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

Elenco delle tabelle

Elenco delle figure

Capitolo 1

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:

- $\mathbf{N_U}$ richieste dell' utente (caratterizzate da un *think time* Z_U);
- $\mathbf{N_S}$ requisiti software di chiamata di procedura remota;
- $\mathbf{N_B}$ 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	$\mathbf{K_{WS}}$	$\mathbf{K_{AS}}$	$\mathbf{K_{DB}}$	$\mathbf{N_U}$	$\mathbf{N_S}$	$\mathbf{N_B}$	$\mathbf{Z_U}$
2	8	30	60	10	50	100	2 *min*

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.

Capitolo 2

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*.

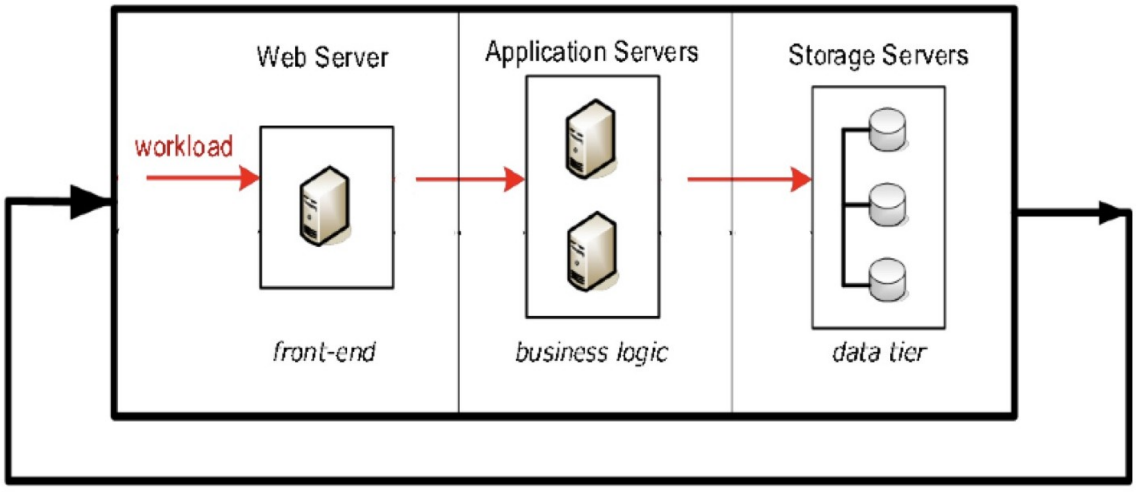

Figura 2.1: architetture del modello di alto livello.

2.2 Le classi

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

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

- $\mathbf{N_S}$ requisiti software di chiamata di procedura remota;
- $\mathbf{N_B}$ 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*. Questa specifica è gestita tramite una stazione di ritardo. La *reference station* per la classe chiamata di procedura remota (S) e per la classe programmi di collaborazione (B), è la stazione *web server*; metre per la classe (richieste dell'utente) è la stazione di ritardo.

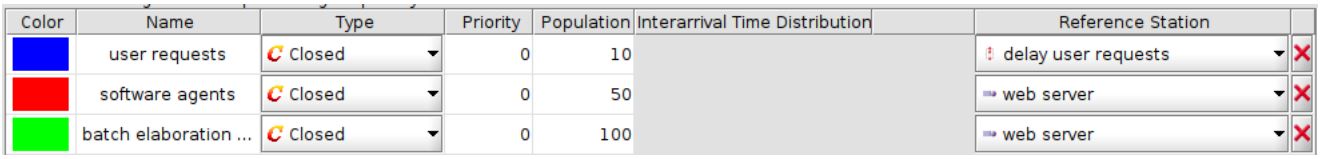

Figura 2.2: classi presenti nel sistema.

2.3 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 ragiunta. 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)*.

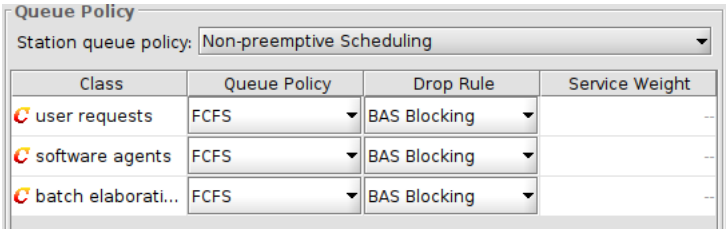

Figura 2.3: politica di gestione delle code adottata nelle stazioni di coda (*web server*, *application server*, *database*)

La distribuzione del *service time* segue una esponenziale la cui media è pari ai valori riportati in tabella [**??**]. 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 stategia 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 dela classe richieste dell' utente (U), esso è indirizzato con probabilità uno alla stazione di ritardo. [1]
- 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).

2.4 L'architettura del modello completo

Riassumendo, nel modello sono presenti le tre stazioni coda, rappresentanti l'architettura *three tier*, disposte in cascate; a valle troviamo il *router* collegato alla stazione di ritardo e al web server.

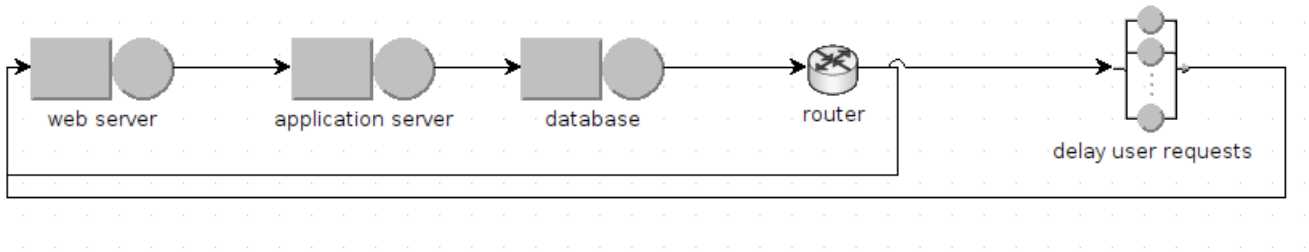

Figura 2.4: architettura del modello completo.

[1] il router puo 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.

Capitolo 3

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 *service center flow balance*: le perdite non assicurano un ugual numero di arrivi e completamenti ad ogni stazione;
- violazione della *device homogeneity property*: l'utilizzo di code con capacità limitata non rispetta questa assunzione.

Dunque il modello non è separabile, 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 [??], [??], 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 di $120ms$, maggior valore tra i *service times*. Dunque ci aspettiamo che la stazione *application server* abbia bisogno di un maggior numero di repliche rispetto le altre due stazione. 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 [1]. 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 quando detto nella sezione [**??**], la prima configurazione a raggiungere l'obiettivo è:

	Utilizzo medio	**Utilizzo massimo**	**numero di repliche (C)**
WS	0.6788	0.6949	1
AS	0,9990	1.0117	7
DB	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 [**??**], si ottiene un *response time* medio del sitema 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)**
WS	0.7741	0.7959	1
AS	0,9704	1.0067	8
DB	0.9868	1.0041	4

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

[1] *JSIMgrapf* 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 [**??**], 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

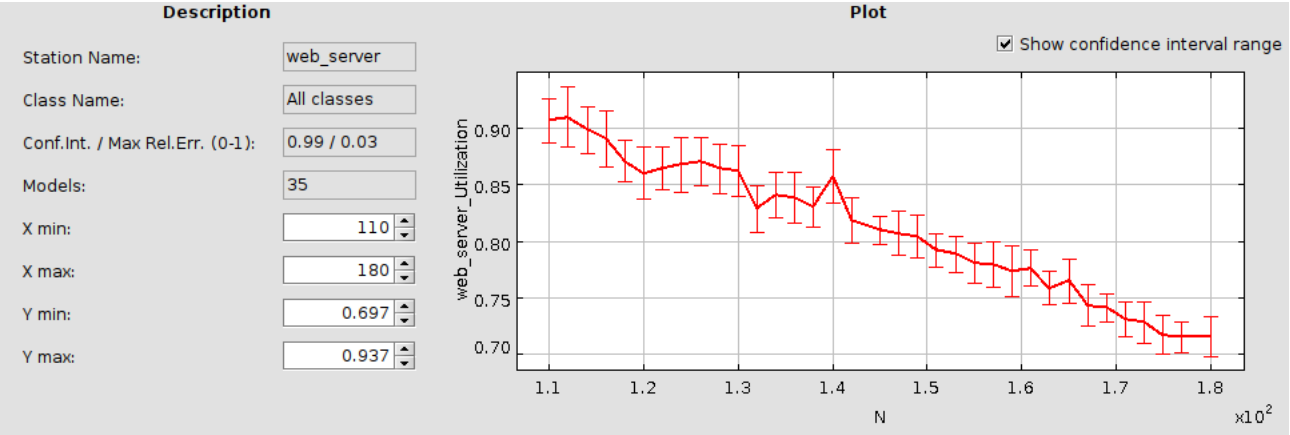

Figura 3.1: Utilizzo della stazione *web server* al variare di N_B.

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.

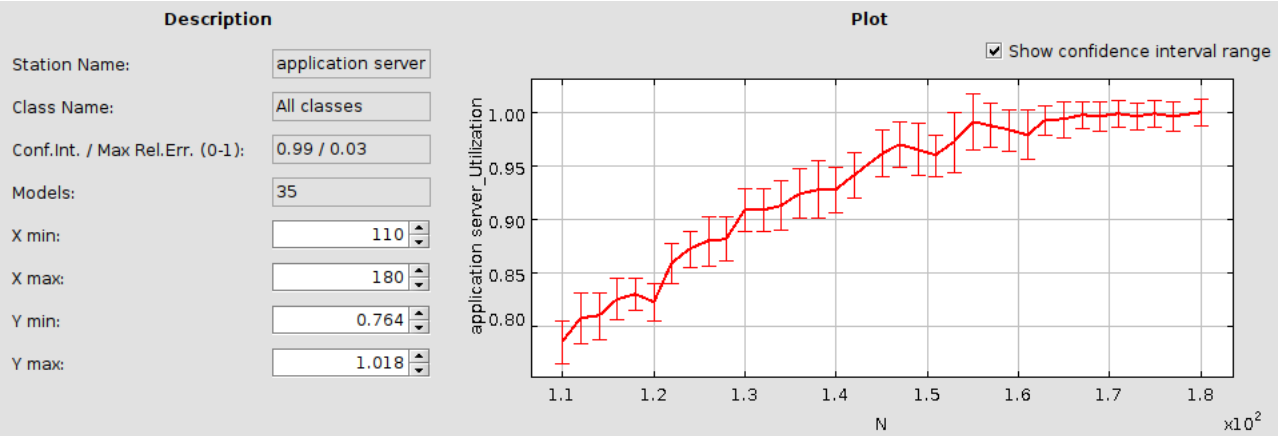

Figura 3.2: Utilizzo della stazione *application server* al variare di N_B.

Il grafico mostra un andamento opposta al precedente grafico [**??**]; 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 gia da valori elevati (0.8) per arrivare a circa 1.

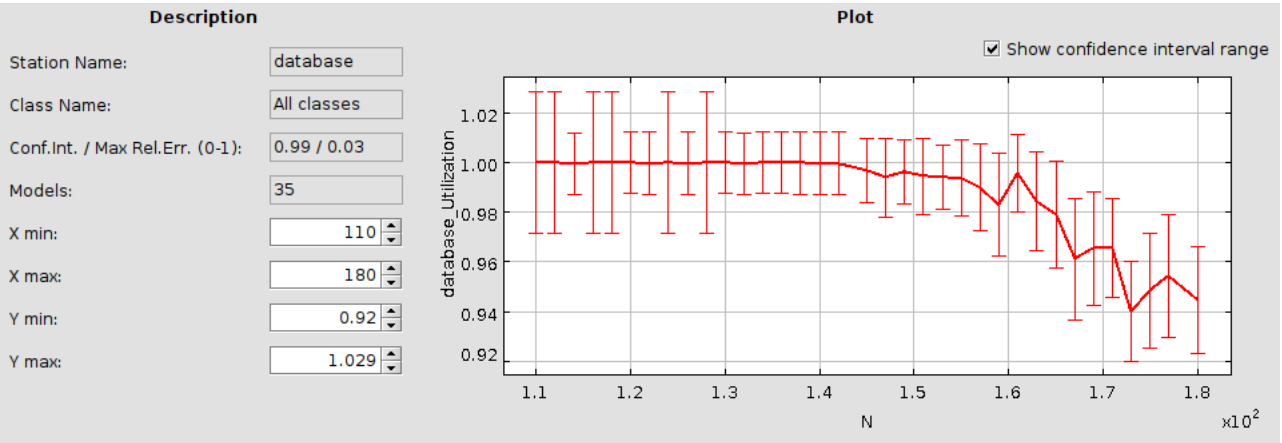

Figura 3.3: Utilizzo della stazione *databse* al variare di N_B.

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

3.3.2 *Queue time* e *residence time*

Di seguito sono riportati i grafici relativi al *residence time*, in quanto quelli relativi al *queue time* presentano lo stesso andamento.

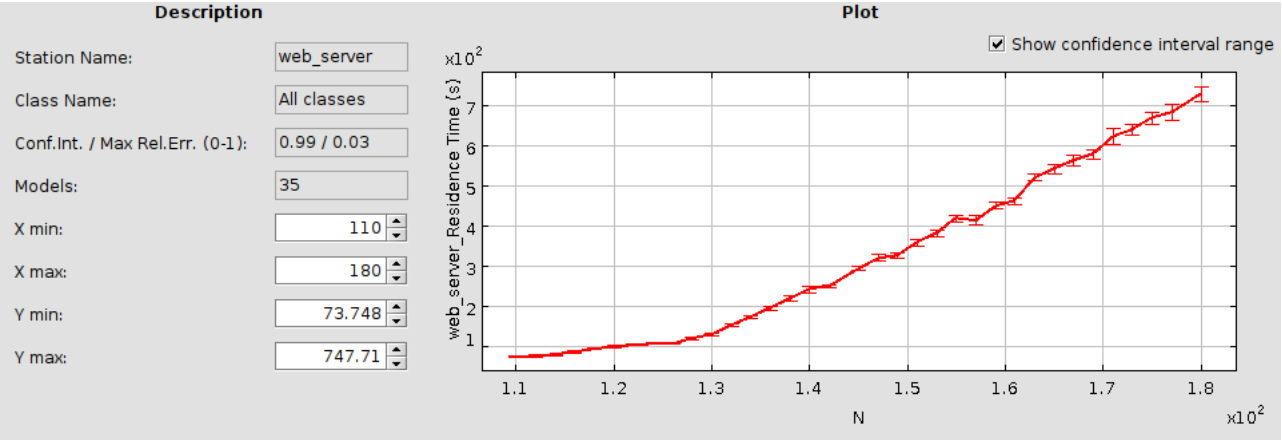

Figura 3.4: *residence time* del *web server*.

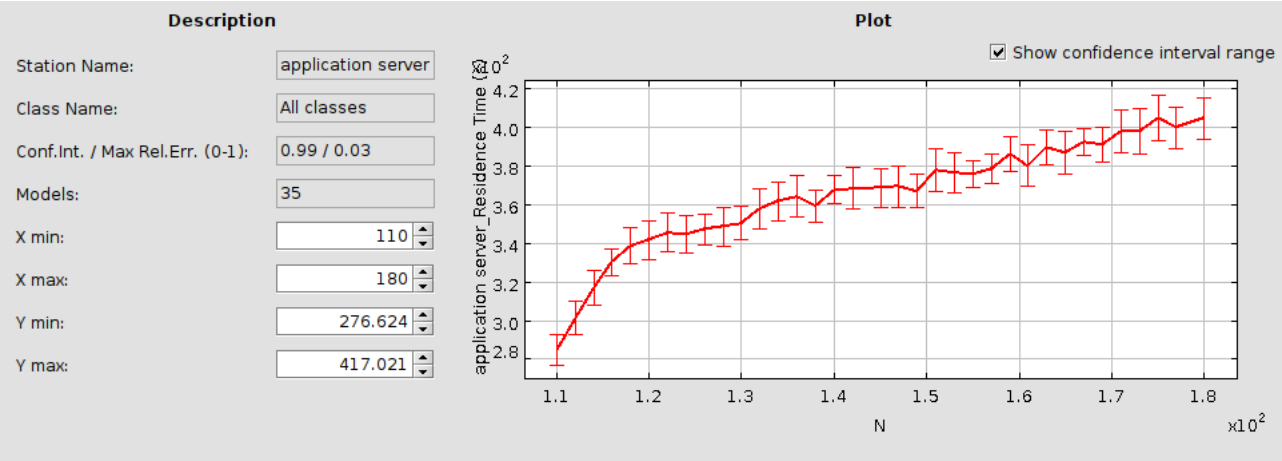

Figura 3.5: *residence time* dell' *application server*.

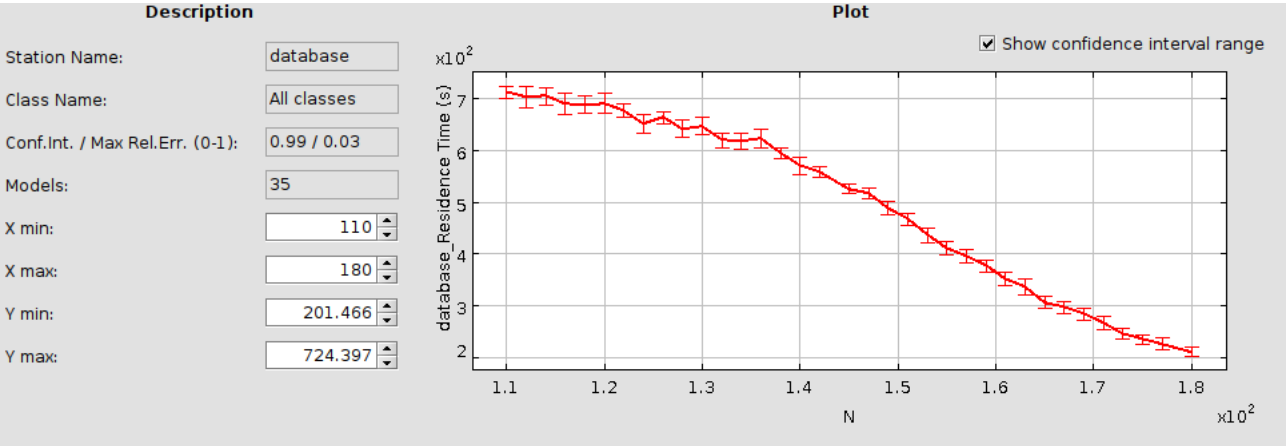

Figura 3.6: *residence time* del *database server*.

Quelli relativi al *web server* e all'*application server* mostrano un incremento del *residence time* e del *queue time* all'aumentare di N_B. Un comportamento opposto risulta per il *database*, i relativi grafici mostrano una diminuizione all' incrementare di N_B. Ancora una

volta, questi comportamenti possono trovare una giustificazione nel *throughput* limitato dalla stazione *application server*.

3.3.3 *Response time* del sistema

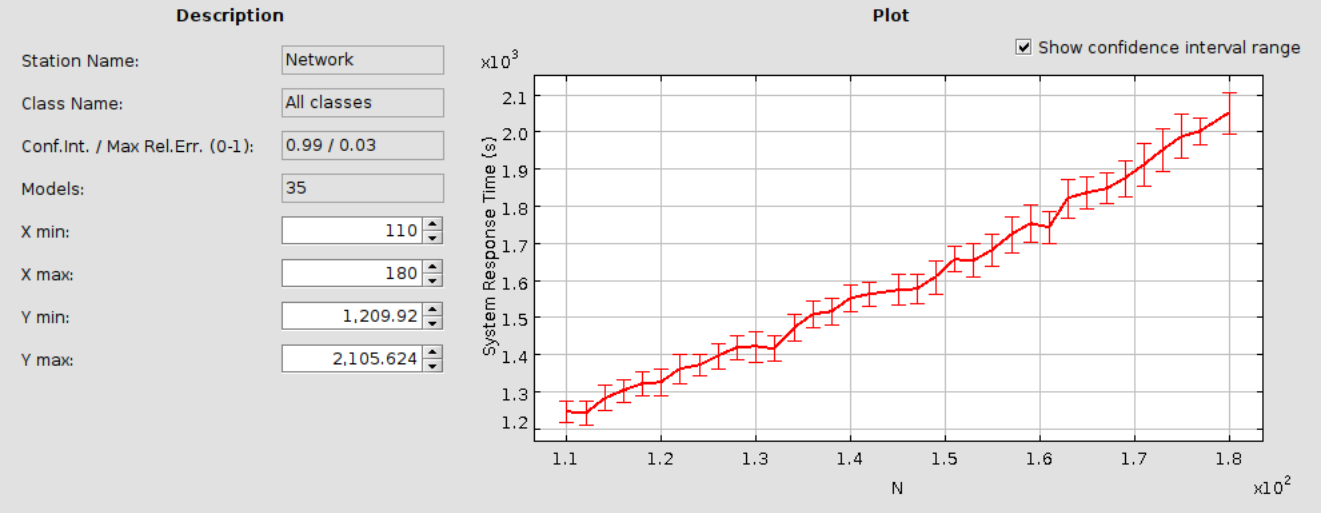

Figura 3.7: *Response time* medio del sistema.

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) [2], il valore del *Response time* del sistema è di circa 2 *sec* come atteso dalle precedenti simulazioni [??].

3.3.4 Soluzione 2

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

[2] sull'asse delle ascisse è riportato il numero totale dei *jobs* nel sistema: $\mathbf{N_U} + \mathbf{N_S} + \mathbf{N_B} = 10 + 50 + 100 = 160$

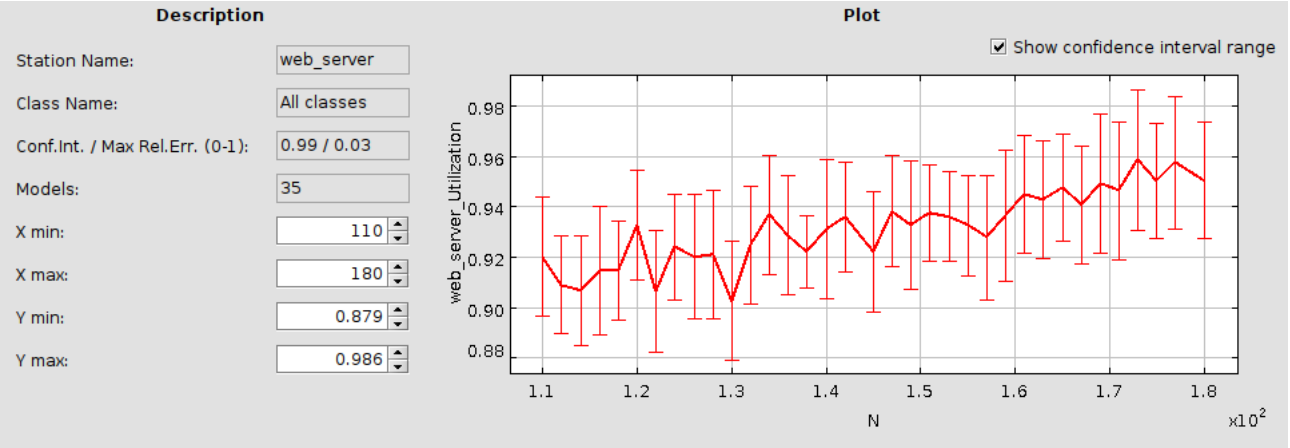

Figura 3.8: Utilizzo della stazione *web server* al variare di N_B, soluzione 2.

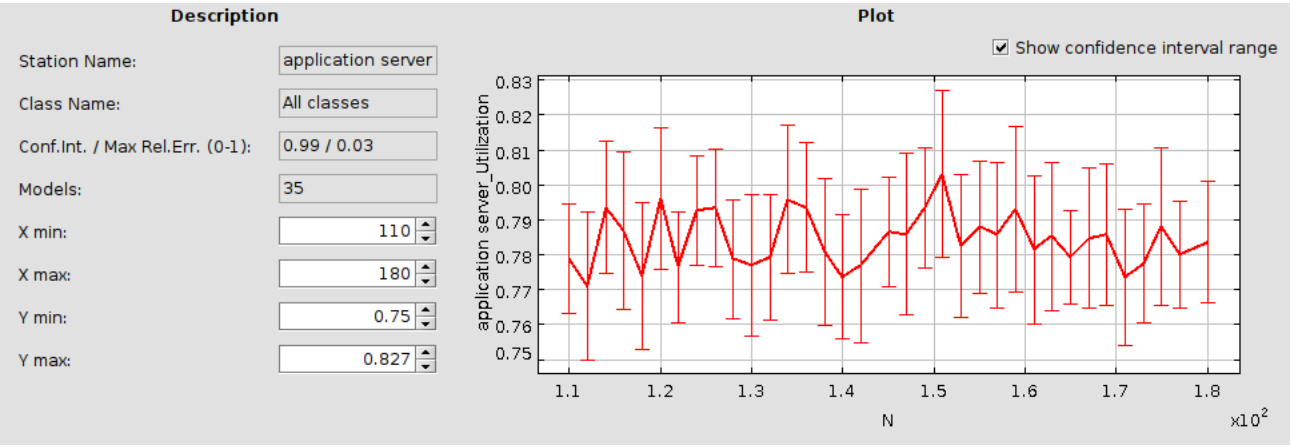

Figura 3.9: Utilizzo della stazione *application server* al variare di N_B, soluzione 2.

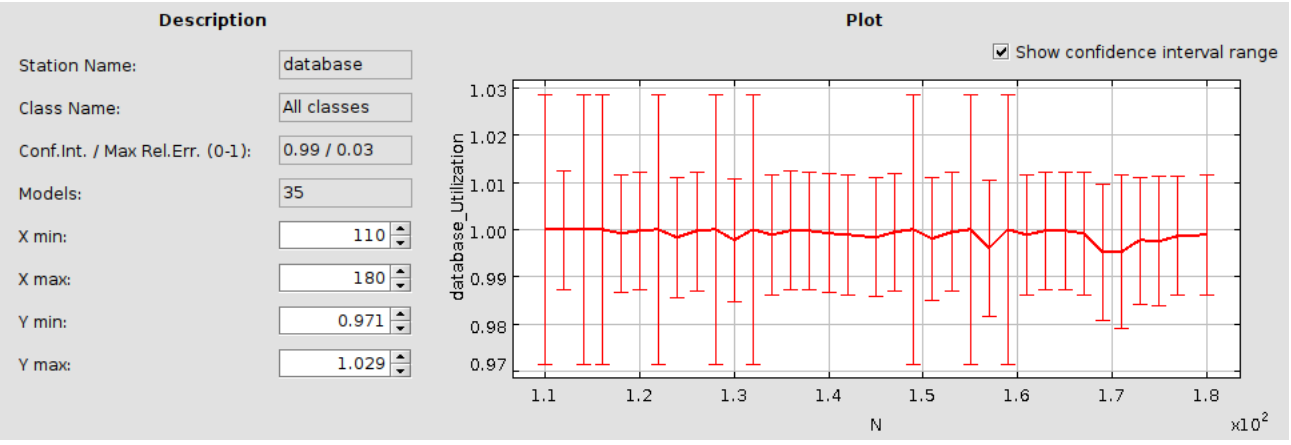

Figura 3.10: Utilizzo della stazione *database* al variare di N_B, soluzione 2.

Come si può notare nel grafico dell'utilizzo del *database*, esso si attesta su valori in un intorno di 1. Dunque, i risultati evidenziano come in questo caso il nuovo collo di bottiglia

del sistema diventi la stazione *database*, differentemente a ciò che accadeva nella soluzione 1 [**??**].

Capitolo 4

Conclusioni

L'analisi condotta ha individuato due soluzioni che raggiungono l'obiettivo fissito. La prima [**??**] è una soluzione che ha una più bassa soglia di errore mentre la seconda [**??**] da maggiori garanzie in termini di affidabilità. Questo vantaggio va ad incidere sui costi di tale soluzione richiedendo una replica in più dell' *application server*.

Capitolo 5

References

- Modello e analisi realizzati con *JSIMgraph*;
- Slide del corso *Computer Systems Performance Evaluation* [090945] - Marco Gribaudo;
- *Quantitative System Evaluation with Java Modeling Tools (Tutorial Paper)*.