



**POLITECNICO**  
**MILANO 1863**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

---

# Computer System Performance Evaluation

---

RELAZIONE PROGETTO E

*Autore:*

**Maddes** Evandro

*Codice persona:*

10539479

19 Febbraio, 2021

---

# Indice

---

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Il problema da affrontare . . . . .	1
1.1.1	Parametri del problema . . . . .	1
1.2	Obiettivo . . . . .	2
1.3	Struttura del documento . . . . .	2
<b>2</b>	<b>Descrizione del modello</b>	<b>3</b>
2.1	Formalizzazione del problema . . . . .	3
2.2	Le classi . . . . .	3
2.3	L'architettura del modello completo . . . . .	4
2.3.1	Le componenti del modello . . . . .	4
<b>3</b>	<b>Analisi delle performance</b>	<b>6</b>
3.1	Approccio utilizzato . . . . .	6
3.1.1	Analisi dei dati a disposizione . . . . .	6
3.1.2	Modus operandi . . . . .	6
3.2	Soluzioni ottenute . . . . .	7
3.3	<i>What-if analysis</i> delle soluzioni ottenute . . . . .	8
3.3.1	Utilizzo delle stazioni di coda soluzione 1 . . . . .	8
3.3.2	<i>Throughput</i> soluzione 1 . . . . .	9
3.3.3	<i>Residence time</i> soluzione 1 . . . . .	11
3.3.4	<i>Response time</i> del sistema . . . . .	12
3.3.5	Soluzione 2 . . . . .	12
<b>4</b>	<b>Conclusioni</b>	<b>14</b>
<b>5</b>	<b>References</b>	<b>15</b>

---

# Elenco delle tabelle

---

1.1	Tempo richiesto da ogni risorsa per tipo di richiesta, espresso in <i>ms.</i> . . .	1
1.2	<i>Set</i> di parametri del problema. . . . .	1
3.1	Numero di repliche e utilizzo per ogni stazione coda presente nel sistema, soluzione 1. . . . .	7
3.2	Numero di repliche e utilizzo per ogni stazione coda presente nel sistema, soluzione 2. . . . .	7

---

# Elenco delle figure

---

2.1	<i>Architettura</i> alto livello . . . . .	3
2.2	<i>Classi</i> del sistema . . . . .	4
2.3	<i>Architettura</i> del modello . . . . .	4
2.4	<i>Politica</i> di gestione delle code . . . . .	5
3.1	<i>Utilizzo</i> della stazione <i>web server</i> . . . . .	8
3.2	<i>Utilizzo</i> della stazione <i>application server</i> . . . . .	9
3.3	<i>Utilizzo</i> della stazione <i>database</i> . . . . .	9
3.4	<i>Throughput</i> del <i>web server</i> . . . . .	10
3.5	<i>Throughput</i> dell' <i>application server</i> . . . . .	10
3.6	<i>Throughput</i> del <i>database server</i> . . . . .	10
3.7	<i>Residence time</i> del <i>web server</i> . . . . .	11
3.8	<i>Residence time</i> del <i>application server</i> . . . . .	11
3.9	<i>Residence time</i> del <i>database</i> . . . . .	11
3.10	<i>Response time</i> medio del sistema . . . . .	12
3.11	<i>Utilizzo</i> della stazione <i>web server</i> , soluzione 2 . . . . .	13
3.12	<i>Utilizzo</i> della stazione <i>application server</i> , soluzione 2 . . . . .	13
3.13	<i>Utilizzo</i> della stazione <i>database</i> , soluzione 2 . . . . .	13

# INTRODUZIONE

## 1.1 Il problema da affrontare

Un' applicazione web a tre livelli è composta da un web server (WB), un application server (AS) e un Database (DB): i servizi sono chiamati in sequenza. Ogni server è replicato rispettivamente in  $C_{WS}$ ,  $C_{AS}$  e  $C_{DB}$  istanze, ognuna delle quali gira su una macchina diversa che condivide una coda comune di capacità finita  $K_{WS}$ ,  $K_{AS}$  e  $K_{DB}$ .

Tre tipi di richieste popolano il sistema:

- $N_U$  richieste dell' utente (caratterizzate da un *think time*  $Z_U$ );
- $N_S$  requisiti software di chiamata di procedura remota;
- $N_B$  programmi di collaborazione.

Ogni tipo di richiesta richiede una diversa quantità di tempo da ogni risorsa, secondo una distribuzione esponenziale che segue la media mostrata nella tabella sottostante.

	U	S	B
WS	80	15	5
AS	30	25	120
DB	20	50	40

Tabella 1.1: Tempo richiesto da ogni risorsa per tipo di richiesta, espresso in *ms*.

Le richieste che arrivano ad una stazione piena sono bloccate dopo essere state servite.

### 1.1.1 Parametri del problema

ID	$K_{WS}$	$K_{AS}$	$K_{DB}$	$N_U$	$N_S$	$N_B$	$Z_U$
2	8	30	60	10	50	100	2 <i>min</i>

Tabella 1.2: *Set* di parametri del problema.

## 1.2 Obiettivo

Si vuole determinare il numero minimo di repliche  $C_{WS}$ ,  $C_{AS}$  e  $C_{DB}$  per ogni server tale che le richieste siano servite in media in meno di 2 *sec*.

## 1.3 Struttura del documento

**Chapter 1: Introduzione.** Il capitolo ha lo scopo di illustrare il problema da affrontare, i dati a disposizione e l'obiettivo da raggiungere.

**Chapter 2: Il modello.** Si descrivono nel dettaglio le scelte implementative e il modello sviluppato.

**Chapter 3: Analisi delle performance.** Si analizza il modello dal punto di vista delle performance; si descrive anche l'approccio utilizzato per raggiungere l'obiettivo prefissato.

**Chapter 4: References.**

# DESCRIZIONE DEL MODELLO

Il modello sviluppato è stato implementato usando l'applicativo *JSIMgraph*.

## 2.1 Formalizzazione del problema

Partendo dalla descrizione del problema, si è deciso di sviluppare un modello multi-classe con tre stazioni di coda. Ogni stazione di coda rappresenta un diverso componente dell'architettura *three tier*. Le stazioni di coda sono disposte in sequenza: stazione del *web server*, la stazione del *application server* e infine la stazione del *database*.

In particolare, ogni stazione è caratterizzata da una coda limitata e dal numero di repliche. Al termine della sequenza delle tre stazioni di coda, i diversi *jobs* vengono instradati verso il *web server*.

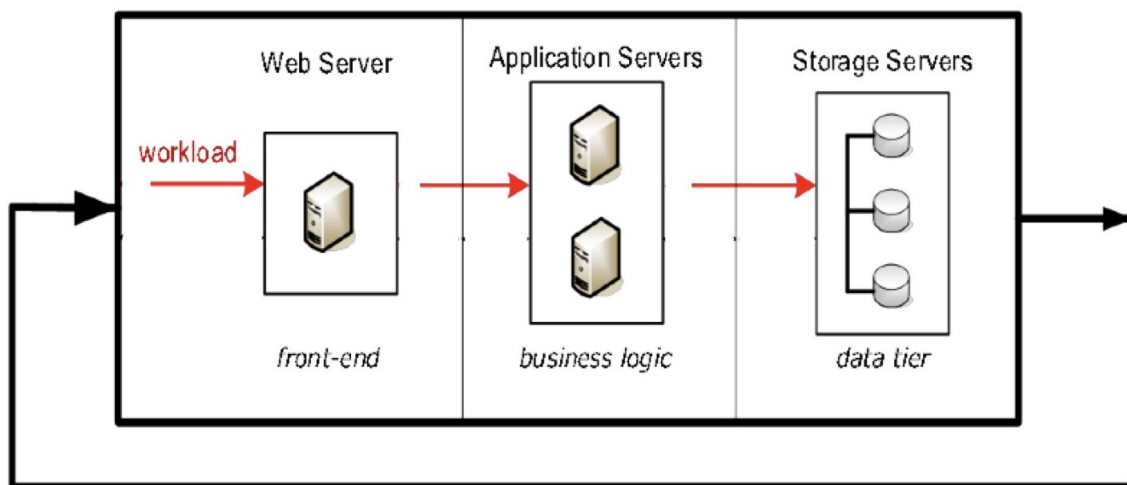


Figura 2.1: architetture del modello di alto livello.

## 2.2 Le classi

I *jobs* che circolano nel sistema sono stati identificati con tre diverse classi: tutti i *jobs* che appartengono alla stessa classe, sono caratterizzati dalle stesse proprietà.

- $N_U$  richieste dell'utente (caratterizzate da un *think time*  $Z_U$ ).

- $N_S$  requisiti software di chiamata di procedura remota;
- $N_B$  programmi di collaborazione.

Tutte e tre le classi sono di tipo chiuso e sono caratterizzate da una diversa popolazione (numero di *jobs*). Tuttavia solo la classe richieste dell'utente (U), presenta un *think time* seguente una distribuzione esponenziale con valore medio 2 *min*. Questa specifica è gestita tramite una stazione di ritardo. Per quanto riguarda le restanti due classi, il relativo *think time* è assunto fisso a zero. La *reference station* per la classe chiamata di procedura remota (S) e per la classe programmi di collaborazione (B), è la stazione *web server*; mentre per la classe (richieste dell'utente) è la stazione di ritardo.













Color	Name	Type	Priority	Population	Interarrival Time Distribution	Reference Station	
	user requests	 Closed	0	10		 delay user requests	
	software agents	 Closed	0	50		 web server	
	batch elaboration ...	 Closed	0	100		 web server	

Figura 2.2: classi presenti nel sistema.

## 2.3 L'architettura del modello completo

Nel modello sono presenti le tre stazioni coda, ciascuna per ogni componente del sistema, disposte in cascate; a valle troviamo il *router* collegato sia alla stazione di ritardo e sia al *web server*.

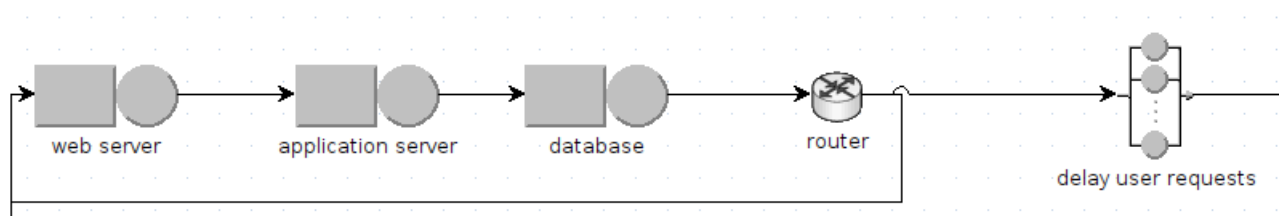
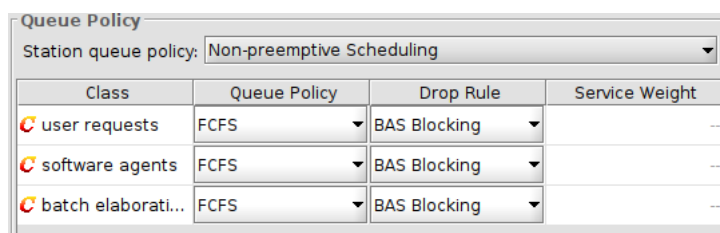


Figura 2.3: architettura del modello completo.

### 2.3.1 Le componenti del modello

A livello architettonico si possono distinguere quattro stazioni ed un router. Tutte le stazioni di coda hanno una capacità limitata e questo implica l'adozione di una politica di gestione nel caso la capacità massima venga raggiunta. La regola utilizzata è *blocking after service (BAS)* per rispettare al meglio le richieste del problema. La politica di selezione del prossimo *job* (o di coda) adottata è una *non-preemptive* di tipo *first come first served (FCFS)*.





Class	Queue Policy	Drop Rule	Service Weight
user requests	FCFS	BAS Blocking	--
software agents	FCFS	BAS Blocking	--
batch elaborati...	FCFS	BAS Blocking	--

Figura 2.4: politica di gestione delle code adottata nelle stazioni di coda (*web server*, *application server*, *database*)

La distribuzione del *service time* segue una esponenziale la cui media è pari ai valori riportati in tabella [1.1]. Analizzando il modello da sinistra verso destra, rispettando l'ordine di visita delle stazioni, si identificano:

- stazione di coda, "*web server*".
- stazione di coda, "*application server*".
- stazione di coda, "*database*".
- router, "*router*". Utilizza una strategia probabilistica per indirizzare i diversi *jobs* presenti nel sistema in base alla loro classe di appartenenza. Se un *job* appartiene alla classe chiamata di procedura remota (S) oppure alla classe programmi di collaborazione (B), essi sono indirizzati al *web server* con probabilità uno; se invece il *job* è della classe richieste dell'utente (U), esso è indirizzato con probabilità uno alla stazione di ritardo.<sup>1</sup>
- stazione di ritardo, "*delay user request*". La distribuzione del *service time* segue una esponenziale la cui media è pari a *12000 ms*; essa equivale al *think time* della classe richieste dell'utente (U).

<sup>1</sup>il router può anche essere eliminato in quanto le sue funzionalità possono essere svolte dalla politica di *routing* del *database*. Tuttavia la sua presenza è giustificata da una maggiore leggibilità del modello.

---

# ANALISI DELLE PERFORMANCE

---

La soluzione per via analitica del problema è impraticabile considerata la natura del modello utilizzato. I fattori principali che impediscono l'utilizzo di tecniche analitiche sono:

- violazione della *device homogeneity property*: l'utilizzo di code con capacità limitata non rispetta questa assunzione.

Anche se le restanti assunzioni sono rispettate, la violazione della *device homogeneity property*, non permette di avere una rete di code separabili. Dunque non si possono utilizzare tecniche come la MVA (che richiederebbe anche un solo server per stazione); si è deciso di procedere con la simulazione a eventi discreti di JMT e con lo strumento *what-if analysis*.

## 3.1 Approccio utilizzato

### 3.1.1 Analisi dei dati a disposizione

Ispezionando i dati a disposizione [1.1], [1.2], si nota che la classe più popolosa presente nel sistema sono i programmi di collaborazione (B). La stazione *application server* ha un *service time* medio per la suddetta classe modellato con un esponenziale di media 120ms, maggior valore tra i dati del problema. Dunque ci aspettiamo che la stazione *application server* abbia bisogno di un maggior numero di repliche rispetto le altre due stazioni. Un ragionamento simile può essere condotto per la stazione *database* rispetto la stazione *web server*. Tali considerazioni però non tengono conto della lunghezza delle code di ogni stazione che incide in modo determinante sulle performance di un sistema. Per quanto riguarda la topologia del modello, la stazione con il *throughput* minore influenza le due stazioni successive essendo il modello "circolare".

### 3.1.2 Modus operandi

L'analisi del modello è stata condotta partendo da una configurazione base rispetto al numero di repliche per ogni stazioni di coda: *web server*, *application server*, *database*. Ognuna ha inizialmente una sola replica. Lo step successivo è stato quello di individuare

la stazione che agiva da collo di bottiglia per il sistema e di aumentarne il numero di repliche. Per individuare il collo di bottiglia si è preso in considerazione l'utilizzo di ogni stazione, il valore medio più elevato decreta a quale stazione aumentare di uno il numero di repliche. Nell'aumentare il numero di repliche, si è anche aggiornata la capacità complessiva della relativa stazione <sup>1</sup>. L'obiettivo è di avere il *response time* del sistema al di sotto dei 2 *sec*, per cui questi passaggi sono stati ripetuti fino al raggiungimento di tale obiettivo.

## 3.2 Soluzioni ottenute

In tutte le soluzioni ottenute, gli indici analizzati sono arrivati a convergenza. Per ognuno di essi l'intervallo di confidenza è pari a 0,99 mentre l'errore relativo è del 3%.

Applicando quanto detto nella sezione [3.1.2], la prima configurazione a raggiungere l'obiettivo è:

	Utilizzo medio	Utilizzo massimo	numero di repliche (C)
<b>WS</b>	0.6788	0.6949	1
<b>AS</b>	0,9990	1.0117	7
<b>DB</b>	0.8782	0.8992	4

Tabella 3.1: Numero di repliche e utilizzo per ogni stazione coda presente nel sistema, soluzione 1.

Con questa configurazione [3.1], si ottiene un *response time* medio del sistema pari a 1999,92 *ms*. Sebbene riesca ad essere sotto i 2 *sec* richiesti dal progetto, analizzando il valore massimo, esso è pari a 2017,92 *ms*, non abbastanza da soddisfare le richieste del problema. Per avere un picco massimo del *response time* del sistema sotto i 2 *sec*, bisogna aggiungere un'ulteriore replica alla stazione del *application server*.

La nuova configurazione raggiunta:

	Utilizzo medio	Utilizzo massimo	numero di repliche (C)
<b>WS</b>	0.7741	0.7959	1
<b>AS</b>	0,9704	1.0067	8
<b>DB</b>	0.9868	1.0041	4

Tabella 3.2: Numero di repliche e utilizzo per ogni stazione coda presente nel sistema, soluzione 2.

---

<sup>1</sup> *JSIMgraph* richiede di definire la capacità di una stazione di coda come somma di *jobs* in coda e *jobs* che sono processati.

Con questa nuova configurazione [3.2], si ottiene un *response time* medio del sistema pari a 1704,9896 *ms*; mentre il massimo valore è pari a 1747,1874 *ms*. Ambedue i valori sono al di sotto del valore richiesto.

### 3.3 *What-if analysis* delle soluzioni ottenute

Si è utilizzato lo strumento *What-if analysis* di *JSIMgraph* per valutare il comportamento delle soluzioni ottenute. Si è deciso di studiare il comportamento del modello rispetto alla classe più popolosa presente nel sistema.

Quindi si è incrementata la popolazione della classe programmi di collaborazione (B) da un valore iniziale di 50 fino ad un valore finale di 120. Tali valori sono stati scelti prendendo in considerazione la reale popolazione della classe programmi di collaborazione (B): 100. Il numero di *steps* è stato selezionato pari a 35, in modo da aggiungere due *jobs* per volta.

#### 3.3.1 Utilizzo delle stazioni di coda soluzione 1

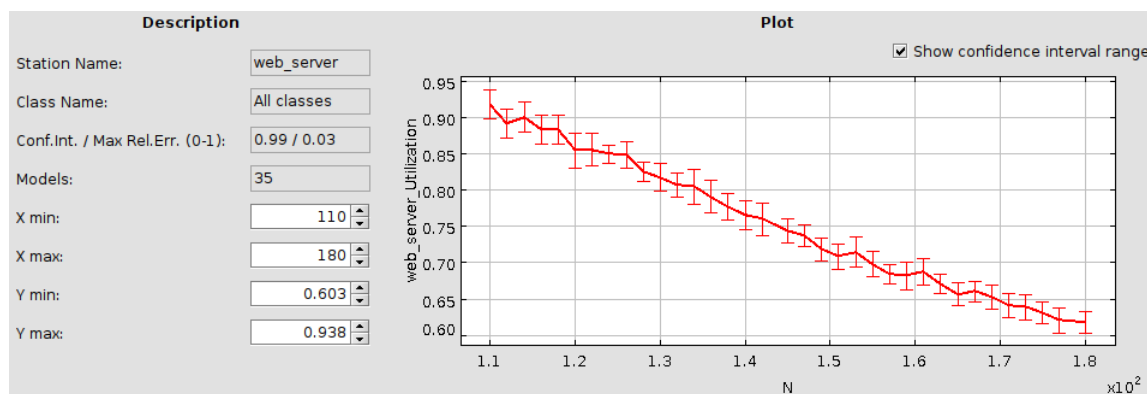


Figura 3.1: Utilizzo della stazione *web server* al variare di  $N_B$ .

Il grafico mostra una riduzione del utilizzo medio del *web server*, ciò è dovuto alla presenza di un collo di bottiglia nel sistema. Il *throughput* di tale stazione incide anche sulle altre stazione limitandone, tra le altre cose anche l'utilizzo.

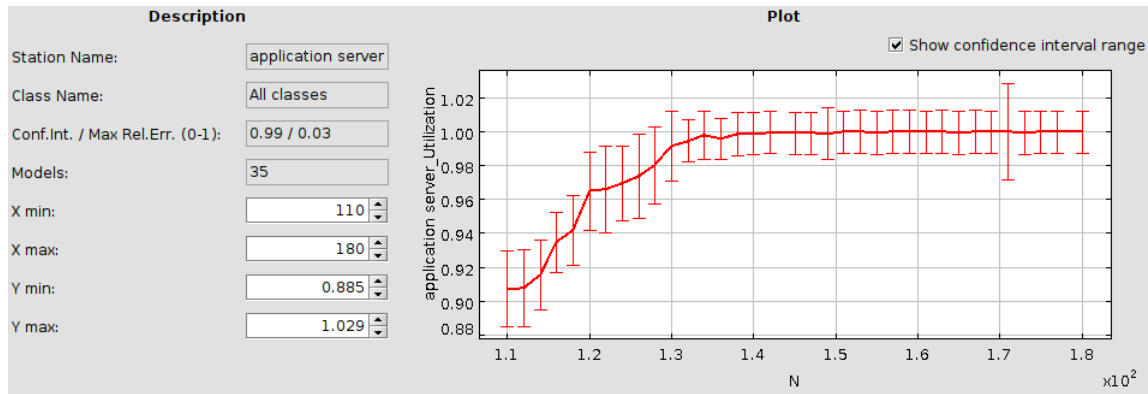


Figura 3.2: Utilizzo della stazione *application server* al variare di  $N_B$ .

Il grafico mostra un andamento opposto al precedente grafico [3.1]; è evidente come l'*application server* rappresenti il bottleneck di tale sistema e il suo throughput limitato, limiti anche le altre risorse del sistema. L'utilizzo medio infatti parte già da valori elevati (0.8) per arrivare a circa 1.

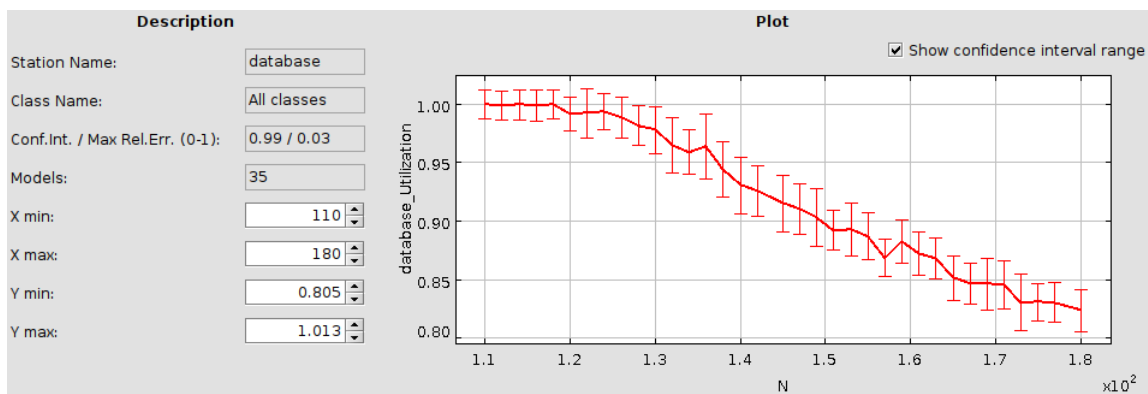


Figura 3.3: Utilizzo della stazione *database* al variare di  $N_B$ .

Il grafico ha un andamento simile al primo grafico mostrato [3.1], per cui considerazioni simili a quelle fatte per il *web server* possono essere ripetute anche per il *database*.

### 3.3.2 Throughput soluzione 1

Di seguito sono riportati i grafici relativi al *throughput* della soluzione 1.

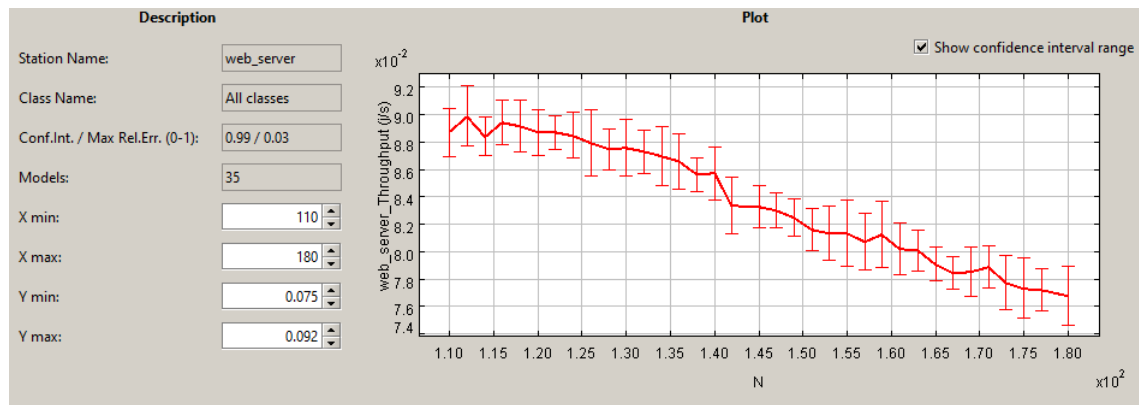


Figura 3.4: *throughput* del *web server*.

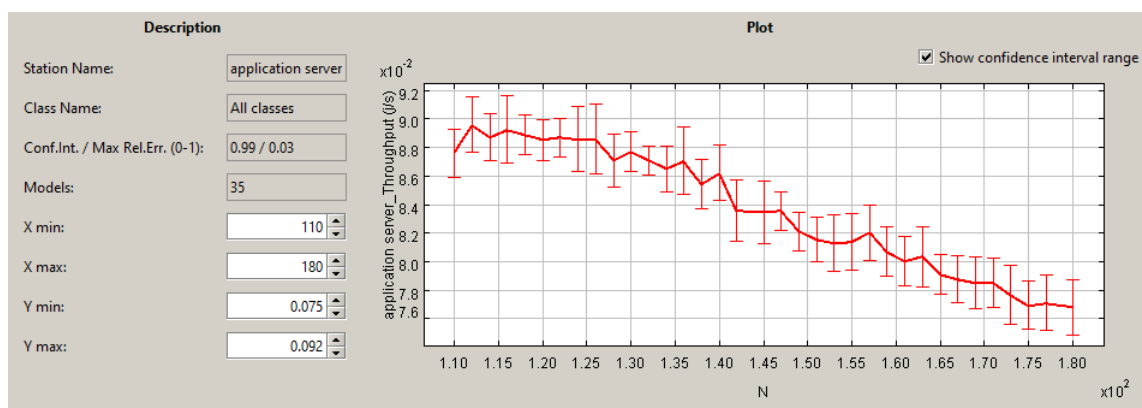


Figura 3.5: *throughput* dell' *application server*.

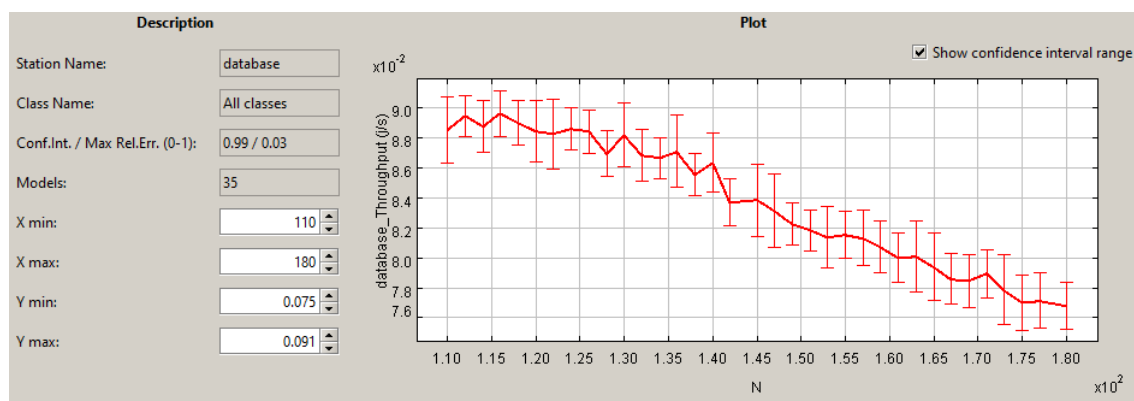


Figura 3.6: *throughput* del *database server*.

Tutti e tre i grafici mostrano lo stesso andamento e gli stessi valori al variare di  $N_B$ . La topologia della rete e la politica di coda *BAS*, non permettono di avere delle perdite nel numero di *jobs* nel sistema, per cui la stazione che agisce da collo di bottiglia serve tutti i lavori in attesa limitando anche il *throughput* delle altre due stazioni.

### 3.3.3 Residence time soluzione 1

Di seguito sono riportati i grafici relativi al *residence time* della soluzione 1.

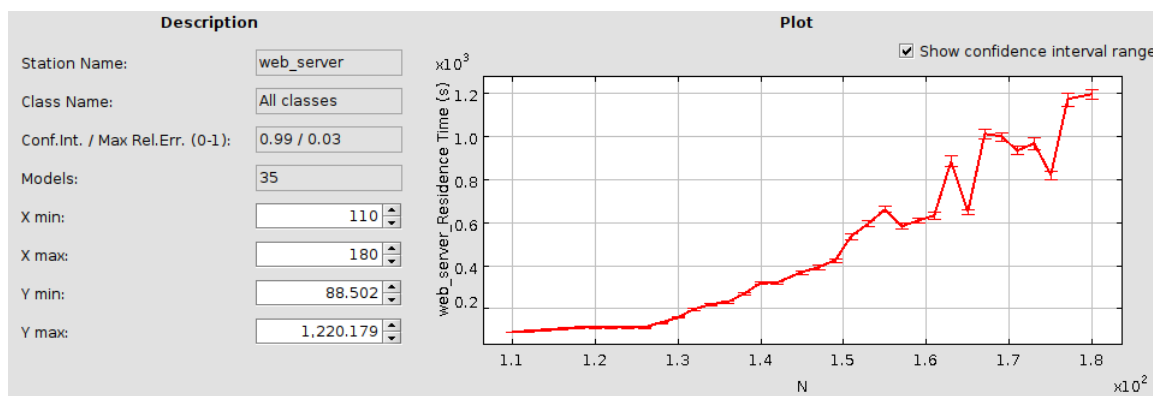


Figura 3.7: *residence time* del web server.

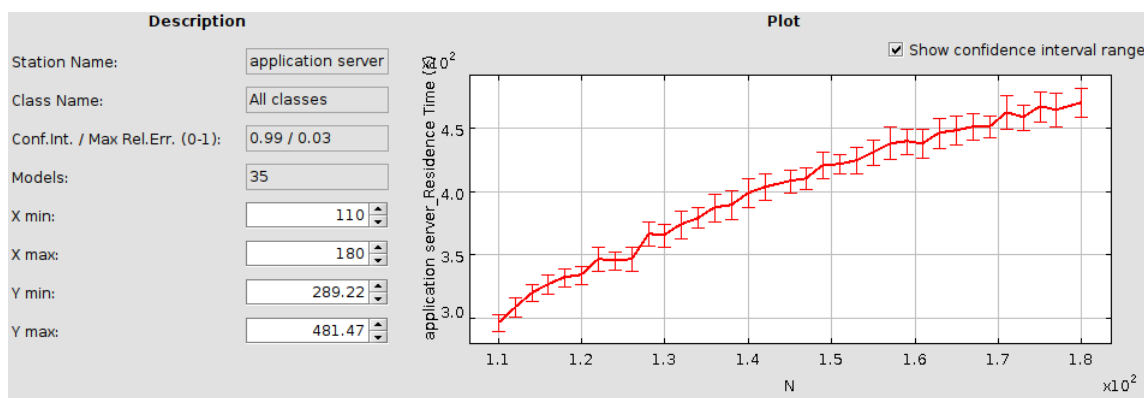


Figura 3.8: *residence time* del application server.

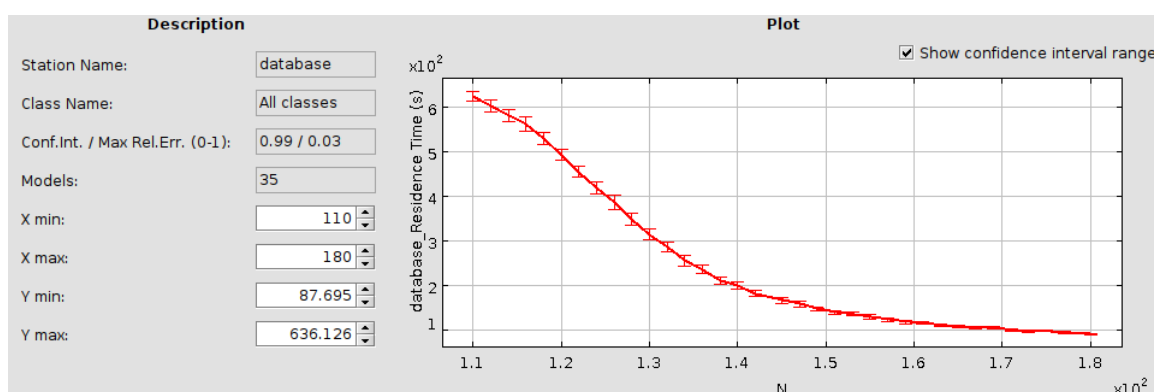


Figura 3.9: *residence time* del database.

È interessante notare come il *residence time* delle stazioni *web server* e dell'*application server* aumenti all'aumentare di  $N_B$ , mentre quello del *database* diminuisce.

Sebbene per la stazione *application server*, collo di bottiglia di questa configurazione,

l'aumento del *residence time* può essere ricondotto ad una condizione di saturazione, discorso differente deve essere affrontato per il *web server*. L'incremento non è dovuto ad una condizione di saturazione, come mostrano i grafici in sezione [3.3.1] bensì può essere ricondotto alla politica di coda utilizzata: *BAS*. Nella stazione precedente al collo di bottiglia (*web server*), il *residence time* aumenta tanto quanto il *service time* medio rimasto alla stazione successiva (*application server*); possiamo definire questo aumento come il *blocking time*. Per quanto riguarda il *database*, il *residence time* diminuisce essendo a valle del collo di bottiglia.

### 3.3.4 Response time del sistema

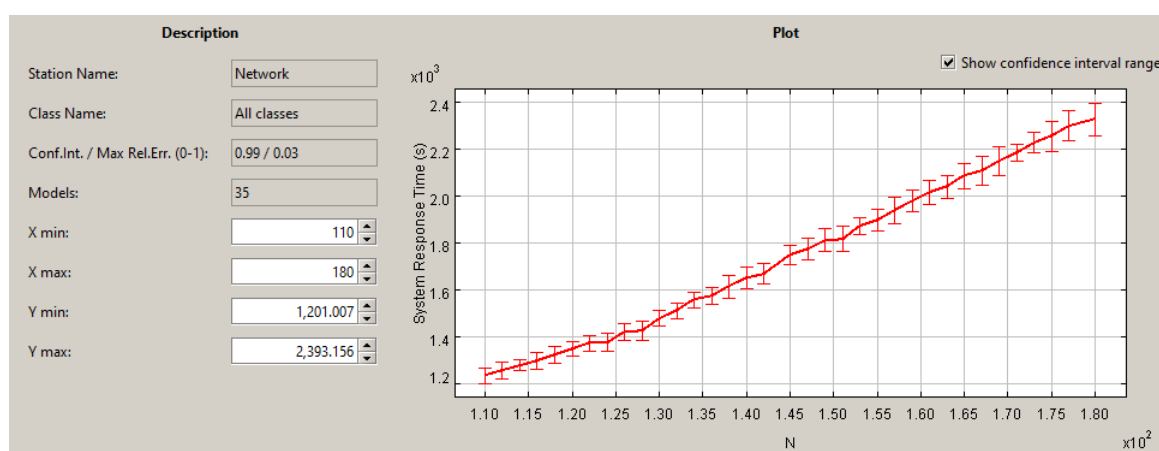


Figura 3.10: *Response time* medio del sistema.

Il grafico mostra un andamento crescente al crescere del numero di *jobs* nel sistema. In corrispondenza di  $N_B$  pari a 100 (sulle ascisse  $1.6 \times 10^2$ )<sup>2</sup>, il valore del *Response time* del sistema è di circa 2 *sec* come atteso dalle precedenti simulazioni [3.1].

### 3.3.5 Soluzione 2

Come per la prima soluzione trovata [3.1], anche per la seconda è stata condotta la stessa *what-if analysis*.

<sup>2</sup>sull'asse delle ascisse è riportato il numero totale dei *jobs* nel sistema:  $N_U + N_S + N_B = 10 + 50 + 100 = 160$



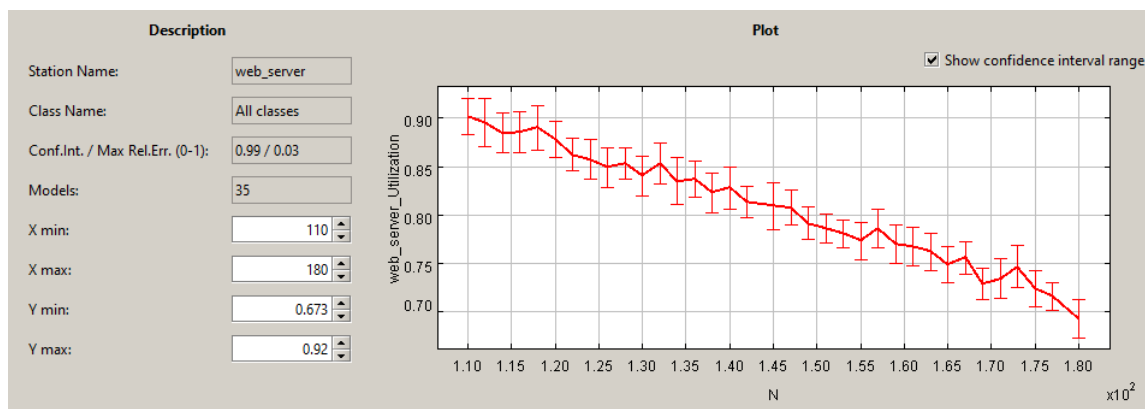


Figura 3.11: Utilizzo della stazione *web server* al variare di  $N_B$ , soluzione 2.

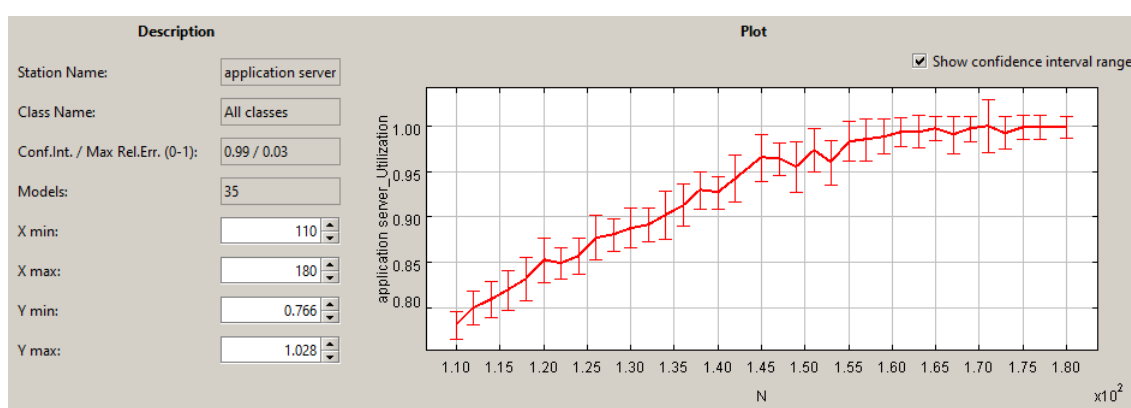


Figura 3.12: Utilizzo della stazione *application server* al variare di  $N_B$ , soluzione 2.

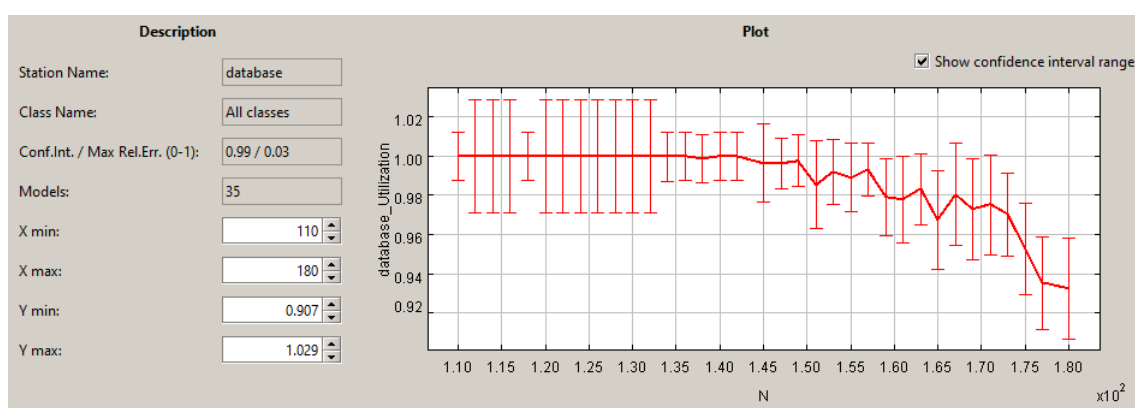


Figura 3.13: Utilizzo della stazione *database* al variare di  $N_B$ , soluzione 2.

I grafici seguono andamenti simili a quelli presentati per la soluzione 1 [3.3.1] per cui simili considerazioni si possono applicare anche a questa soluzione. Confrontando i valori notiamo che l'utilization del *web server* e dell' *application server* sono aumentati rispetto alla soluzione 1. Questo è giustificato dal fatto che il response time, relativo al sistema, della seconda soluzione è minore.

# CONCLUSIONI

---

L'analisi condotta ha individuato due soluzioni che raggiungono l'obiettivo fissito. La prima [3.1] è una soluzione che presenta un minor margine rispetto all'obiettivo della nostra analisi. La seconda [3.2], invece, da maggiori garanzie in termini di affidabilità. Questo vantaggio va ad incidere sui costi di tale soluzione richiedendo una replica in più per l' *application server*. Entrambe le soluzioni rispettano le aspettative enunciate [3.1.1].

## REFERENCES

---

- Modello e analisi realizzati con *JSIMgraph*;
- Slide del corso *Computer Systems Performance Evaluation* [090945] - Marco Gribaudo;