

Performance Analysis of a Web App Workflow

DANIELE LA PROVA* and ELISA VERZA†

1 INTRODUZIONE

In questo lavoro presentiamo un'analisi delle prestazioni di una applicazione di e-commerce sviluppata usando uno stack tecnologico WEB. Il lavoro è ispirato a uno studio analogo delle performance effettuato da Serazzi [2024].

1.1 Contesto

Il sistema oggetto di studio consiste in un'applicazione distribuita su tre server:

- Server A: Gestisce l'autenticazione del cliente, la riserva in magazzino del prodotto, la generazione della ricevuta di acquisto, la pianificazione della spedizione e la finalizzazione dell'acquisto.
- Server B: Gestisce la logica di business, ovvero implementa tutte le funzioni di shopping quali la gestione del carrello, la consultazione del catalogo e l'acquisto dei prodotti;
- Server P: Gestisce l'elaborazione dei pagamenti.

Figure 2 mostra in maggiore dettaglio le mansioni dei servers. Si assume che il server P sia un attore esterno al sistema e dunque sotto una diversa proprietà rispetto ai server A e B. Inoltre, si assumono latenze di rete e think times nulli per studiare le performance del sistema nel contesto di carico più pesante e serrato possibile.

Le richieste elaborate dal sistema appartengono a una determinata **classe**, che ne determina la domanda media di servizio a seconda del server che la deve elaborare. Una stessa richiesta può essere più volte promossa a una classe diversa durante il suo viaggio nel sistema, come illustrato nella Figure 1.

2 METODOLOGIA

2.1 Obiettivi

Procediamo con l'individuazione degli obiettivi della nostra analisi:

- **Obiettivo 1:** ha lo scopo di implementare un modello in grado di rappresentare il sistema oggetto di studio e misurare tempo di risposta, popolazione e throughput medi. Tutte le metriche devono essere considerati sia locali, per singolo nodo, sia globalmente su tutto il sistema.
- **Obiettivo 2:** la WebApp decide di implementare un sistema di autenticazione a due fattori per rendere i pagamenti più sicuri. Ci siamo posti l'obiettivo di andare a valutare l'impatto di questa modifica sul sistema, valutando le stesse metriche dell'obiettivo precedente.
- **Obiettivo 3:** vogliamo ora osservare il comportamento del sistema in caso di carico di richieste maggiore. Per i primi due obiettivi il massimo carico era di 4320 job/h (ovvero 1.2 job/s) incrementato a circa 5000 job/h (ovvero 1.4 job/s). Il

*Both authors contributed equally to this research.

Authors' address: Daniele La Prova, daniele.laprova@hotmail.it; Elisa Verza, elisaverza.gm@gmail.com.

2025. XXXX-XXXX/2025/1-ART \$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

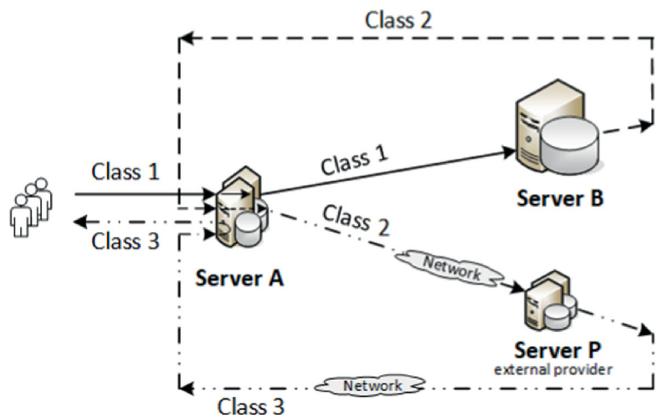


Fig. 1. Viaggio che le richieste utente intraprendono nel sistema e quali classi vengono loro assegnate [Serazzi 2024]

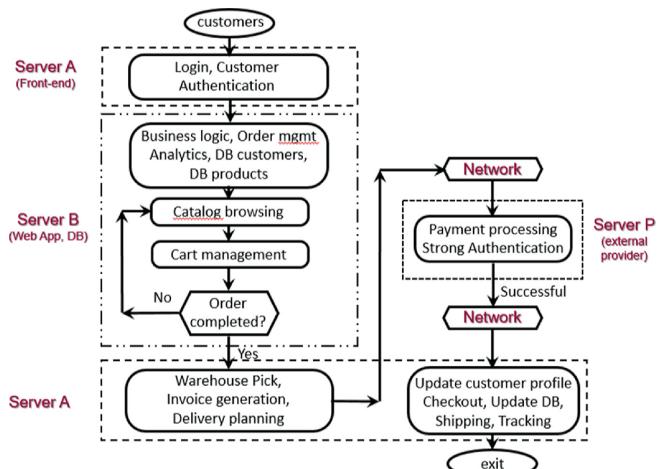


Fig. 2. Dettaglio delle mansioni svolte dai server durante le tappe del viaggio di una richiesta all'interno del sistema. [Serazzi 2024]

confronto viene eseguito sulle stesse metriche dei primi due obiettivi.

- **Obiettivo 4:** l'ultimo obiettivo ha lo scopo di migliorare il sistema andando ad individuare eventuali bottleneck per migliorarne le prestazioni.

2.2 Modello Concettuale

Il sistema in analisi è stato modellato come una rete di code aperta il cui schema è riportato in Figure 3.

I job rappresentano gli utenti della WebApp dal momento in cui effettuano il login fino all'uscita successiva al checkout a seguito del pagamento.

Il percorso del job all'interno del sistema consiste in tre visite al

centro A ed una visita ai centri B e P seguendo l'ordine: A-B-A-P-A. Dopo il terzo passaggio in A il job esce dal sistema. Per questo motivo è stata assegnata ai job una classe seguendo il seguente criterio:

- **Classe 1:** sono i job che arrivano dall'esterno e restano in tale classe fino all'entrata in servizio in B.
- **Classe 2:** sono i job in arrivo nel server A tramite il feedback del server B e restano in tale classe fino all'entrata in servizio in P
- **Classe 3:** sono i job in arrivo nel server A tramite il feedback del server P. Sono anche la classe di job che escono dal sistema.

Con questa premessa le variabili di stato che meglio descrivono il sistema sono di due tipologie:

- $N_{node, class}$: numero di job di classe $class \in \{1, 2, 3\}$ nel nodo $node \in \{A, B, P\}$
- C_j con $j \in [0, \infty)$: valore che indica la classe di appartenenza del job j.

La simulazione utilizzata è una *next-event simulation*. Gli eventi sono di due tipologie, arrivi (Arrival) caratterizzati da server di destinazione e classe del job in arrivo e partenze (Departure) caratterizzato da server di partenza e classe del job in partenza. Non abbiamo inserito nessun evento artificiale

Di seguito l'evoluzione del sistema in base al verificarsi degli eventi divisi per server.

Server A

- Arrivo dall'esterno del sistema: vengono modificate le variabile di stato incrementando N_A , 1 e ponendo $C_j = 1$. Vengono generati l'evento di Departure per il job di classe 1 dal server A e l'evento di arrival dall'esterno successivo;
- Departure di job di classe 1: viene decrementata la variabile di stato N_A , 1 e generato l'arrivo del job di classe 1 nel server B;
- Arrival di job di classe 2: viene incrementata la variabile di stato N_A , 2, viene generato l'evento di Departure per il job di classe 2 dal server A.
- Departure di job di classe 2: viene decrementata la variabile di stato N_A , 2. Viene generato l'arrivo del job di classe 2 nel server P;
- Arrival di job di classe 3: viene incrementata la variabile di stato N_A , 3, viene generato l'evento di Departure per il job di classe 3 dal server A.
- Departure di job di classe 3: E' l'evento di uscita dal sistema. Viene decrementata la variabile di stato N_A , 3.

Server B

- Arrival di job di classe 1: viene incrementata la variabile di stato N_B , 1, viene generato l'evento di Departure per il job di classe 1 dal server B.
- Departure di job di classe 1: vengono modificate le variabile di stato decrementando N_B , 1 e ponendo $C_j = 2$. Viene generato l'arrivo del job di classe 2 nel server A;

Server P

- Arrival di job di classe 2: viene incrementata la variabile di stato N_P , 2, viene generato l'evento di Departure per il job di classe 2 dal server P.

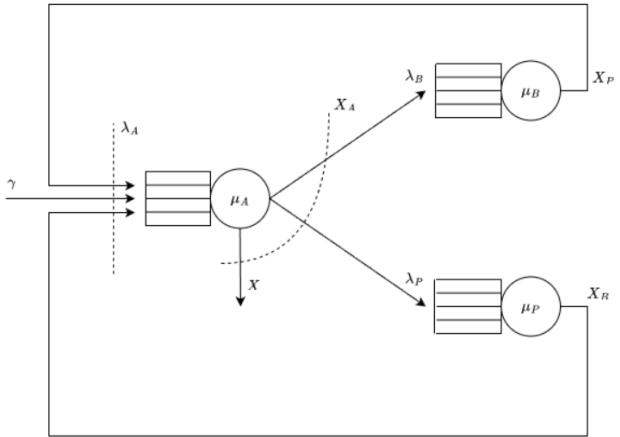


Fig. 3. Diagramma del modello a reti di code del sistema

Stations	Classes		
	1	2	3
Server A (Login, Front end, ...)	0.2	0.4	0.1
Server B (Web App Serv., DBs, ...)	0.8	0	0
Server P (Payment Provider)	0	0.4	0

Stations	Classes		
	1	2	3
Server A (Login, Front end, ...)	0.2	0.4	0.15
Server B (Web App Serv., DBs, ...)	0.8	0	0
Server P (Payment Provider)	0	0.7	0

Fig. 4. Tempi di servizio medi per classe di job in ogni nodo, nella versione vanilla (sinistra) e con 2FA (destra) [Serazzi 2024]

- Departure di job di classe 2: vengono modificate le variabile di stato decrementando N_P , 2 e ponendo $C_j = 3$. Viene generato l'arrivo del job di classe 3 nel server P;

2.3 Modello delle Specifiche

I valori di input, se non diversamente specificato, sono presi dal lavoro di Serazzi [2024].

Le variabili indipendenti sono le seguenti:

- Distribuzioni di probabilità dei servizi: Esponenziali;
- Distribuzioni di probabilità degli arrivi esterni: Esponenziali;
- Tempi di servizio medi per classe di job in ogni nodo: vedere Figure 4
- Rate medi di arrivi esterni: valori che spaziano da 0.50 a 1.20 con un passo di $0.05req/s$, per poi estendere fino a $1.40req/s$ nel caso di carico pesante;
- Politiche di scheduling: PS per tutti i server coinvolti.
- Matrice di routing: consultando Figure 5 e Figure 1 è possibile produrre la matrice di routing mostrata in Table 1.

Le variabili dipendenti sono le seguenti metriche di performance:

- Tempo di risposta T : detto anche Tempo di Residenza, è l'intervallo di tempo che una richiesta trascorre all'interno di un nodo (o del Sistema) dal momento in cui entra fino al momento in cui esce;
- Popolazione N : numero di richieste all'interno di un nodo (o nel Sistema);

CS Parameters Definition				
	Class Switch Matrix \ Routing Section			
CS Strategies				
*	Class1	Class2	Class3	
Class1	0.0 (0%)	1.0 (100%)	0.0 (0%)	
Class2	0.0 (0%)	0.0 (0%)	1.0 (100%)	
Class3	0.0 (0%)	0.0 (0%)	1.0 (100%)	

Fig. 5. Class switch matrix [Serazzi 2024]

Table 1. Routing Matrix per jobs di una certa classe che arrivano in un server. I valori nelle celle implicano una probabilità di routing pari a 1 verso i server specificati e 0 verso tutti gli altri. Il valore EXIT indica l'uscita dal sistema.

	class 1	class 2	class3
[HTML]FFFFFFServer A	Server B	Server P	EXIT
[HTML]FFFFFFServer B	Server A		
[HTML]FFFFFFServer P		Server A	

- Throughput X : numero di richieste soddisfatte sull'unità di tempo da un nodo (o dal Sistema).
- Utilizzazione ρ : proporzione di tempo in cui il nodo è occupato dall'esecuzione delle richieste.

2.4 Modello Computazionale

Il codice del simulatore è stato realizzato in Python ed è disponibile al seguente indirizzo: <https://github.com/caballo-domestico/webapp-workflow-pa>

2.4.1 Scheduler Next-Event. Figure 8 illustra l'architettura dello scheduler Next-Event. Per implementare una Next-Event simulation è stato definito un oggetto `NextEventScheduler`, che mantiene una event list, ovvero una lista di oggetti Event ordinata secondo il tempo di schedulazione. È possibile aggiungere un evento allo scheduler attraverso il metodo `schedule(event, delay)`, il quale verrà inserito alla posizione corrispondente al suo tempo di schedulazione, con l'aggiunta di un delay opzionale. È inoltre possibile annullare la schedulazione di un evento attraverso il metodo `cancel(event)`, che dunque non verrà più processato pur rimanendo nella event list. Lo scheduler espone un'interfaccia da iteratore, ovvero attraverso i metodi `has_next()` e `next()` è possibile consumare gli eventi della event list, ovvero processarli. Per processare un evento, ne viene invocato l'`EventHandler` corrispondente che ne implementa i dovuti cambiamenti di stato.

2.4.2 Arrivi e Partenze dei jobs. Figure 6 mostra i passaggi da intraprendere per gestire l'arrivo di un job a un nodo. In sintesi, l'handler calcola il tempo di departure del nodo come la somma dei seguenti tempi:

- Arrival time: tempo di simulazione all'istante di arrivo del nodo;

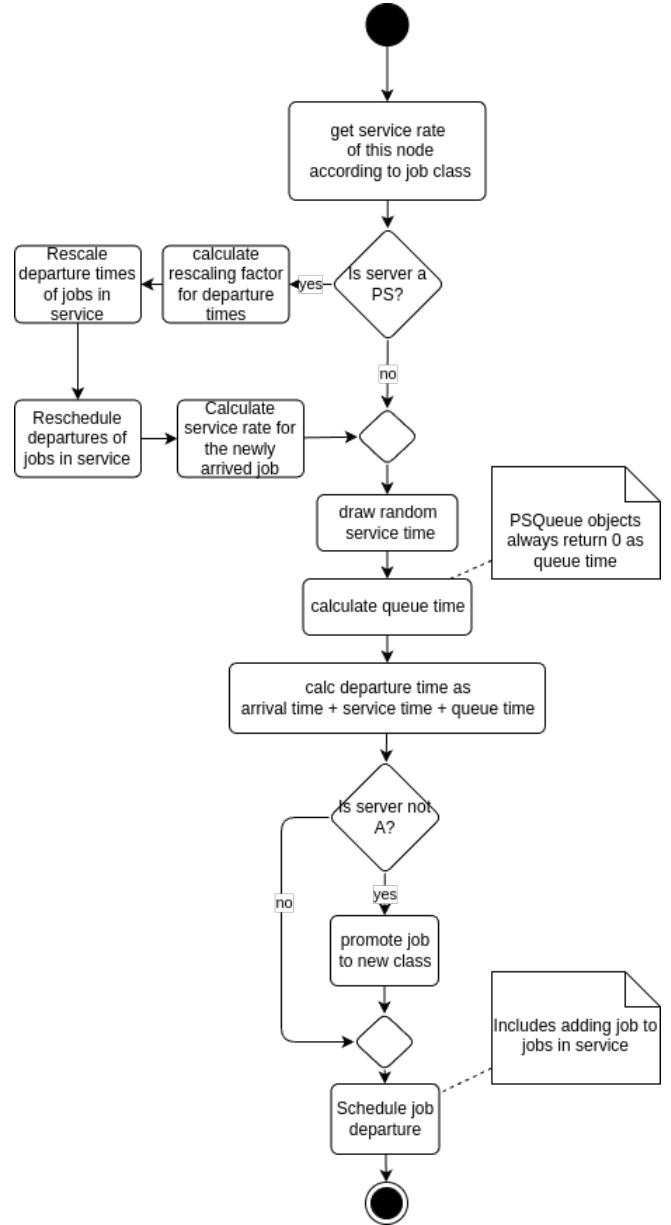


Fig. 6. Activity diagram dell'handler degli arrivi.

- Service time: tempo che il job deve spendere in servitù, estratto casualmente dalla distribuzione dei servizi del nodo con parametri che dipendono dalla classe del job;
- Queue time: tempo che il job spende nella coda del nodo, che dipende dalla politica di scheduling e dai jobs residenti nel nodo.

Nel caso in cui la politica di scheduling sia PS è necessaria particolare attenzione, infatti il ricambio di jobs residenti nel nodo implica un aggiornamento dei tempi di servizio dei jobs già in servitù. Se chiamiamo S_{rem} il tempo di servizio rimanente per un certo job con

N jobs in servitura nel nodo otteniamo Equation 1, dove D_{rem} è la domanda rimanente da smaltire per completare il servizio su quel job.

$$S_{rem} = \frac{D_{rem}}{\frac{\mu}{N}} \quad (1)$$

Nel momento in cui arriva un nuovo job nel nodo, il suo tempo rimanente andrebbe ora calcolato come in Equation 2, dove N^* è il nuovo numero di jobs nel nodo che include il nuovo job arrivato.

$$S_{rem}^* = \frac{D_{rem}}{\frac{\mu}{N^*}} \quad (2)$$

Si noti come D_{rem} sia ovviamente la stessa quantità sia in Equation 1 che in Equation 2. Mettendo a sistema le due equazioni otteniamo Equation 3, dove $\frac{N^*}{N}$ è il Rescaling Factor menzionato in Figure 6.

$$S_{rem}^* = S_{rem} \frac{N^*}{N} \quad (3)$$

Usando Equation 3 è possibile aggiornare i tempi delle partenze dei job già in servizio nel nodo nel momento in cui arriva un nuovo job. L'effetto che si ottiene è che con l'arrivo di nuovi job, i tempi di servizio rimanente si allungano. Nel caso particolare in cui il job arrivi in un nodo vuoto, allora sia N che N^* sono pari a 1.

Figure 7 mostra l'Activity Diagram dell'handler che gestisce la partenza di un job da un nodo. Essenzialmente, rimuove il job dall'insieme dei job residenti al nodo in questione, e se necessario genera un nuovo evento di arrivo al nodo di destinazione. Se il nodo ha politica PS è necessario aggiornare le departures dei nodi ancora residenti usando sempre Equation 3, con l'effetto di accorciare i tempi di servizio rimanenti.

2.4.3 Event-Oriented Programming. È possibile iscrivere degli EventHandler alla ricezione di uno o più tipi di eventi poiché lo scheduler implementa un pattern Pub-Sub push-notify. L'iscrizione agli eventi può avvenire attraverso i seguenti metodi dello scheduler:

- `subscribe(event_type, event_handler)`: l'EventHandler viene invocato dopo che l'evento è stato elaborato, ovvero il suo handler ha terminato l'esecuzione;
- `intercept(event_type, event_handler)`: l'EventHandler viene invocato prima che l'evento è stato elaborato, e può modificare l'Event stesso.

Non c'è garanzia sull'ordine della ricezione delle notifiche tra subscribers e tra interceptors. Inoltre, iscriversi a un tipo di Event comporta la ricezione delle notifiche anche per eventi che specializzano quel tipo di eventi. Ad esempio, iscriversi al tipo Event implica la ricezione di una notifica ad ogni evento processato. Gli eventi annullati non comportano l'invio di notifiche.

A supporto della realizzazione delle simulazioni sono stati implementati i seguenti subscribers e interceptors:

- `BatchMeansInterceptor`: Intercetta eventi di arrivo esterni per tenerne il conto. Se il conteggio supera il numero di arrivi previsto per il batch, salva le statistiche finora calcolate per questo batch e ripristina le statistiche calcolate dagli estimatori per il prossimo batch.
- `ArrivalsGeneratorSubscriber`: Si mette in ascolto di arrivi esterni e ne tiene il conto. Per ogni arrivo ricevuto, schedula un nuovo arrivo esterno dopo un tempo di interarrivo casuale,

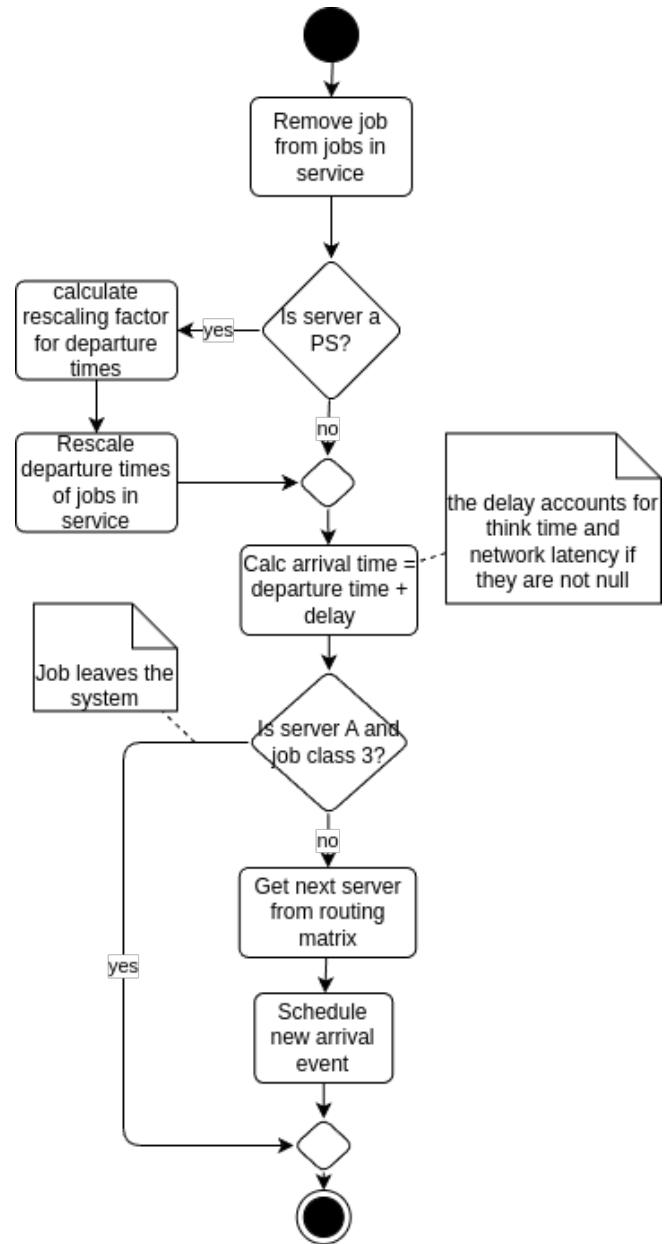


Fig. 7. Activity diagram dell'handler delle partenze.

fino a quando gli arrivi contati superano il numero massimo di arrivi previsto.

2.4.4 Misurazione metriche di performance. Gli estimatori delle misurazioni delle metriche di performance sono stati implementati come subscribers e sono i seguenti:

- `BusytimeEstimator`: Si mette in ascolto di arrivi e partenze dei job. Per ogni nodo, traccia gli arrivi e le partenze e tiene conto del numero di jobs ancora presenti nel nodo. Se a causa di un arrivo tale numero passa da zero a maggiore di zero,

- l'istante di tempo di tale arrivo segna l'inizio di un busy period per quel nodo, che terminerà solo quando tale numero scende di nuovo a zero, oppure lo stimatore è resettato (i.e. quando cambia il batch);
- **ObservationTimeEstimator:** Si iscrive a tutti gli eventi e tiene il conto del tempo di simulazione trascorso dalla sua iscrizione o dal suo ultimo reset;
 - **CompletionsEstimator:** Si iscrive alle partenze dei jobs dai nodi. Per ogni nodo, conta il numero di partenze dall'inizio della simulazione o dal reset dello stimatore;
 - **ResponseTimeEstimator:** Si iscrive agli arrivi e alle partenze dei jobs. Per ogni nodo, si segna il tempo di arrivo e di partenza di ogni job e ne calcola il tempo di risposta come la differenza dei due valori.
 - **PopulationEstimator:** Si iscrive agli arrivi e alle partenze dei jobs. Per ogni nodo, ne incrementa il conteggio della popolazione dei jobs per ogni arrivo ricevuto e la decrementa per ogni partenza.

Gli stimatori **ResponseTimeEstimator** e **PopulationEstimator** mantengono una media dei valori campionati usando un oggetto **WelfordEstimator**, che implementa l'algoritmo di Welford per il calcolo online della media [Lawrence M. Leemis 2004]. Inoltre, tale oggetto calcola anche la Standard Deviation, il Min value e il Max value dei campioni raccolti durante una run. Gli altri stimatori mantengono invece la somma dei campioni.

2.4.5 Modello. Figure 9 mostra come il modello a rete di code è implementato nel sistema. Un oggetto **Network** si compone di diversi **Node**, ognuno dei quali è composto da un **Server** e da una **Queue**. Il **Server** implementa il servente e attraverso il metodo `get_service(service_rate)` genera un tempo di servizio casuale secondo un certo rate e distribuzione di probabilità, dopo aver selezionato lo stream corretto. La **Queue** calcola il tempo che un job deve spendere in coda prima che sia il suo turno di essere servito in base alla politica di scheduling. I tempi di interrarrivo esterni sono calcolati dalla **Network** secondo la sua distribuzione di probabilità degli arrivi e il rate medio di arrivi.

2.4.6 Processi stocastici. La simulazione dei processi casuali è stata implementata usando il modulo **rngs** [Lawrence M. Leemis 2004], il quale mette a disposizione diversi streams di generatori di Lehmer in grado di produrre una sequenza di numeri pseudo-casuali secondo una distribuzione desiderata. Per ogni processo casuale indipendente simulato nel sistema è selezionato uno stream dedicato, ovvero si imposta il seed del generatore al valore corrente per quello stream prima di pescare il prossimo numero della sequenza. I processi pseudo-casuali indipendenti implementati nel sistema sono:

- Gli arrivi dall'esterno del sistema al server A;
- I servizi del server A;
- I servizi del server B;
- I servizi del server P;

2.4.7 Simulation Runs. Una simulazione è implementata come un oggetto **Simulation**, come illustrato in Figure 9. Attraverso l'uso di decoratori, sono state implementate due varianti di simulazione:

- **BatchMeansSimulation:** Iscrive alla simulazione decorata un **BatchMeansInterceptor** per implementare una simulazione Batch Means;
- **ReplicatedSimulation:** Decora una sequenza di simulazioni identiche e le esegue in sequenza, impostando il seed iniziale della prossima replica come l'ultimo valore della sequenza prng (DEFAULT stream) generato dalla replica precedente.

È possibile implementare una run di simulazione aggiungendo a runtime i comportamenti desiderati utilizzando opportunamente decoratori, subscribers e interceptors, oltre a costruire la rete di code desiderata andando a comporre opportunamente oggetti **Server** e **Queue** in oggetti **Node**, e questi ultimi in una **Network** da associare poi alla **Simulation**. Inoltre, è necessario impostare un evento iniziale che di fatto dà il via alla simulazione. Nel nostro caso, abbiamo delegato la complessità della costruzione di una simulazione in una **SimulationFactory**. Queste caratteristiche rendono il simulatore molto potente e flessibile in termini di features mantenendone contenuta la complessità della implementazione.

Una volta costruita una **Simulation**, per eseguirla è sufficiente invocarne il metodo `run()`, che imposta il seed iniziale per i vari streams del prng e avvia la consumazione degli eventi. Quando non ci sono più eventi da consumare, la simulazione termina e viene effettuato il salvataggio delle statistiche su dei file .csv nella cartella **statistics**. I parametri della simulazione sono definiti in un file **config.json**, dove a ogni entry corrisponde un set di valori di parametri che realizzano un **Simulation Study**.

2.5 Verifica

Per la verifica della correttezza del simulatore abbiamo studiato il modello analitico del sistema. Il tale modello però non può tenere conto delle tre classi servite dal server A che richiederebbero tre tassi di servizio differenti.

Abbiamo osservato che nel server A entrano tre flussi di job: uno per gli arrivi dall'esterno, due dai feedback dei server B e P. Essendo il percorso dei job deterministico abbiamo concluso che la probabilità di trovare un job di una specifica classe in servizio nel server A è uguale ad $\frac{1}{3}$ per ogni classe. Abbiamo quindi trattato tutti i job del server A con un rate ottenuto facendo la media delle tre classi:

$$\mu_A = \sum_{c=1}^3 \mu_{A,c} p_{A,c} \quad (4)$$

Dove $\mu_{A,c}$ è il tasso di servizio della classe c nel server A e $p_{A,c}$ è la probabilità di trovare un job di classe c nel server A se quest'ultimo è occupato. Considerato quanto detto fin'ora e con l'ulteriore assunzione che il flusso dei job è bilanciato possiamo scrivere Equation 5 che esprime i rate di arrivi a ogni server, calcolati risolvendo il bilanciamento dei flussi riportato in Equation 6 e ottenendo così Equation 7.

$$\begin{aligned} \lambda_A &= \gamma + X_B + X_P \\ \lambda_B &= X_B \\ \lambda_P &= X_P \end{aligned} \quad (5)$$

$$\begin{cases} \gamma + X_B + X_P = (p_{A,1} + p_{A,2} + p_{A,3}) X_A \\ X_A p_{A,1} = X_B \\ X_A p_{P,2} = X_P \end{cases} \quad (6)$$

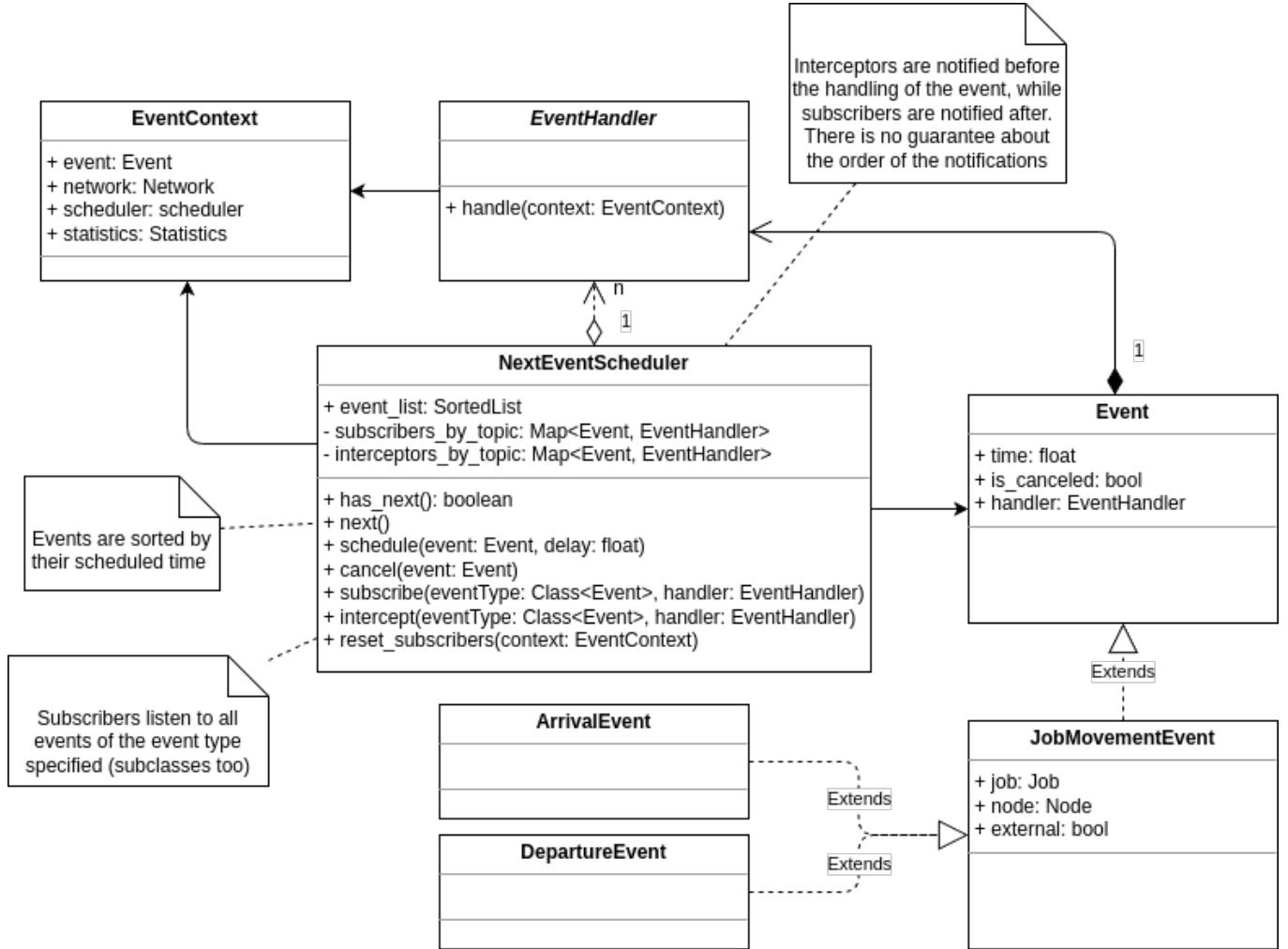


Fig. 8. Vista delle classi che implementano uno scheduler Next-Event.

$$\begin{aligned} X_A &= \lambda_A = 3\gamma \\ X_B &= \lambda_B = \gamma \\ X_P &= \lambda_P = \gamma \end{aligned} \quad (7)$$

Per ogni obiettivo prima di procedere al calcolo degli indici locali e globali andiamo a calcolare l'utilizzazione al variare del tasso di arrivo dall'esterno γ per ogni server per assicurarci che siano stabili.

$$\rho_s = \frac{\lambda_s}{\mu_s} \quad (8)$$

Per il calcolo del tempo di risposta per ogni server abbiamo utilizzato Equation 9.

$$E[T]_s = \frac{1}{(1 - \rho_s) \mu_s} \quad (9)$$

Mentre per la popolazione media abbiamo utilizzato la legge di Little riportata in Equation 10.

$$E[N]_s = E[T]_s \lambda_s \quad (10)$$

Infine per quanto riguarda gli indici globali sono stati calcolati come in Equation 12. Abbiamo calcolato per ogni nodo le visite medie con l'Equation 11. Con le visite medie abbiamo potuto calcolare il tempo di risposta medio e, di conseguenza, utilizzando il teorema di Little la popolazione media. Per quanto riguarda il throughput viene preso in considerazione quello del server A dei soli job di classe 3 ovvero quelli che escono dal sistema. Essendo dipendente dal tasso di arrivo γ ciò è valido solo se il sistema è stabile.

$$v_s = \frac{\lambda_s}{\gamma} \quad (11)$$

$$\begin{aligned} E[T] &= \sum_{s=A}^P E[T]_s v_s \\ E[N] &= \gamma E[T] \\ X &= X_A p_{A,3} \end{aligned} \quad (12)$$

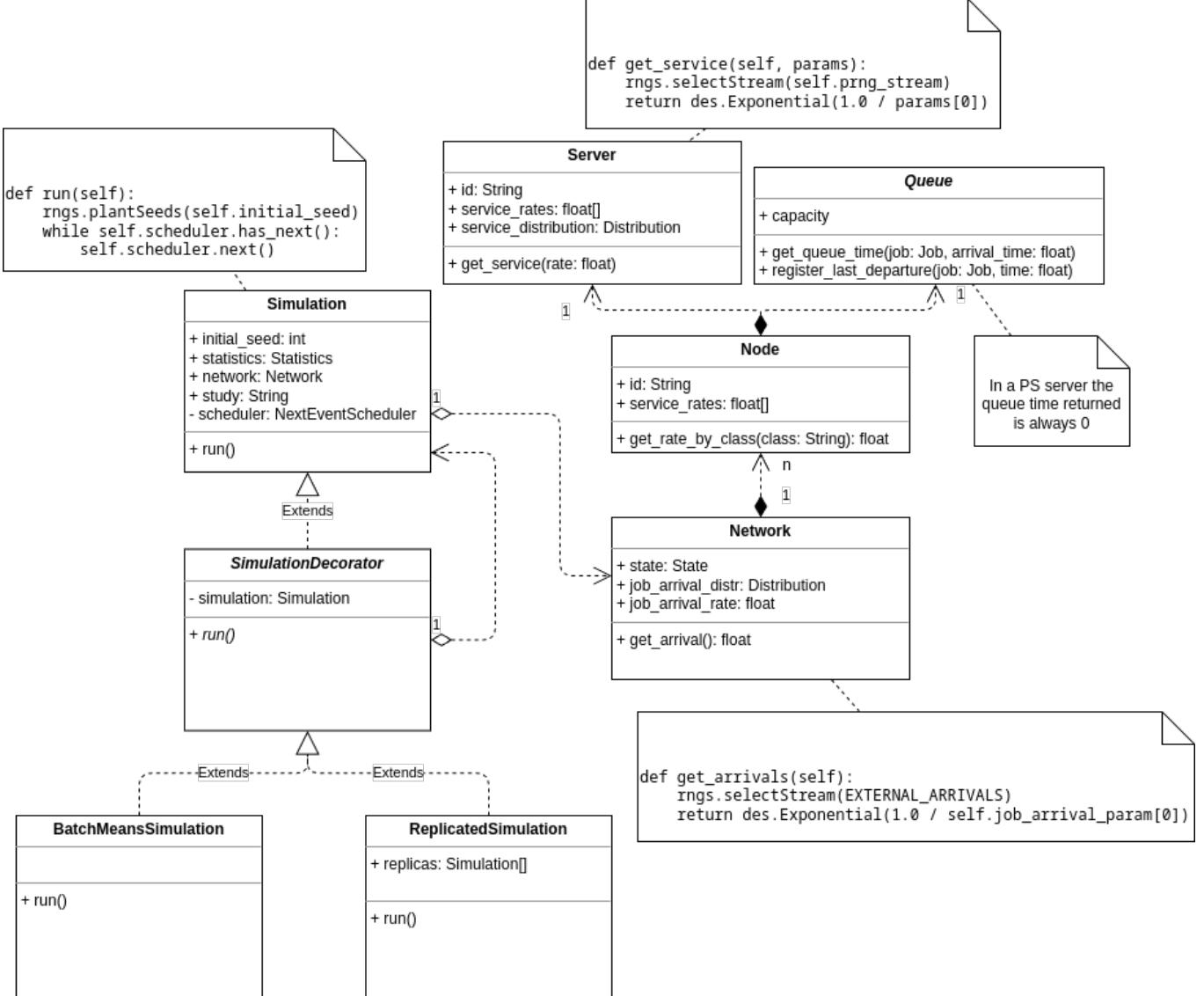


Fig. 9. Vista delle classi che implementano una Simulazione.

2.6 Validazione

In assenza di dati sperimentali riguardanti una implementazione reale della Web App non è possibile effettuare una validazione vera e propria. Tuttavia, assumendo che il simulatore descritto da Serazzi [2024] sia stato validato su un caso di studio reale, abbiamo validato i nostri risultati con quelli della fonte citata.

2.7 Simulation Studies

Gli obiettivi che ci siamo posti ci hanno portato ad eseguire diverse run con configurazioni diverse di parametri. Per ogni obiettivo viene eseguita una run per ogni tasso di arrivo. Le run effettuate sono simulazioni ad orizzonte infinito implementate come Batch-Means,

con 100 batch da 64 job ognuno.

- **Obiettivo 1:** nel primo esperimento consideriamo i server nel loro assetto base con rate di servizio elencati in Table 2. Inoltre solo per questo obiettivo abbiamo voluto osservare il comportamento dello stato transitorio effettuando anche una replicated simulation di 100 repliche da 64 job ciascuna in modo simmetrico rispetto alla simulazione batch means.
- **Obiettivo 2:** in questo caso i server subiscono una diminuzione dei rate di servizio a causa dell'autenticazione a due fattori. Per il server A nel caso di job di classe 3, diventa 6.6667 job/s , mentre per il server P diventa 1.4285 job/s , resta invariato per B.

Table 2. Tassi di servizio medi per classe di job offerti da ogni nodo.

	Classe 1	Classe 2	Classe 3
Server A	5	2.5	10
Server B	1.25	0	0
Server C	0	2.5	0

• **Obiettivo 3:** in questo caso abbiamo esteso i tassi di arrivo fino al nuovo carico pesante richiesto, mentre per i tassi di servizio si torna alla configurazione base dei server.

• **Obiettivo 4:** per questo obiettivo, come approfondiremo più avanti, sono state fatte più run al variare del rate di servizio del server B nel tentativo di migliorare le performance del sistema.

Gli altri server hanno mantenuto la loro configurazione base.

Per ogni obiettivo abbiamo generato i seguenti grafici per ogni metrica (tempo medio di risposta, popolazione media, throughput, utilizzazione), per ogni nodo e globalmente per il sistema:

Per ogni run differente viene fatta la media dei valori medi delle metriche in modo da aggregare i dati ottenuti intra-run (le medie delle metriche per singolo batch nel caso di simulazione ad orizzonte infinito e della singola replica per l'orizzonte finito). Allo stesso modo viene calcolata la deviazione standard ed il numero di sample points. Abbiamo quindi calcolato l'intervallo di confidenza secondo Equation 13:

$$CI = \bar{x} \pm z \cdot \frac{\sigma}{\sqrt{n}} \quad (13)$$

Dove \bar{x} è la media campionaria, z è il valore critico della distribuzione normale standard considerato a livello di confidenza del 95% (ha valore 1.96), σ è la deviazione standard ed n il numero di sample points.

Inoltre abbiamo utilizzato i valori delle metriche ottenuti per batch (o per replica) per osservare la loro distribuzione.

3 RISULTATI E DISCUSSIONE

I valori grezzi delle misurazioni sperimentali sono disponibili a https://github.com/caballo-domestico/webapp-workflow-pa/raw/refs/heads/main/docs/replication_data_kit/statistics.zip. I dati in forma tabellare del modello analitico sono disponibili a <https://github.com/caballo-domestico/webapp-workflow-pa/blob/main/docs/model/classless.ipynb>, mentre i risultati elaborati di questo studio sono disponibili a https://github.com/caballo-domestico/webapp-workflow-pa/blob/main/docs/plots/model_plots.ipynb.

3.1 Obiettivo 1

3.1.1 *Risultati sperimentali.* In Figure 10 andiamo ad analizzare il comportamento dell'utilizzazione. Per tutti i nodi l'andamento è crescente, come ci aspettavamo, la dispersione è piuttosto simile tra i vari box ad eccezione del server B con tasso di arrivo 1.2 job/s, in questo caso il box è schiacciato verso 1. Sia il server A che il server B presentano valori di massimo che raggiungono 1, invece per quanto riguarda i valori medi di utilizzazione il valore più alto che troviamo è di 0.97 per il server B quando il tasso di arrivo è 1.2 job/s, seguito dal caso con tasso di arrivo 1.15 job/s ed utilizzazione 0.92 sempre per il server B. Per i valori medi sono gli unici due casi che superano

lo 0.9.

Da notare anche come le performance del server B influenzino l'andamento del sistema. Per quanto riguarda la popolazione media in Figure 11 il server P riesce a mantenere dei valori in un range abbastanza ristretto, non superando mai l'1.5 job di media, rispetto al server A che presenta un andamento costantemente crescente fino ad arrivare a 5.78 job. Per quanto riguarda il server B presenta un andamento costantemente crescente fino al rate di arrivi 1.1 job/s, mentre per gli ultimi due valori (1.15 job/s e 1.12 job/s) presenta dei picchi di 11 job e 31 job, comportamento atteso visto il valore prossimo all'1 dell'utilizzazione. Comportamento analogo segue il tempo medio di risposta in Figure 12, il server P presenta come valore medio massimo 0.78s in corrispondenza del tasso di arrivo massimo, mentre il server A arriva a 1.47s. Il server B è in un range più ampio con picchi, per i tassi di arrivo 1.15 job/s e 1.2 job/s, che arrivano a 9.38s e 25.50s. Infine per il throughput in Figure 14 tutti e tre i server hanno un andamento costantemente crescente, indice che comunque riescono a servire i job in arrivo. Il server A ha il throughput più alto, mentre B e P hanno un andamento molto simile come si può vedere anche dal grafico in Figure 13

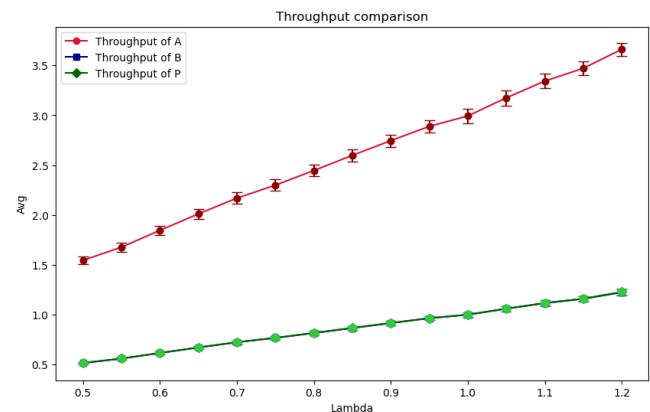


Fig. 13. Confronto throughput per i server A, B e P per l'obiettivo 1

3.1.2 *Verifica con il modello analitico.* Per il tasso di servizio del server A nel modello analitico abbiamo utilizzato i seguenti parametri: $\mu_A = 5.833 \text{ job/s}$. Mentre i tassi di servizio per i server B e P restano invariati rispetto alla simulazione: $\mu_B = 1.25 \text{ job/s}$ e $\mu_P = 2.5 \text{ job/s}$. Per ogni lambda i nodi risultano stabili, riportiamo i valori per tasso di arrivo minimo e massimo:

γ	ρ_A	ρ_B	ρ_P
0.5	0.257	0.4	0.2
1.2	0.617	0.96	0.48

Anche in questo caso riportiamo i valori di ogni indice locale per tasso di arrivo minimo e massimo:

Infine segue la tabella per gli indici globali della rete:

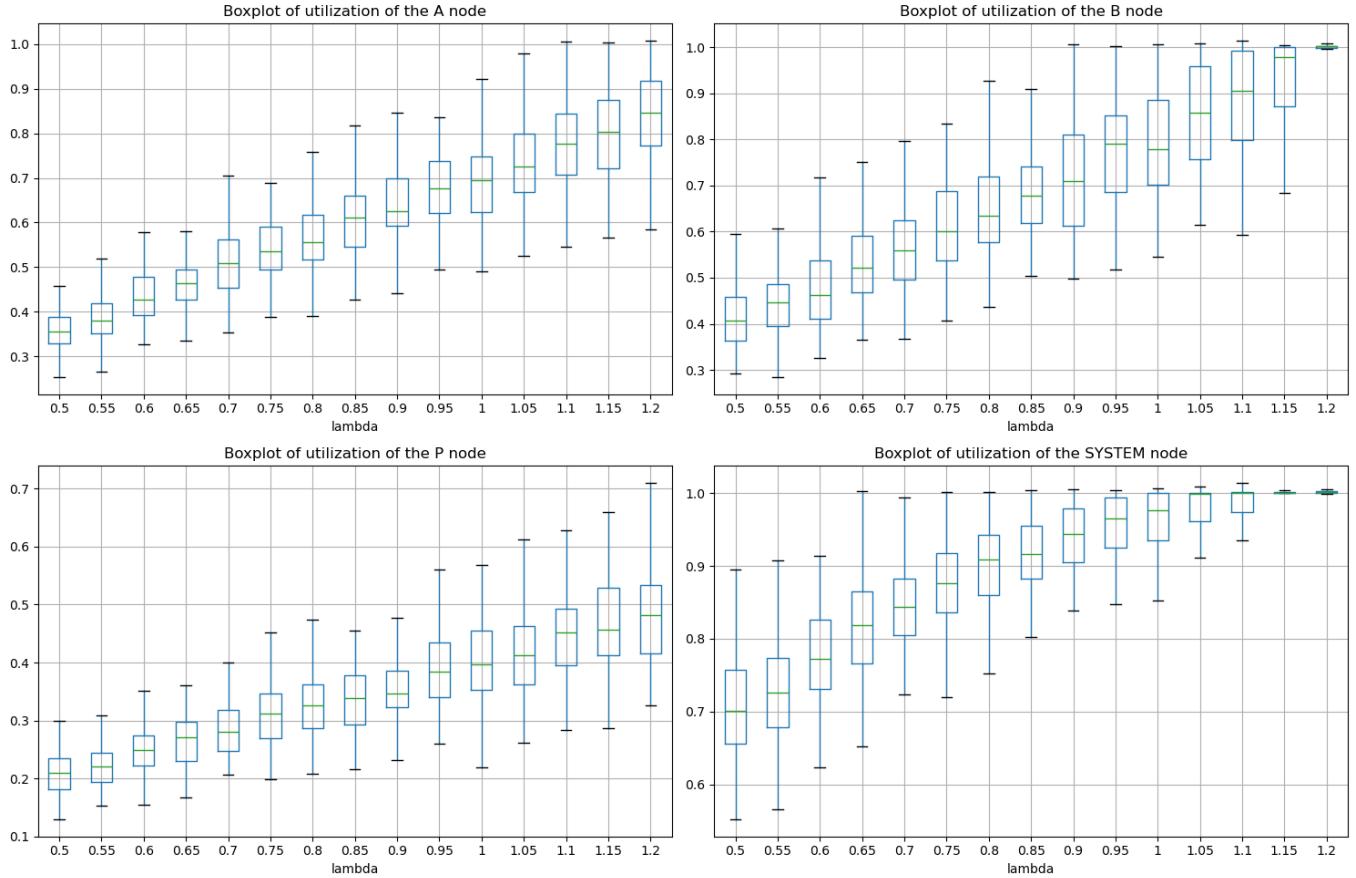


Fig. 10. Distribuzione dell'utilizzazione dei risultati sperimentali dell'obiettivo 1.

$$\begin{array}{cccc} \gamma & E[T]_A & E[T]_B & E[T]_P \\ 0.5 & 0.230 & 1.333 & 0.5 \\ 1.2 & 0.447 & 20 & 0.769 \end{array}$$

$$\begin{array}{cccc} \gamma & E[N]_A & E[N]_B & E[N]_P \\ 0.5 & 0.346 & 0.666 & 0.25 \\ 1.2 & 1.611 & 24 & 0.923 \end{array}$$

$$\begin{array}{cccc} \gamma & X_A & X_B & X_P \\ 0.5 & 1.5 & 0.5 & 0.5 \\ 1.2 & 3.6 & 1.2 & 1.2 \end{array}$$

$$\begin{array}{cccc} \gamma & E[T]_S & E[N]_S & X_S \\ 0.5 & 2.525 & 1.262 & 0.5 \\ 1.2 & 22.112 & 26.535 & 1.2 \end{array}$$

In Figure 15, Figure 16, Figure 17 è possibile vedere i grafici dell'intervallo di confidenza delle metriche ed il confronto con il modello analitico. Per quanto riguarda la popolazione media i server P e B si discostano molto poco, la differenza è mediamente di 0.5, ad eccezione del server B in corrispondenza del tasso di arrivo massimo

dove si vede un discostamento maggiore. Il server A invece presenta un discostamento leggermente superiore, ma comprensibile dato il diverso rate di servizio utilizzato. Possiamo osservare un comportamento analogo per i tempi medi di risposta. Infine il throughput è la metrica che presenta il comportamento migliore aderendo in modo piuttosto preciso.

Abbiamo effettuato in Figure 18 i grafici anche per l'utilizzazione dei singoli nodi per effettuare i confronti con il modello analitico. Come per il throughput anche in questo caso notiamo una forte aderenza tra i due modelli sempre con il server A più discostato rispetto agli altri due server.

3.1.3 Validazione con il caso di studio. Per la validazione dell'obiettivo 1 si faccia riferimento a subsubsection 3.2.3

3.1.4 Analisi orizzonte finito. Abbiamo infine eseguito una simulazione ad orizzonte finito con 100 run da 64 job ciascuna di cui si possono vedere i risultati in Figure 19, Figure 21, Figure 20 sempre facendo il confronto con i dati ottenuti dal modello analitico, in questo caso notiamo come il modello differisce in maniera più evidente rispetto al caso dell'orizzonte finito, risultato atteso dato che sono statistiche ottenute dal sistema in stato transitorio.

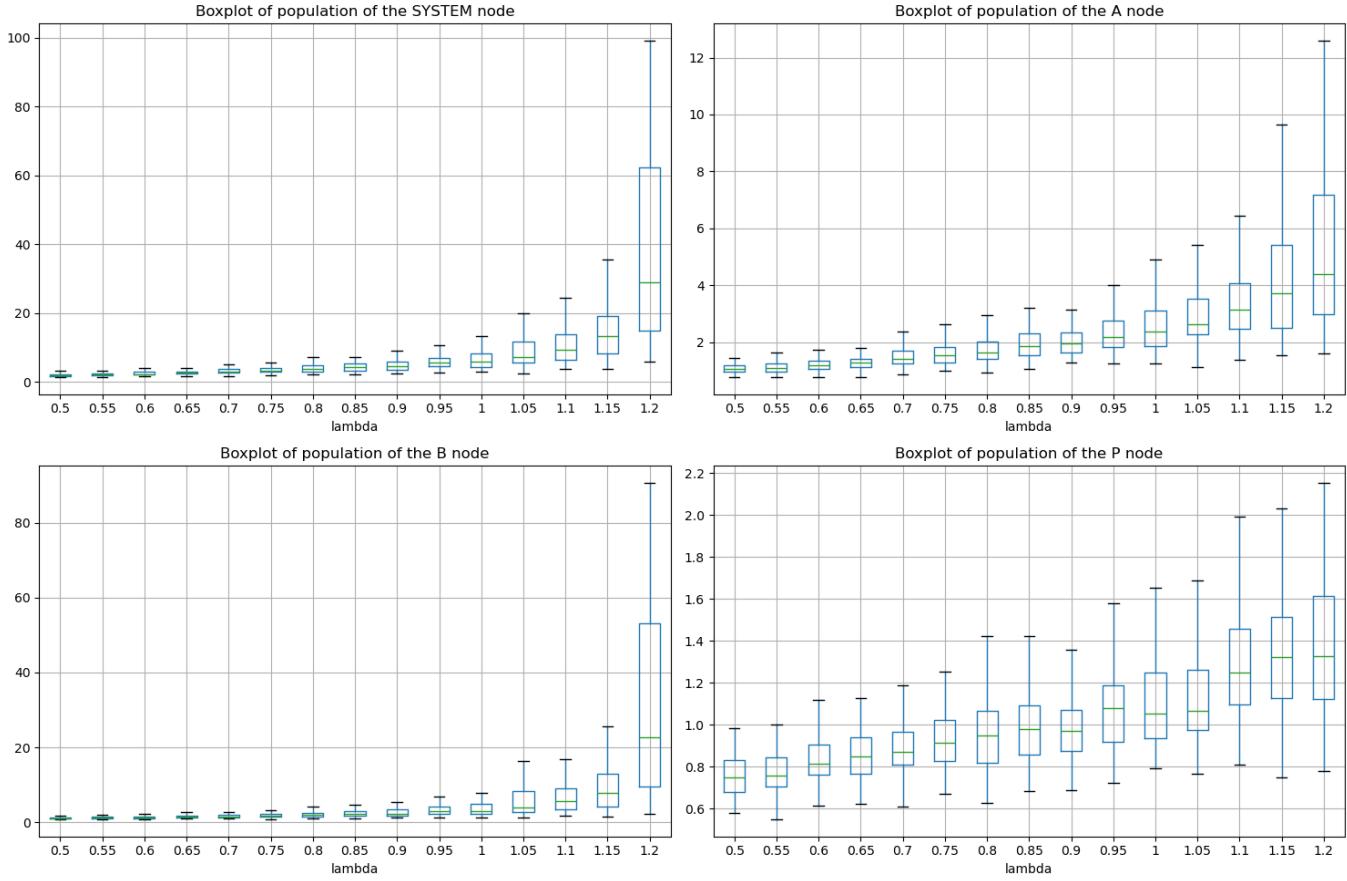


Fig. 11. Distribuzione della popolazione media dei risultati sperimentali dell'obiettivo 1.

3.2 Obiettivo 2

3.2.1 Risultati sperimentali. Figure 22 mostra la distribuzione della Popolazione media al variare del rate medio di arrivi esterni per ogni nodo del sistema e globale. È possibile notare come l'andamento assuma una forma esponenziale.

Figure 27 mostra la distribuzione del Tempo di Risposta medio al variare del rate medio di arrivi esterni per ogni nodo del sistema e globale. È possibile notare come l'andamento assuma una forma esponenziale, con una maggiore variabilità corrispondente ai rate di arrivo più alti.

Figure 24 mostra la distribuzione del Throughput medio al variare del rate medio di arrivi esterni per ogni nodo del sistema e globale. Il throughput sembra crescere linearmente all'aumentare del rate, segno che il sistema rimane stabile fino a 1.2req/s . Il throughput del sistema sembra avere una leggera flessione verso il basso verso i rate più alti, segno che il sistema, pur rimanendo stabile, si avvicina al punto di saturazione.

Figure 25 mostra la distribuzione dell'Utilizzazione media al variare del rate medio di arrivi esterni per ogni nodo del sistema e globale. L'Utilizzazione del sistema sembra crescere linearmente all'aumentare del rate, fino a colpire il tetto massimo di 1 nei rate

più alti, il che spiegherebbe l'alta variabilità del tempo di risposta in prossimità di tali rates.

3.2.2 Verifica con il modello analitico. Figure 26, Figure 27, Figure 28, e mostrano l'allineamento dei valori sperimentali di Popolazione, Tempo di Risposta e Throughput medi con quelli teorici forniti dal modello analitico. È opportuno notare che il modello analitico è stato definito modellando il Server A senza tener conto che job di classe diversa hanno tempi medi di servizio diversi, per cui è possibile notare delle discrepanze soprattutto per le metriche del Server A. Infatti, l'effetto finale è che il modello analitico risulta essere "ottimista" rispetto ai risultati di simulazione.

Figure 29 mostra l'allineamento dei valori sperimentali dell'Utilizzazione media con quelli teorici forniti dal modello analitico dei server A, B, P. In Figure 29a si vede ancora una volta come il modello analitico sia ottimista rispetto ai valori sperimentali.

3.2.3 Validazione con il caso di studio. Figure 30 riporta i valori medi del Tempo di Risposta e della Popolazione ottenuti tramite il nostro simulatore confrontati con i valori riportati da Serazzi [2024]. È possibile notare come in entrambi i risultati le metriche di performance siano più alte con la two-factor authentication rispetto alla

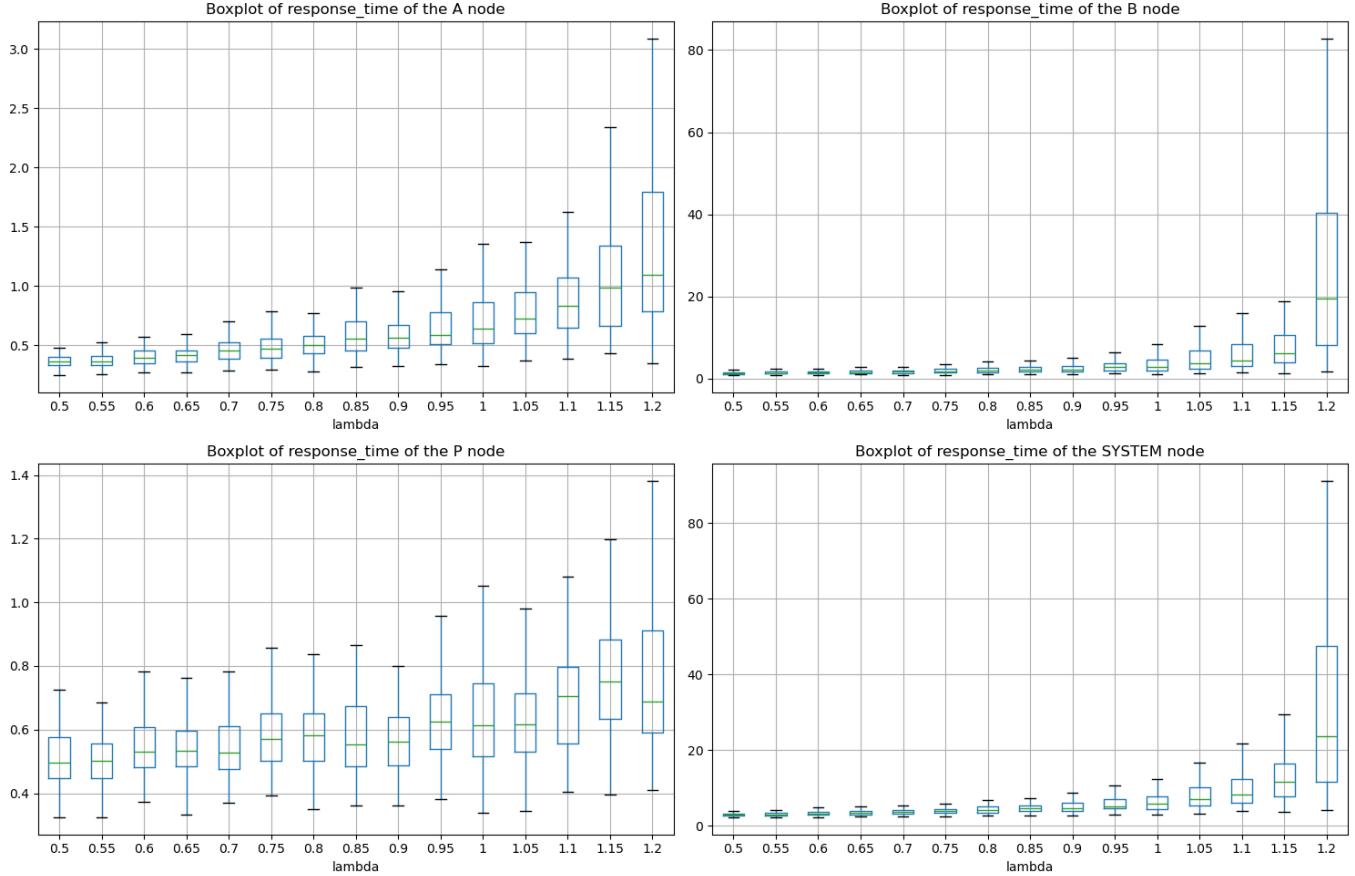


Fig. 12. Distribuzione dei tempi di risposta medi dei risultati sperimentali dell'obiettivo 1.

single-factor. Inoltre, gli andamenti sembrano essere tutti esponenziali. I risultati sembrano essere in linea con quanto riportato nel caso di studio.

3.3 Obiettivo 3

3.3.1 Risultati sperimentali. Figure 31 mostra la distribuzione della Popolazione media al variare del rate di arrivi esterni. È possibile notare come i valori esplosano al superamento di $\lambda = 1.2 \text{req/s}$, valore oltre il quale il sistema risulta instabile. Ciò provoca anche un'esplosione dei tempi di risposta, visibile in Figure 32.

Figure 33 mostra come il Throughput globale medio non vada oltre un upper bound posto tra 1.2 e 1.4req/s , ulteriore conferma del fatto che il sistema non riesca a gestire efficacemente uno scenario di carico così pesante.

Figure 34 mostra che di fatti è il server B a soffrire di più il carico poiché la sua utilizzazione diventa fissa a 1 per tassi di arrivo oltre 1.2req/s . Il suo throughput è anche tra i più bassi tra i server, come è visibile in Figure 35.

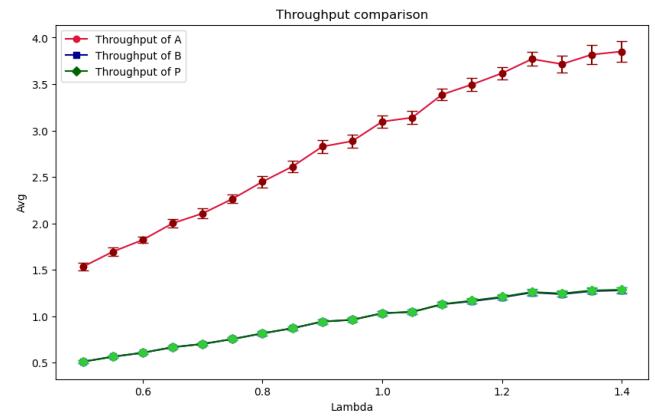


Fig. 35. Confronto dei Throughput medi tra i Server della Web App per l'obiettivo 3.

3.3.2 Verifica con il modello analitico. Il modello analitico è analogo a quanto riportato in subsection 3.2.2 fino a $\lambda = 1.2 \text{req/s}$, oltre il quale non si hanno a disposizione dati analitici poiché il modello assume un regime di stazionarietà.

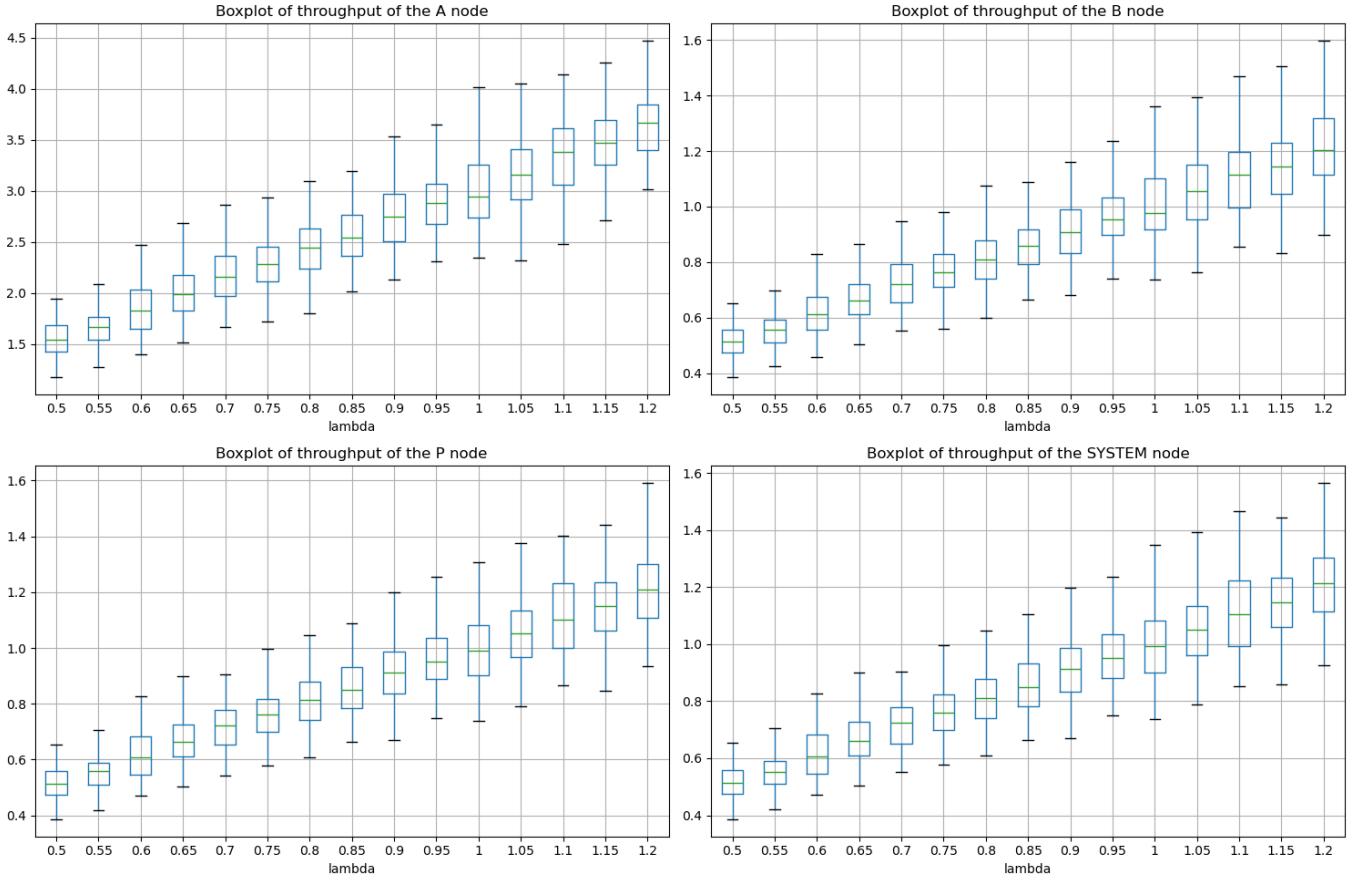


Fig. 14. Distribuzione del throughput medio dei risultati sperimentali dell'obiettivo 1.

3.3.3 Validazione con il caso di studio. Per la validazione dei risultati si faccia riferimento a subsubsection 3.4.3.

3.4 Obiettivo 4

3.4.1 Risultati sperimentali. Con l'obiettivo di migliorare le performance del sistema in modo che possa supportare il nuovo carico pesante abbiamo innanzitutto individuato il collo di bottiglia nel server B. Questo risultato era già evidente dai grafici dei precedenti obiettivi, confermato dal fatto che il bound per il throughput è:

$$\frac{1}{D_{max}}$$

dove D_{max} = domanda massima

Per il server B è 1.25 job/s quindi non può soddisfare il carico di 1.4 job/s richiesto. Inoltre dal seguente grafico in Figure 35 si nota come i throughput più bassi siano del server B e P. Siccome non si può agire sul server P in quanto servizio esterno al sistema andiamo ad operare sul server B.

Il miglioramento al sistema consiste nella sostituzione del server B con uno più performante. A tale scopo abbiamo selezionato tempi di servizio a partire da 0.8 s (attuale tempo di servizio) decrementando di 0.05 s fino ad arrivare a 0.4 s ed effettuando per ogni valore una run

Table 3. Minimi e massimi di utilizzazione e throughput per tempi di servizio 0.4 del server B

λ	metrica	valore
0.5	utilizzazione	0.20484
1.4	utilizzazione	0.574275
0.5	throughput	0.516347
1.4	throughput	1.40302

per ogni tasso di arrivo. Per ogni run abbiamo messo a confronto utilizzazione in Figure 36a e throughput in Figure 36b.

Dai grafici si nota come l'utilizzazione decresca notevolmente al aumentare del rate di servizio, mentre per i throughput la differenza è meno netta.

Abbiamo così deciso di riportare i risultati di due modelli diversi. Il primo è il più efficiente ovvero con tempi di servizio medi di 0.4 s ed è la stessa soluzione adottata dal caso di studio di riferimento che per throughput e utilizzazione riporta i seguenti valori minimi e massimi come si può vedere in Table 3: Per quanto riguarda il secondo modello abbiamo valutato che la spesa da sostenere per comprare un server in grado di dimezzare i tempi di servizio può

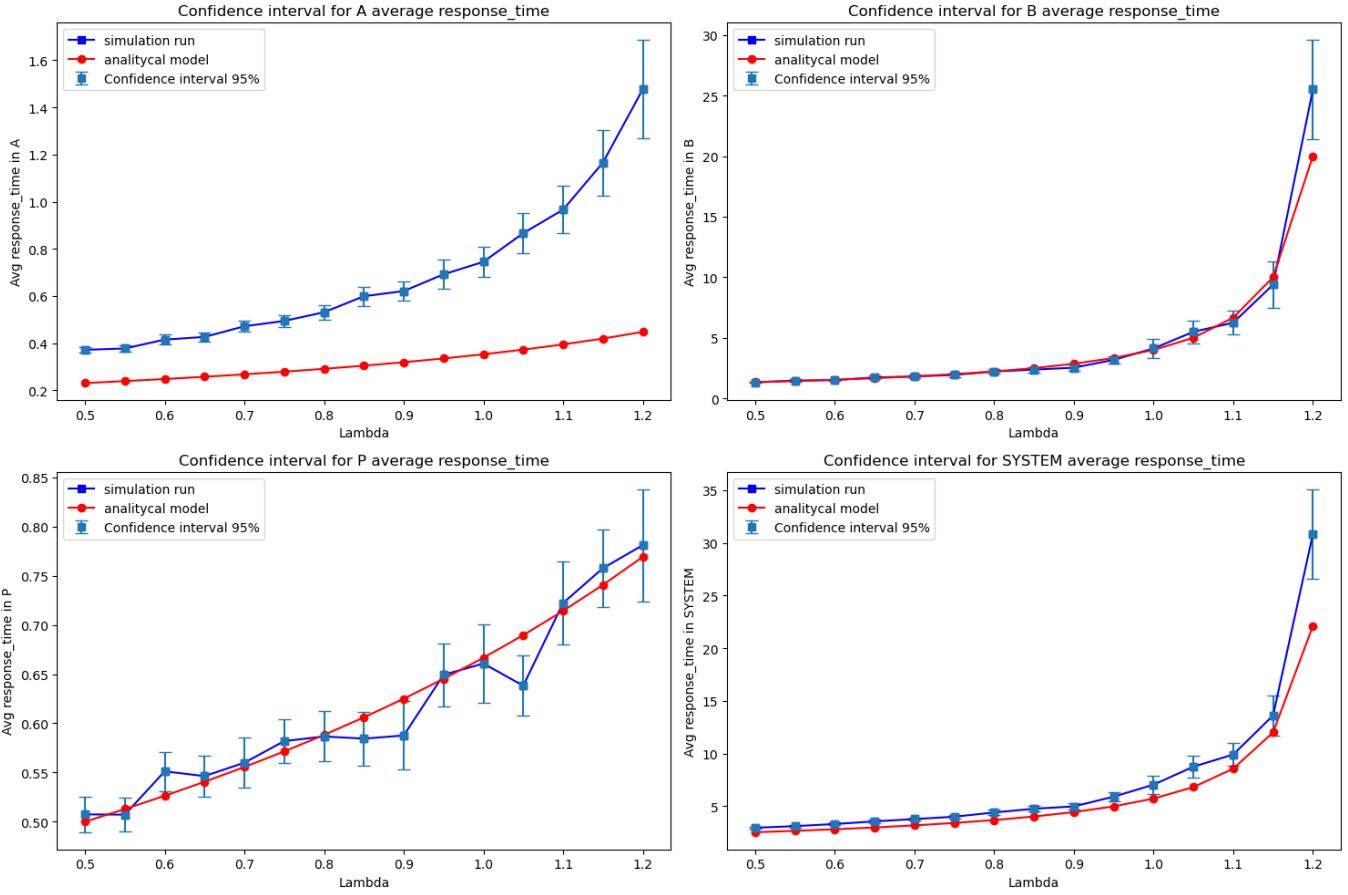


Fig. 15. Intervallo di confidenza del tempo di risposta medio e confronto con modello analitico per l’obiettivo 1

Table 4. Minimi e massimi di utilizzazione e throughput per tempi di servizio 0.65 del server B

λ	metrica	valore
0.5	utilizzazione	0.333898
1.4	utilizzazione	0.921304
0.5	throughput	0.507752
1.4	throughput	1.40273

essere notevole, quando potrebbe bastare un altro server anche di poco più performante. In tale ottica abbiamo scelto il modello con tempi di servizio 0.65s ottenendo un throughput massimo è di 1.4 job/s. Abbiamo ottenuto i risultati minimi e massimi riportati in Table 4: Mentre per l’utilizzazione le prestazioni degradano notevolmente, il throughput varia molto poco, possiamo quindi considerarla comunque una valida miglioria. Rappresenta inoltre, tra quelle considerate, la prima configurazione che soddisfi i requisiti, infatti già per tempi di servizio di 0.70s il throughput massimo è 1.38 job/s e l’utilizzazione 0.98.

3.4.2 Verifica con il modello analitico. Per la verifica abbiamo scelto comunque il modello con tempo di servizio medi 0.4s. Riportiamo

i grafici che confrontano l’andamento con il modello analitico per il throughput in Figure 37 e l’utilizzazione in Figure 38. Notiamo come la differenza tra i due modelli sia poca, risultato che verifica la correttezza del modello.

3.4.3 Validazione con il caso di studio. Per la validazione facciamo riferimento al modello con tempo medio di servizio 0.4s in quanto è l’unico con cui possiamo effettuare un confronto con i risultati riportati dal caso di studio. Seguono i grafici dove vengono messe a confronto le due configurazioni, quella originale e quella migliorata nel nostro modello di simulazione Figure 39a e nel caso di studio Figure 39b

REFERENCES

- Stephen K. Park Lawrence M. Leemis. 2004. *Discrete-Event Simulation: A First Course*. Prentice Hall. <https://www.math.wm.edu/~leemis/>
Giuseppe Serazzi. 2024. *Performance Engineering - Learning Through Applications Using JMT*. Springer. <https://doi.org/10.1007/978-3-031-36763-2>

A APPENDICE: STUDIO DELLO STATO TRANSITORIO
In questa appendice sono riportati i risultati per lo studio dello stato transitorio, divisi per obiettivi. Per ogni obiettivo, dato un set di 5 seed diversi, è stata simulata una replica per ogni seed

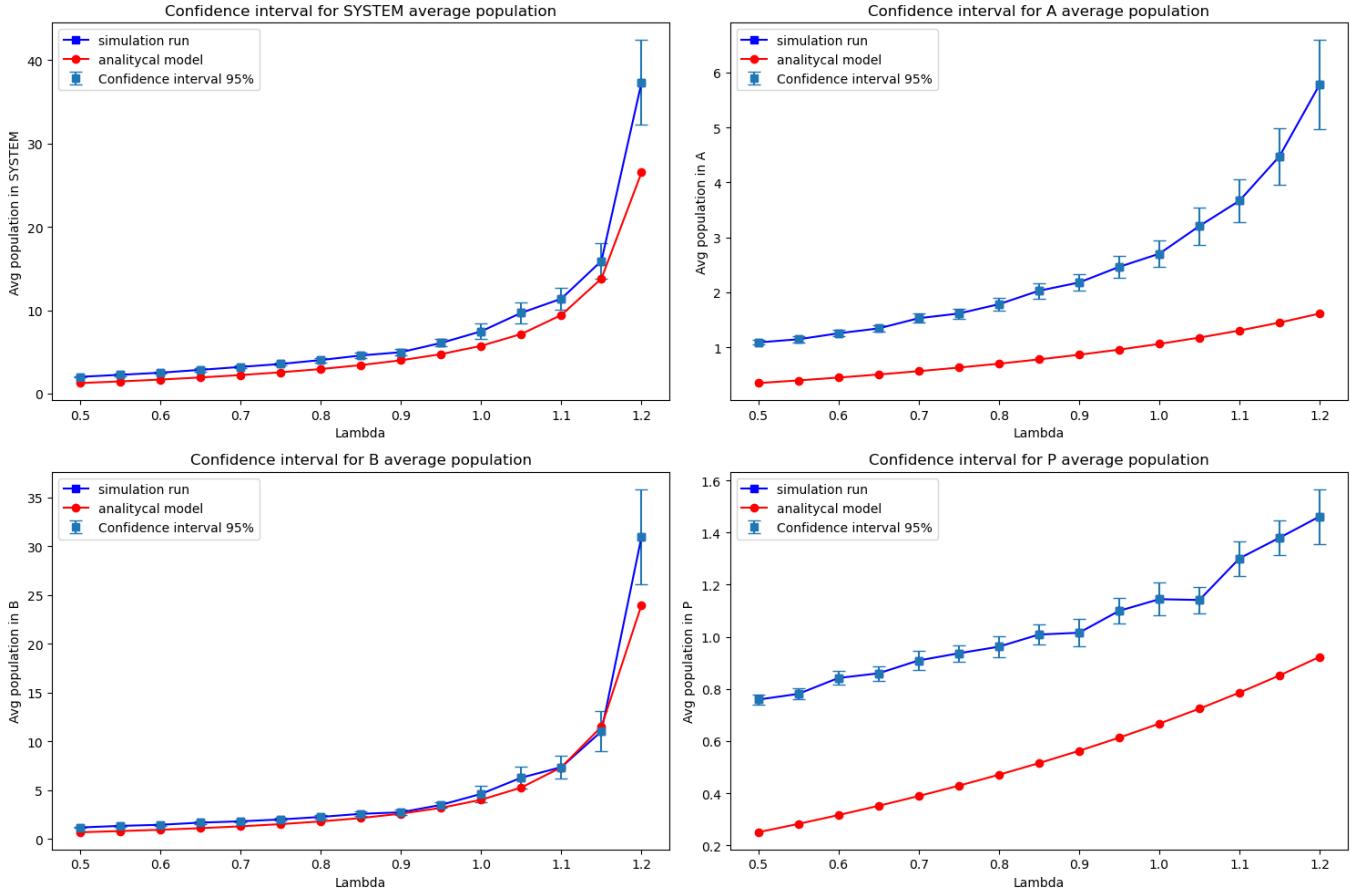


Fig. 16. Intervallo di confidenza della popolazione media e confronto con modello analitico per l'obiettivo 1

con il rate di arrivi più alto previsto per poter studiarne il comportamento nel contesto di stress maggiore. Ricordiamo che i valori tabellari dei risultati esposti in questa sezione sono disponibili a https://github.com/caballo-domestico/webapp-workflow-pa/blob/main/docs/plots/model_plots.ipynb.

A.1 Obiettivo 1

In Figure 40 è riportato il valore del Tempo di Risposta medio al variare del tempo di simulazione e del seed iniziale, con un rate di arrivi medio di 1.2 job/s . Il valore, nonostante cresca notevolmente entro i primi momenti della simulazione, sembra assestarsi verso un valore compreso tra 1.3 e 0.5s per il Server A. Inoltre, sembra che il server A riceva la maggior parte degli arrivi entro i primi 25 secondi di simulazione, per poi smaltire le richieste accumulate. Il Tempo di risposta per il sistema sembra crescere continuamente, per quattro dei cinque seed di simulazione. I server A e P sembrano avvicinarsi ai corrispettivi valori teorici, mentre B trascina l'intero sistema lontano da essi.

A.2 Obiettivo 2

In Figure 41 è riportato il valore del Tempo di Risposta medio al variare del tempo di simulazione e del seed iniziale, con un rate di arrivi medio di 1.2 job/s . L'aumento dei tempi di servizio sembra comportare una maggiore impennata dei tempi di risposta di P, con conseguente aumento dei tempi di risposta del sistema. Tuttavia, anche qui i server A e P sembrano tendere ai corrispettivi valori teorici.

A.3 Obiettivo 3

In Figure 42 è riportato il valore del Tempo di Risposta medio al variare del tempo di simulazione e del seed iniziale, con un rate di arrivi medio di 1.4 job/s , che in questo scenario rappresenta una situazione di instabilità del sistema e per questo non è disponibile un valore teorico di riferimento. Il tempo di risposta medio del server B sembra crescere esponenzialmente, e causa dei valori instabili del tempo di risposta medio del sistema.

A.4 Obiettivo 4

In Figure 43 e Figure 44 è riportato il valore del Tempo di Risposta medio al variare del tempo di simulazione e del seed iniziale, con

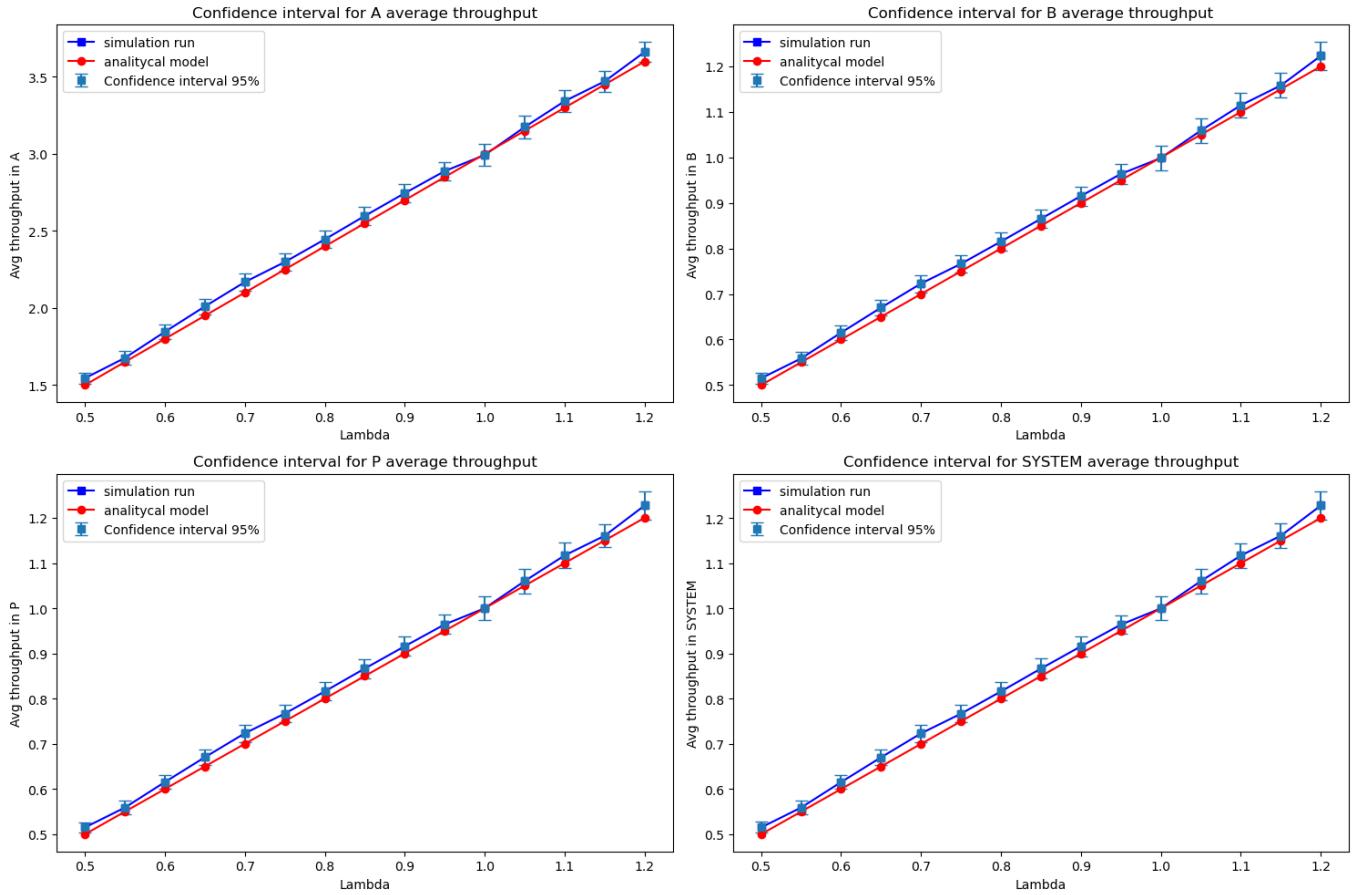


Fig. 17. Intervallo di confidenza del throughput medio e confronto con modello analitico per l'obiettivo 1

un rate di arrivi medio di $1.4 job/s$, rispettivamente con un tempo di servizio medio per il server B di 0.4 e 0.65s. Sebbene entrambi i casi sembrino mostrare situazioni di stabilità, il caso con 0.65s

sembra avere ragionevolmente valori più alti. La Figure 44b sembra mostrare un avvicinamento maggiore ai valori teorici rispetto a Figure 43b.

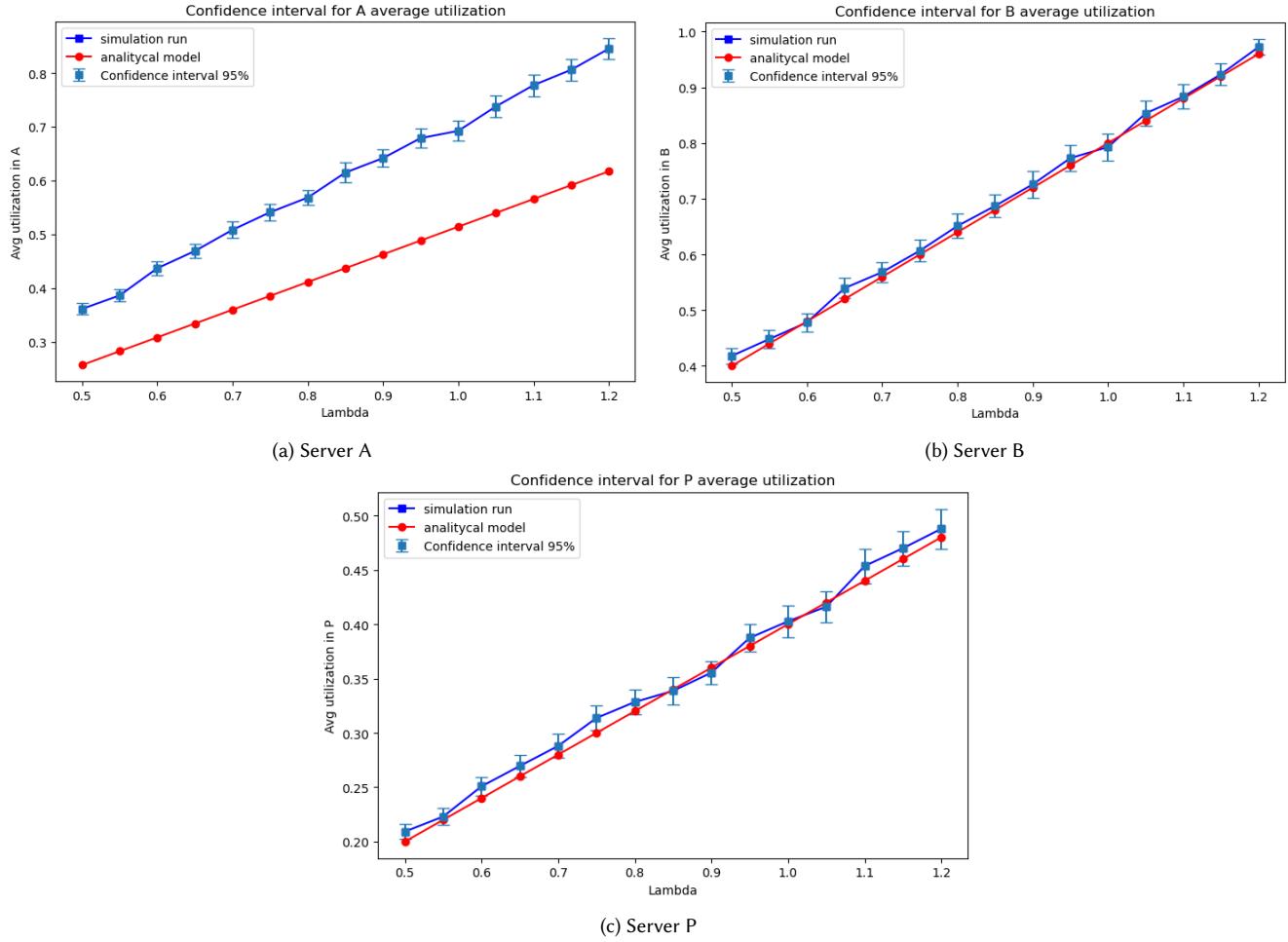


Fig. 18. Intervallo di confidenza dell'utilizzazione e confronto con modello analitico per l'obiettivo 1.

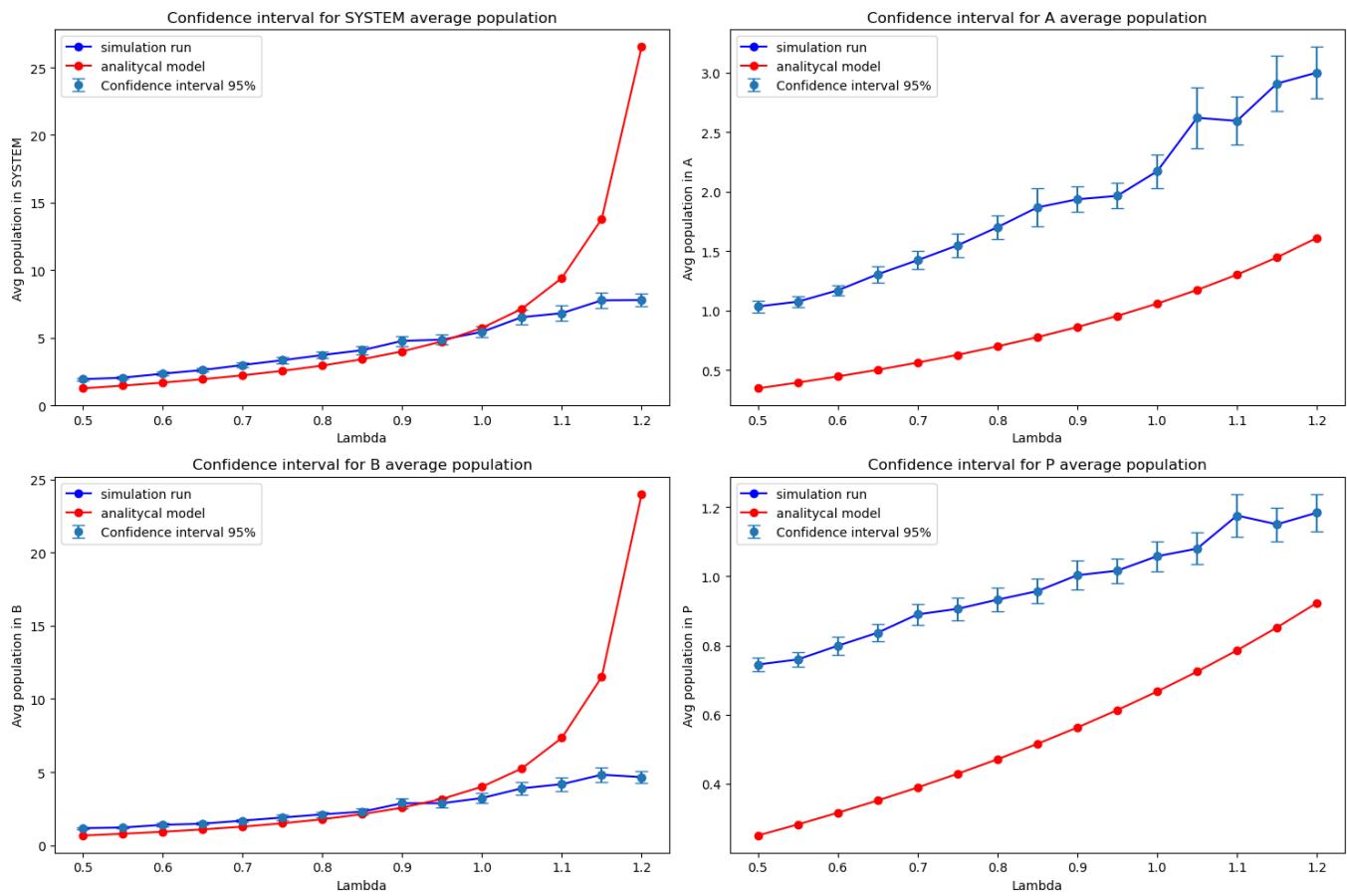


Fig. 19. Intervallo di confidenza della popolazione media e confronto con modello analitico per l'obiettivo 1 ad orizzonte finito.

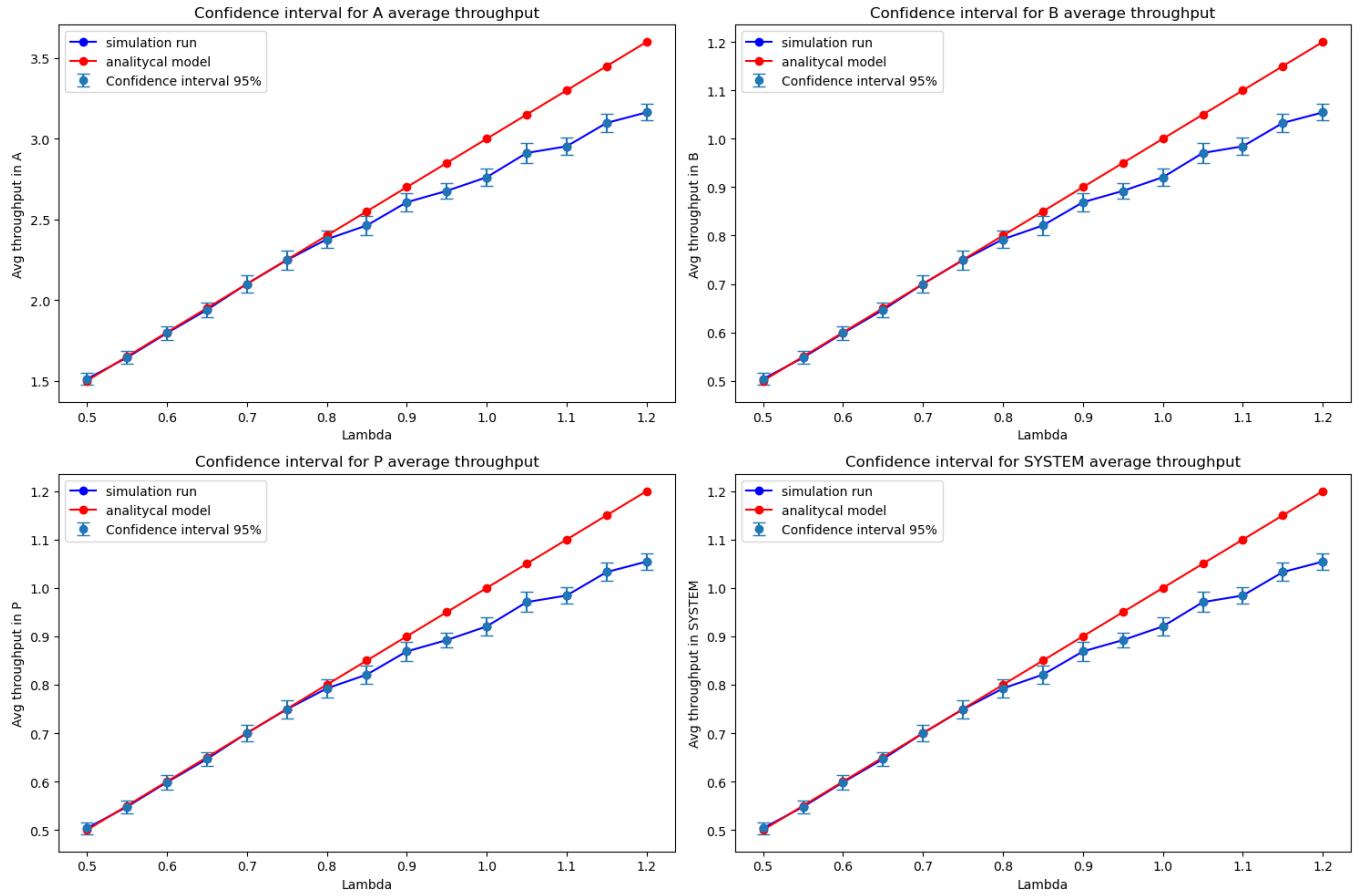


Fig. 20. Intervallo di confidenza del throughput medio e confronto con modello analitico per l'obiettivo 1 ad orizzonte finito.

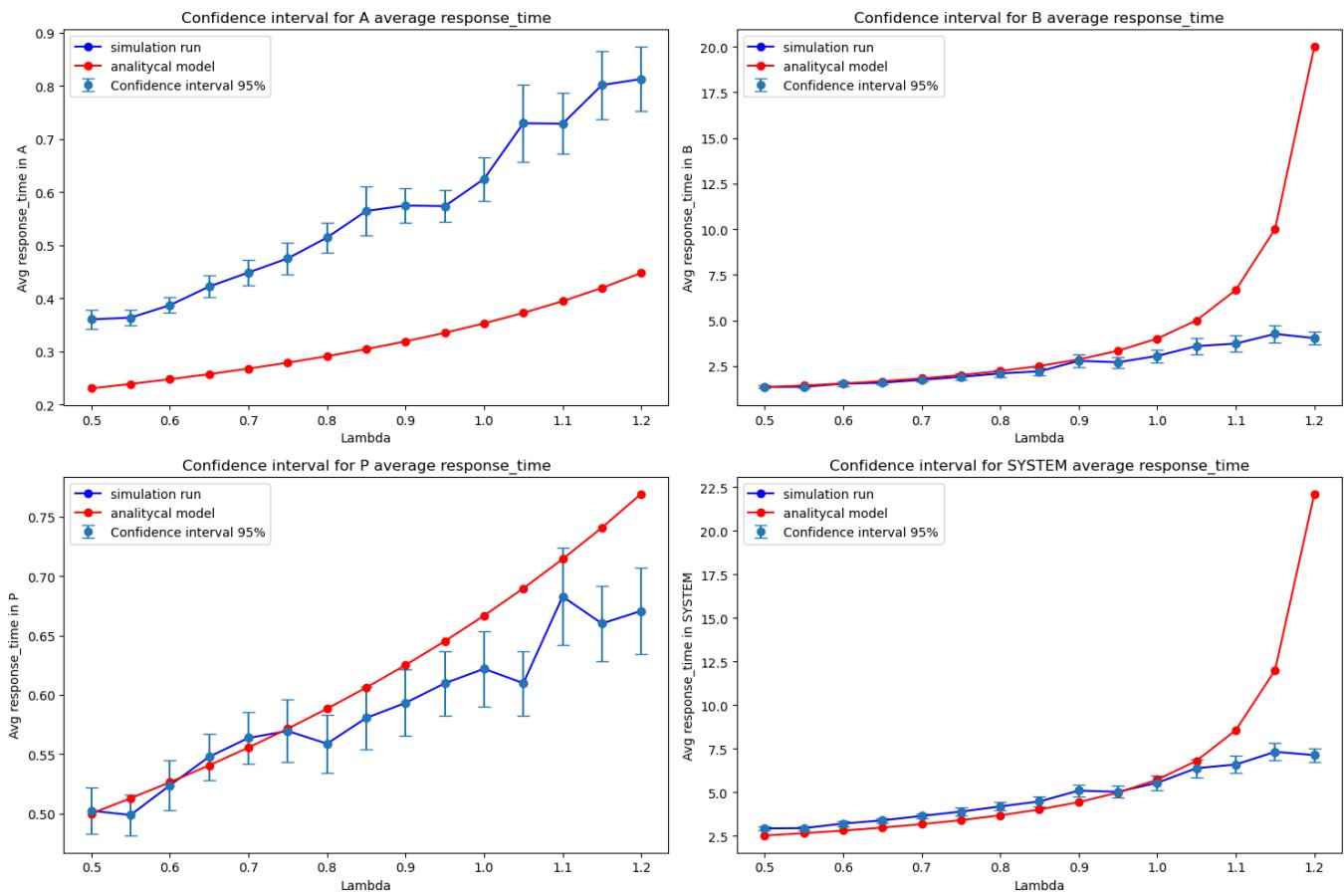


Fig. 21. Intervallo di confidenza del tempo di risposta medio e confronto con modello analitico per l'obiettivo 1 ad orizzonte finito.

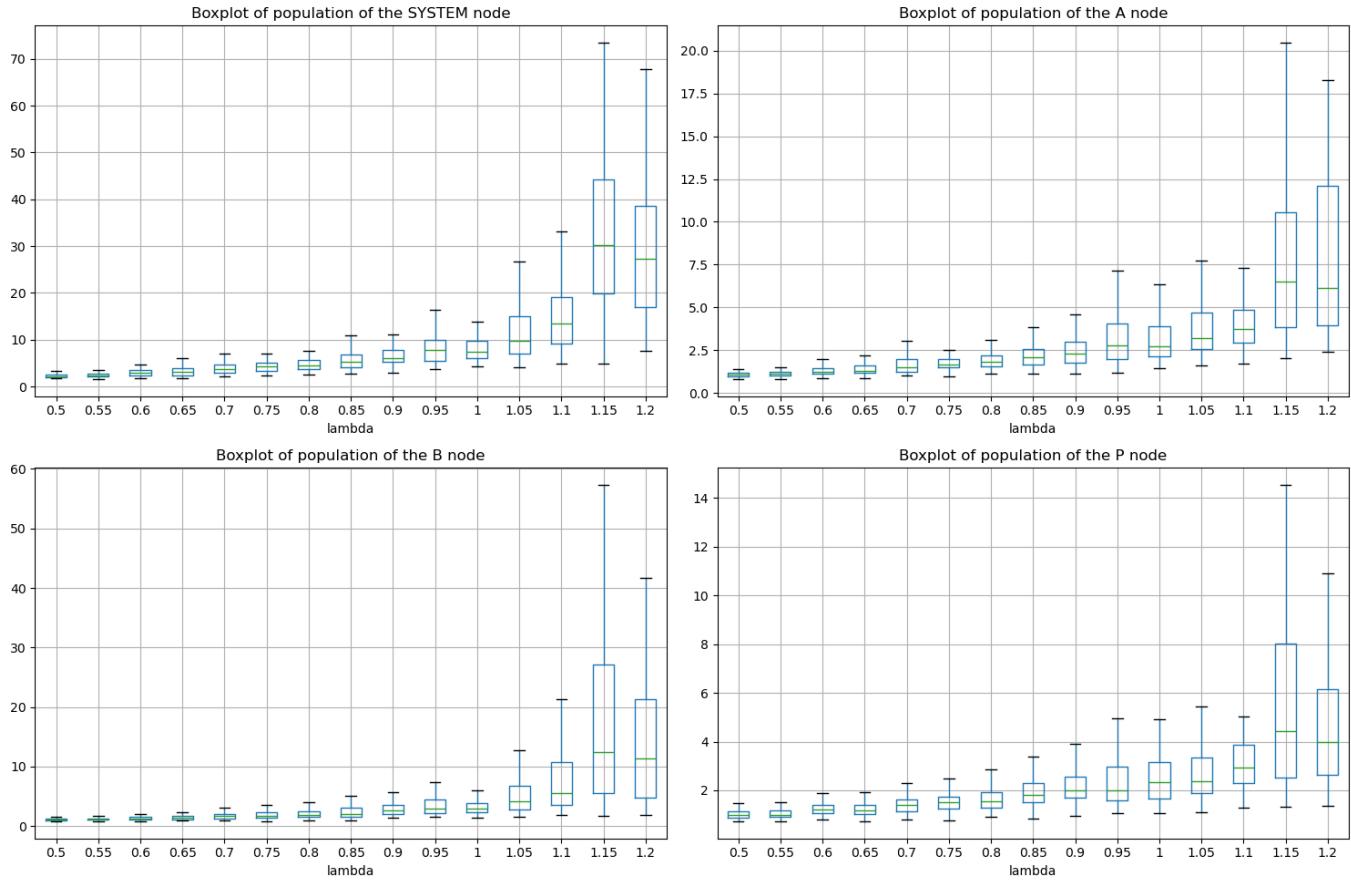


Fig. 22. Distribuzione della popolazione media dei risultati sperimentali dell'obiettivo 2.

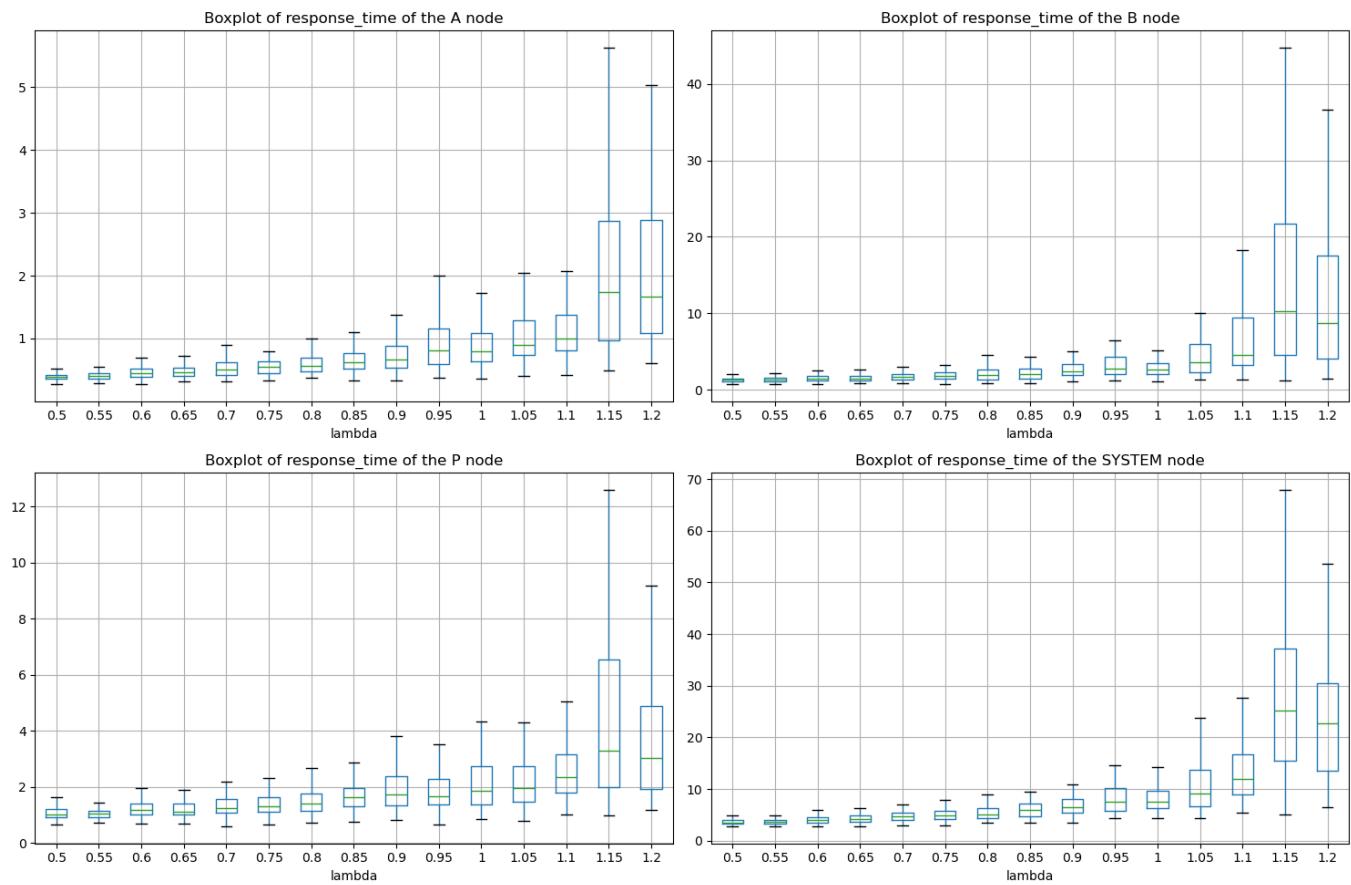


Fig. 23. Distribuzione del Tempo di Risposta medio dei risultati sperimentali dell'obiettivo 2.

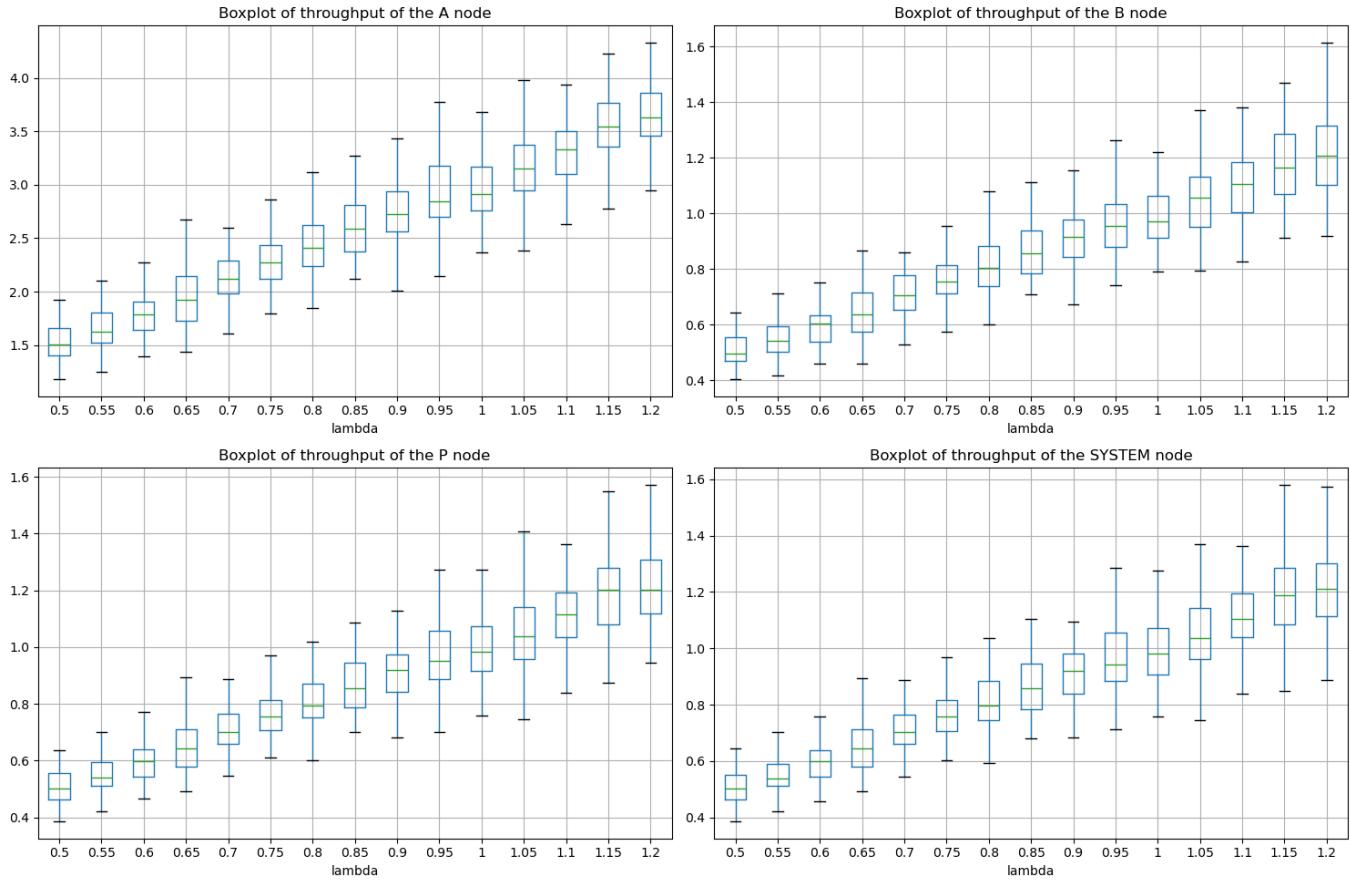


Fig. 24. Distribuzione del Throughput medio dei risultati sperimentali dell'obiettivo 2.

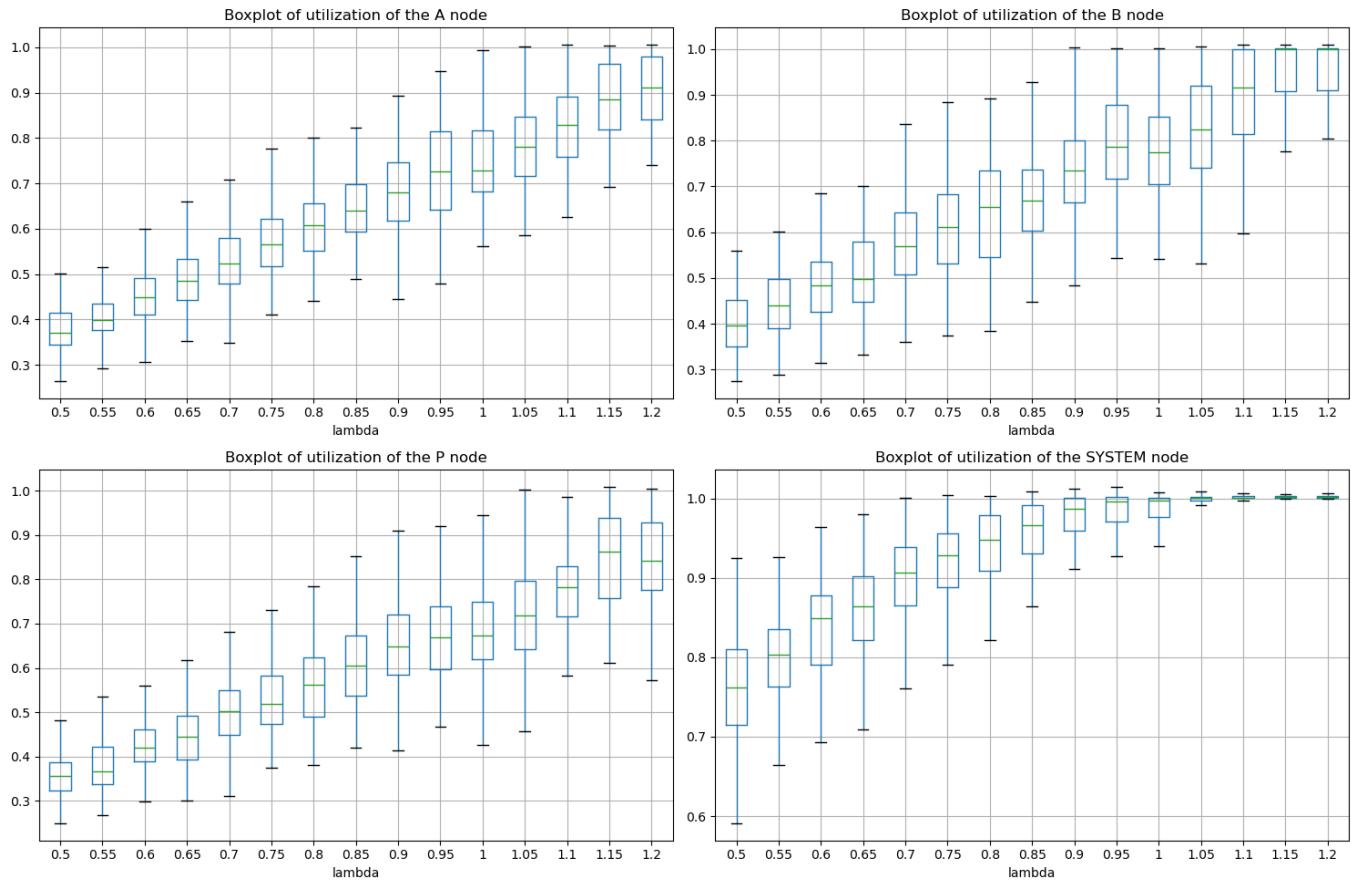


Fig. 25. Distribuzione dell'Utilizzazione media dei risultati sperimentali dell'obiettivo 2.

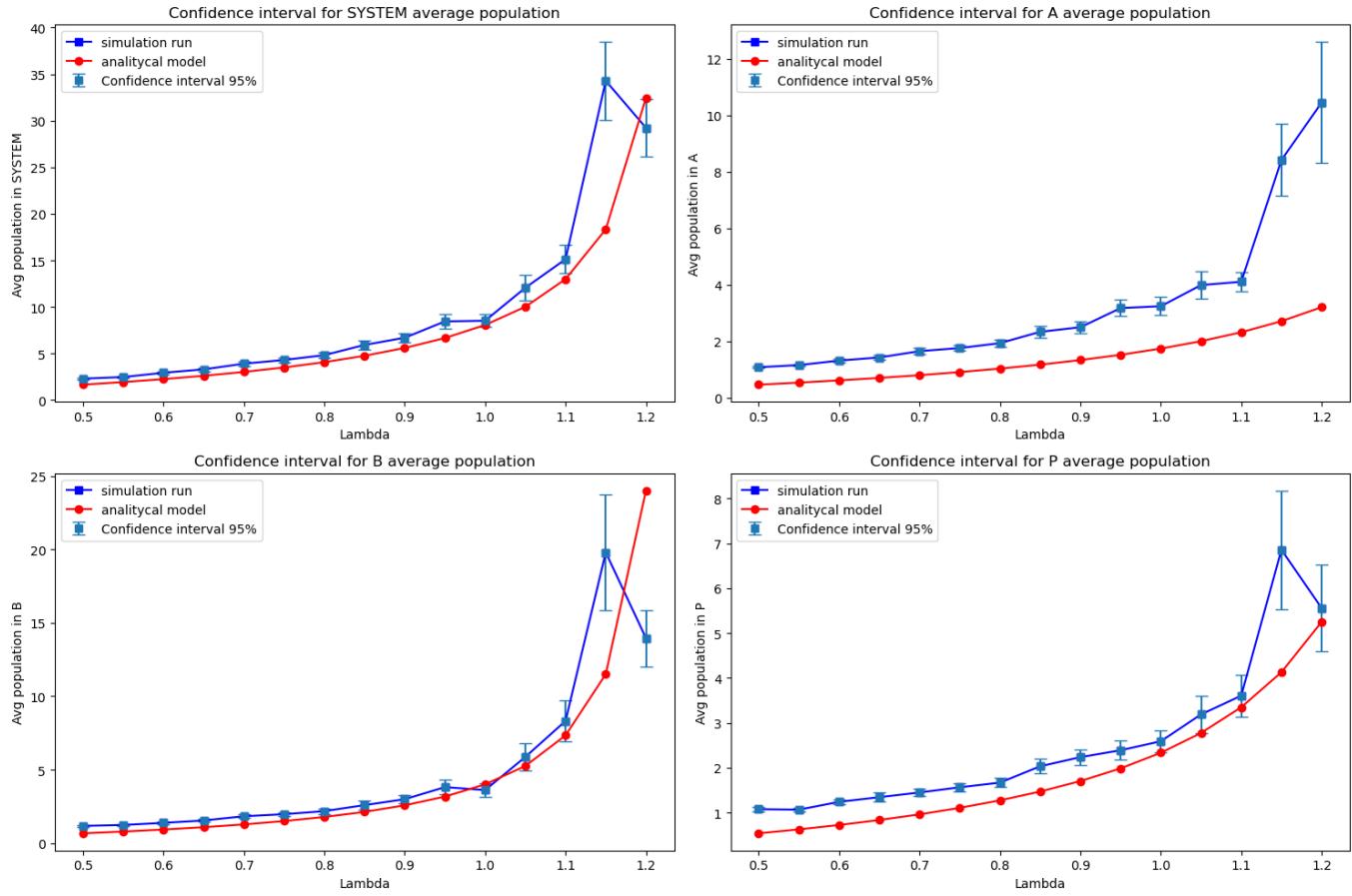


Fig. 26. Confronto tra valori medi della simulazione e del modello analitico della Popolazione per l'Obiettivo 2.

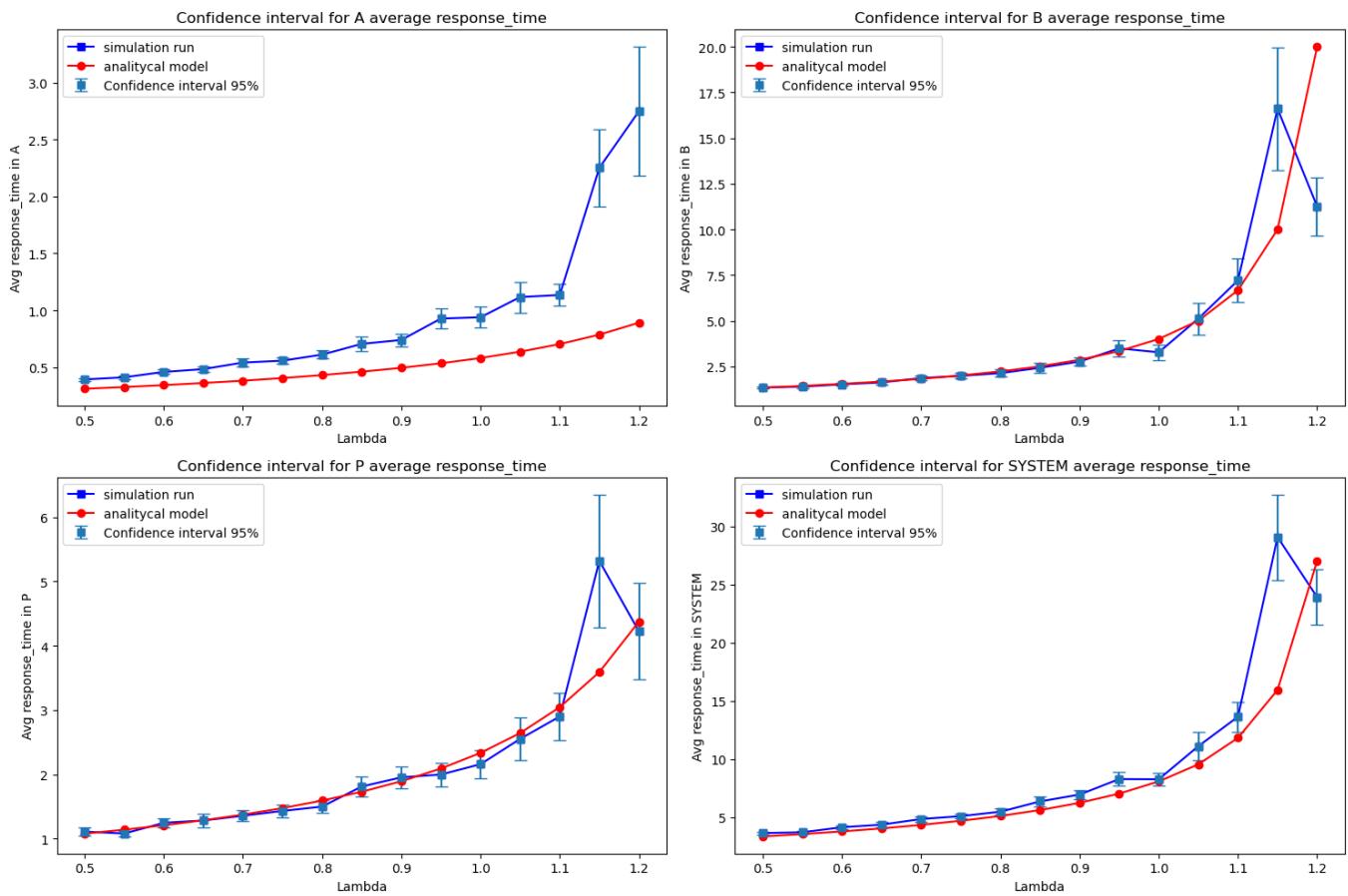


Fig. 27. Confronto tra valori medi della simulazione e del modello analitico dell'Tempo di Risposta per l'Obiettivo 2.

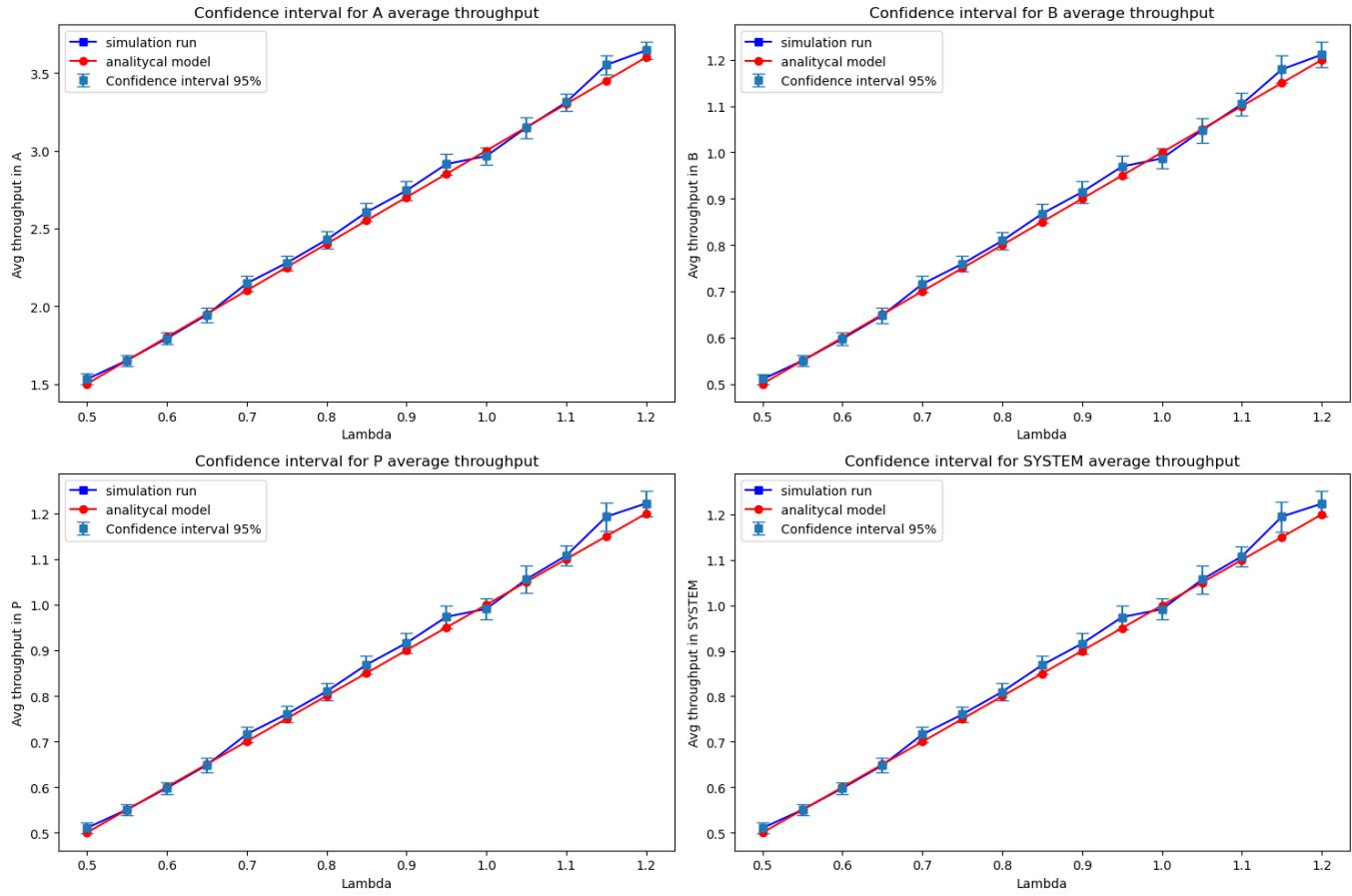


Fig. 28. Confronto tra valori medi della simulazione e del modello analitico del Throughput per l'Obiettivo 2.

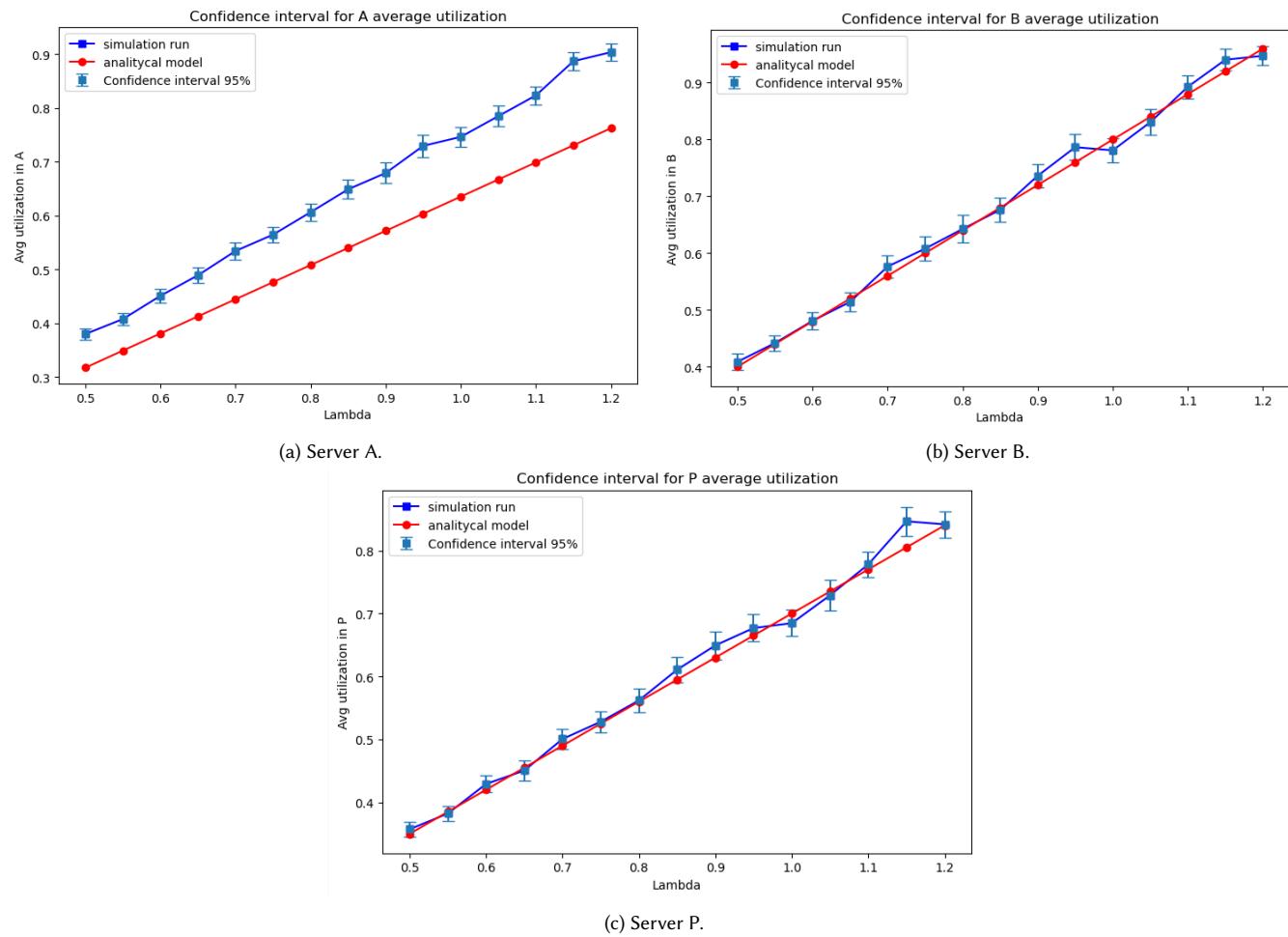


Fig. 29. Confronto tra valori medi della simulazione e del modello analitico dell'Utilizzazione per l'Obiettivo 2.

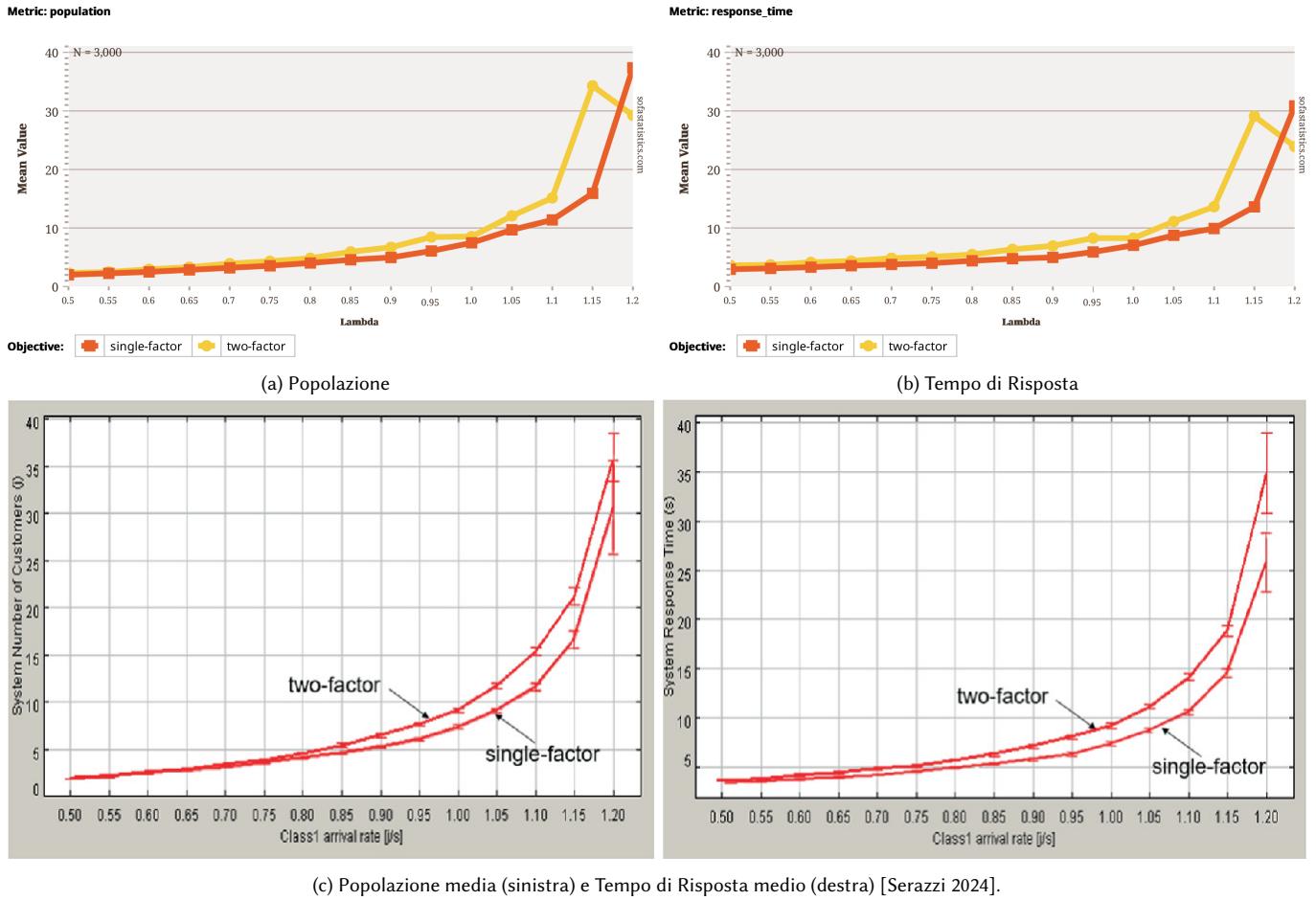


Fig. 30. Popolazione e Tempo di Risposta medi del sistema in funzione del rate di arrivi esterni nel caso con e senza 2FA.

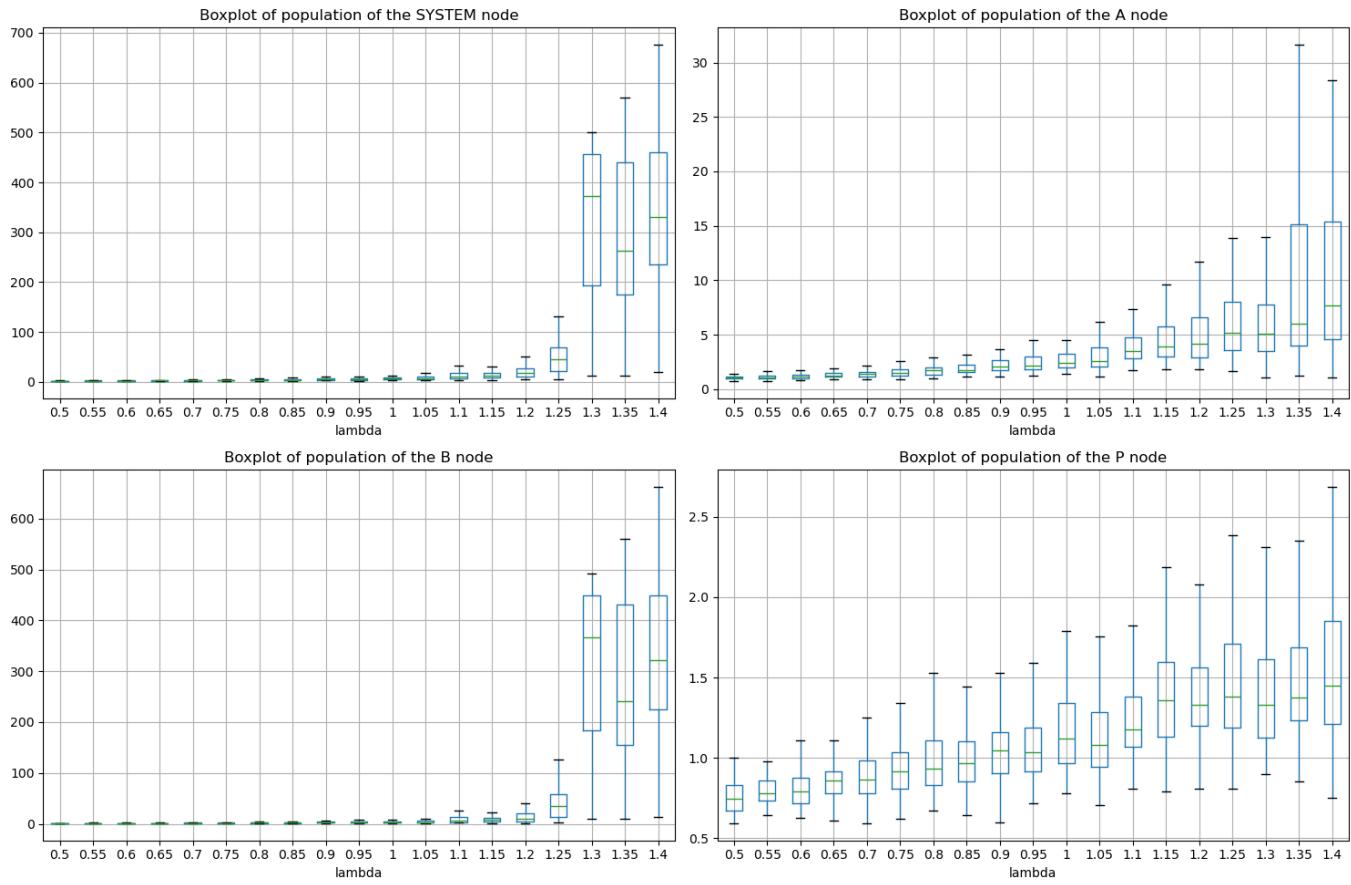


Fig. 31. Distribuzione della Popolazione media dei risultati sperimentali dell'obbiettivo 3.

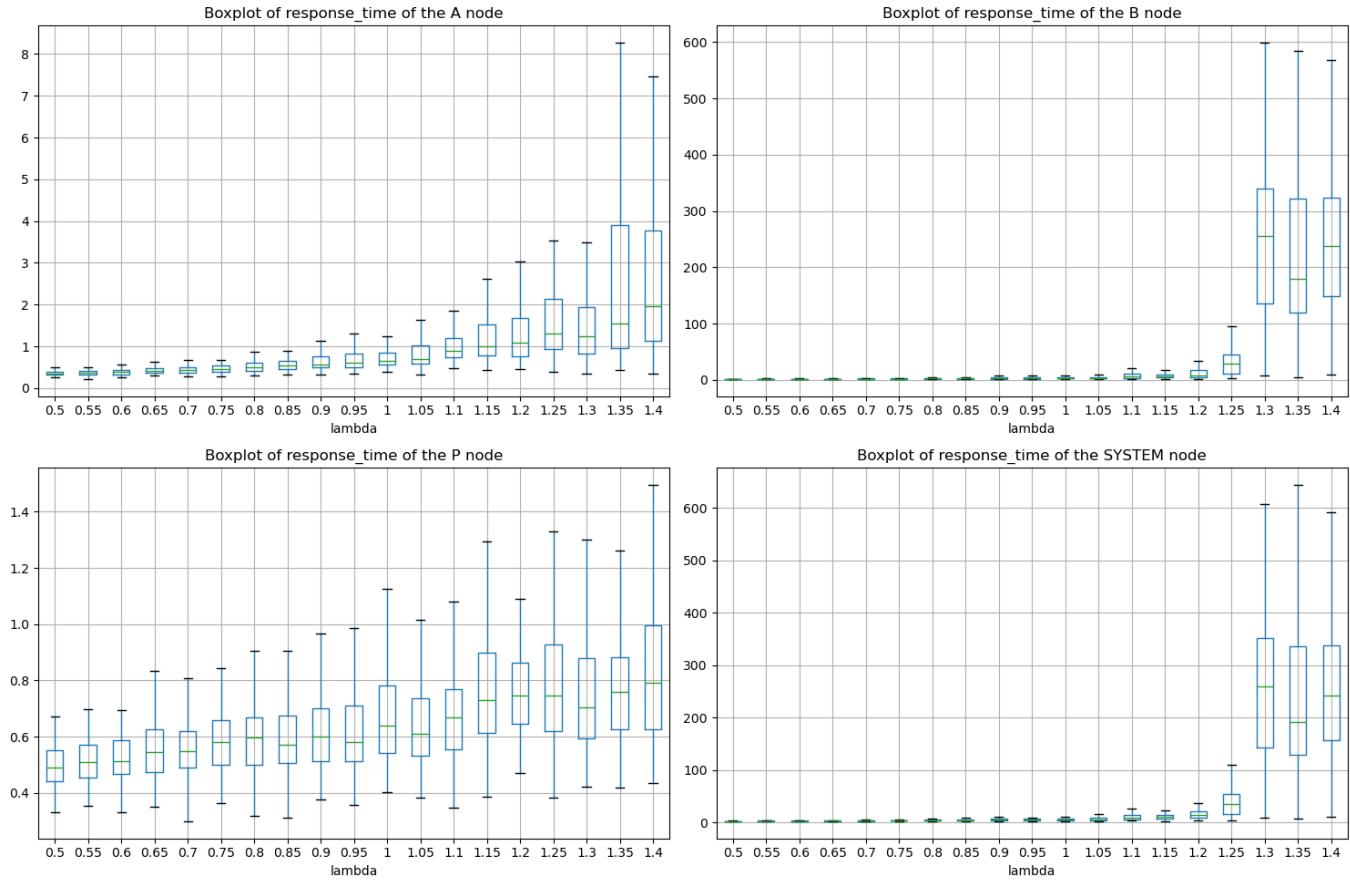


Fig. 32. Distribuzione del Tempo di Risposta medio dei risultati sperimentali dell'obiettivo 3.

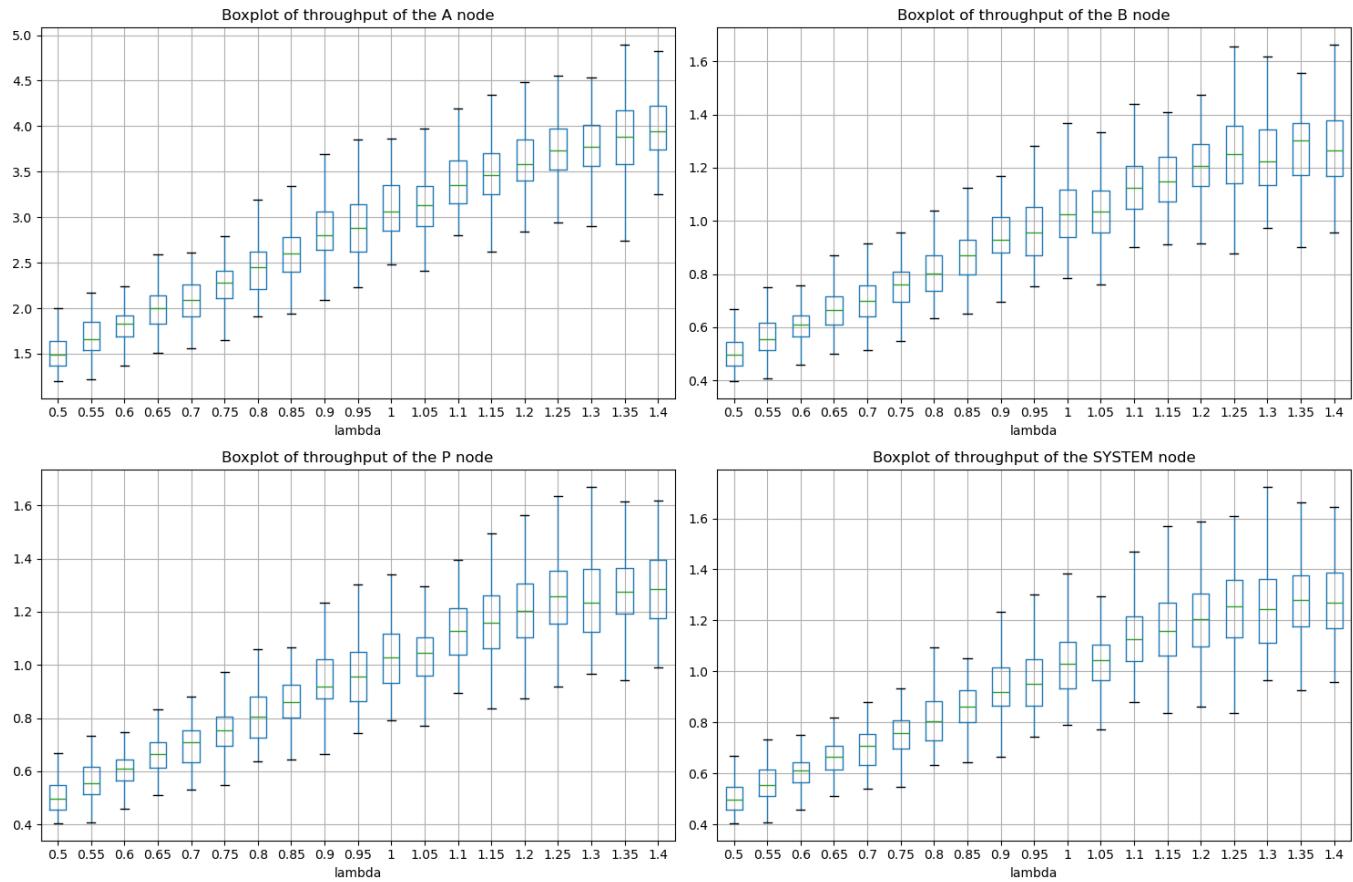


Fig. 33. Distribuzione del Throughput medio dei risultati sperimentali dell'obiettivo 3.

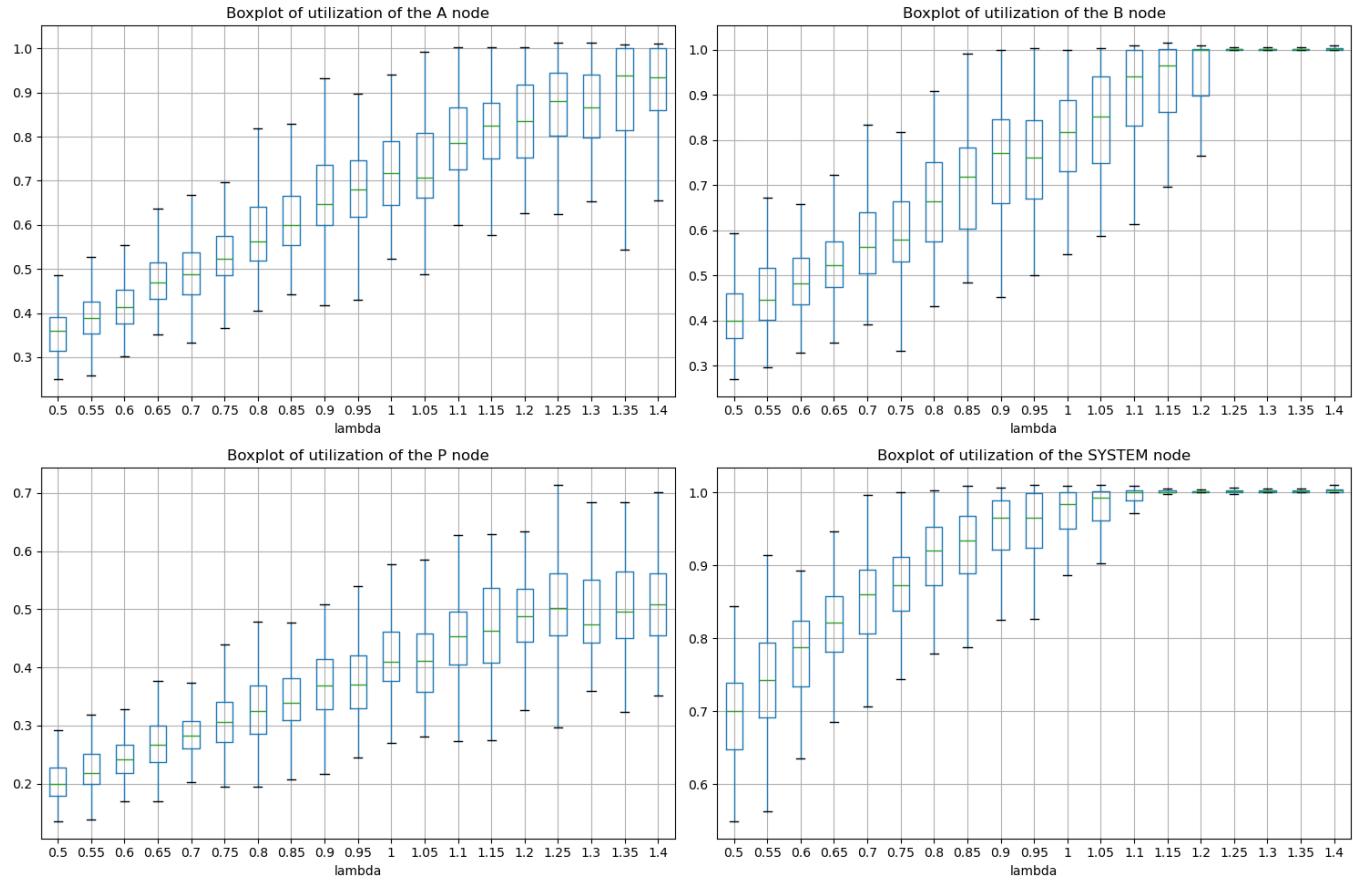


Fig. 34. Distribuzione dell'utilizzazione media dei risultati sperimentali dell'obiettivo 3.

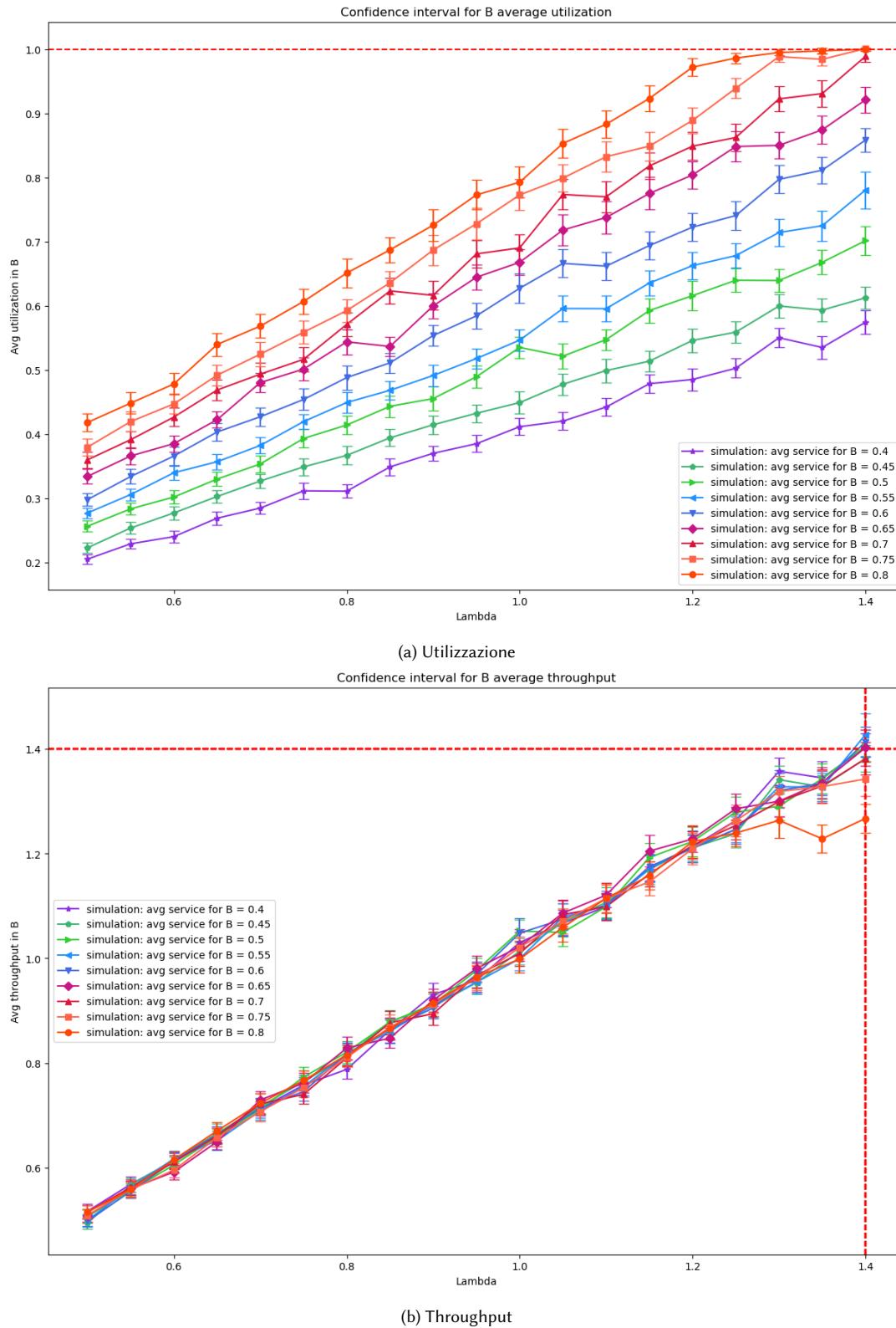


Fig. 36. Confronto Throughput e Utilizzazione medi al variare dei valori dei tempi di servizio

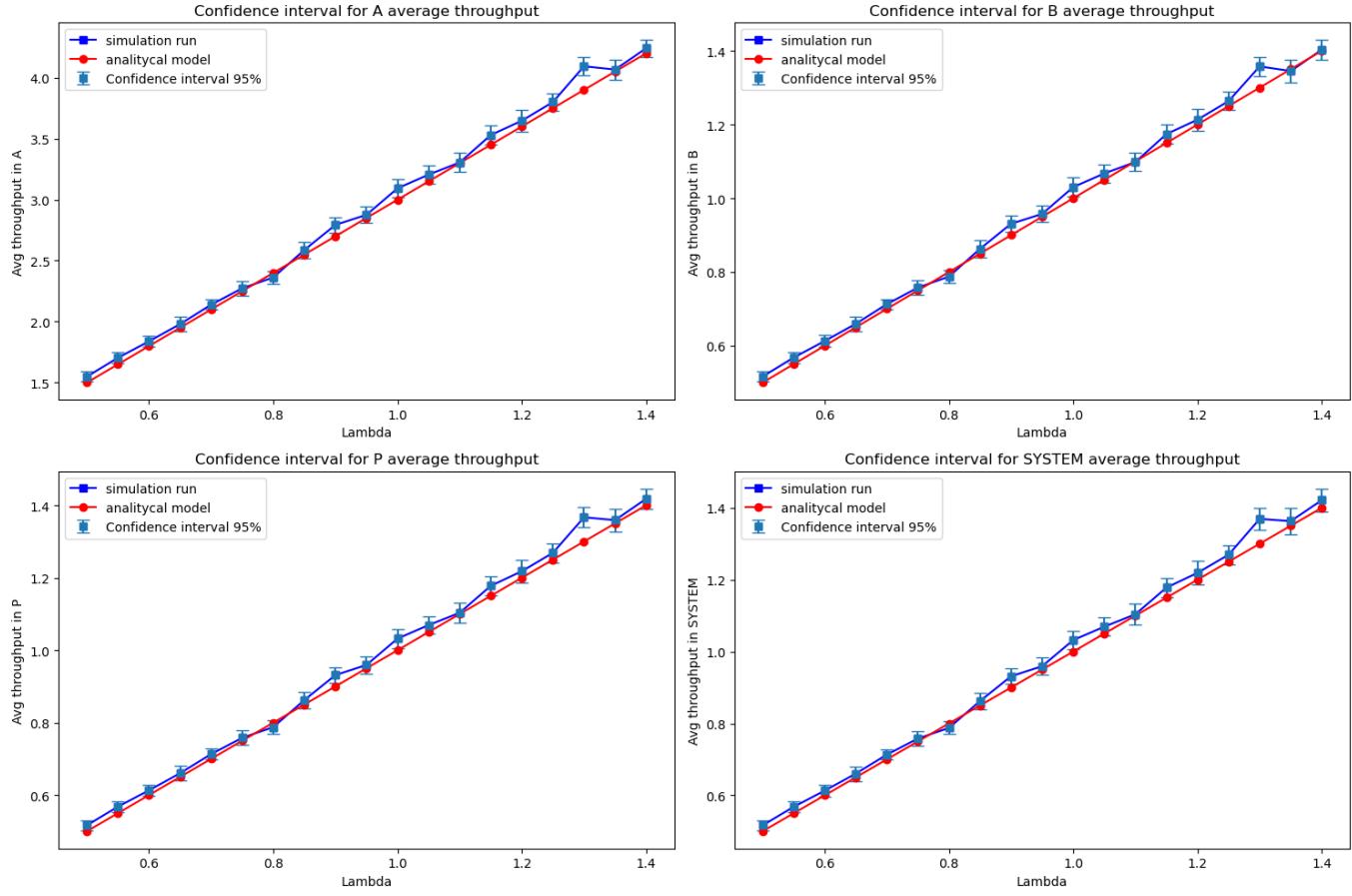


Fig. 37. Intervallo di confidenza del throughput e confronto con modello analitico per l'obiettivo 4.

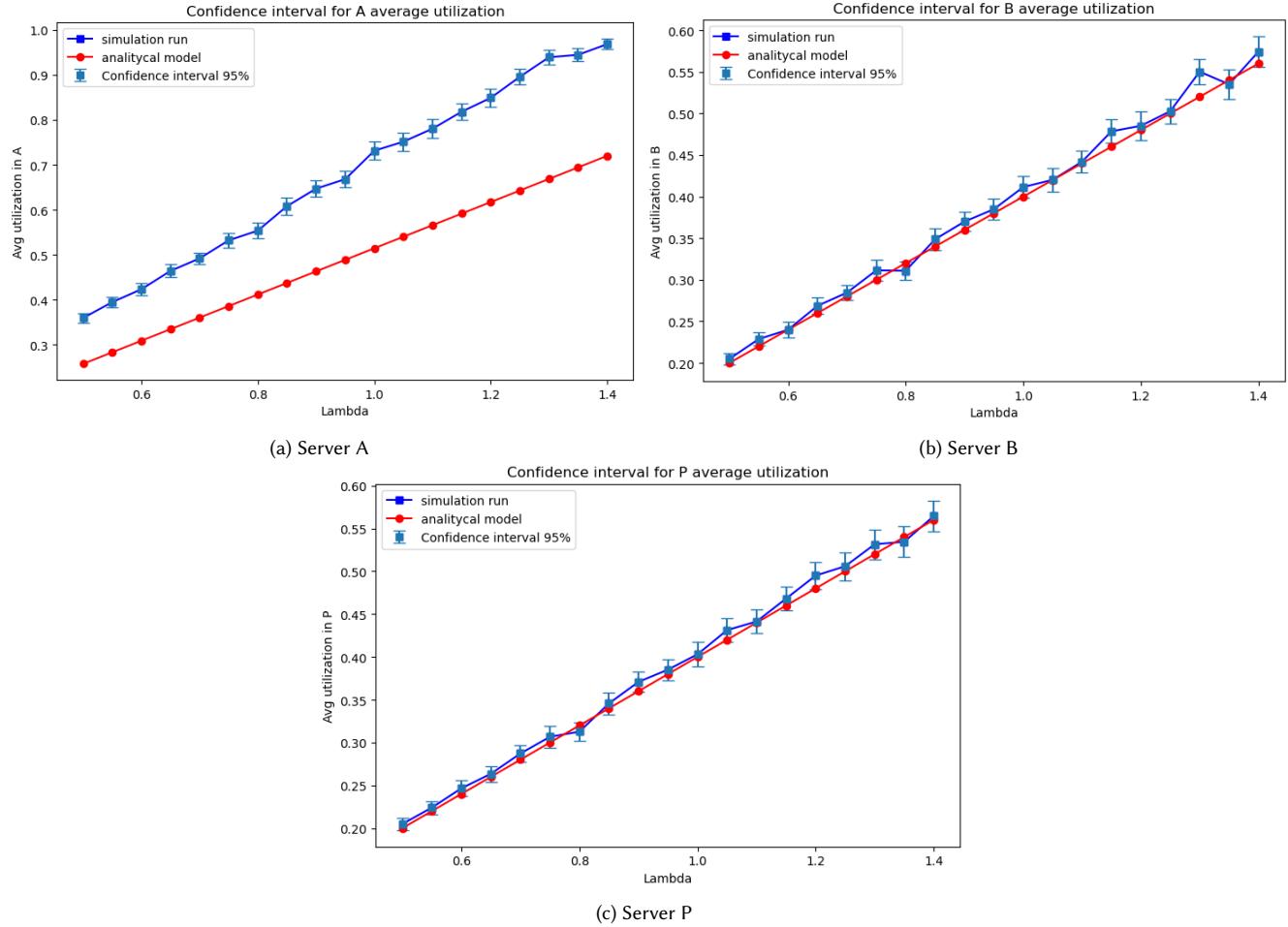


Fig. 38. Intervallo di confidenza dell'utilizzazione e confronto con modello analitico per l'obiettivo 4.

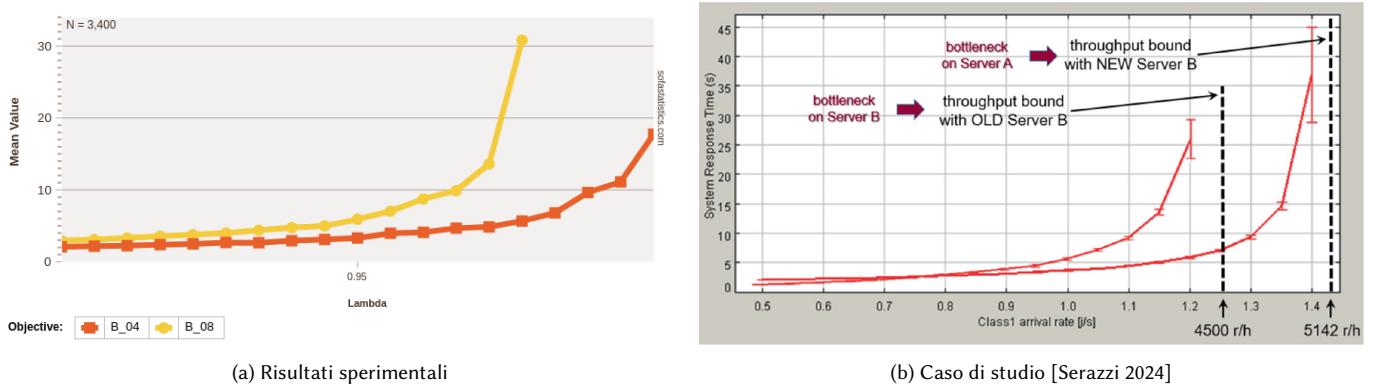


Fig. 39. Tempo di risposta medio prima e dopo la miglioria del server B in funzione del rate medio degli arrivi esterni

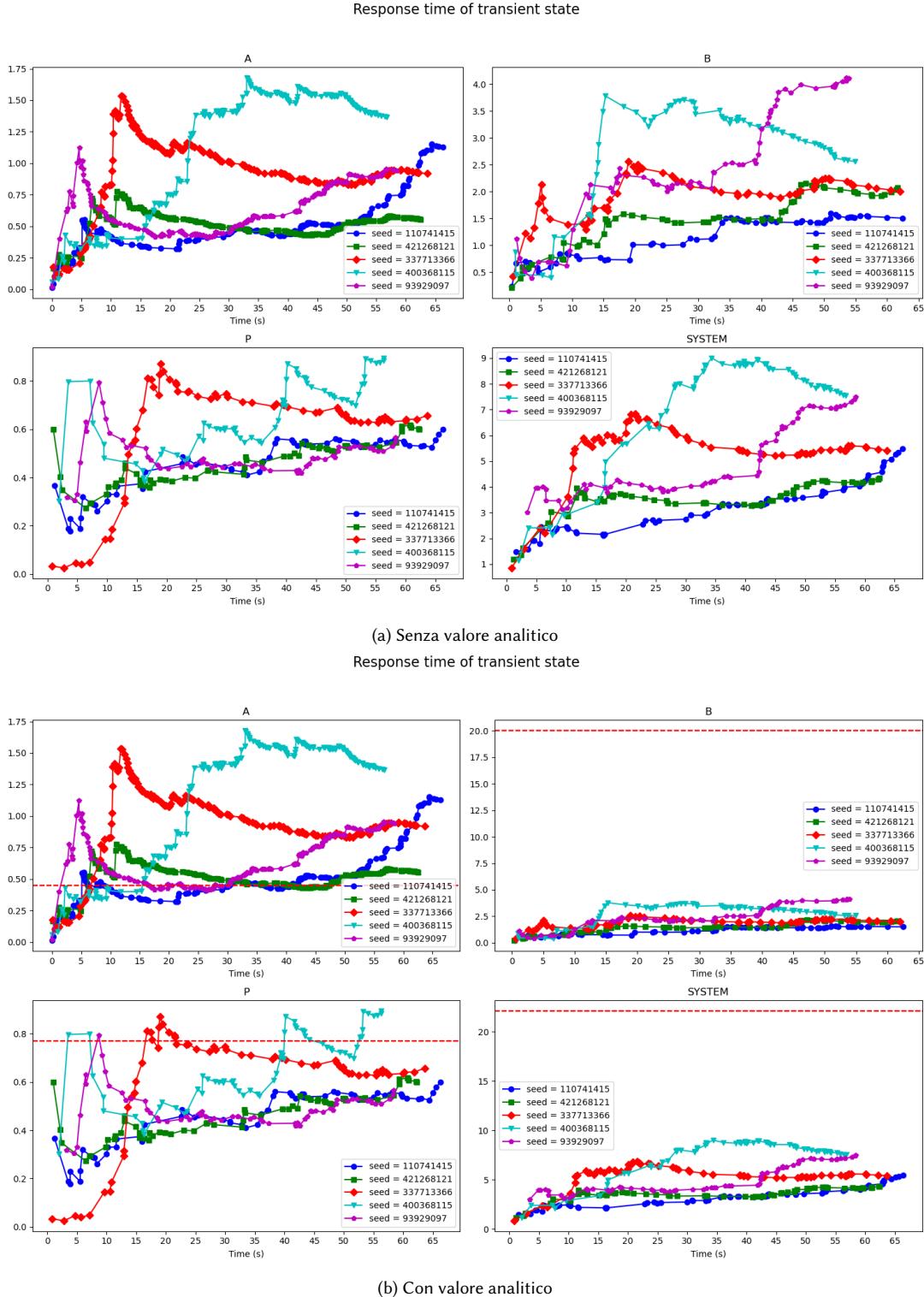


Fig. 40. Tempo di risposta per l'obiettivo 1 in funzione del tempo di simulazione nello stato transiente del sistema con un rate di arrivi 1.2job/s al variare del seed.

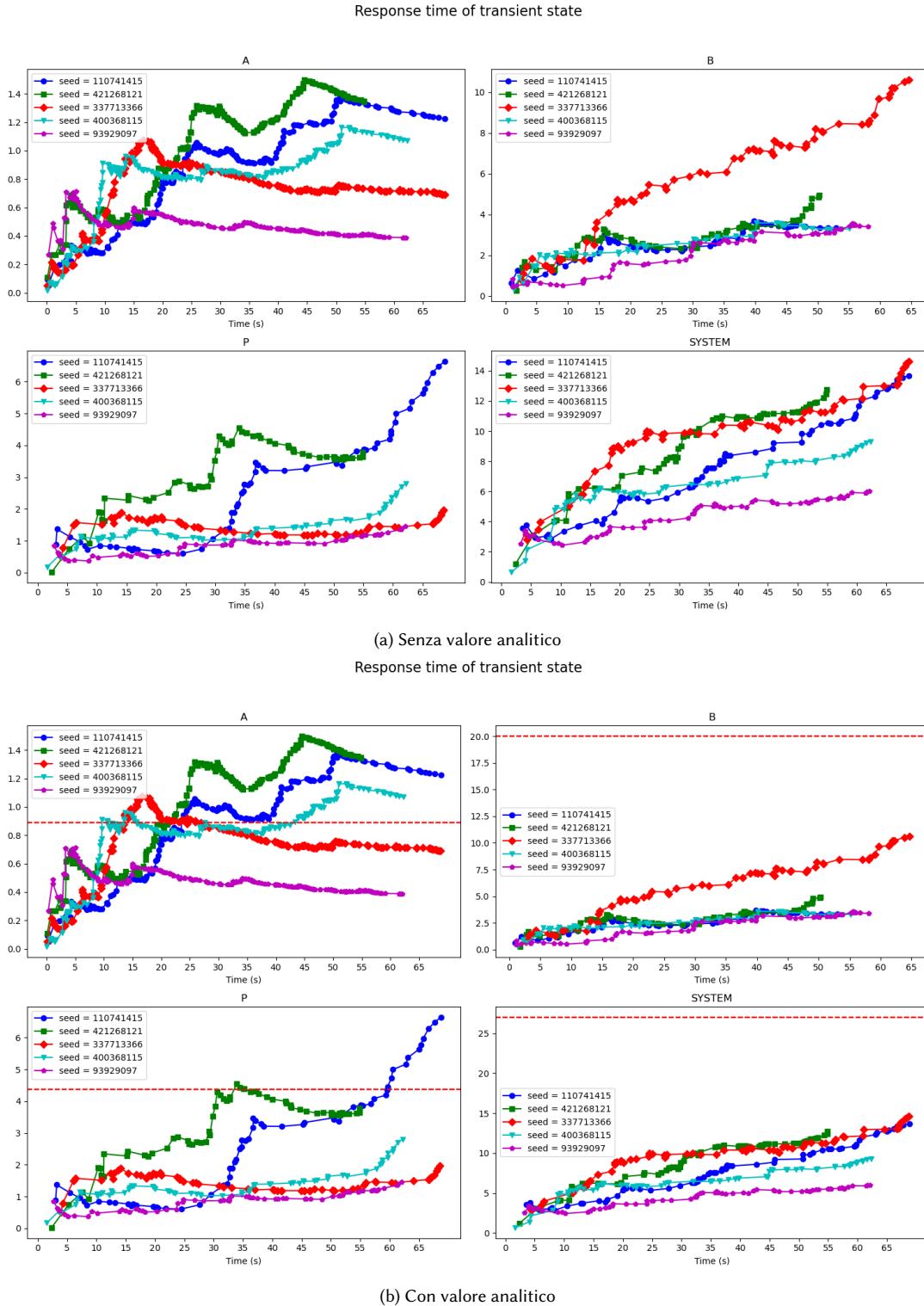


Fig. 41. Tempo di risposta per l'obiettivo 2 in funzione del tempo di simulazione nello stato transiente del sistema con un rate di arrivi 1.2 job/s al variare del seed.

Response time of transient state

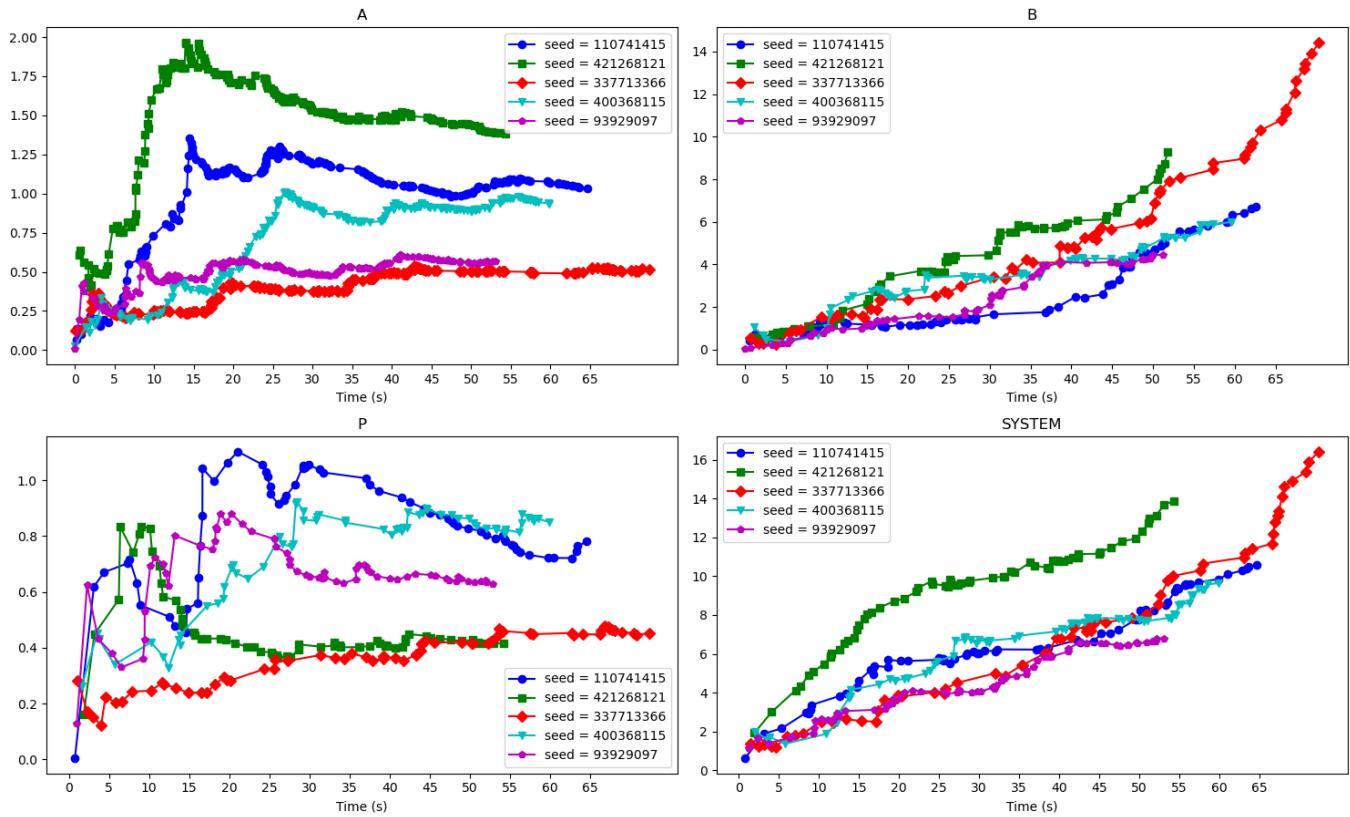


Fig. 42. Tempo di risposta per l'obiettivo 3 in funzione del tempo nello stato transiente del sistema con un rate di arrivi 1.4job/s al variare del seed usato per la simulazione

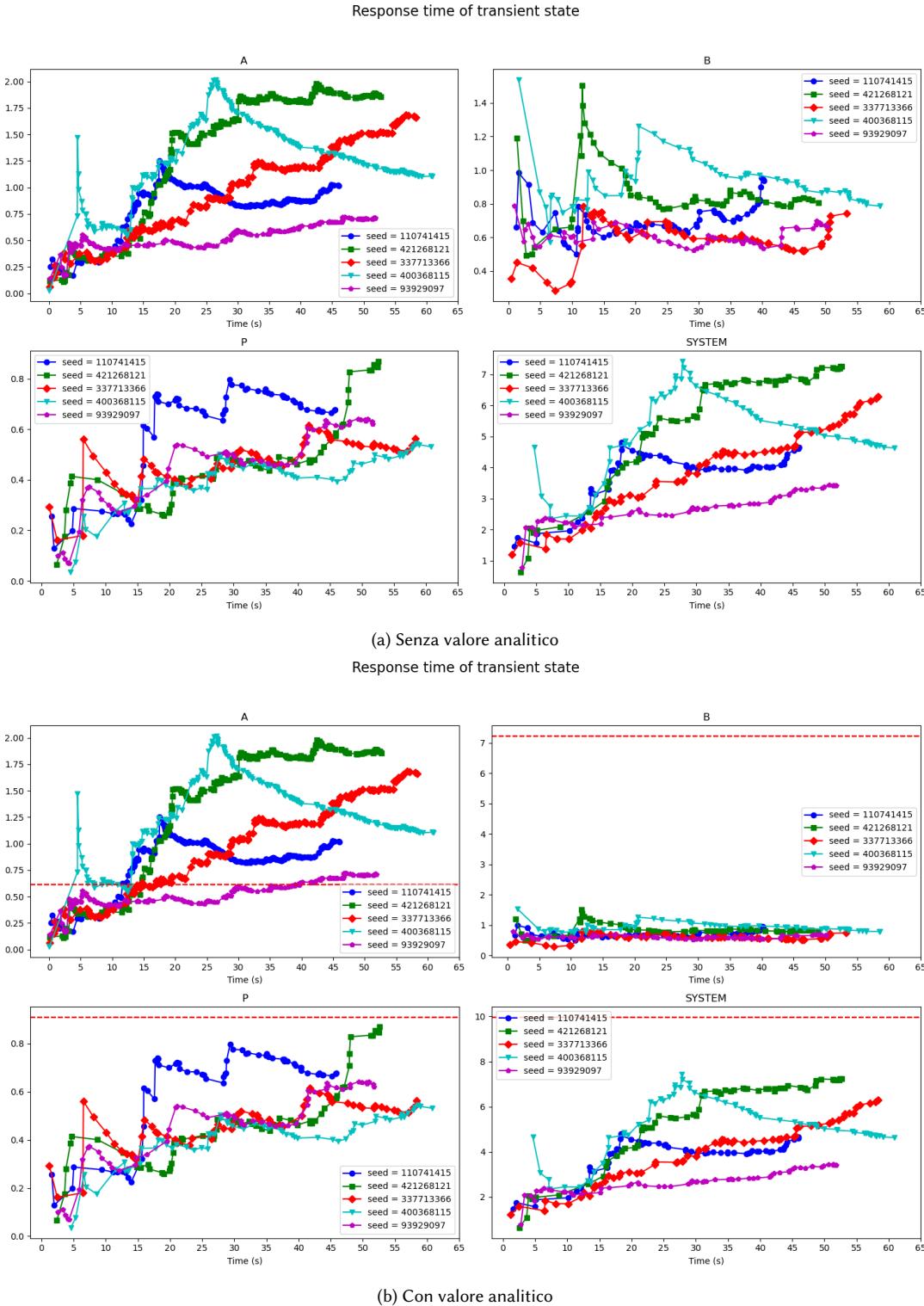


Fig. 43. Tempo di risposta per l'obiettivo 4 (con il server B con tempi di servizio 0.4s) in funzione del tempo di simulazione nello stato transiente del sistema con un rate di arrivi 1.4 job/s al variare del seed.

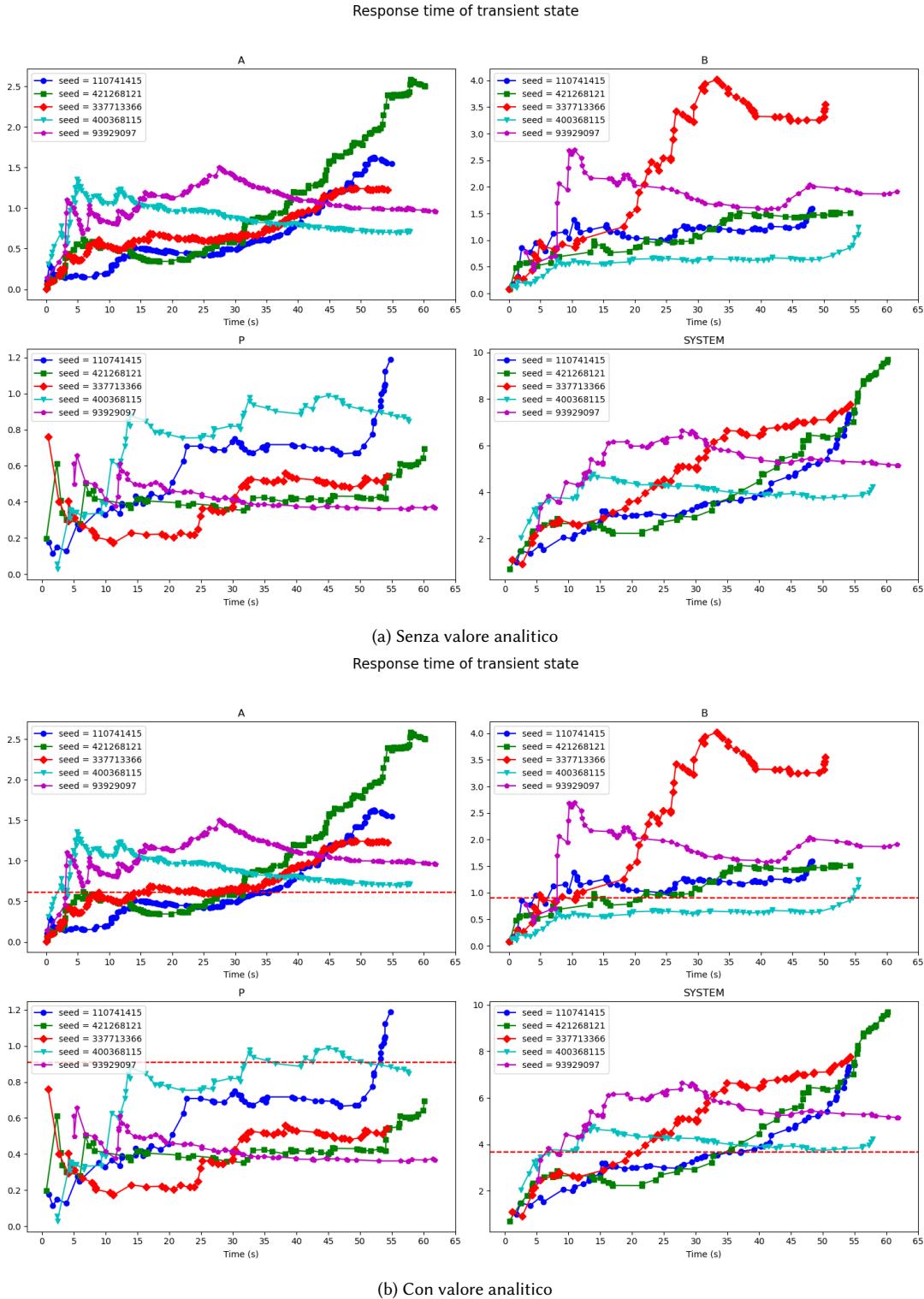


Fig. 44. Tempo di risposta per l'obiettivo 4 (con il server B con tempi di servizio 0.65s) in funzione del tempo di simulazione nello stato transiente del sistema con un rate di arrivi 1.4 job/s al variare del seed.