RightSizing-SpikeServer

Contents

1 Caso di Studio

Il sistema oggetto di studio è un'architettura di data center per un Internet Service Provider, progettata per gestire dinamicamente le fluttuazioni di carico e garantire la Quality of Service (QoS) ottimizzando al contempo l'uso delle risorse.

Il problema principale affrontato è il "right-sizing", ovvero come evitare sia il sovradimensionamento (spreco di risorse) sia il sottodimensionamento (violazione degli SLA e degrado delle prestazioni), specialmente in presenza di fluttuazioni di carico a breve e lungo termine.

L'architettura proposta, come descritto nel caso di studio 6.2 del libro di testo "Performance Engineer", si basa su un livello di scaling verticale che gestisce i picchi di carico improvvisi e di breve durata. Questo livello introduce uno Spike Server dedicato. Un Load Controller monitora un indicatore di picco (Spike Indicator, SI), definito come il numero di richieste concorrenti totali presenti su un Web Server (tutte le richieste vengono gestite con un scheduler processor sharing).

Il comportamento del sistema seguirebbe quanto descritto:

- Quando l'indicatore SI supera una soglia di allarme SI_max, le nuove richieste in arrivo non vengono più inviate al Web Server congestionato, ma vengono reindirizzate allo Spike Server.
- Quando il carico sul Web Server diminuisce e SI scende al di sotto della soglia, il routing delle richieste torna alla normalità.

2 Obiettivi dello studio

Lo studio si pone l'obiettivo di analizzare e validare l'efficacia del modello di autoscaling basato su spike server attraverso la simulazione. Gli obiettivi specifici sono:

- Determinare il valore di SI_{max} più alto possibile (provando vari valori di SI) che mantenga comunque il tempo di risposta medio $E[R] \leq 8$ secondi, con un tasso di arrivo iperesponenziale con media di 6.66 req/s (400 req/min).
- Analizzare come varia il tempo di risposta medio al variare del coefficiente di variazione degli arrivi iperesponenziali (fluttuazioni a breve termine) per carichi di lavoro crescenti (da 1 req/s a 12 req/s).
- Verificare come cambia il contesto (sempre sotto carichi crescenti e al variare di SI_max) se lo spike server ha un tasso di servizio doppio rispetto al webserver principale, invece che uguale come nel caso base.

3 Modello concettuale

Il modello descritto può essere schematizzato come in figura 1.

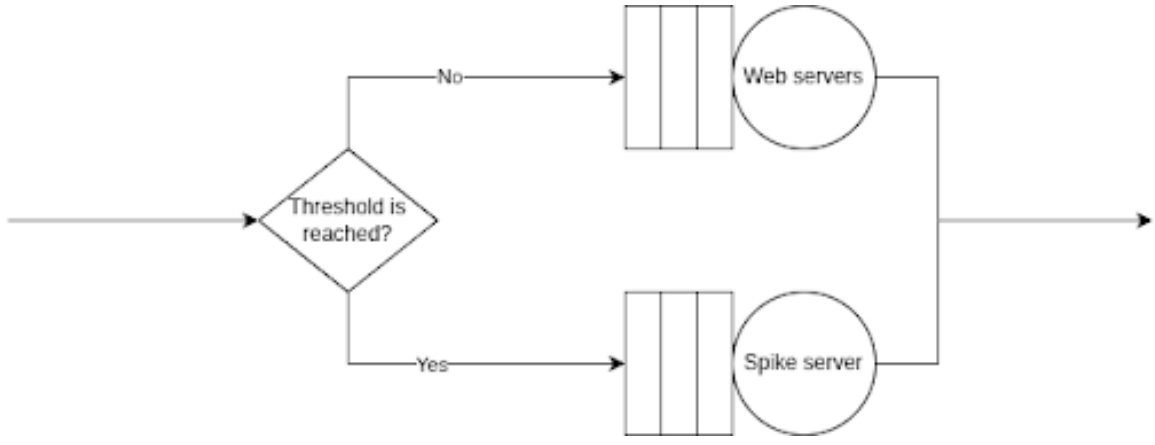


Figure 1: Modello concettuale del sistema di autoscaling gerarchico con Spike Server

I job arrivano e a seconda del livello di pienezza del webserver vengono instradati o al webserver principale o allo spike server. Nonostante il sistema possa sembrare troppo semplice per una analisi simulativa, il fatto che il routing non sia probalistico lo rende molto complesso da analizzare in modo analitico. il routing dei job, dipende strettamente dallo stato del webserver nel momento del routing.

3.1 Tempi di servizio

Nel caso di studio affrontato nel libro viene utilizzata una distribuzione iperesponenziale nel tassi di servizio per modellare il fatto che ad un server arrivano job di dimensione molto variabile. Tale comportamento potrebbe essere modellato anche utilizzando delle size dei job differenti anziché agire sul tasso di servizio. Tuttavia ho deciso di attenermi al testo originale e utilizzare anche io dei tassi di servizio iperesponenziali per modellare questo comportamento per poter confrontare meglio i risultati ottenuti.

3.2 Spike server

Lo Spike Server avrà la stessa potenza rispetto al webserver nei primi 2 obiettivi. Nell'obiettivo 3, invece, avrà un tasso di servizio doppio rispetto a quello del webserver principale, in modo da poter smaltire rapidamente i picchi di carico. Per il calcolo di SI_{max} ottimale è importante sottolineare che lo spike server è bene usarlo il meno possibile, in quanto è una risorsa più costosa rispetto al webserver. Quindi anziché cercare di minimizzare il tempo di risposta medio del sistema, si cerca di rispettare lo SLA (8 secondi) utilizzando il meno possibile lo spike server.

3.3 Web Server

Il web server sarà solamente uno, per semplicità di modellazione. Per questo motivo non vengono gestite le fluttuazioni a lungo termine, in quanto queste richiederebbero un sistema con più webserver per utilizzare una scaling orizzontale.

4 Modello delle specifiche

Queste variabili matematiche, sono la rappresentazione dello stato del sistema:

- $SI(t) \in [0, SI_{max})$: Spike indicator, indica il numero di job presenti nel web server al tempo t.
- $n_{spike}(t) \in [0, \infty)$: numero di job presenti nello spike server al tempo t.

per cui lo stato del sistema è definito come la tupla $S(t) = (SI(t), n_{spike}(t))$.

4.1 Componenti fisici del sistema:

- Web server principale: server principale che gestisce le richieste in arrivo.
- Spike server: server secondario che viene attivato quando il carico sul web server principale supera una certa soglia (SI_{max}).

Componenti logici del sistema:

- Load Controller: componente che monitora l'indicatore di picco (Spike Indicator, SI) e decide il routing delle richieste in base al valore di SI e alla soglia SI_{max}.

4.2 Specifiche del carico di lavoro:

- Arrivi:
  - Valore base: Processo di arrivo iperesponenziale con cv = 4 e media 0.15 secondi (6.66 req/s)
  - Valore stress: Processo di arrivo iperesponenziale con carico variabile da 1 req/s a 12 req/s

- Servizio: Distribuzione iperesponenziale con cv = 4 e media 0.16 secondi (web server e spike server), media 0.08 secondi (spike server nell'obiettivo 3)
- Scheduling: Processor Sharing

Vale la pena notare che, nel caso base del webserver, siccome $\rho > 1$, il sistema è instabile e senza spike server il tempo di risposta divergerebbe a infinito.

4.3 Logica di controllo:

- Arrivi nuovi job:
  - Se $SI < SI_{max}$: invia al webserver principale
  - Se $SI \geq SI_{max}$: invia allo spike server
- Completamento job:
  - Decrementa il contatore SI se il job era nel webserver principale
  - Decrementa il contatore n_spike se il job era nello spike server

Nella logica originale del caso di studio, il valore di SI parte da SI_{max} e viene decrementato ad ogni completamento di un job. La logica da me adottata, nonostante sia inversa, è perfettamente equivalente.

4.4 Metrica di valutazione delle prestazioni:

Lo SLA da rispettare è un tempo di risposta medio $E[R] \leq 8$ secondi.

4.5 Gestione della fine della simulazione:

Dato che il sistema viene studiato a regime, il sistema viene simulato per un tempo fisso di 5000 secondi, scartando i primi 500 secondi di fase transitoria. Inoltre una volta raggiunto il tempo di stop, vengono completati tutti i job ancora in servizio, fermando gli arrivi di nuovi job allo scadere del tempo di stop.

5 Modello Computazionale

Il sistema di simulazione è implementato in Python e segue un modello ad eventi discreti. Sono state utilizzate alcuni file della libreria di Steve Park & Dave Geyer, tradotti in python da Philip Steele. In particolare sono state utilizzate le librerie per la generazione di numeri casuali per il multistream (rngs.py) e per le distribuzioni (rvgs.py).

5.1 Stato del sistema

Le variabili di programmazione che rappresentano lo stato del sistema sono:

- web_jobs : list: è una lista che contiene i job in fase di processamento nel web server, la sua lunghezza rappresenta l'indicatore SI
- spike_jobs : list: è una lista che contiene i job in fase di processamento nello spike server.

5.2 Descrizione del job

Ogni job è rappresentato da una classe Job:

```
class Job:
    def __init__(self,
                 arrival_time,
                 service_demand,
                 is_spike=False):
        self.arrival_time = arrival_time
        self.service_demand = service_demand
        self.remaining_work = service_demand
        self.is_spike = is_spike
```

Dove:

- arrival_time: tempo di arrivo del job
- service_demand: tempo di servizio totale richiesto dal job
- remaining_work: lavoro rimanente da completare (utile per lo scheduling processor sharing)
- is_spike: booleano che indica se il job è stato assegnato allo spike server

5.3 Identificazione degli eventi

- arrivo di un nuovo job
- completamento di un job nel web server
- completamento di un job nello spike server

5.4 Logica di controllo e routing

Ogni volta che arriva un nuovo job:

- Se len(web_jobs) $\leq SI_{max}$, il job viene aggiunto a web_jobs.
- Se len(web_jobs) $> SI_{max}$, il job viene aggiunto a spike_jobs.

Ogni volta che job completa rimuove dalla lista corrispondente il job (web_jobs o spike_jobs).

5.5 Configurazione dei parametri delle distribuzioni

- Stream 0: Inter-arrivi (Iperesponenziale, media 0.15s, cv=4).
- Stream 1: Servizio Web Server (Iperesponenziale, media 0.16s, cv=4).
- Stream 2: Servizio Spike Server (Iperesponenziale, media 0.08s, cv=4).

La distribuzione iperesponenziale è stata implementata utilizzando l'esponenziale e l'uniforme della libreria rvgs.py. Con l'uniforme calcoliamo la probabilità di scegliere una delle due esponenziali, e con l'esponenziale calcoliamo il servizio vero e proprio (Figura 2).

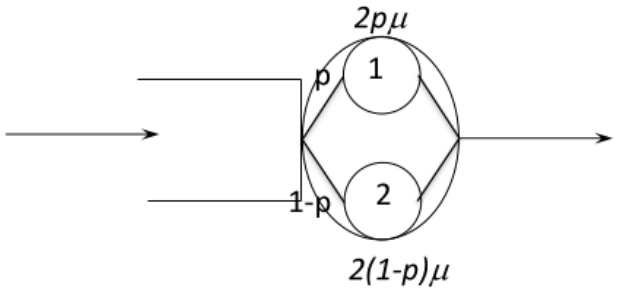


Figure 2: Generazione di una variabile casuale iperesponenziale

In particolare date in input la media e il coefficiente di variazione cv. Sapendo che $g(p) = cv^2$, possiamo calcolare p_1, p_2, m_1 e m_2 come:

$$c = \frac{cv^2 - 1}{cv^2 + 1}$$

$$p_1 = \frac{1 + c}{2}$$

$$p_2 = 1 - p_1$$

$$m_1 = \frac{mean}{2 \cdot p_1}$$

$$m_2 = \frac{mean}{2 \cdot p_2}$$

e quindi utilizzare un semplice if per scegliere quale esponenziale utilizzare in base ad un campione uniforme:

```
u = Uniform(0,1)
if u < p1:
    return Exponential(m1)
else:
    return Exponential(m2)
```

Da notare che per ogni campionamento dell'esponenziale si utilizzano, quindi, due estrazioni dallo stream del generatore di numeri casuali.

5.6 Struttura del simulatore

Il simulatore è strutturato come una classe Simulator che gestisce l'inizializzazione, l'esecuzione e la raccolta delle metriche della simulazione. La classe ha una serie di costanti di configurazione:

```
START        = 0.0
BIAS_PHASE   = 500.0
STOP         = 5000.0
INFINITY     = 1e15
SEED         = 8
REPLICAS     = 100
N_PROCESSES  = 40
```

Dove:

- START: tempo di inizio della simulazione
- BIAS_PHASE: durata della fase transitoria (500 secondi)
- STOP: tempo di fine della simulazione (5000 secondi)

- INFINITY: un valore molto grande usato per inizializzare i tempi di evento (deve essere molto più grande di STOP)
- SEED: seme per il generatore di numeri casuali, che è stato utilizzato insieme alla libreria rngs.py per tutti i generatori di numeri casuali
- REPLICAS: numero di repliche della simulazione (100 repliche)
- N_PROCESSES: numero di processi paralleli per l'esecuzione delle repliche (12 processi)

5.6.1 Parallelismo

Per velocizzare l'esecuzione delle repliche della simulazione, è stata implementata una logica di multiprocessing per sfruttare tutti i core della mia CPU disponibili. Tuttavia per farlo ho dovuto prendere alcune accortezze che permettessero di non minare nè l'indipendenza delle repliche nè la riproducibilità dei risultati. In particolare:

- ad ogni processo vengono assegnati 3 stream dei 256 disponibili, in modo che ogni processo abbia il proprio spazio di numeri casuali indipendenti dagli altri processi, ma in maniera tale che potessero procedere in parallelo senza che la non deterministicità del parallelismo influisse sull'ordine di estrazione dei numeri casuali.
- Per il numero di repliche introdotte, ogni processo non si avvicinava nemmeno lontanamente al limite dello stream assegnato.

  $$MODULUS/STREAM \approx 8388607$$

  Ho comunque implementato un meccanismo in cui una volta sforato il limite dello stream, gli venivano assegnati 3 nuovi stream. Ed essendo il numero di processi 12, e ogni processo usa 3 stream, questo è tranquillamente possibile senza sforare il limite dei 256 stream e senza ulteriori meccanismi di controllo.
- Sempre per garantire la riproducibilità dei risultati, ad ogni processo vengono assegnate sempre lo stesso numero di repliche da eseguire, in modo tale che le ultime repliche non vengano eseguite da processi diversi (e quindi strem diversi) in base all'assegnazione dinamica di essi.

5.7 Raccolta delle metriche

Per raccogliere le metriche è stata utilizzata una classe Track:

```
class Track:
    def __init__(self):
        self.area_node_web = 0.0
        self.area_node_spike = 0.0

        self.area_busy_web = 0.0
        self.area_busy_spike = 0.0

        self.completed_web = 0
        self.completed_spike = 0

        self.scaling_actions = 0
```

Dove:

- area_node_web: area sotto la curva del numero di job nel web server

  $$\int_{BIAS\_PHASE}^{STOP} N_{web}(t)dt$$

- area_node_spike: area sotto la curva del numero di job nello spike server

$$\int_{BIAS\_PHASE}^{STOP} N_{spike}(t)dt$$

- area_busy_web: area sotto la curva del tempo di utilizzo del web server

$$\int_{BIAS\_PHASE}^{STOP} U_{web}(t)dt$$

- area_busy_spike: area sotto la curva del tempo di utilizzo dello spike server

$$\int_{BIAS\_PHASE}^{STOP} U_{spike}(t)dt$$

- completed_web: numero di job completati nel web server
- completed_spike: numero di job completati nello spike server
- scaling_actions: numero di azioni di scaling (numero di volte in cui lo spike server è stato attivato)

Gli integrali sono calcolati ad ogni evento in maniera incrementale durante la simulazione.

Le metriche finali raccolte sono state:

- **web_response_time**: tempo di risposta medio nel web server calcolato come l'integrale del numero di job completati nel web server diviso per il numero di job completati nel web server

$$\frac{area\_node\_web}{completed\_web}$$

- **spike_response_time**: tempo di risposta medio nello spike server

$$\frac{area\_node\_spike}{completed\_spike}$$

- **total_response_time**: tempo di risposta medio totale

$$\frac{area\_node\_web + area\_node\_spike}{completed\_web + completed\_spike}$$

- **utilization_web**: utilizzo medio del web server

$$\frac{area\_busy\_web}{STOP - BIAS\_PHASE}$$

- **utilization_spike**: utilizzo medio dello spike server

$$\frac{area\_busy\_spike}{STOP - BIAS\_PHASE}$$

- **throughput_web**: throughput medio del web server

$$\frac{completed\_web}{STOP - BIAS\_PHASE}$$

- **throughput_spike**: throughput medio dello spike server

$$\frac{completed\_spike}{STOP - BIAS\_PHASE}$$

- **throughput_total**: throughput medio totale

$$\frac{completed\_web + completed\_spike}{STOP - BIAS\_PHASE}$$

- **scaling_actions**: numero di azioni di scaling

Per quanto riguarda le metriche finali su tutte le repliche, è stato utilizzato l'algoritmo di Welford per il calcolo delle medie e varianze e poi degli intervalli di confidenza al 95%.

5.8 Scheduler

Siccome lo scheduler è di tipo processor sharing, il tempo prima del prossimo evento da schedulare è calcolato come:

$$min\{next\_arrival, time\_to\_complete\_web\_job, time\_to\_complete\_spike\_job\}$$

Dove:

- next_arrival: tempo rimanente al prossimo arrivo
- time_to_complete_web_job: tempo rimanente al completamento del job con il minor lavoro rimanente nel web server
- time_to_complete_spike_job: tempo rimanente al completamento del job con il minor lavoro rimanente nello spike server

Per calcolare i tempi rimanenti al completamento dei job, non basta trovare il job con il minor lavoro rimanente, perché essendo in processor sharing il tempo di completamento dipende anche dal numero di job in servizio. Quindi il tempo di completamento del job con il minor lavoro rimanente è calcolato come:

$$time\_to\_complete\_web\_job = min\_remaining\_work \times len(web\_jobs)$$

$$time\_to\_complete\_spike\_job = min\_remaining\_work \times len(spike\_jobs)$$

Una volta trovato il tempo del prossimo evento, si avanza il tempo di simulazione di tale intervallo e si aggiorna il lavoro rimanente di tutti i job in servizio sottraendo il lavoro svolto in tale intervallo:

$$j.remaining\_work = j.remaining\_work - \frac{time\_advance}{len(web\_jobs)} \forall j \in web\_jobs$$

$$j.remaining\_work = j.remaining\_work - \frac{time\_advance}{len(spike\_jobs)} \forall j \in spike\_jobs$$

6 Verifica

Ci sono varie verifiche che si possono fare andando a controllare se tutta una serie di caratteristiche volute si riflettano veramente nel modello simulativo costruito e quindi nei risultati ottenuti. In particolare, andrò ad effettuare le seguenti verifiche:

- Verifica del corretto funzionamento di routing basato su soglia si SI_{max}.
  - Controllo che impostando $SI_{max} = 0$ venga utilizzato solo lo spike server.
  - Controllo che impostando $SI_{max} = \infty$ non venga mai utilizzato lo spike server.
- Verifica delle metriche prestazionali:
  - Verifico che aumentando SI_{max} l'utilizzazione del web server aumenta
  - Verifico che il throughput sia pari al tasso di arrivo quando il sistema è stabile e uguale al tasso di servizio quando il sistema è sovraccarico.
- Verifica della distribuzione
  - Verifica che media e cv della distribuzione iperesponenziale corrispondano ai valori impostati.

6.1 Verifica del routing basato su soglia SI_{max}

6.1.1 Verifica con $SI_{max} = 0$

Impostando $SI_{max} = 0$ si può verificare che tutti i job vengano instradati allo spike server. Infatti, come mostrato in figura 3, l'utilizzazione del web server è nulla, mentre tutto il carico viene gestito dallo spike server.

```
Simulazione completata con i seguenti parametri:
  SI_max: 0
  Arrival Mean: 0.15
  Web Mean: 0.16
  Spike Mean: 0.16
  Coefficiente di Variazione: 4.0
Statistiche raccolte con intervallo di confidenza al 95%:
spike_response_time      : 12.9032 +/- 1.1099 (Var: 32.069075)
total_response_time      : 12.9032 +/- 1.1099 (Var: 32.069075)
utilization_web          : 0.0000 +/- 0.0000 (Var: 0.000000)
utilization_spike        : 0.9910 +/- 0.0028 (Var: 0.000205)
throughput_web           : 0.0000 +/- 0.0000 (Var: 0.000000)
throughput_spike         : 6.5423 +/- 0.0339 (Var: 0.029877)
throughput_total         : 6.5423 +/- 0.0339 (Var: 0.029877)
scaling_actions          : 44.9200 +/- 7.3205 (Var: 1394.983434)
```

Figure 3: Verifica routing con $SI_{max} = 0$

6.1.2 Verifica con $SI_{max} = \infty$

Impostando $SI_{max} = \infty$ si può verificare che nessun job venga instradato allo spike server. Infatti, come mostrato in figura 4, l'utilizzazione dello spike server è nulla, mentre tutto il carico viene gestito dal web server.

```
Simulazione completata con i seguenti parametri:
  SI_max: 1000000000000000.0
  Arrival Mean: 0.15
  Web Mean: 0.16
  Spike Mean: 0.16
  Coefficiente di Variazione: 4.0
Statistiche raccolte con intervallo di confidenza al 95%:
web_response_time        : 13.1808 +/- 1.0418 (Var: 28.251530)
total_response_time      : 13.1808 +/- 1.0418 (Var: 28.251530)
utilization_web          : 0.9914 +/- 0.0028 (Var: 0.000202)
utilization_spike        : 0.0000 +/- 0.0000 (Var: 0.000000)
throughput_web           : 6.5389 +/- 0.0351 (Var: 0.032065)
throughput_spike         : 0.0000 +/- 0.0000 (Var: 0.000000)
throughput_total         : 6.5389 +/- 0.0351 (Var: 0.032065)
scaling_actions          : 0.0000 +/- 0.0000 (Var: 0.000000)
```

Figure 4: Verifica routing con $SI_{max} = \infty$

6.2 Verifica delle metriche prestazionali

6.2.1 Verifica dell'utilizzazione al variare di SI_{max}

Impostando un carico di lavoro fisso e variando il valore di SI_{max}, si può verificare che l'utilizzazione del web server aumenti al crescere di SI_{max}, come mostrato in figura 5.

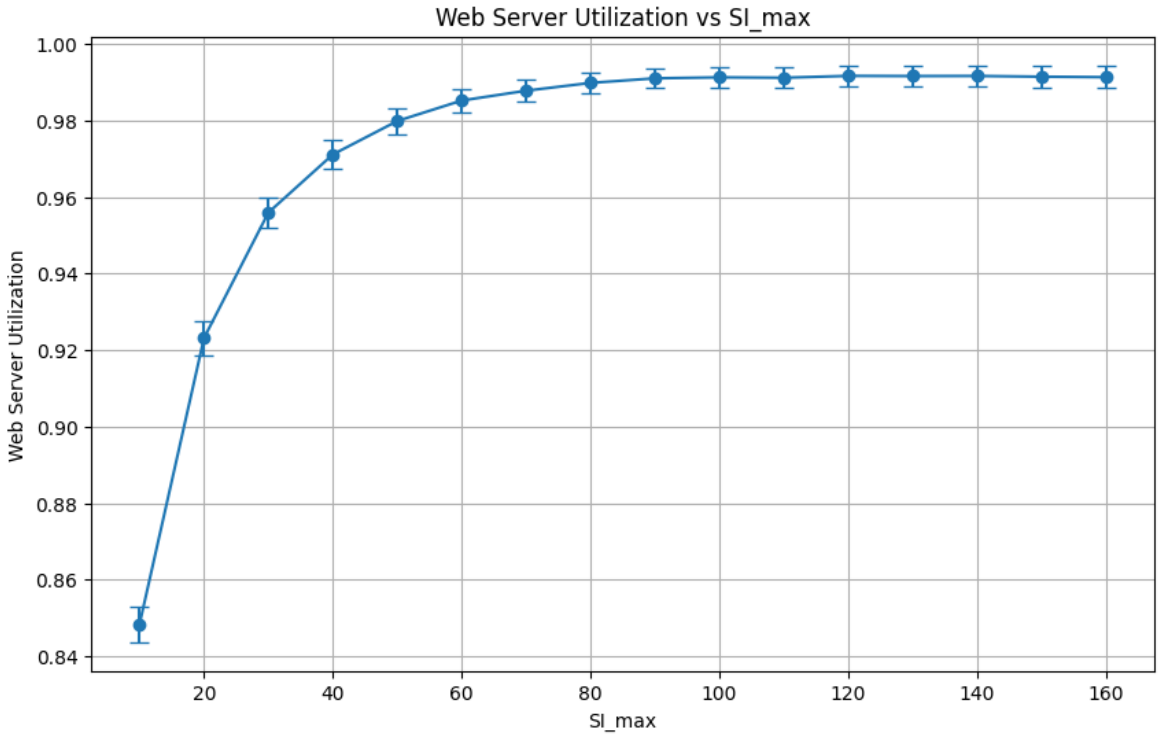


Figure 5: Verifica dell'utilizzazione del web server al variare di SI_{max}

Table 1: Verifica dell'utilizzazione del web server al variare di SI_{max}

SI_{max}	U_{web}	U_{web} CI95
10	0.8197	0.0066
20	0.9058	0.0064
30	0.9427	0.0061
40	0.9619	0.0057
50	0.9728	0.0051
60	0.9788	0.0047
70	0.9818	0.0046
80	0.9832	0.0044
90	0.9838	0.0043
100	0.9844	0.0042
110	0.9847	0.0041
120	0.9847	0.0041
130	0.9849	0.0040
140	0.9851	0.0040
150	0.9850	0.0040
160	0.9848	0.0040

6.2.2 Verifica del throughput

Variando il carico di lavoro da 1 req/s a 12 req/s e impostando $SI_{max} = \infty$, in modo da misurare solo il web server, si può verificare che il throughput del sistema non superi mai il suo limite teorico, che nel caso considerato è uguale a $\min(\mu_{web}, \lambda_{web})$. Infatti, come evidenziato in Figura 6, il throughput misurato $X_{measured} = N_{completed}/T_{sim}$ non supera il limite teorico e si mantiene sempre al di sotto di esso.

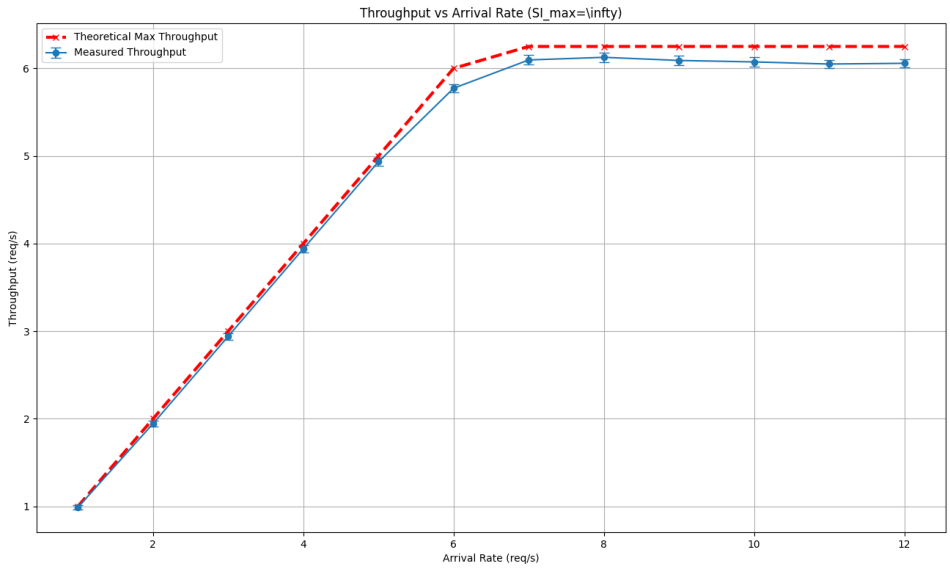


Figure 6: Throughput misurato che supera il valore teorico

Table 2: Verifica del Throughput

λ	$X_{measured}$	$X_{measured}$ CI95	$X_{theoretical}$
1	0.9889	0.0214	1.0000
2	1.9442	0.0304	2.0000
3	2.9411	0.0367	3.0000
4	3.9406	0.0442	4.0000
5	4.9337	0.0469	5.0000
6	5.7728	0.0475	6.0000
7	6.0958	0.0552	6.2500
8	6.1260	0.0553	6.2500
9	6.0902	0.0522	6.2500
10	6.0736	0.0512	6.2500
11	6.0492	0.0474	6.2500
12	6.0581	0.0463	6.2500

6.3 Verifica della distribuzione iperesponenziale

Per verificare che la distribuzione iperesponenziale implementata nel simulatore corrisponda a quella voluta, sono state effettuate delle misurazioni della media e del coefficiente di variazione dei tempi di servizio generati, confrontandoli con i valori teorici impostati. In particolare sono state effettuate sempre 100 repliche di simulazione con $SI_{max} = 80$. Effettivamente, come mostrato in figura 7, i valori teorici rientrano perfettamente negli intervalli di confidenza al 95% delle misurazioni effettuate.

```
meas_arrival_mean         : 0.1499 +/- 0.0007 (Var: 0.000014)
meas_arrival_cv           : 3.9961 +/- 0.0168 (Var: 0.007359)
meas_web_service_mean     : 0.1606 +/- 0.0006 (Var: 0.000011)
meas_web_service_cv       : 4.0032 +/- 0.0177 (Var: 0.008145)
meas_spike_service_mean   : 0.1592 +/- 0.0030 (Var: 0.000230)
meas_spike_service_cv     : 3.9284 +/- 0.0819 (Var: 0.174522)
```

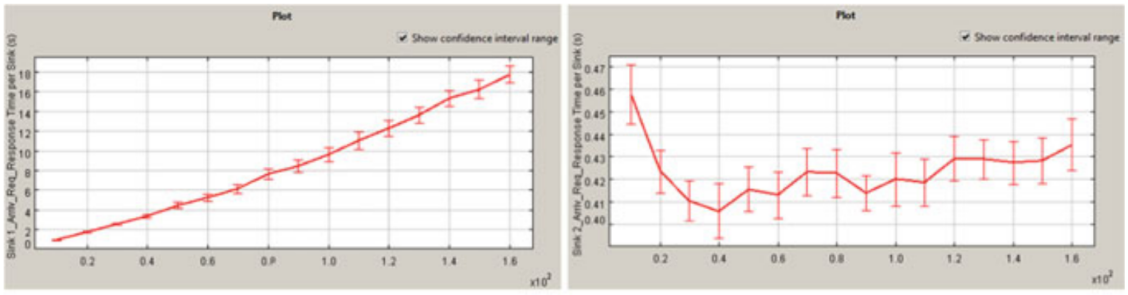
Figure 7: Verifica della distribuzione iperesponenziale dei tempi di servizio

7 Validazione

Dato che in questo studio non sono disponibili dati reali con cui confrontare i risultati ottenuti dalla simulazione, la validazione del modello simulativo confronterà i risultati ottenuti con quelli ottenuti dallo studio di riferimento nella prossima sezione del report.

7.1 Confronto tempi di risposta

Nello studio di riferimento vengono effettuati dei test sui tempi di risposta sul webserver e sullo spike server.



(a) Response times of Web Server1 (b) Response times of Spike Server

Figure 8: Tempi di risposta del webserver e dello spike server in funzione del carico nel caso di studio di Serazzi

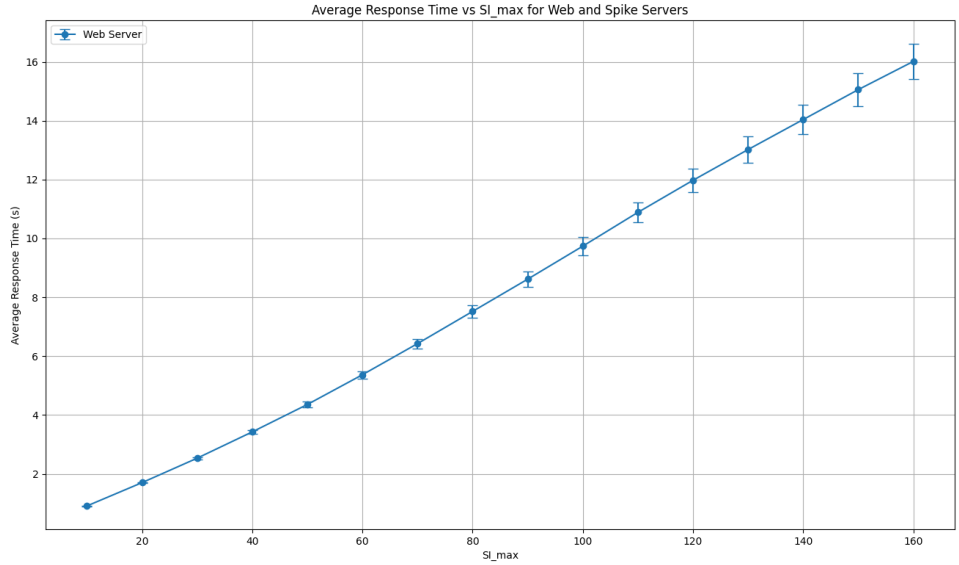


Figure 9: Tempi di risposta del webserver in funzione del SI_max ottenuti dalla simulazione.

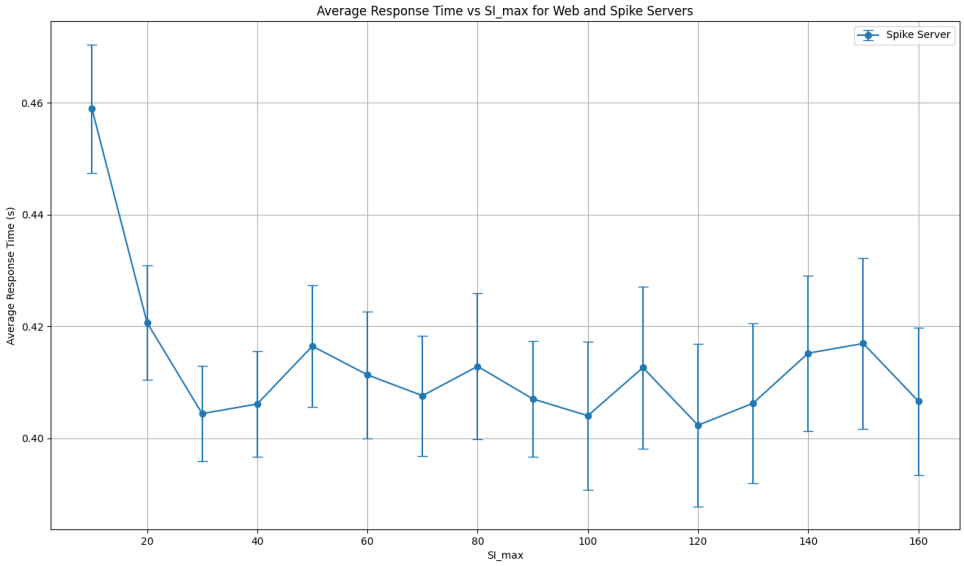


Figure 10: Tempi di risposta dello spike server in funzione del SI_max ottenuti dalla simulazione.

Table 3: Validazione tempi di risposta del webserver

SI_{max}	$E[T_s]_{web}$	$E[T_s]_{web}$ CI95
10	0.9096	0.0075
20	1.7015	0.0197
30	2.5320	0.0375
40	3.4252	0.0605
50	4.3567	0.0912
60	5.3675	0.1252
70	6.4253	0.1667
80	7.5198	0.2103
90	8.6157	0.2536
100	9.7376	0.3011
110	10.8851	0.3462
120	11.9747	0.3967
130	13.0199	0.4450
140	14.0386	0.5004
150	15.0560	0.5574
160	16.0146	0.6099

Table 4: Validazione tempi di risposta dello spike server

SI_{max}	$E[T_s]_{spike}$	$E[T_s]_{spike}$ CI95
10	0.4589	0.0115
20	0.4207	0.0102
30	0.4044	0.0085
40	0.4061	0.0095
50	0.4165	0.0108
60	0.4113	0.0113
70	0.4076	0.0107
80	0.4129	0.0130
90	0.4070	0.0103
100	0.4040	0.0133
110	0.4126	0.0145
120	0.4023	0.0145
130	0.4063	0.0143
140	0.4152	0.0139
150	0.4169	0.0153
160	0.4066	0.0132

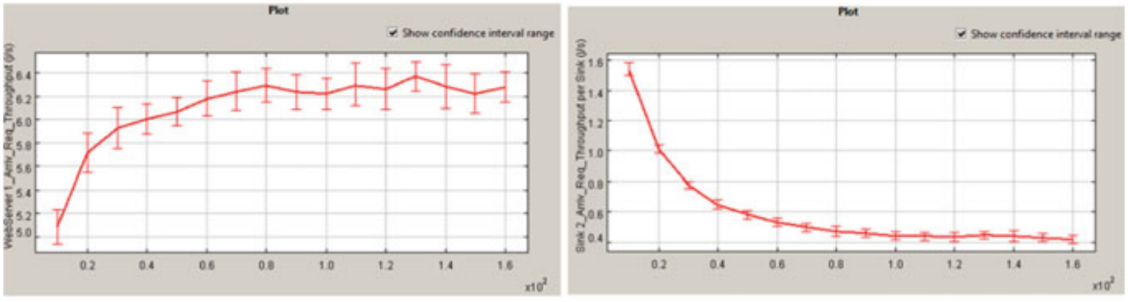
Controntandoli con i risultati ottenuti dalla mia simulazione, si nota come l'andamento sia essenzialmente lo stesso, tuttavia i valori dei tempi differiscono. Questo è probabilmente dovuto dal tempo di simulazione che Serazzi non esplicita direttamente e quindi probabilmente differente dai 5000 secondi utilizzati nelle mie simulazione. Tuttavia per non rendere troppo lunghi gli esperimenti, ho eseguito una simulazione con $SImax = 160$ e aumentato il tempo di simulazione a 20000 secondi, per dimostrare che il motivo della differenza nei tempi di risposta è dovuto proprio al tempo di simulazione.

```
  SI_max: 160
  Arrival Mean: 0.15
  Web Mean: 0.16
  Spike Mean: 0.16
  Coefficiente di Variazione: 4.0
Statistiche raccolte con intervallo di confidenza al 95%:
web_response_time        : 17.8339 +/- 0.2711 (Var: 1.913843)
spike_response_time      : 0.4290 +/- 0.0071 (Var: 0.001299)
total_response_time      : 16.6907 +/- 0.2125 (Var: 1.175528)
utilization_web          : 0.9991 +/- 0.0003 (Var: 0.000003)
utilization_spike        : 0.0688 +/- 0.0033 (Var: 0.000279)
throughput_web           : 6.1952 +/- 0.0146 (Var: 0.005573)
throughput_spike         : 0.4294 +/- 0.0198 (Var: 0.010167)
throughput_total         : 6.6247 +/- 0.0158 (Var: 0.006467)
scaling_actions          : 2282.1700 +/- 96.0887 (Var: 240343.314242)
```

Figure 11: Tempi di risposta del webserver con $SI_{max} = 160$ ottenuti dalla simulazione con 20000 secondi di simulazione.

7.2 Confronto Throughput

Nello studio di riferimento vengono effettuati dei test sul throughput del webserver.



(a) Throughput of Web Server1 (b) Throughput of Spike Server

Figure 12: Throughput del webserver e dello spike server in funzione del carico nel caso di studio di Serazzi

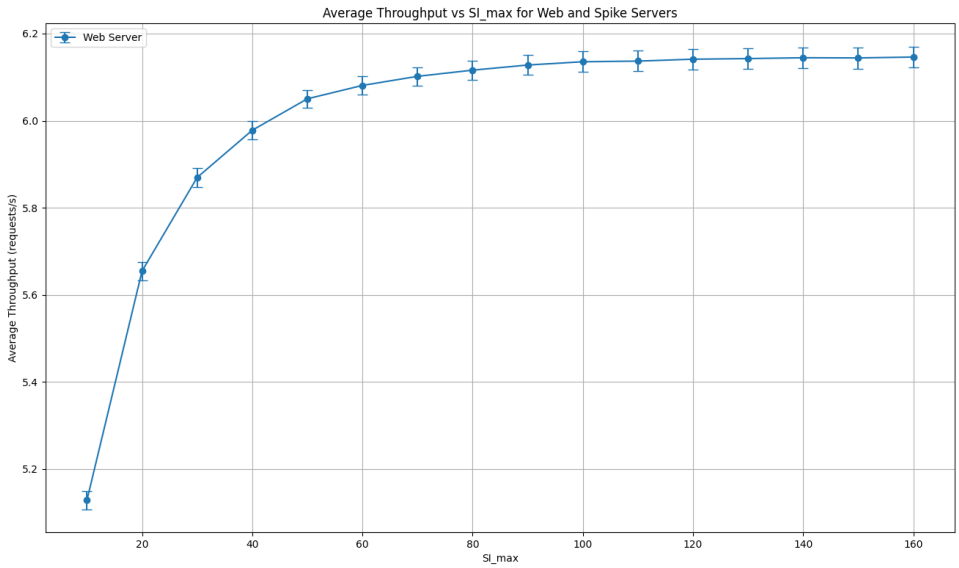


Figure 13: Throughput del webserver in funzione del SI_max ottenuti dalla simulazione.

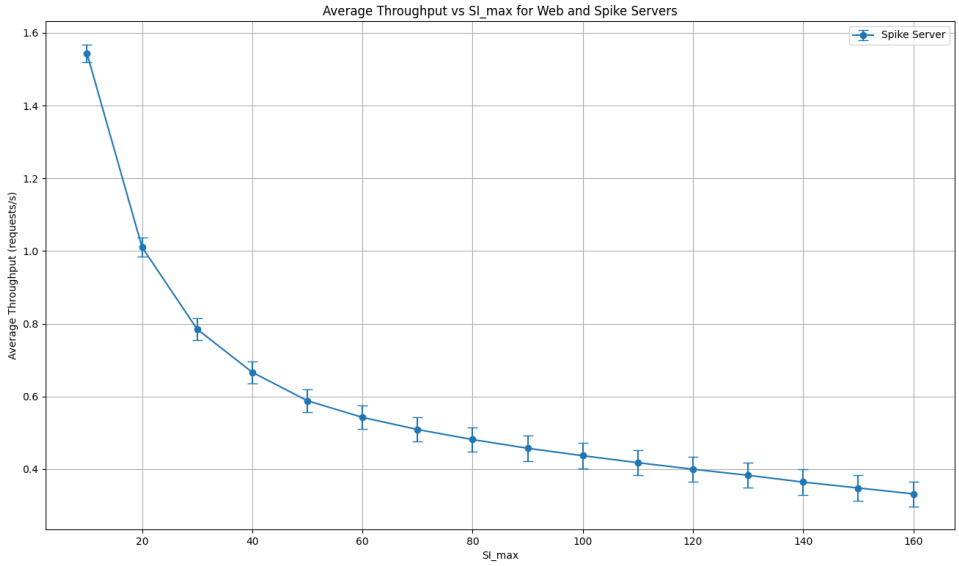


Figure 14: Throughput dello spike server in funzione del SI_max ottenuti dalla simulazione.

L'andamento e i valori qui sono essenzialmente gli stessi. Purtoppo serazzi non fornisce i valori numerici precisi su cui ha costruito i grafici per confrontarli direttamente.

8 Risultati

Gli esperimenti condotti mirano a raggiungere gli obiettivi delineati nella sezione 2.

8.1 Obiettivo 1

Come si può evincere dalla figura 15, il tempo di risposta medio $E[R]$ rimane al di sotto della soglia di 8 secondi per valori di SI_{max} fino a circa 90. Esattamente come suggerito nello studio di riferimento, il valore ottimale di SI_{max} che massimizza l'utilizzo del web server mantenendo il rispetto dello SLA si attesta intorno a 80-90. Considerando che con l'intervallo di confidenza con 90 si sfora, il valore più prudente da adottare per SI_{max} tra i due risulta essere 80.

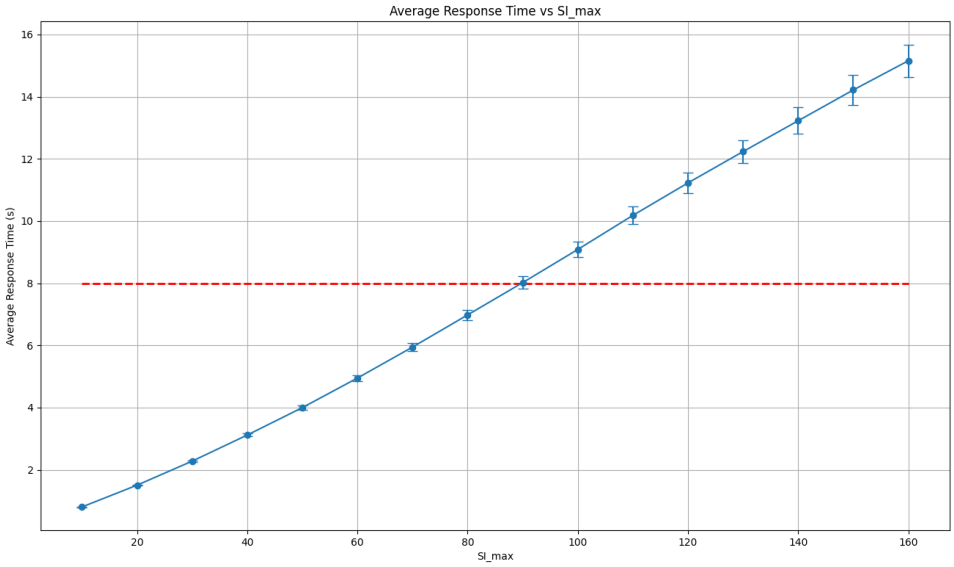


Figure 15: Tempo di risposta medio in funzione di SI_{max}

Table 5: Tempo di risposta medio in funzione di SI_{max}

SI_{max}	$E[T_s]$	$E[T_s]$ CI95
10	0.8053	0.0060
20	1.5063	0.0132
30	2.2786	0.0258
40	3.1180	0.0439
50	4.0000	0.0684
60	4.9507	0.0962
70	5.9467	0.1309
80	6.9813	0.1671
90	8.0204	0.2033
100	9.0866	0.2438
110	10.1823	0.2833
120	11.2269	0.3283
130	12.2347	0.3722
140	13.2254	0.4227
150	14.2151	0.4754
160	15.1546	0.5239

8.2 Obiettivo 2

Come mostrato in figura 16, accade lo stesso fenomeno descritto nello studio di riferimento: al crescere del tasso di arrivo, il tempo di risposta medio aumenta all'aumentare degli arrivi, fino a che non si arriva a superare la soglia SI_{max}. A questo punto, il tempo di risposta comincia a calare fino a che anche lo spike server si satura e il tempo di risposta ricomincia a salire per valori superiori a 11. Nelle tabelle sottostanti sono riportati i valori numerici però mostrando solo i valori di SI_{max} a multipli di 20 per la grande quantità di dati generata.

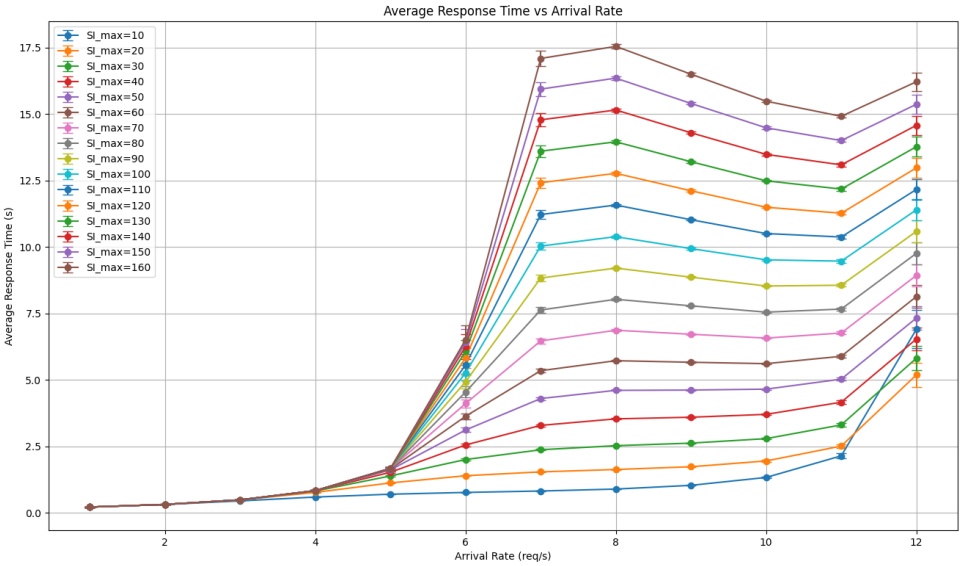


Figure 16: Tempo di risposta medio in funzione del tasso di arrivo e di SI_{max}

Table 6: Tempo di risposta medio in funzione del tasso di arrivo e di SI_{max}

SI max	λ	$E[T_s]$	$E[T_s]$ CI95
20	1	0.2195	0.0038
20	2	0.3176	0.0053
20	3	0.4876	0.0099
20	4	0.7702	0.0126
20	5	1.1277	0.0163
20	6	1.3977	0.0158
20	7	1.5423	0.0123
20	8	1.6321	0.0111
20	9	1.7357	0.0140
20	10	1.9532	0.0320
20	11	2.5122	0.0685
20	12	5.1907	0.4602
40	1	0.2195	0.0038
40	2	0.3177	0.0053
40	3	0.4899	0.0111
40	4	0.8327	0.0185
40	5	1.5277	0.0391
40	6	2.5559	0.0532
40	7	3.2913	0.0396
40	8	3.5365	0.0226
40	9	3.5982	0.0174
40	10	3.7059	0.0273
40	11	4.1629	0.0642
40	12	6.5401	0.4375
60	1	0.2195	0.0038
60	2	0.3177	0.0053
60	3	0.4899	0.0111
60	4	0.8358	0.0194
60	5	1.6461	0.0525
60	6	3.6343	0.1164
60	7	5.3484	0.0756
60	8	5.7243	0.0319
60	9	5.6653	0.0209
60	10	5.6126	0.0262
60	11	5.8901	0.0594
60	12	8.1362	0.4258
80	1	0.2195	0.0038
80	2	0.3177	0.0053
80	3	0.4899	0.0111
80	4	0.8363	0.0197
80	5	1.6704	0.0576
80	6	4.5467	0.2029
80	7	7.6317	0.1148
80	8	8.0334	0.0378
80	9	7.7872	0.0240
80	10	7.5497	0.0255
80	11	7.6654	0.0564
80	12	9.7624	0.4081

SI max	λ	$E[T_s]$	$E[T_s]$ CI95
100	1	0.2195	0.0038
100	2	0.3177	0.0053
100	3	0.4899	0.0111
100	4	0.8363	0.0197
100	5	1.6729	0.0580
100	6	5.2647	0.2931
100	7	10.0359	0.1499
100	8	10.3884	0.0433
100	9	9.9396	0.0280
100	10	9.5185	0.0286
100	11	9.4683	0.0585
100	12	11.3979	0.3965
120	1	0.2195	0.0038
120	2	0.3177	0.0053
120	3	0.4899	0.0111
120	4	0.8363	0.0197
120	5	1.6729	0.0580
120	6	5.8265	0.3824
120	7	12.4191	0.1921
120	8	12.7711	0.0496
120	9	12.1152	0.0353
120	10	11.4965	0.0340
120	11	11.2706	0.0602
120	12	12.9782	0.3712
140	1	0.2195	0.0038
140	2	0.3177	0.0053
140	3	0.4899	0.0111
140	4	0.8363	0.0197
140	5	1.6729	0.0580
140	6	6.2580	0.4695
140	7	14.7827	0.2410
140	8	15.1544	0.0592
140	9	14.2970	0.0419
140	10	13.4849	0.0394
140	11	13.0898	0.0575
140	12	14.5766	0.3567
160	1	0.2195	0.0038
160	2	0.3177	0.0053
160	3	0.4899	0.0111
160	4	0.8363	0.0197
160	5	1.6729	0.0580
160	6	6.5128	0.5297
160	7	17.0925	0.2905
160	8	17.5471	0.0714
160	9	16.4999	0.0509
160	10	15.4854	0.0454
160	11	14.9160	0.0544
160	12	16.2153	0.3499

8.3 Obiettivo 3

Nell'obiettivo 3 si va ad duplicare la potenza dello spike server, per verificare se questo evita il saturamento dello spike server e quindi il successivo aumento del tempo di risposta medio. Come mostrato in figura 17, lo spike server non arriva più a saturarsi e il tempo di risposta continua a diminuire all'aumentare del tasso di arrivo.

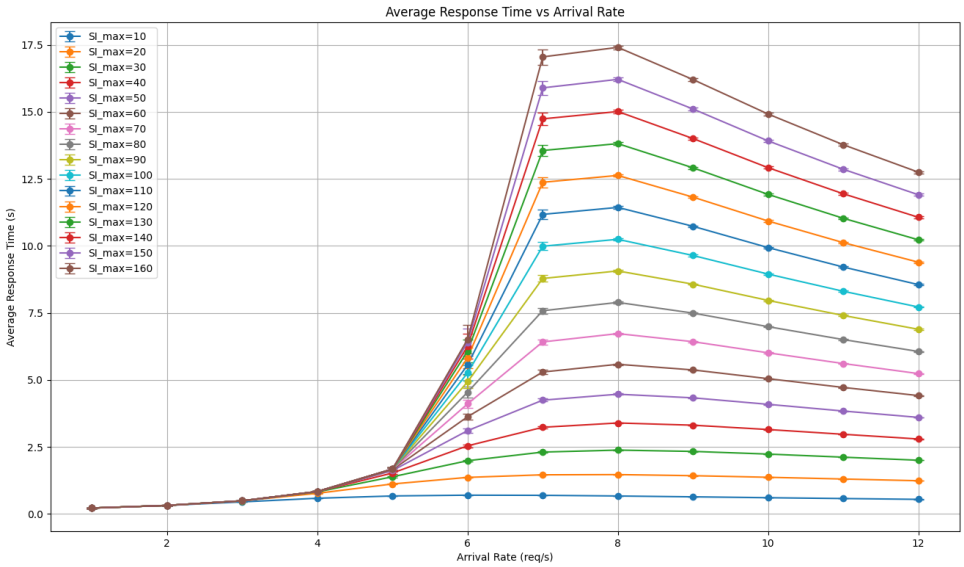


Figure 17: Tempo di risposta medio in funzione del tasso di arrivo e di SI_{max} con spike server potenziato

Table 7: Tempo di risposta medio in funzione del tasso di arrivo e di SI_{max} con spike server potenziato

SI max	λ	$E[T_s]$	$E[T_s]$ CI95
20	1	0.2195	0.0038
20	2	0.3176	0.0053
20	3	0.4875	0.0099
20	4	0.7690	0.0126
20	5	1.1181	0.0158
20	6	1.3633	0.0146
20	7	1.4595	0.0104
20	8	1.4677	0.0074
20	9	1.4267	0.0058
20	10	1.3677	0.0045
20	11	1.3027	0.0035
20	12	1.2382	0.0032
40	1	0.2195	0.0038
40	2	0.3177	0.0053
40	3	0.4899	0.0111
40	4	0.8327	0.0185
40	5	1.5265	0.0390
40	6	2.5407	0.0523
40	7	3.2317	0.0374
40	8	3.3925	0.0201
40	9	3.3098	0.0130
40	10	3.1488	0.0104
40	11	2.9712	0.0078
40	12	2.7961	0.0075
60	1	0.2195	0.0038
60	2	0.3177	0.0053
60	3	0.4899	0.0111
60	4	0.8358	0.0194
60	5	1.6459	0.0525
60	6	3.6257	0.1156
60	7	5.2940	0.0735
60	8	5.5813	0.0292
60	9	5.3716	0.0177
60	10	5.0454	0.0153
60	11	4.7184	0.0131
60	12	4.4150	0.0121
80	1	0.2195	0.0038
80	2	0.3177	0.0053
80	3	0.4899	0.0111
80	4	0.8363	0.0197
80	5	1.6704	0.0576
80	6	4.5417	0.2023
80	7	7.5794	0.1126
80	8	7.8899	0.0357
80	9	7.4914	0.0227
80	10	6.9846	0.0218
80	11	6.5039	0.0192
80	12	6.0600	0.0175

SI max	λ	$E[T_s]$	$E[T_s]$ CI95
100	1	0.2195	0.0038
100	2	0.3177	0.0053
100	3	0.4899	0.0111
100	4	0.8363	0.0197
100	5	1.6729	0.0580
100	6	5.2617	0.2926
100	7	9.9856	0.1477
100	8	10.2436	0.0414
100	9	9.6399	0.0287
100	10	8.9465	0.0284
100	11	8.3056	0.0251
100	12	7.7184	0.0227
120	1	0.2195	0.0038
120	2	0.3177	0.0053
120	3	0.4899	0.0111
120	4	0.8363	0.0197
120	5	1.6729	0.0580
120	6	5.8247	0.3820
120	7	12.3706	0.1900
120	8	12.6261	0.0482
120	9	11.8170	0.0362
120	10	10.9252	0.0350
120	11	10.1203	0.0312
120	12	9.3900	0.0282
140	1	0.2195	0.0038
140	2	0.3177	0.0053
140	3	0.4899	0.0111
140	4	0.8363	0.0197
140	5	1.6729	0.0580
140	6	6.2569	0.4693
140	7	14.7358	0.2390
140	8	15.0097	0.0583
140	9	14.0020	0.0434
140	10	12.9146	0.0416
140	11	11.9422	0.0379
140	12	11.0653	0.0341
160	1	0.2195	0.0038
160	2	0.3177	0.0053
160	3	0.4899	0.0111
160	4	0.8363	0.0197
160	5	1.6729	0.0580
160	6	6.5122	0.5296
160	7	17.0474	0.2883
160	8	17.4033	0.0705
160	9	16.2019	0.0534
160	10	14.9150	0.0497
160	11	13.7714	0.0447
160	12	12.7474	0.0402