defined( _RVGS_ )
#define _RVGS_

long Bernoulli(double p);
long Binomial(long n, double p);
long Equilikely(long a, long b);
long Geometric(double p);
long Pascal(long n, double p);
long Poisson(double m);

double Uniform(double a, double b);
double Exponential(double m);
double Erlang(long n, double b);
double Normal(double m, double s);
double Lognormal(double a, double b);
double Chisquare(long n);
double Student(long n);

#endif
```

3.3 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);

```

```
51 double pdfNormal(double m, double s, double x);
52 double cdfNormal(double m, double s, double x);
53 double idfNormal(double m, double s, double u);
54
55 double pdfLognormal(double a, double b, double x);
56 double cdfLognormal(double a, double b, double x);
57 double idfLognormal(double a, double b, double u);
58
59 double pdfChisquare(long n, double x);
60 double cdfChisquare(long n, double x);
61 double idfChisquare(long n, double u);
62
63 double pdfStudent(long n, double x);
64 double cdfStudent(long n, double x);
65 double idfStudent(long n, double u);
66
67 #endif
```

3.4 structures.h

```
1  #ifndef _STRUCTURES_
2  #define _STRUCTURES_
3
4  /*-----Modella la sessione-----*/
5  struct node
6  {
7      struct node* next;
8
9      long    length; //Lunghezza (rimanente) delle richieste della sessione
10
11     double arrival_FS;  //Tempo (assoluto) di arrivo della richiesta al FS
12     double ended_FS;    //Tempo (assoluto) di uscita dal FS = Tempo di arrivo al
           BE
13     double ended;   //Tempo (assoluto) di uscita della richiesta dal BE
14
15     double sessionStart;    //Tempo (assoluto) di inizio sessione
16     double sessionEnded;    //Tempo (assoluto) di fine sessione
17
18     double endThinkTime;    //Tempo (assoluto) di think del nodo di thinkTime
19 };
20
21 struct node* create_node();
22 void destroy_node(struct node** n);
23
24 /*-----Struttura per la modellazione di liste-----*/
25 struct list
26 {
27     struct node* head;
28     struct node* tail;
29     int size; // Numero di Node in coda
30 };
31
32 struct list* create_list();
33 void destroy_list(struct list** l);
34
35 /*-----Modellazione delle list in modalità FIFO-----*/
36 void add_node_FIFO_mod(struct list* l, struct node* n);
37 struct node* fetch_node(struct list* l);
38
39 /*-----Modellazione delle list in modalità THINK (la funzione fetch node è la
       stessa)-----*/
40 void add_node_THINK_mod(struct list* l, struct node* n);
41
42 /*-----Modella server di fs o di be-----*/
43 struct server
44 {
45     struct node* internal_node; // Richiesta in lavorazione
46     struct list* fifo;
47 };
48
```

```
49 struct server* create_server();
50 struct node* update_internal_node(struct server* s);
51 void add_new_req(struct server* s, struct node* n);
52 void destroy_server(struct server** s);
53
54 /*-----Modella quantità cumulative time-dependent utili per il monitoraggio
       dell'occupazione dei nodi-----*/
55 struct area
56 {
57     double x;
58     double q;
59     double l;
60 };
61
62 struct area* create_area();
63 void update_x(struct area* a, double x);
64 void update_q(struct area* a, double q);
65 void update_l(struct area* a, double l);
66 void destroy_area(struct area** a);
67
68
69 #endif
```

3.5 time_events.h

```
1  #ifndef _TIME_EVENTS_
2  #define _TIME_EVENTS_
3
4  typedef int bool;
5  #define true 1
6  #define false 0
7
8  #define STOP_SIMULATION  10000  //Tempo di fine simulazione (oltre non entrano
       nuove sessioni e vengono servite le restanti sessioni attive)
9  #define T_SAMPLE 1  //Periodo di sampling
10
11 #define EVENTS_N 5
12
13 #define NEW_SESSION_INDEX 0
14 #define EXIT_FS_INDEX 1
15 #define EXIT_BE_INDEX 2
16 #define EXIT_TH_INDEX 3
17 #define SAMPLING_INDEX 4
18
19 /*-----Modella il clock di simulazione-----*/
20 struct clock
21 {
22     double current;
23     double next;
24 };
25
26 struct clock* create_clock(double next);
27 void destroy_clock(struct clock** c);
28
29
30 /*-----Modella una struttura che mantiene l'istante di prossima occorrenza per
       ogni tipo di evento-----*/
31 struct calendar
32 {
33     double* events_times;
34 };
35
36 struct calendar* create_calendar(double new_session_time);
37 void set_new_session_time(struct calendar* c, double new_session_time);
38 void set_exit_fs_time(struct calendar* c, double exit_fs_time);
39 void set_exit_be_time(struct calendar* c, double exit_be_time);
40 void set_exit_th_time(struct calendar* c, double exit_th_time);
41 void set_sample_time(struct calendar* c, double smp_time);
42 void destroy_calendar(struct calendar** c);
43
44 int min(struct calendar* c);
45
46 #endif
```

3.6 rngs.c

```c
/* ------------------------------------------------------------------------- */
 * This is an ANSI C library for multi-stream 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 library supplies 256 streams of random numbers; use
 * SelectStream(s) to switch between streams indexed s = 0,1,...,255.
 *
 * The streams must be initialized.  The recommended way to do this is by
 * using the function PlantSeeds(x) with the value of x used to initialize
 * the default stream and all other streams initialized automatically with
 * values dependent on the value of x.  The following convention is used
 * to initialize the default stream:
 *    if x > 0 then x is the state
 *    if x < 0 then the state is obtained from the system clock
 *    if x = 0 then the state is to be supplied interactively.
 *
 * The generator used in this library is a so-called 'Lehmer random number
 * generator' which returns a pseudo-random number uniformly distributed
 * 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
 *
 * Name             : rngs.c  (Random Number Generation - Multiple Streams)
 * Authors          : Steve Park & Dave Geyer
 * Language         : ANSI C
 * Latest Revision  : 09-22-98
 * ------------------------------------------------------------------------- */
 */

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

#define MODULUS    2147483647 /* DON'T CHANGE THIS VALUE                  */
#define MULTIPLIER 48271      /* DON'T CHANGE THIS VALUE                  */
#define CHECK      399268537  /* DON'T CHANGE THIS VALUE                  */
#define STREAMS    256        /* # of streams, DON'T CHANGE THIS VALUE    */
#define A256       22925      /* jump multiplier, DON'T CHANGE THIS VALUE */
#define DEFAULT    123456789  /* initial seed, use 0 < DEFAULT < MODULUS  */

static long seed[STREAMS] = {DEFAULT};  /* current state of each stream   */
static int  stream        = 0;          /* stream index, 0 is the default */
static int  initialized   = 0;          /* test for stream initialization */


```

```
51    double Random(void)
52  /* ------------------------------------------------------------------
53   * Random returns a pseudo-random real number uniformly distributed
54   * between 0.0 and 1.0.
55   * ------------------------------------------------------------------
56   */
57  {
58    const long Q = MODULUS / MULTIPLIER;
59    const long R = MODULUS % MULTIPLIER;
60          long t;
61
62    t = MULTIPLIER * (seed[stream] % Q) - R * (seed[stream] / Q);
63    if (t > 0)
64      seed[stream] = t;
65    else
66      seed[stream] = t + MODULUS;
67    return ((double) seed[stream] / MODULUS);
68  }
69
70
71    void PlantSeeds(long x)
72  /* ------------------------------------------------------------------
73   * Use this function to set the state of all the random number generator
74   * streams by "planting" a sequence of states (seeds), one per stream,
75   * with all states dictated by the state of the default stream.
76   * The sequence of planted states is separated one from the next by
77   * 8,367,782 calls to Random().
78   * ------------------------------------------------------------------
79   */
80  {
81    const long Q = MODULUS / A256;
82    const long R = MODULUS % A256;
83          int  j;
84          int  s;
85
86    initialized = 1;
87    s = stream;                             /* remember the current stream */
88    SelectStream(0);                        /* change to stream 0          */
89    PutSeed(x);                             /* set seed[0]                 */
90    stream = s;                             /* reset the current stream    */
91    for (j = 1; j < STREAMS; j++) {
92      x = A256 * (seed[j - 1] % Q) - R * (seed[j - 1] / Q);
93      if (x > 0)
94        seed[j] = x;
95      else
96        seed[j] = x + MODULUS;
97    }
98  }
99
100
101   void PutSeed(long x)
102 /* ------------------------------------------------------------------
```

```
103  * Use this function to set the state of the current random number
104  * generator stream according to the following conventions:
105  *     if x > 0 then x is the state (unless too large)
106  *     if x < 0 then the state is obtained from the system clock
107  *     if x = 0 then the state is to be supplied interactively
108  * ------------------------------------------------------------------
109  */
110 {
111   char ok = 0;
112
113   if (x > 0)
114     x = x % MODULUS;                          /* correct if x is too large  */
115   if (x < 0)
116     x = ((unsigned long) time((time_t *) NULL)) % MODULUS;
117   if (x == 0)
118     while (!ok) {
119       printf("\nEnter a positive integer seed (9 digits or less) >> ");
120       scanf("%ld", &x);
121       ok = (0 < x) && (x < MODULUS);
122       if (!ok)
123         printf("\nInput out of range ... try again\n");
124     }
125   seed[stream] = x;
126 }
127
128
129    void GetSeed(long *x)
130 /* ------------------------------------------------------------------
131  * Use this function to get the state of the current random number
132  * generator stream.
133  * ------------------------------------------------------------------
134  */
135 {
136   *x = seed[stream];
137 }
138
139
140    void SelectStream(int index)
141 /* ------------------------------------------------------------------
142  * Use this function to set the current random number generator
143  * stream -- that stream from which the next random number will come.
144  * ------------------------------------------------------------------
145  */
146 {
147   stream = ((unsigned int) index) % STREAMS;
148   if ((initialized == 0) && (stream != 0))    /* protect against        */
149     PlantSeeds(DEFAULT);                      /* un-initialized streams */
150 }
151
152
153    void TestRandom(void)
154 /* ------------------------------------------------------------------
```

```
155  * Use this (optional) function to test for a correct implementation.
156  * ------------------------------------------------------------------------
157  */
158 {
159   long    i;
160   long    x;
161   double  u;
162   char    ok = 0;
163
164   SelectStream(0);                    /* select the default stream */
165   PutSeed(1);                         /* and set the state to 1    */
166   for(i = 0; i < 10000; i++)
167     u = Random();
168   GetSeed(&x);                        /* get the new state value   */
169   ok = (x == CHECK);                  /* and check for correctness */
170
171   SelectStream(1);                    /* select stream 1                 */
172   PlantSeeds(1);                      /* set the state of all streams    */
173   GetSeed(&x);                        /* get the state of stream 1       */
174   ok = ok && (x == A256);             /* x should be the jump multiplier */
175   if (ok)
176     printf("\n The implementation of rngs.c is correct.\n\n");
177   else
178     printf("\n\a ERROR -- the implementation of rngs.c is not correct.\n\n");
179 }
```

3.7 rvgs.c

```
/* ------------------------------------------------------------------------
 * This is an ANSI C library for generating random variates from six discrete
 * distributions
 *
 *       Generator           Range (x)      Mean           Variance
 *
 *       Bernoulli(p)        x = 0,1        p              p*(1-p)
 *       Binomial(n, p)      x = 0,...,n    n*p            n*p*(1-p)
 *       Equilikely(a, b)    x = a,...,b    (a+b)/2        ((b-a+1)*(b-a+1)-1)/12
 *       Geometric(p)        x = 0,...      p/(1-p)        p/((1-p)*(1-p))
 *       Pascal(n, p)        x = 0,...      n*p/(1-p)      n*p/((1-p)*(1-p))
 *       Poisson(m)          x = 0,...      m              m
 *
 * and seven continuous distributions
 *
 *       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 a Lognormal(a, b) random variable, the mean and variance are
 *
 *                           mean = exp(a + 0.5*b*b)
 *                       variance = (exp(b*b) - 1) * exp(2*a + b*b)
 *
 * Name                : rvgs.c  (Random Variate GeneratorS)
 * Author              : Steve Park & Dave Geyer
 * Language            : ANSI C
 * Latest Revision     : 10-28-98
 * ------------------------------------------------------------------------
 */

#include <math.h>
#include "rngs.h"
#include "rvgs.h"


  long Bernoulli(double p)
/* ========================================================
 * Returns 1 with probability p or 0 with probability 1 - p.
 * NOTE: use 0.0 < p < 1.0
 * ========================================================
 */
{
  return ((Random() < (1.0 - p)) ? 0 : 1);
}

```

```
51    long Binomial(long n, double p)
52  /* ============================================================
53   * Returns a binomial distributed integer between 0 and n inclusive.
54   * NOTE: use n > 0 and 0.0 < p < 1.0
55   * ============================================================
56   */
57  {
58    long i, x = 0;
59
60    for (i = 0; i < n; i++)
61      x += Bernoulli(p);
62    return (x);
63  }
64
65    long Equilikely(long a, long b)
66  /* ================================================================
67   * Returns an equilikely distributed integer between a and b inclusive.
68   * NOTE: use a < b
69   * ================================================================
70   */
71  {
72    return (a + (long) ((b - a + 1) * Random()));
73  }
74
75    long Geometric(double p)
76  /* ===================================================
77   * Returns a geometric distributed non-negative integer.
78   * NOTE: use 0.0 < p < 1.0
79   * ===================================================
80   */
81  {
82    return ((long) (log(1.0 - Random()) / log(p)));
83  }
84
85    long Pascal(long n, double p)
86  /* ================================================
87   * Returns a Pascal distributed non-negative integer.
88   * NOTE: use n > 0 and 0.0 < p < 1.0
89   * ================================================
90   */
91  {
92    long i, x = 0;
93
94    for (i = 0; i < n; i++)
95      x += Geometric(p);
96    return (x);
97  }
98
99    long Poisson(double m)
100 /* ===================================================
101  * Returns a Poisson distributed non-negative integer.
102  * NOTE: use m > 0
```

```
103  * ================================================
104  */
105 {
106   double t = 0.0;
107   long   x = 0;
108
109   while (t < m) {
110     t += Exponential(1.0);
111     x++;
112   }
113   return (x - 1);
114 }
115
116   double Uniform(double a, double b)
117 /* ================================================================
118  * Returns a uniformly distributed real number between a and b.
119  * NOTE: use a < b
120  * ================================================================
121  */
122 {
123   return (a + (b - a) * Random());
124 }
125
126   double Exponential(double m)
127 /* ================================================================
128  * Returns an exponentially distributed positive real number.
129  * NOTE: use m > 0.0
130  * ================================================================
131  */
132 {
133   return (-m * log(1.0 - Random()));
134 }
135
136   double Erlang(long n, double b)
137 /* ================================================
138  * Returns an Erlang distributed positive real number.
139  * NOTE: use n > 0 and b > 0.0
140  * ================================================
141  */
142 {
143   long   i;
144   double x = 0.0;
145
146   for (i = 0; i < n; i++)
147     x += Exponential(b);
148   return (x);
149 }
150
151   double Normal(double m, double s)
152 /* ========================================================================
153  * Returns a normal (Gaussian) distributed real number.
154  * NOTE: use s > 0.0
```

```
155  *
156  * Uses a very accurate approximation of the normal idf due to Odeh & Evans,
157  * J. Applied Statistics, 1974, vol 23, pp 96-97.
158  * ============================================================================
159  */
160 {
161   const double p0 = 0.322232431088;     const double q0 = 0.099348462606;
162   const double p1 = 1.0;                const double q1 = 0.588581570495;
163   const double p2 = 0.342242088547;     const double q2 = 0.531103462366;
164   const double p3 = 0.204231210245e-1;  const double q3 = 0.103537752850;
165   const double p4 = 0.453642210148e-4;  const double q4 = 0.385607006340e-2;
166   double u, t, p, q, z;
167
168   u   = Random();
169   if (u < 0.5)
170     t = sqrt(-2.0 * log(u));
171   else
172     t = sqrt(-2.0 * log(1.0 - u));
173   p   = p0 + t * (p1 + t * (p2 + t * (p3 + t * p4)));
174   q   = q0 + t * (q1 + t * (q2 + t * (q3 + t * q4)));
175   if (u < 0.5)
176     z = (p / q) - t;
177   else
178     z = t - (p / q);
179   return (m + s * z);
180 }
181
182    double Lognormal(double a, double b)
183 /* ============================================================
184  * Returns a lognormal distributed positive real number.
185  * NOTE: use b > 0.0
186  * ============================================================
187  */
188 {
189   return (exp(a + b * Normal(0.0, 1.0)));
190 }
191
192    double Chisquare(long n)
193 /* ============================================================
194  * Returns a chi-square distributed positive real number.
195  * NOTE: use n > 0
196  * ============================================================
197  */
198 {
199   long    i;
200   double z, x = 0.0;
201
202   for (i = 0; i < n; i++) {
203     z  = Normal(0.0, 1.0);
204     x += z * z;
205   }
206   return (x);
```

```
207 }
208
209    double Student(long n)
210 /* ============================================
211  * Returns a student-t distributed real number.
212  * NOTE: use n > 0
213  * ============================================
214  */
215 {
216   return (Normal(0.0, 1.0) / sqrt(Chisquare(n) / n));
217 }
```

3.8 rvms.c

```
1
2  /* ------------------------------------------------------------------------
3   * This is an ANSI C library that can be used to evaluate the probability
4   * density functions (pdf's), cumulative distribution functions (cdf's), and
5   * inverse distribution functions (idf's) for a variety of discrete and
6   * continuous random variables.
7   *
8   * The following notational conventions are used
9   *                  x : possible value of the random variable
10  *                  u : real variable (probability) between 0.0 and 1.0
11  *   a, b, n, p, m, s : distribution-specific parameters
12  *
13  * There are pdf's, cdf's and idf's for 6 discrete random variables
14  *
15  *       Random Variable     Range (x)  Mean          Variance
16  *
17  *       Bernoulli(p)        0..1       p             p*(1-p)
18  *       Binomial(n, p)      0..n       n*p           n*p*(1-p)
19  *       Equilikely(a, b)    a..b       (a+b)/2       ((b-a+1)*(b-a+1)-1)/12
20  *       Geometric(p)        0...       p/(1-p)       p/((1-p)*(1-p))
21  *       Pascal(n, p)        0...       n*p/(1-p)     n*p/((1-p)*(1-p))
22  *       Poisson(m)          0...       m             m
23  *
24  * and for 7 continuous random variables
25  *
26  *       Uniform(a, b)       a < x < b  (a+b)/2       (b-a)*(b-a)/12
27  *       Exponential(m)      x > 0      m             m*m
28  *       Erlang(n, b)        x > 0      n*b           n*b*b
29  *       Normal(m, s)        all x      m             s*s
30  *       Lognormal(a, b)     x > 0          see below
31  *       Chisquare(n)        x > 0      n             2*n
32  *       Student(n)          all x      0  (n > 1)    n/(n-2)   (n > 2)
33  *
34  * For the Lognormal(a, b), the mean and variance are
35  *
36  *                       mean = Exp(a + 0.5*b*b)
37  *                   variance = (Exp(b*b) - 1)*Exp(2*a + b*b)
38  *
39  * Name             : rvms.c (Random Variable ModelS)
40  * Author           : Steve Park & Dave Geyer
41  * Language         : ANSI C
42  * Latest Revision  : 11-22-97
43  * ------------------------------------------------------------------------
44  */
45
46 #include <math.h>
47 #include "rvms.h"
48
49 #define TINY      1.0e-10
50 #define SQRT2PI 2.506628274631              /* sqrt(2 * pi) */
```

```
51
52 static double pdfStandard(double x);
53 static double cdfStandard(double x);
54 static double idfStandard(double u);
55 static double LogGamma(double a);
56 static double LogBeta(double a, double b);
57 static double InGamma(double a, double b);
58 static double InBeta(double a, double b, double x);
59
60
61    double pdfBernoulli(double p, long x)
62 /* ============================================
63  * NOTE: use 0.0 < p < 1.0 and 0 <= x <= 1
64  * ============================================
65  */
66 {
67    return ((x == 0) ? 1.0 - p : p);
68 }
69
70    double cdfBernoulli(double p, long x)
71 /* ============================================
72  * NOTE: use 0.0 < p < 1.0 and 0 <= x <= 1
73  * ============================================
74  */
75 {
76    return ((x == 0) ? 1.0 - p : 1.0);
77 }
78
79    long idfBernoulli(double p, double u)
80 /* ============================================
81  * NOTE: use 0.0 < p < 1.0 and 0.0 < u < 1.0
82  * ============================================
83  */
84 {
85    return ((u < 1.0 - p) ? 0 : 1);
86 }
87
88    double pdfEquilikely(long a, long b, long x)
89 /* ============================================
90  * NOTE: use a <= x <= b
91  * ============================================
92  */
93 {
94    return (1.0 / (b - a + 1.0));
95 }
96
97    double cdfEquilikely(long a, long b, long x)
98 /* ============================================
99  * NOTE: use a <= x <= b
100 * ============================================
101 */
102 {
```

```
103     return ((x - a + 1.0) / (b - a + 1.0));
104 }
105
106     long idfEquilikely(long a, long b, double u)
107 /* ================================================
108  * NOTE: use a <= b and 0.0 < u < 1.0
109  * ================================================
110  */
111 {
112     return (a + (long) (u * (b - a + 1)));
113 }
114
115     double pdfBinomial(long n, double p, long x)
116 /* ================================================
117  * NOTE: use 0 <= x <= n and 0.0 < p < 1.0
118  * ================================================
119  */
120 {
121     double s, t;
122
123     s = LogChoose(n, x);
124     t = x * log(p) + (n - x) * log(1.0 - p);
125     return (exp(s + t));
126 }
127
128     double cdfBinomial(long n, double p, long x)
129 /* ================================================
130  * NOTE: use 0 <= x <= n and 0.0 < p < 1.0
131  * ================================================
132  */
133 {
134     if (x < n)
135       return (1.0 - InBeta(x + 1, n - x, p));
136     else
137       return (1.0);
138 }
139
140     long idfBinomial(long n, double p, double u)
141 /* ================================================
142  * NOTE: use 0 <= n, 0.0 < p < 1.0 and 0.0 < u < 1.0
143  * ================================================
144  */
145 {
146     long x = (long) (n * p);              /* start searching at the mean */
147
148     if (cdfBinomial(n, p, x) <= u)
149       while (cdfBinomial(n, p, x) <= u)
150         x++;
151     else if (cdfBinomial(n, p, 0) <= u)
152       while (cdfBinomial(n, p, x - 1) > u)
153         x--;
154     else
```

```
155       x = 0;
156     return (x);
157 }
158
159     double pdfGeometric(double p, long x)
160 /* ================================== */
161  * NOTE: use 0.0 < p < 1.0 and x >= 0
162  * ================================== */
163  */
164 {
165     return ((1.0 - p) * exp(x * log(p)));
166 }
167
168     double cdfGeometric(double p, long x)
169 /* ================================== */
170  * NOTE: use 0.0 < p < 1.0 and x >= 0
171  * ================================== */
172  */
173 {
174     return (1.0 - exp((x + 1) * log(p)));
175 }
176
177     long idfGeometric(double p, double u)
178 /* ========================================== */
179  * NOTE: use 0.0 < p < 1.0 and 0.0 < u < 1.0
180  * ========================================== */
181  */
182 {
183     return ((long) (log(1.0 - u) / log(p)));
184 }
185
186     double pdfPascal(long n, double p, long x)
187 /* ========================================== */
188  * NOTE: use n >= 1, 0.0 < p < 1.0, and x >= 0
189  * ========================================== */
190  */
191 {
192     double  s, t;
193
194     s = LogChoose(n + x - 1, x);
195     t = x * log(p) + n * log(1.0 - p);
196     return (exp(s + t));
197 }
198
199     double cdfPascal(long n, double p, long x)
200 /* ========================================== */
201  * NOTE: use n >= 1, 0.0 < p < 1.0, and x >= 0
202  * ========================================== */
203  */
204 {
205     return (1.0 - InBeta(x + 1, n, p));
206 }
```

```
207
208     long idfPascal(long n, double p, double u)
209 /* ================================================
210  * NOTE: use n >= 1, 0.0 < p < 1.0, and 0.0 < u < 1.0
211  * ================================================
212  */
213 {
214     long x = (long) (n * p / (1.0 - p));     /* start searching at the mean */
215
216     if (cdfPascal(n, p, x) <= u)
217       while (cdfPascal(n, p, x) <= u)
218         x++;
219     else if (cdfPascal(n, p, 0) <= u)
220       while (cdfPascal(n, p, x - 1) > u)
221         x--;
222     else
223       x = 0;
224     return (x);
225 }
226
227     double pdfPoisson(double m, long x)
228 /* ================================
229  * NOTE: use m > 0 and x >= 0
230  * ================================
231  */
232 {
233     double t;
234
235     t = - m + x * log(m) - LogFactorial(x);
236     return (exp(t));
237 }
238
239     double cdfPoisson(double m, long x)
240 /* ================================
241  * NOTE: use m > 0 and x >= 0
242  * ================================
243  */
244 {
245     return (1.0 - InGamma(x + 1, m));
246 }
247
248     long idfPoisson(double m, double u)
249 /* ================================
250  * NOTE: use m > 0 and 0.0 < u < 1.0
251  * ================================
252  */
253 {
254     long x = (long) m;                       /* start searching at the mean */
255
256     if (cdfPoisson(m, x) <= u)
257       while (cdfPoisson(m, x) <= u)
258         x++;
```

```
259     else if (cdfPoisson(m, 0) <= u)
260       while (cdfPoisson(m, x - 1) > u)
261         x--;
262     else
263       x = 0;
264     return (x);
265 }
266
267     double pdfUniform(double a, double b, double x)
268 /* ============================================
269  * NOTE: use a < x < b
270  * ============================================
271  */
272 {
273     return (1.0 / (b - a));
274 }
275
276     double cdfUniform(double a, double b, double x)
277 /* ============================================
278  * NOTE: use a < x < b
279  * ============================================
280  */
281 {
282     return ((x - a) / (b - a));
283 }
284
285     double idfUniform(double a, double b, double u)
286 /* ============================================
287  * NOTE: use a < b and 0.0 < u < 1.0
288  * ============================================
289  */
290 {
291     return (a + (b - a) * u);
292 }
293
294     double pdfExponential(double m, double x)
295 /* ============================================
296  * NOTE: use m > 0 and x > 0
297  * ============================================
298  */
299 {
300     return ((1.0 / m) * exp(- x / m));
301 }
302
303     double cdfExponential(double m, double x)
304 /* ============================================
305  * NOTE: use m > 0 and x > 0
306  * ============================================
307  */
308 {
309     return (1.0 - exp(- x / m));
310 }
```

```
311
312     double idfExponential(double m, double u)
313 /* ============================================
314  * NOTE: use m > 0 and 0.0 < u < 1.0
315  * ============================================
316  */
317 {
318     return (- m * log(1.0 - u));
319 }
320
321     double pdfErlang(long n, double b, double x)
322 /* ============================================
323  * NOTE: use n >= 1, b > 0, and x > 0
324  * ============================================
325  */
326 {
327     double t;
328
329     t = (n - 1) * log(x / b) - (x / b) - log(b) - LogGamma(n);
330     return (exp(t));
331 }
332
333     double cdfErlang(long n, double b, double x)
334 /* ============================================
335  * NOTE: use n >= 1, b > 0, and x > 0
336  * ============================================
337  */
338 {
339     return (InGamma(n, x / b));
340 }
341
342     double idfErlang(long n, double b, double u)
343 /* ============================================
344  * NOTE: use n >= 1, b > 0 and 0.0 < u < 1.0
345  * ============================================
346  */
347 {
348     double t, x = n * b;                    /* initialize to the mean, then */
349
350     do {                                    /* use Newton-Raphson iteration */
351       t = x;
352       x = t + (u - cdfErlang(n, b, t)) / pdfErlang(n, b, t);
353       if (x <= 0.0)
354         x = 0.5 * t;
355     } while (fabs(x - t) >= TINY);
356     return (x);
357 }
358
359     static double pdfStandard(double x)
360 /* ============================================
361  * NOTE: x can be any value
362  * ============================================
```

```
363  */
364 {
365     return (exp(- 0.5 * x * x) / SQRT2PI);
366 }
367
368     static double cdfStandard(double x)
369 /* ==========================================
370  * NOTE: x can be any value
371  * ==========================================
372  */
373 {
374     double t;
375
376     t = InGamma(0.5, 0.5 * x * x);
377     if (x < 0.0)
378       return (0.5 * (1.0 - t));
379     else
380       return (0.5 * (1.0 + t));
381 }
382
383     static double idfStandard(double u)
384 /* ==========================================
385  * NOTE: 0.0 < u < 1.0
386  * ==========================================
387  */
388 {
389     double t, x = 0.0;                    /* initialize to the mean, then  */
390
391     do {                                  /* use Newton-Raphson iteration  */
392       t = x;
393       x = t + (u - cdfStandard(t)) / pdfStandard(t);
394     } while (fabs(x - t) >= TINY);
395     return (x);
396 }
397
398     double pdfNormal(double m, double s, double x)
399 /* ==================================================
400  * NOTE: x and m can be any value, but s > 0.0
401  * ==================================================
402  */
403 {
404     double t = (x - m) / s;
405
406     return (pdfStandard(t) / s);
407 }
408
409     double cdfNormal(double m, double s, double x)
410 /* ==================================================
411  * NOTE: x and m can be any value, but s > 0.0
412  * ==================================================
413  */
414 {
```

```
415     double t = (x - m) / s;
416
417     return (cdfStandard(t));
418 }
419
420     double idfNormal(double m, double s, double u)
421 /* ================================================
422  * NOTE: m can be any value, but s > 0.0 and 0.0 < u < 1.0
423  * ================================================
424  */
425 {
426     return (m + s * idfStandard(u));
427 }
428
429     double pdfLognormal(double a, double b, double x)
430 /* ================================================
431  * NOTE: a can have any value, but b > 0.0 and x > 0.0
432  * ================================================
433  */
434 {
435     double t = (log(x) - a) / b;
436
437     return (pdfStandard(t) / (b * x));
438 }
439
440     double cdfLognormal(double a, double b, double x)
441 /* ================================================
442  * NOTE: a can have any value, but b > 0.0 and x > 0.0
443  * ================================================
444  */
445 {
446     double t = (log(x) - a) / b;
447
448     return (cdfStandard(t));
449 }
450
451     double idfLognormal(double a, double b, double u)
452 /* ================================================
453  * NOTE: a can have any value, but b > 0.0 and 0.0 < u < 1.0
454  * ================================================
455  */
456 {
457     double t;
458
459     t = a + b * idfStandard(u);
460     return (exp(t));
461 }
462
463     double pdfChisquare(long n, double x)
464 /* ================================
465  * NOTE: use n >= 1 and x > 0.0
466  * ================================
```

```
467  */
468 {
469     double t, s = n / 2.0;
470 
471     t = (s - 1.0) * log(x / 2.0) - (x / 2.0) - log(2.0) - LogGamma(s);
472     return (exp(t));
473 }
474 
475     double cdfChisquare(long n, double x)
476 /* ====================================
477  * NOTE: use n >= 1 and x > 0.0
478  * ====================================
479  */
480 {
481     return (InGamma(n / 2.0, x / 2));
482 }
483 
484     double idfChisquare(long n, double u)
485 /* ====================================
486  * NOTE: use n >= 1 and 0.0 < u < 1.0
487  * ====================================
488  */
489 {
490     double t, x = n;                          /* initialize to the mean, then */
491 
492     do {                                     /* use Newton-Raphson iteration */
493       t = x;
494       x = t + (u - cdfChisquare(n, t)) / pdfChisquare(n, t);
495       if (x <= 0.0)
496         x = 0.5 * t;
497     } while (fabs(x - t) >= TINY);
498     return (x);
499 }
500 
501     double pdfStudent(long n, double x)
502 /* ====================================
503  * NOTE: use n >= 1 and x > 0.0
504  * ====================================
505  */
506 {
507     double s, t;
508 
509     s = -0.5 * (n + 1) * log(1.0 + ((x * x) / (double) n));
510     t = -LogBeta(0.5, n / 2.0);
511     return (exp(s + t) / sqrt((double) n));
512 }
513 
514     double cdfStudent(long n, double x)
515 /* ====================================
516  * NOTE: use n >= 1 and x > 0.0
517  * ====================================
518  */
```

```
519 {
520     double s, t;
521
522     t = (x * x) / (n + x * x);
523     s = InBeta(0.5, n / 2.0, t);
524     if (x >= 0.0)
525       return (0.5 * (1.0 + s));
526     else
527       return (0.5 * (1.0 - s));
528 }
529
530     double idfStudent(long n, double u)
531 /* ======================================
532  * NOTE: use n >= 1 and 0.0 < u < 1.0
533  * ======================================
534  */
535 {
536     double t, x = 0.0;                          /* initialize to the mean, then */
537
538     do {                                        /* use Newton-Raphson iteration */
539       t = x;
540       x = t + (u - cdfStudent(n, t)) / pdfStudent(n, t);
541     } while (fabs(x - t) >= TINY);
542     return (x);
543 }
544
545 /* ====================================================================
546  * The six functions that follow are a 'special function' mini-library
547  * used to support the evaluation of pdf, cdf and idf functions.
548  * ====================================================================
549  */
550
551     static double LogGamma(double a)
552 /* ====================================================================
553  * LogGamma returns the natural log of the gamma function.
554  * NOTE: use a > 0.0
555  *
556  * The algorithm used to evaluate the natural log of the gamma function is
557  * based on an approximation by C. Lanczos, SIAM J. Numerical Analysis, B,
558  * vol 1, 1964.  The constants have been selected to yield a relative error
559  * which is less than 2.0e-10 for all positive values of the parameter a.
560  * ====================================================================
561  */
562 {
563     double s[6], sum, temp;
564     int     i;
565
566     s[0] =  76.180091729406 / a;
567     s[1] = -86.505320327112 / (a + 1.0);
568     s[2] =  24.014098222230 / (a + 2.0);
569     s[3] =  -1.231739516140 / (a + 3.0);
570     s[4] =   0.001208580030 / (a + 4.0);
```

```
571     s[5] =   -0.000005363820 / (a + 5.0);
572     sum  =    1.000000000178;
573     for (i = 0; i < 6; i++)
574       sum += s[i];
575     temp = (a - 0.5) * log(a + 4.5) - (a + 4.5) + log(SQRT2PI * sum);
576     return (temp);
577 }
578
579     double LogFactorial(long n)
580 /* ================================================================
581  * LogFactorial(n) returns the natural log of n!
582  * NOTE: use n >= 0
583  *
584  * The algorithm used to evaluate the natural log of n! is based on a
585  * simple equation which relates the gamma and factorial functions.
586  * ================================================================
587  */
588 {
589     return (LogGamma(n + 1));
590 }
591
592     static double LogBeta(double a, double b)
593 /* ================================================================
594  * LogBeta returns the natural log of the beta function.
595  * NOTE: use a > 0.0 and b > 0.0
596  *
597  * The algorithm used to evaluate the natural log of the beta function is
598  * based on a simple equation which relates the gamma and beta functions.
599  *
600  */
601 {
602     return (LogGamma(a) + LogGamma(b) - LogGamma(a + b));
603 }
604
605     double LogChoose(long n, long m)
606 /* ================================================================
607  * LogChoose returns the natural log of the binomial coefficient C(n,m).
608  * NOTE: use 0 <= m <= n
609  *
610  * The algorithm used to evaluate the natural log of a binomial coefficient
611  * is based on a simple equation which relates the beta function to a
612  * binomial coefficient.
613  * ================================================================
614  */
615 {
616     if (m > 0)
617       return (-LogBeta(m, n - m + 1) - log(m));
618     else
619       return (0.0);
620 }
621
622     static double InGamma(double a, double x)
```

```
623 /* ================================================================
624  * Evaluates the incomplete gamma function.
625  * NOTE: use a > 0.0 and x >= 0.0
626  *
627  * The algorithm used to evaluate the incomplete gamma function is based on
628  * Algorithm AS 32, J. Applied Statistics, 1970, by G. P. Bhattacharjee.
629  * See also equations 6.5.29 and 6.5.31 in the Handbook of Mathematical
630  * Functions, Abramowitz and Stegum (editors).  The absolute error is less
631  * than 1e-10 for all non-negative values of x.
632  * ================================================================
633  */
634 {
635    double t, sum, term, factor, f, g, c[2], p[3], q[3];
636    long   n;
637
638    if (x > 0.0)
639      factor = exp(-x + a * log(x) - LogGamma(a));
640    else
641      factor = 0.0;
642    if (x < a + 1.0) {                  /* evaluate as an infinite series - */
643      t    = a;                         /* A & S equation 6.5.29           */
644      term = 1.0 / a;
645      sum  = term;
646      while (term >= TINY * sum) {      /* sum until 'term' is small */
647        t++;
648        term *= x / t;
649        sum  += term;
650      }
651      return (factor * sum);
652    }
653    else {                              /* evaluate as a continued fraction - */
654      p[0] = 0.0;                       /* A & S eqn 6.5.31 with the extended */
655      q[0] = 1.0;                       /* pattern 2-a, 2, 3-a, 3, 4-a, 4,... */
656      p[1] = 1.0;                       /* - see also A & S sec 3.10, eqn (3) */
657      q[1] = x;
658      f    = p[1] / q[1];
659      n    = 0;
660      do {                              /* recursively generate the continued */
661        g = f;                          /* fraction 'f' until two consecutive */
662        n++;                            /* values are small                   */
663        if ((n % 2) > 0) {
664          c[0] = ((double) (n + 1) / 2) - a;
665          c[1] = 1.0;
666        }
667        else {
668          c[0] = (double) n / 2;
669          c[1] = x;
670        }
671        p[2] = c[1] * p[1] + c[0] * p[0];
672        q[2] = c[1] * q[1] + c[0] * q[0];
673        if (q[2] != 0.0) {              /* rescale to avoid overflow */
674          p[0] = p[1] / q[2];
```

```
675           q[0] = q[1] / q[2];
676           p[1] = p[2] / q[2];
677           q[1] = 1.0;
678           f    = p[1];
679         }
680       } while ((fabs(f - g) >= TINY) || (q[1] != 1.0));
681       return (1.0 - factor * f);
682     }
683 }
684
685     static double InBeta(double a, double b, double x)
686 /* ==========================================================
687  * Evaluates the incomplete beta function.
688  * NOTE: use a > 0.0, b > 0.0 and 0.0 <= x <= 1.0
689  *
690  * The algorithm used to evaluate the incomplete beta function is based on
691  * equation 26.5.8 in the Handbook of Mathematical Functions, Abramowitz
692  * and Stegum (editors).  The absolute error is less than 1e-10 for all x
693  * between 0 and 1.
694  * ==========================================================
695  */
696 {
697     double t, factor, f, g, c, p[3], q[3];
698     int    swap;
699     long   n;
700
701     if (x > (a + 1.0) / (a + b + 1.0)) { /* to accelerate convergence   */
702       swap = 1;                          /* complement x and swap a & b */
703       x    = 1.0 - x;
704       t    = a;
705       a    = b;
706       b    = t;
707     }
708     else                                 /* do nothing */
709       swap = 0;
710     if (x > 0)
711       factor = exp(a * log(x) + b * log(1.0 - x) - LogBeta(a,b)) / a;
712     else
713       factor = 0.0;
714     p[0] = 0.0;
715     q[0] = 1.0;
716     p[1] = 1.0;
717     q[1] = 1.0;
718     f    = p[1] / q[1];
719     n    = 0;
720     do {                                /* recursively generate the continued */
721       g = f;                            /* fraction 'f' until two consecutive */
722       n++;                              /* values are small                   */
723       if ((n % 2) > 0) {
724         t = (double) (n - 1) / 2;
725         c = -(a + t) * (a + b + t) * x / ((a + n - 1.0) * (a + n));
726       }
```

```
727       else {
728         t = (double) n / 2;
729         c = t * (b - t) * x / ((a + n - 1.0) * (a + n));
730       }
731       p[2] = p[1] + c * p[0];
732       q[2] = q[1] + c * q[0];
733       if (q[2] != 0.0) {                  /* rescale to avoid overflow */
734         p[0] = p[1] / q[2];
735         q[0] = q[1] / q[2];
736         p[1] = p[2] / q[2];
737         q[1] = 1.0;
738         f    = p[1];
739       }
740     } while ((fabs(f - g) >= TINY) || (q[1] != 1.0));
741     if (swap)
742       return (1.0 - factor * f);
743     else
744       return (factor * f);
745 }
```

3.9 time_events.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "time_events.h"

/*-----CLOCK-----*/

struct clock* create_clock(double next)
{
    /*Crea un'istanza di clock*/

    struct clock* c = (struct clock*)malloc(sizeof(struct clock));

    if(c == NULL)
    {
        perror("Error in malloc of create_clock()!\n");
        exit(1);
    }

    c->current = 0; // Tempo corrente inizio simulazione
    c->next = next; // Tempo primo evento prossimo

    return c;
}

void destroy_clock(struct clock** c)
{
    /*Rimuove dallo heap un clock*/

    if(c != NULL)
    {
        free(*c);
        *c = NULL;
    }
}

/*-----CALENDAR-----*/

struct calendar* create_calendar(double new_session_time)
{
    /*Crea un'istanza di calendar*/

    struct calendar* c = (struct calendar*)malloc(sizeof(struct calendar));

    if(c == NULL)
    {
        perror("Error in malloc of create_calendar()!\n");
        exit(1);
    }

```

```
50      c->events_times = (double*)malloc(EVENTS_N*sizeof(double)); // Array dei
            tempi per ciascun tipo di evento
51
52      c->events_times[NEW_SESSION_INDEX] = new_session_time;    // new_sessione
53      c->events_times[EXIT_FS_INDEX] = 100*STOP_SIMULATION; // exit_fs
54      c->events_times[EXIT_BE_INDEX] = 100*STOP_SIMULATION; // exit_be
55      c->events_times[EXIT_TH_INDEX] = 100*STOP_SIMULATION; // exit_th
56      c->events_times[SAMPLING_INDEX] = T_SAMPLE; // sample
57
58      return c;
59  }
60
61  void set_new_session_time(struct calendar* c, double new_session_time)
62  {
63      /*Setta l'istante dell'arrivo della prossima nuova sessione (tempo assoluto)*/
64
65      if(c == NULL)
66      {
67          perror("Null pointers in set_new_session_time() parameters!\n");
68          exit(1);
69      }
70
71      c->events_times[0] = new_session_time;
72
73  }
74
75  void set_exit_fs_time(struct calendar* c, double exit_fs_time)
76  {
77      /*Setta l'istante prossimo di uscita di una richiesta dal fs (tempo
            assoluto)*/
78
79      if(c == NULL)
80      {
81          perror("Null pointers in set_exit_fs_time() parameters!\n");
82          exit(1);
83      }
84
85      c->events_times[1] = exit_fs_time;
86
87  }
88
89  void set_exit_be_time(struct calendar* c, double exit_be_time)
90  {
91      /*Setta l'istante prossimo di uscita di una richiesta dal be (tempo
            assoluto)*/
92
93      if(c == NULL)
94      {
95          perror("Null pointers in set_exit_be_time() parameters!\n");
96          exit(1);
97      }
98
```

```
99      c->events_times[2] = exit_be_time;
100
101 }
102
103 void set_exit_th_time(struct calendar* c, double exit_th_time)
104 {
105     /*Setta l'istante prossimo di uscita di una richiesta dal nodo di think
            (tempo assoluto)*/
106
107     if(c == NULL)
108     {
109         perror("Null pointers in set_exit_th_time() parameters!\n");
110         exit(1);
111     }
112
113     c->events_times[3] = exit_th_time;
114
115 }
116
117 void set_sample_time(struct calendar* c, double smp_time)
118 {
119     /*Setta l'istante prossimo di sampling (tempo assoluto)*/
120
121     if(c == NULL)
122     {
123         perror("Null pointers in set_exit_th_time() parameters!\n");
124         exit(1);
125     }
126
127     c->events_times[4] = smp_time;
128
129 }
130
131 void destroy_calendar(struct calendar** c)
132 {
133     /*Rimuove dallo heap il calendar*/
134
135     if(*c != NULL)
136     {
137         free((*c)->events_times);
138
139         free(*c);
140         *c = NULL;
141     }
142 }
143
144 int min(struct calendar* c)
145 {
146     /* Restituisce l'indice dell'elemento minimo di events_times */
147
148     int pos = 0;
149     int j;
```

```
150
151     for(j = pos+1 ; j < EVENTS_N ; j++)
152     {
153         if(c->events_times[pos] > c->events_times[j])
154             pos = j;
155     }
156
157     return pos;
158 }
159
160 /*
161 int main(void)
162 {
163     struct calendar* c = create_calendar(3);
164     set_exit_fs_time(c, 2.3);
165     set_exit_be_time(c, 1.3);
166     set_exit_th_time(c, 0.4);
167
168     printf("%f\n",c->events_times[min(c)]);
169
170     return EXIT_SUCCESS;
171 }
172 */
```

3.10 structures.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "structures.h"

struct list* fifo;

/*-----NODI-----*/

struct node* create_node()
{
    /*Creazione di Node*/

    struct node* n = (struct node*)malloc(sizeof(struct node));

    if(n==NULL) {
            perror("Error in malloc of create_node()\n");
            exit(1);
    }

    n->next = NULL;
    n->length = 0;
    n->arrival_FS = 0;
    n->ended_FS = 0;
    n->ended = 0;
    n->sessionStart = 0;
    n->sessionEnded = 0;

    n->endThinkTime = 0;

    return n;
}

void destroy_node(struct node** n)
{
    /*Rimuove dallo heap un nodo*/

    if(*n != NULL)
    {
        free(*n);
        *n = NULL;
    }
}

/*-----LISTE-----*/

struct list* create_list()
{
    /*Creazione di lista vuota*/

    struct list* l = (struct list*)malloc(sizeof(struct list));
```

```
51
52      if(l==NULL) {
53              perror("Error in malloc of create_list()\n");
54              exit(1);
55      }
56
57      l->head = NULL;
58      l->tail = NULL;
59      l->size = 0;
60
61      return l;
62  }
63
64  void add_node_FIFO_mod(struct list* l, struct node* n)
65  {
66      /*Mette in coda il nodo n alla coda FIFO l*/
67
68      if(n == NULL || l == NULL)
69      {
70          perror("Null pointers in add_node_FIFO_mod() parameters!\n");
71          exit(1);
72      }
73
74      else if(l->tail == NULL) // FIFO vuota
75      {
76          l->tail = n;
77          l->head = n;
78          l->size += 1;
79      }
80
81      else // Il nodo si mette in coda
82      {
83          (l->tail)->next = n;
84          l->tail = n;
85          l->size += 1;
86      }
87  }
88
89  void add_node_THINK_mod(struct list* l, struct node* n)
90  {
91      /*Aggiuge un nodo alla lista nella posizione corretta a seconda del suo tempo
            di Think*/
92
93      if(n == NULL || l == NULL)
94      {
95          perror("Null pointers in add_node_THINK_mod() parameters!\n");
96          exit(1);
97      }
98
99      else if(l->tail == NULL) // lista vuota
100     {
101         l->tail = n;
```

```
102         l->head = n;
103         l->size += 1;
104     }
105 
106     else // Il nodo si mette in lista nella posizione che gli compete (lista non
            vuota)
107     {
108         struct node* current = l->head; // Parto dalla testa
109         struct node* previous = NULL;
110 
111         while(current != NULL && current->endThinkTime < n->endThinkTime)
112         {
113             previous = current;
114             current = current->next;
115         }
116 
117         if(current == NULL) // Inserisco in coda
118         {
119             (l->tail)->next = n;
120             l->tail = n;
121             l->size += 1;
122         }
123 
124         else if(current == l->head) // Inserimento in testa
125         {
126             n->next = l->head;
127             l->head = n;
128             l->size += 1;
129         }
130 
131         else // Inserimento interno alla lista
132         {
133             previous->next = n;
134             n->next = current;
135             l->size += 1;
136         }
137     }
138 }
139 
140 struct node* fetch_node(struct list* l)
141 {
142     /*Preleva il nodo in testa alla lista l*/
143 
144     struct node* n = NULL;
145 
146     if(l == NULL)
147     {
148         perror("Null pointers in fetch_node() parameters!\n");
149         exit(1);
150     }
151 
152     if(l->head == NULL)
```

```
153     {
154         //printf("%s\n","No Node to fetch!");
155     }
156 
157     else if(l->head == l->tail) // Un solo nodo in lista
158     {
159         n = l->head;
160         l->head = NULL;
161         l->tail = NULL;
162         l->size -= 1;
163     }
164 
165     else // Più di un nodo in lista
166     {
167         n = l->head;
168         l->head = n->next;
169         n->next = NULL;
170         l->size -= 1;
171     }
172 
173     return n;
174 }
175 
176 void destroy_list(struct list** l)
177 {
178     /*Rimuove dallo heap una lista e tutti i nodi ad essa associati*/
179 
180     if(*l != NULL)
181     {
182         struct node* n = NULL;
183 
184         while((*l)->size > 0)
185         {
186             n = fetch_node(*l);
187             destroy_node(&n);
188         }
189 
190         free(*l);
191         *l = NULL;
192     }
193 }
194 
195 /*-----SERVER-----*/
196 
197 struct server* create_server()
198 {
199     /*Creazione di Server*/
200 
201     struct server* s = (struct server*)malloc(sizeof(struct server));
202 
203     if(s==NULL) {
204             perror("Error in malloc of create_server()\n");
```

```
205             exit(1);
206     }
207
208     s->fifo = create_list(); // FIFO vuota inizialmente
209     s->internal_node = NULL;
210
211     return s;
212 }
213
214 struct node* update_internal_node(struct server* s)
215 {
216     /*Estrae internal_node e (se la fifo non è vuota) fa fetch sulla fifo,
            aggiorna internal_node*/
217
218     if(s == NULL)
219     {
220         perror("Null pointers in update_internal_node() parameters!\n");
221         exit(1);
222     }
223
224     struct node* old = NULL;
225
226     if(s->internal_node == NULL)
227     {
228         printf("%s\n","No internal node to update");
229     }
230
231     else // Se vi è qualche richiesta in lavorazione
232     {
233         old = s->internal_node;
234         s->internal_node = fetch_node(s->fifo);
235     }
236
237     return old;
238 }
239
240 void add_new_req(struct server* s, struct node* n)
241 {
242         /*Inserisce nel server il nodo n in coda o nel servente (se esso è
                vuoto)*/
243
244         if(s == NULL || n == NULL)
245         {
246             perror("Null pointers in add_new_req() parameters!\n");
247             exit(1);
248         }
249
250         if(s->internal_node == NULL) // Il nodo n entra direttamente nel servente
                (fifo vuota)
251             s->internal_node = n;
252
253         else    // Il nodo entra in fifo
```

```
254             add_node_FIFO_mod(s->fifo , n);
255
256 }
257
258 void destroy_server(struct server** s)
259 {
260     /*Rimuove dallo heap un server*/
261
262     if(*s != NULL)
263     {
264         destroy_list(&((*s)->fifo));
265
266         destroy_node(&((*s)->internal_node));
267
268         free(*s);
269         *s = NULL;
270     }
271 }
272
273 /*-----AREA-----*/
274
275 struct area* create_area()
276 {
277     /*Creazione di Area*/
278
279     struct area* a = (struct area*)malloc(sizeof(struct area));
280
281     if(a==NULL) {
282             perror("Error in malloc of create_area()\n");
283             exit(1);
284     }
285
286     a->x = 0;
287     a->q = 0;
288     a->l = 0;
289
290     return a;
291 }
292
293 void update_x(struct area* a, double x)
294 {
295     /*Aggiorna x*/
296
297     if(a == NULL)
298     {
299         perror("Null pointers in update_x() parameters!\n");
300         exit(1);
301     }
302
303     a->x += x;
304 }
305
```

```
306 void update_q(struct area* a, double q)
307 {
308     /*Aggiorna q*/
309 
310     if(a == NULL)
311     {
312         perror("Null pointers in update_q() parameters!\n");
313         exit(1);
314     }
315 
316     a->q += q;
317 }
318 
319 void update_l(struct area* a, double l)
320 {
321     /*Aggiorna l*/
322 
323     if(a == NULL)
324     {
325         perror("Null pointers in update_l() parameters!\n");
326         exit(1);
327     }
328 
329     a->l += l;
330 }
331 
332 void destroy_area(struct area** a)
333 {
334     /*Rimuove la struttura area dallo heap*/
335 
336     if(*a != NULL)
337     {
338         free(*a);
339         *a = NULL;
340     }
341 }
```

3.11 simulation.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include "rngs.h"  // Generatore di Lehmer multistream
#include "rvgs.h" // Distribuzioni utilizzate
#include "rvms.h" // Quantità pivotali
#include "structures.h"
#include "time_events.h"

#define LOWER_EQ 5
#define UPPER_EQ 35

#define INTER_ARRIVAL_T 0.02857 // Tempo medio di interarrivo nuove sessioni (1/35)
#define FS_AVG_T 0.00456     // E[D] del fs
#define BE_AVG_T 0.00117     // E[D] del be
#define TH_AVG_T 7  // Tempo medio di attesa nel nodo di think

#define ABDR_THRESHOLD 0.85 // Soglia utilizzazione fs per attivazione ab_dr
#define ABDR_FLOOR 0.75 // Soglia sotto la quale in caso di omm attivato il sistema torna a condizioni standard (ab_dr disattivato)

/** METRICA SELEZIONATA (TREND) **/

double* trend;
char choice[15]; // Metrica selezionata

/** FINE METRICA SELEZIONATA (TREND) **/

/**----GESTIONE DROP E ABORT----**/

bool omm = false; // Overload Managment Mechanism (TUNABLE a inizio programma)
bool ab_dr = false; // Abort e Drop attivi se omm è true e se l'utilizzazione supera l'85%

/**----FINE GESTIONE DROP E ABORT----**/

/**----STATO----**/

/* Variabili di stato*/
long session_fifo_fs = 0; // Numero di sessioni in fifo a fs
int xfs = 0; // Stato di occupazione del servente del fs (0 o 1)
long session_fifo_be = 0; // Numero di sessioni in fifo a be
int xbe = 0; // Stato di occpuazione del servente del be (0 o 1)
long session_th = 0; // Numero di sessioni nel nodo di think
/*Fine variabili di stato*/

```

```
47 long total_generated_sessions = 0; // Totale delle sessioni create (anche se
       droppate)
48 long total_completed_sessions = 0; // Totale sessioni completate
49 long total_dropped_sessions = 0; // Totale sessioni droppate
50
51 long total_processed_requests = 0; // Totale delle richieste transitate per il be
       + richieste abortite (nodo di think)
52 long total_aborted_requests = 0; // Totale richieste abortite
53
54 long num_ended_req_fs = 0; // Richieste espletate dal fs
55 long num_ended_req_be = 0; // Richieste espletate dal be
56 long num_entered_be_serv = 0; // Numero richieste entrate nel servente del be in
       un certo istante di simulazione
57
58 /**-----RESET STATE----**/
59
60 void reset_state(void)
61 {
62     /*Resetta le varabili di stato e quelle di appoggio*/
63
64     ab_dr = false;
65     session_fifo_fs = 0;
66     xfs = 0;
67     session_fifo_be = 0;
68     xbe = 0;
69     session_th = 0;
70     total_generated_sessions = 0;
71     total_completed_sessions = 0;
72     total_dropped_sessions = 0;
73     total_processed_requests = 0;
74     total_aborted_requests = 0;
75     num_ended_req_fs = 0;
76     num_ended_req_be = 0;
77     num_entered_be_serv = 0;
78 }
79
80 /**----- FINE RESET STATE----**/
81
82 /**-----FINE STATO-----**/
83
84 /**-----METRICHE MISURATE----**/
85
86 struct Metrics{
87     double sys_resp; // Job average
88     double sys_thr; // (total_completed_sessions/cl->current)
89
90     double fs_resp; // Job average
91     double fs_thr; // (num_ended_req_fs/cl->current)
92     double fs_util; // Area
93     double fs_pop; // Popolazione del FS
94
95     double be_resp; // Job average
```

```
96      double be_delay; // Tempo medio in coda del be (Job average)
97      double be_thr; // (num_ended_req_be/cl->current)
98      double be_util; // Area
99
100     double th_pop; // Popolazione del nodo di Think
101     double drop_ratio; // (total_dropped_sessions/total_generated_sessions)
102     double abort_ratio; // (total_aborted_requests/total_processed_requests)
103
104     double total_time; // Tempo totale di esecuzione della simulazione (può
            essere più di 10000)
105 };
106
107 void reset_metrics(struct Metrics* m)
108 {
109     m->sys_resp = 0.0;
110     m->sys_thr = 0.0;
111     m->fs_resp = 0.0;
112     m->fs_thr = 0.0;
113     m->fs_util = 0.0;
114     m->fs_pop = 0;
115     m->be_resp = 0.0;
116     m->be_delay = 0.0;
117     m->be_thr = 0.0;
118     m->be_util = 0.0;
119     m->th_pop = 0;
120     m->drop_ratio = 0.0;
121     m->abort_ratio = 0.0;
122     m->total_time = 0.0;
123 }
124
125 struct Metrics* create_metrics(void)
126 {
127     struct Metrics* m = (struct Metrics*)malloc(sizeof(struct Metrics));
128
129     if(m == NULL)
130     {
131         perror("Error in malloc() of create_metrics!\n");
132         exit(1);
133     }
134
135     reset_metrics(m);
136
137     return m;
138 }
139
140 void destroy_metrics(struct Metrics** m)
141 {
142     /*Rimuove dallo heap una metrica*/
143     if(*m != NULL)
144     {
145         free(*m);
146         *m = NULL;
```

```
147     }
148 }
149 /**-----FINE METRICHE MISURATE-----**/
150
151 /**-----GET INTER-TIME MULTISTREAM-----**/
152
153 int GetSessionLength(void)
154 {
155     /*Genera la lunghezza (numero di richieste) delle nuove sessioni*/
156
157   SelectStream(0);
158   return Equilikely(LOWER_EQ, UPPER_EQ);
159 }
160
161 double GetInterArrival(void)
162 {
163     /*Genera il tempo di interarrivo delle nuove sessioni*/
164
165   SelectStream(1);
166   return Exponential((double)INTER_ARRIVAL_T);
167 }
168
169 double GetEndFs(void)
170 {
171     /*Genera il tempo di uscita dal fs*/
172
173   SelectStream(2);
174   return Exponential((double)FS_AVG_T);
175 }
176
177 double GetEndBe(void)
178 {
179     /*Genera il tempo di uscita dal be*/
180
181   SelectStream(3);
182   return Exponential((double)BE_AVG_T);
183 }
184
185 double GetEndTh(void)
186 {
187     /*Genera il tempo di uscita dal th*/
188
189   SelectStream(4);
190   return Exponential((double)TH_AVG_T);
191 }
192
193 /**-----GET INTER-TIME MULTISTREAM FINE-----**/
194
195 /**
196                         SIMULATION:
197 **/
198
```

```
199 void simulation(struct Metrics* measures)
200 {
201
202     PlantSeeds(-1);
203
204     double previous;
205
206     struct Metrics* metrics = create_metrics();
207
208     double new_session_time = GetInterArrival(); // Genera il primo tempo di arrivo
209     struct calendar* c = create_calendar(new_session_time); // Tutti i tipi di
            evento
210
211     struct server* fs = create_server(); // Front End
212     struct area* a_fs = create_area(); // Area del Front End
213     struct server* be = create_server(); // Back End
214     struct area* a_be = create_area(); // Area del Back End
215
216     struct list* thinklist = create_list(); // Coda di think (coda con priorità)
217
218     struct clock* cl = create_clock(c->events_times[min(c)]);
219     int index = min(c); // Primo evento (arrivo nuova sessione o sampling)
220
221     struct node *n = malloc(sizeof(intptr_t));
222
223     while(c->events_times[NEW_SESSION_INDEX] < STOP_SIMULATION || xfs + xbe +
            session_th > 0)
224     {
225         previous = cl->current;
226         cl->current = cl->next;
227
228         //AGGIORNAMENTO METRICHE E AREA:
229
230         //Area fs
231         update_x(a_fs, (cl->current - previous)*xfs);
232         update_q(a_fs, (cl->current - previous)*(session_fifo_fs));
233         update_l(a_fs, (cl->current - previous)*(session_fifo_fs + xfs));
234         //Area be
235         update_x(a_be ,(cl->current - previous)*xbe);
236         update_q(a_be, (cl->current - previous)*(session_fifo_be));
237         update_l(a_be, (cl->current - previous)*(session_fifo_be + xbe));
238
239         metrics->fs_util = (a_fs->x)/(cl->current); // Utilizzazione fs
240         metrics->be_util = (a_be->x)/(cl->current); // Utilizzazione be
241
242         metrics->fs_thr = (num_ended_req_fs)/(cl->current); // Throughput fs
243         metrics->be_thr = (num_ended_req_be)/(cl->current); // Throughput be
244
245         metrics->sys_thr = (total_completed_sessions/(cl->current)); // Throughput
                sys
246
```

```
247         metrics->drop_ratio =
                ((double)(total_dropped_sessions)/(double)(total_generated_sessions));
248         metrics->abort_ratio =
                ((double)(total_aborted_requests)/(double)(total_processed_requests));
249
250         //FINE AGGIORNAMENTO METRICHE E AREA
251
252         //Modifica delle measures -->> PER INTERVALLI DI CONFIDENZA A 10.000
                SECONDI (TAGLIO DELLA SIMULAZIONE)
253
254         if(cl->current == STOP_SIMULATION)
255         {
256             measures->sys_resp = metrics->sys_resp;
257             measures->sys_thr = metrics->sys_thr;
258             measures->fs_resp = metrics->fs_resp;
259             measures->fs_thr = metrics->fs_thr;
260             measures->fs_pop = metrics->fs_pop;
261             measures->fs_util = metrics->fs_util;
262             measures->be_resp = metrics->be_resp;
263             measures->be_delay = metrics->be_delay;
264             measures->be_thr = metrics->be_thr;
265             measures->th_pop = metrics->th_pop;
266             measures->drop_ratio = metrics->drop_ratio;
267             measures->abort_ratio = metrics->abort_ratio;
268         }
269
270         //FINE Modifica delle measures -->> PER INTERVALLI DI CONFIDENZA A 10.000
                SECONDI (TAGLIO DELLA SIMULAZIONE)
271
272         //ATTIVAZIONE DROP E ABORT:
273
274         if(omm == true)
275         {
276             if (ab_dr == false && metrics->fs_util >= ABDR_THRESHOLD)
277                 ab_dr = true;
278
279             else if(ab_dr == true && metrics->fs_util <= ABDR_FLOOR)
280                 ab_dr = false;
281         }
282
283         //FINE ATTIVAZIONE DROP E ABORT
284
285         switch (index) {
286
287             case NEW_SESSION_INDEX:
288
289                 total_generated_sessions++; // Anche se droppate
290
291                 if(ab_dr == false)
292                 {
293                     n = create_node();
294                     n->length = GetSessionLength(); // Numero di richieste
```

```
295                     n->arrival_FS = cl->current;
296                     n->sessionStart = cl->current;
297                     add_new_req(fs, n);
298                     xfs = 1;
299                     session_fifo_fs = fs->fifo->size;
300                     metrics->fs_pop = xfs + session_fifo_fs;
301 
302                     if(fs->fifo->size == 0) {
303                         c->events_times[EXIT_FS_INDEX] = cl->current + GetEndFs();
304                         n->ended_FS = c->events_times[EXIT_FS_INDEX];
305                     }
306 
307                 }
308 
309                 else
310                 {
311                     total_dropped_sessions++;
312                 }
313 
314                 c->events_times[NEW_SESSION_INDEX] = cl->current + GetInterArrival();
315                 if(c->events_times[NEW_SESSION_INDEX] >= STOP_SIMULATION) {
316                     c->events_times[NEW_SESSION_INDEX] = 100 * STOP_SIMULATION;
317                 }
318 
319                 break;
320                     /**FINE ARRIVO NUOVA SESSIONE (FINE CASE 0)**/
321 
322             case EXIT_FS_INDEX: /**USCITA RICHIESTA DA FS**/
323 
324                 //printf("EXIT FS\n");
325 
326                 n = update_internal_node(fs);
327 
328                 if(fs->internal_node != NULL) {
329                     c->events_times[EXIT_FS_INDEX] = cl->current + GetEndFs();
330                     fs->internal_node->ended_FS = c->events_times[EXIT_FS_INDEX];
331                 } else {
332                     c->events_times[EXIT_FS_INDEX] = 100 * STOP_SIMULATION;
333                     xfs = 0;
334                 }
335 
336                 session_fifo_fs = fs->fifo->size;
337                 metrics->fs_pop = xfs + session_fifo_fs;
338                 num_ended_req_fs++;
339                 //AGGIORNAMENTO METRICA fs_resp (TRAMITE WELFORD ONE PASS)
340                 (metrics->fs_resp) = (metrics->fs_resp) + ((n->ended_FS -
                        n->arrival_FS) - (metrics->fs_resp))/num_ended_req_fs;
341 
342                 add_new_req(be, n);
343 
344                 xbe = 1;
345                 session_fifo_be = be->fifo->size;
```

```
346
347                 if(be->fifo->size == 0) {
348                     c->events_times[EXIT_BE_INDEX] = cl->current + GetEndBe();
349                     n->ended = c->events_times[EXIT_BE_INDEX];
350
351                     num_entered_be_serv++;
352                     //AGGIORNAMENTO METRICA be_delay (TRAMITE WELFORD ONE PASS)
353                     (metrics->be_delay) = (metrics->be_delay) + ((cl->current -
                            n->ended_FS) - (metrics->be_delay))/num_entered_be_serv;
354                 }
355
356                 break;
357                     /**FINE USCITA RICHIESTA DA FS (FINE CASE 1)**/
358
359             case EXIT_BE_INDEX: /**USCITA RICHIESTA DA BE**/
360                 //printf("EXIT BE\n");
361
362                 n = update_internal_node(be);
363                 if(be->internal_node != NULL) {
364                     c->events_times[EXIT_BE_INDEX] = cl->current + GetEndBe();
365                     be->internal_node->ended = c->events_times[EXIT_BE_INDEX];
366
367                     num_entered_be_serv++;
368                     //AGGIORNAMENTO METRICA be_delay (TRAMITE WELFORD ONE PASS)
369                     (metrics->be_delay) = (metrics->be_delay) + ((cl->current -
                            be->internal_node->ended_FS) -
                            (metrics->be_delay))/num_entered_be_serv;
370                 } else {
371                     c->events_times[EXIT_BE_INDEX] = 100 * STOP_SIMULATION;
372                     xbe = 0;
373                 }
374
375                 session_fifo_be = be->fifo->size;
376                 num_ended_req_be++;
377                 total_processed_requests++;
378
379                 n->length--;
380
381                 metrics->be_thr = (num_ended_req_be)/(cl->current); // Throughput be
382
383                 //AGGIORNAMENTO METRICA be_resp (TRAMITE WELFORD ONE PASS)
384                 (metrics->be_resp) = (metrics->be_resp) + ((n->ended - n->ended_FS)
                        - (metrics->be_resp))/num_ended_req_be;
385
386                 //AGGIORNAMENTO METRICA sys_resp (TRAMITE WELFORD ONE PASS)
387                 (metrics->sys_resp) = (metrics->sys_resp) + ((n->ended -
                        n->arrival_FS) - (metrics->sys_resp))/num_ended_req_be;
388
389                 if(n->length == 0) {
390                     n->sessionEnded = cl->current;
391                     total_completed_sessions++;
```

```
392                     metrics->sys_thr = (total_completed_sessions/(cl->current)); //
                            Throughput sys
393                     destroy_node(&n);
394                 } else {
395
396                     n->endThinkTime = cl->current + GetEndTh();
397                     add_node_THINK_mod(thinklist, n);
398                     session_th = thinklist->size;
399                     metrics->th_pop = session_th;
400                     c->events_times[EXIT_TH_INDEX] = thinklist->head->endThinkTime;
401                 }
402
403                 break;
404                     /**FINE USCITA RICHIESTA DA BE (FINE CASE 2)**/
405
406             case EXIT_TH_INDEX: /**USCITA RICHIESTA DA TH**/
407                 //printf("EXIT TH\n");
408
409                 n = fetch_node(thinklist);
410
411                 if(thinklist->size == 0) {
412                     c->events_times[EXIT_TH_INDEX] = 100 * STOP_SIMULATION;
413                 } else {
414                     c->events_times[EXIT_TH_INDEX] = thinklist->head->endThinkTime;
415                 }
416
417                 if(ab_dr == false)
418                 {
419                     n->arrival_FS = cl->current;
420
421                     add_new_req(fs, n);
422                     xfs = 1;
423                     session_fifo_fs = fs->fifo->size;
424                     metrics->fs_pop = xfs + session_fifo_fs;
425
426                     if(fs->fifo->size == 0) {
427                         c->events_times[EXIT_FS_INDEX] = cl->current + GetEndFs();
428                         n->ended_FS = c->events_times[EXIT_FS_INDEX];
429                     }
430
431
432                 }
433                 // ab_dr attivo -->> ABORT REQUEST
434                 else
435                 {
436                     destroy_node(&n);
437                     total_dropped_sessions++;
438                     total_aborted_requests++;
439                     total_processed_requests++;
440                 }
441
442                 session_th = thinklist->size;
```

```
443                 metrics->th_pop = session_th;
444
445                 break;
446                     /**FINE USCITA RICHIESTA DA TH (FINE CASE 3)**/
447
448             case SAMPLING_INDEX:
449
450                 /** AGGIORNAMENTO DEL TREND TEMPORALE DELLA METRICA SELEZIONATA **/
451                 if(cl->current <= STOP_SIMULATION)
452                 {
453                         if(strcmp(choice,"sys_resp") == 0)
454                             trend[(int)(cl->current - 1)] += metrics->sys_resp;
455
456                         else if (strcmp(choice,"sys_thr") == 0)
457                             trend[(int)(cl->current - 1)] += metrics->sys_thr;
458
459                         else if (strcmp(choice,"fs_resp") == 0)
460                             trend[(int)(cl->current - 1)] += metrics->fs_resp;
461
462                         else if (strcmp(choice,"fs_thr") == 0)
463                             trend[(int)(cl->current - 1)] += metrics->fs_thr;
464
465                         else if (strcmp(choice,"fs_popolation") == 0)
466                             trend[(int)(cl->current - 1)] += (metrics->fs_pop);
467
468                         else if (strcmp(choice,"fs_util") == 0)
469                             trend[(int)(cl->current - 1)] += (metrics->fs_util);
470
471                         else if (strcmp(choice,"be_resp") == 0)
472                             trend[(int)(cl->current - 1)] += metrics->be_resp;
473
474                         else if (strcmp(choice,"be_delay") == 0)
475                             trend[(int)(cl->current - 1)] += metrics->be_delay;
476
477                         else if (strcmp(choice,"be_thr") == 0)
478                             trend[(int)(cl->current - 1)] += metrics->be_thr;
479
480                         else if (strcmp(choice,"th_popolation") == 0)
481                             trend[(int)(cl->current - 1)] += metrics->th_pop;
482
483                         else if (strcmp(choice,"drop_ratio") == 0)
484                             trend[(int)(cl->current - 1)] += metrics->drop_ratio;
485
486                         else // Abort ratio
487                             trend[(int)(cl->current - 1)] += metrics->abort_ratio;
488
489                         //printf("%f\n", trend[(int)(cl->current - 1)]);
490                 }
491
492                 set_sample_time(c, cl->current + T_SAMPLE);
493                 break;
494
```

```
495             default:
496                 printf("Bad event!\n");
497         }
498 
499         cl->next = c->events_times[min(c)]; // Aggiorno il tempo del next event
500         index = min(c); // Il prossimo evento potrebbe essere cambiato
501 
502     }
503 
504     measures->total_time = cl->current;
505 
506     //FREE MEMORY
507 
508     destroy_calendar(&c);
509     destroy_clock(&cl);
510     destroy_server(&fs);
511     destroy_server(&be);
512     destroy_list(&thinklist);
513     destroy_area(&a_fs);
514     destroy_area(&a_be);
515     destroy_metrics(&metrics);
516 
517     free(n);
518 }
519 
520 /** CONFIDENCE INTERVAL **/
521 
522 double confidence_interval(int n, double s)
523 {
524     double t = idfStudent(n-1, 0.975);
525     double sqnum = sqrt((double) n);
526     double sqs = sqrt(s);
527 
528     return (t * sqs) / sqnum;
529 }
530 
531 /** FINE CONFIDENCE INTERVAL **/
532 
533 /**
534     MAIN:
535 **/
536 
537 int main(void)
538 {
539     printf("TYPE 1 TO ACTIVATE THE OMM MECHANISM, 0 OTHERWISE [ENTER]:\n\n");
540 
541     scanf("%d",&omm);
542 
543     printf("\nENTER THE METRIC TO MEASURE (ITS TIME TREND)
            [ENTER]:\n1.sys_resp\n2.sys_thr\n3.fs_resp\n4.fs_thr\n5.fs_popolation\n6.5.fs_util\n7.be_resp
544 
545     scanf("%s", choice); // Selezione della metrica da monitorare (trend temporale)
```

```
546
547     printf("\n");
548
549     //Generazione della struttura che raccoglie le metriche (per ogni simulazione)
550     struct Metrics* metrics = create_metrics();
551
552     struct Metrics* mean = create_metrics(); // Utile per il calcolo
            dell'intervallo di confidenza
553     struct Metrics* var = create_metrics(); // Utile per il calcolo
            dell'intervallo di confidenza
554
555     //INIZIALIZZAZIONE TREND METRICA SELEZIONATA (MEDIATA PER OGNI RUN):
556
557     trend = (double*)malloc(10000*sizeof(double));
558
559     if(trend == NULL)
560     {
561         perror("ERROR IN MALLOC!\n");
562     }
563
564     memset(trend, 0.0, sizeof(double)*10000);
565
566     int i;
567
568     for(i = 1 ; i < 51 ; i++)
569     {
570         printf("SIMULATION IN PROGRESS (RUN %d) \n", i);
571
572         simulation(metrics);
573
574         //VARIANZE:
575         var->sys_resp = var->sys_resp + pow((metrics->sys_resp - mean->sys_resp) ,
                2.0)*(i-1)/i;
576         var->sys_thr = var->sys_thr + pow((metrics->sys_thr - mean->sys_thr) ,
                2.0)*(i-1)/i;
577
578         var->fs_resp = var->fs_resp + pow((metrics->fs_resp - mean->fs_resp) ,
                2.0)*(i-1)/i;
579         var->fs_thr = var->fs_thr + pow((metrics->fs_thr - mean->fs_thr) ,
                2.0)*(i-1)/i;
580         var->fs_pop = var->fs_pop + pow((metrics->fs_pop - mean->fs_pop) ,
                2.0)*(i-1)/i;
581         var->fs_util = var->fs_util + pow((metrics->fs_util - mean->fs_util) ,
                2.0)*(i-1)/i;
582
583         var->be_resp = var->be_resp + pow((metrics->be_resp - mean->be_resp) ,
                2.0)*(i-1)/i;
584         var->be_delay = var->be_delay + pow((metrics->be_delay - mean->be_delay) ,
                2.0)*(i-1)/i;
585         var->be_thr = var->be_thr + pow((metrics->be_thr - mean->be_thr) ,
                2.0)*(i-1)/i;
```

```
586         var->th_pop = var->th_pop + pow((metrics->th_pop - mean->th_pop) ,
                2.0)*(i-1)/i;
587
588         var->drop_ratio = var->drop_ratio + pow((metrics->drop_ratio -
                mean->drop_ratio) , 2.0)*(i-1)/i;
589         var->abort_ratio = var->abort_ratio + pow((metrics->abort_ratio -
                mean->abort_ratio) , 2.0)*(i-1)/i;
590
591         var->total_time = var->total_time + pow((metrics->total_time -
                mean->total_time) , 2.0)*(i-1)/i;
592
593         //MEDIE:
594         mean->sys_resp = (mean->sys_resp) + (metrics->sys_resp - mean->sys_resp)/i;
595         mean->sys_thr = (mean->sys_thr) + (metrics->sys_thr - mean->sys_thr)/i;
596
597         mean->fs_resp = (mean->fs_resp) + (metrics->fs_resp - mean->fs_resp)/i;
598         mean->fs_thr = (mean->fs_thr) + (metrics->fs_thr - mean->fs_thr)/i;
599         mean->fs_pop = (mean->fs_pop) + (metrics->fs_pop - mean->fs_pop)/i;
600         mean->fs_util = (mean->fs_util) + (metrics->fs_util - mean->fs_util)/i;
601
602         mean->be_resp = (mean->be_resp) + (metrics->be_resp - mean->be_resp)/i;
603         mean->be_delay = (mean->be_delay) + (metrics->be_delay - mean->be_delay)/i;
604         mean->be_thr = (mean->be_thr) + (metrics->be_thr - mean->be_thr)/i;
605         mean->th_pop = (mean->th_pop) + (metrics->th_pop - mean->th_pop)/i;
606
607         mean->drop_ratio = (mean->drop_ratio) + (metrics->drop_ratio -
                mean->drop_ratio)/i;
608         mean->abort_ratio = (mean->abort_ratio) + (metrics->abort_ratio -
                mean->abort_ratio)/i;
609
610         mean->total_time = (mean->total_time) + (metrics->total_time -
                mean->total_time)/i;
611
612         reset_state();
613         reset_metrics(metrics);
614
615         printf("\n");
616     }
617
618     printf("System Response Time: \n");
619     printf("Mean: %6.6f\n",mean->sys_resp);
620     printf("Var: %6.6f\n",(var->sys_resp/(i-2)));
621     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->sys_resp/(i-2))));
622
623     printf("System Useful Throughput: \n");
624     printf("Mean: %6.6f\n",mean->sys_thr);
625     printf("Var: %6.6f\n",(var->sys_thr/(i-2)));
626     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->sys_thr/(i-2))));
627
628     printf("Fs Response Time: \n");
```

```
629     printf("Mean: %6.6f\n",mean->fs_resp);
630     printf("Var: %6.6f\n",(var->fs_resp/(i-2)));
631     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->fs_resp/(i-2))));
632
633     printf("Fs Throughput: \n");
634     printf("Mean: %6.6f\n",mean->fs_thr);
635     printf("Var: %6.6f\n",(var->fs_thr/(i-2)));
636     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->fs_thr/(i-2))));
637
638     printf("Popolation FS: \n");
639     printf("Mean: %6.6f\n",mean->fs_pop);
640     printf("Var: %6.6f\n",(var->fs_pop/(i-2)));
641     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->fs_pop/(i-2))));
642
643     printf("Utilization FS: \n");
644     printf("Mean: %6.6f\n",mean->fs_util);
645     printf("Var: %6.6f\n",(var->fs_util/(i-2)));
646     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->fs_util/(i-2))));
647
648     printf("Be Response Time: \n");
649     printf("Mean: %6.6f\n",mean->be_resp);
650     printf("Var: %6.6f\n",(var->be_resp/(i-2)));
651     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->be_resp/(i-2))));
652
653     printf("Be Delay Time: \n");
654     printf("Mean: %6.6f\n",mean->be_delay);
655     printf("Var: %6.6f\n",(var->be_delay/(i-2)));
656     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->be_delay/(i-2))));
657
658     printf("Be Throughput: \n");
659     printf("Mean: %6.6f\n",mean->be_thr);
660     printf("Var: %6.6f\n",(var->be_thr/(i-2)));
661     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->be_thr/(i-2))));
662
663     printf("Popolation TH: \n");
664     printf("Mean: %6.6f\n",mean->th_pop);
665     printf("Var: %6.6f\n",(var->th_pop/(i-2)));
666     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->th_pop/(i-2))));
667
668     printf("Drop Ratio: \n");
669     printf("Mean: %6.6f\n",mean->drop_ratio);
670     printf("Var: %6.6f\n",(var->drop_ratio/(i-2)));
671     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->drop_ratio/(i-2))));
```

```
672
673     printf("Abort Ratio: \n");
674     printf("Mean: %6.6f\n",mean->abort_ratio);
675     printf("Var: %6.6f\n",(var->abort_ratio/(i-2)));
676     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->abort_ratio/(i-2))));
677
678     printf("Total Time: \n");
679     printf("Mean: %6.6f\n",mean->total_time);
680     printf("Var: %6.6f\n",(var->total_time/(i-2)));
681     printf("Confidence Interval (level 0.95): (+/-)%6.6f\n\n",
            confidence_interval(i-1 , (var->total_time/(i-2))));
682
683     /** SCRITTURA SU FILE DEL TREND TEMPORALE DELLA METRICA SELEZIONATA **/
684
685     FILE* trend_f = fopen(choice, "a+");
686     int index;
687
688     for(index = 0 ; index < STOP_SIMULATION ; index ++)
689     {
690         fprintf(trend_f, "%6.6f,", (double)(trend[index]/(i-1)));
691     }
692
693     fclose(trend_f);
694
695     destroy_metrics(&metrics);
696     destroy_metrics(&mean);
697     destroy_metrics(&var);
698
699     free(trend);
700
701     printf("END SIMULATIONS: ALL DATA STORED\n");
702
703     return EXIT_SUCCESS;
704 }
```

Bibliografia

DISCRETE-EVENT SIMULATION: A FIRST COURSE, Lawrence Leemis