

Performance Analysis of a Web App Workflow

DISSAN UDDIN AHMED, Università di Tor Vergata, Matricola: 0334869, Italy

EDOARDO MARCHIONNI, Università di Tor Vergata Matricola: 0359614, Italy

LEONARDO POLIDORI, Università di Tor Vergata Matricola: 0357314, Italy

Authors' addresses: Dissan Uddin Ahmed, dissanahmed@gmail.com, Università di Tor Vergata, Matricola: 0334869, Rome, Italy; Edoardo Marchionni, edoardo.marchionni99@gmail.com, Università di Tor Vergata Matricola: 0359614, Rome, Italy; Leonardo Polidori, leo.polidori99@gmail.com, Università di Tor Vergata Matricola: 0357314, Rome, Italy.

1 INTRODUZIONE E CASO DI STUDIO

In questo studio analizziamo il workflow di una web application di e-commerce, inteso come la sequenza di azioni necessarie al completamento di un acquisto. Il sistema viene modellato tramite teoria delle code, con tre server principali: **Server A** (front-end e orchestrazione), **Server B** (core business logic), **Server P** (payment hub esterno). Il routing semplificato è deterministico: $A \rightarrow B \rightarrow A \rightarrow P \rightarrow A$.

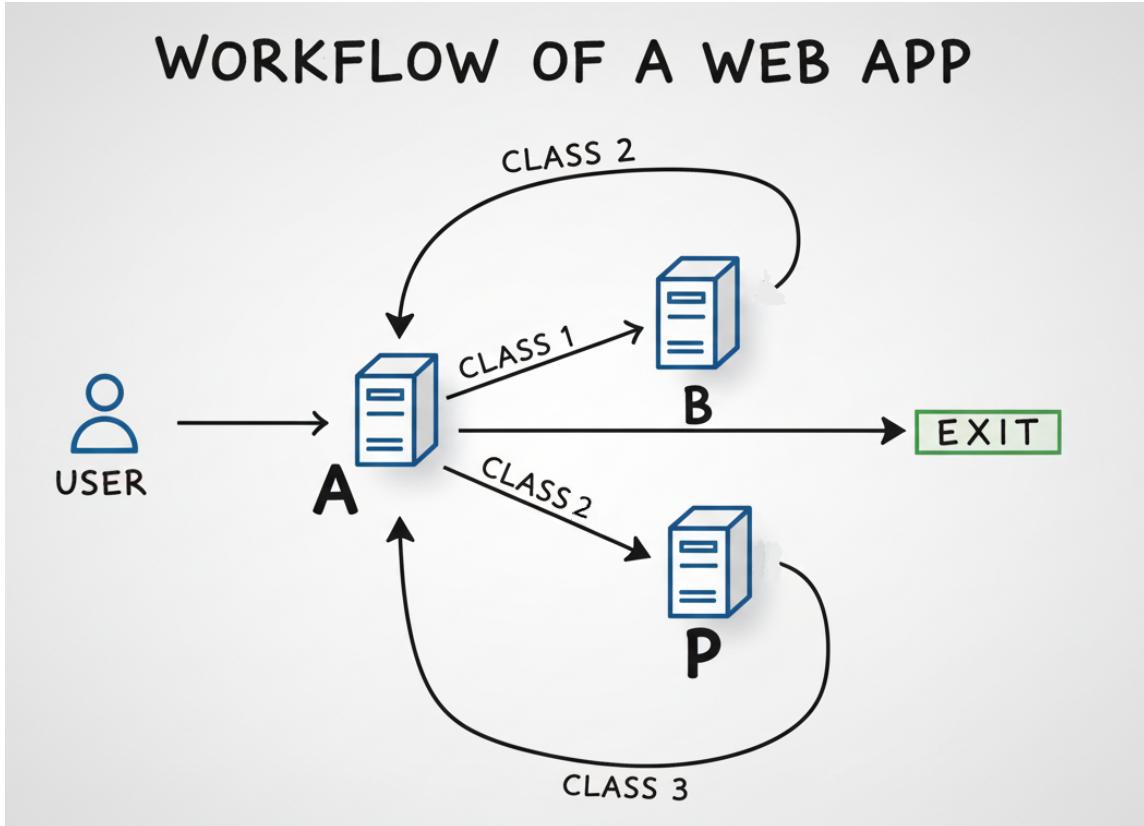


Fig. 1. Introduzione del sistema

1.1 Descrizione del sistema

Il sistema in esame rappresenta un'applicazione cloud multi-tier, distribuita su tre nodi logici: La Figura 2 mostra i compiti dei tre nodi e i flussi di chiamata tra essi. Per semplicità, assumiamo latenze di rete trascurabili e assenza di tempi di think-time lato utente, così da valutare le prestazioni in condizioni di carico massimo.

- Nodo A (Front-End Gateway) Gestisce l'autenticazione degli utenti, la prenotazione delle scorte, la generazione della ricevuta elettronica e l'attivazione del flusso di spedizione. In sintesi, funge da punto di ingresso e di orchestrazione delle transazioni.
- Nodo B (Core Services) Implementa la logica di business: mantiene il carrello, consente la consultazione del catalogo, applica sconti e coupon e valida l'ordine prima del pagamento. Tutte le richieste applicative che coinvolgono la logica di dominio passano attraverso questo nodo.

- Nodo P (Payment Hub) È un servizio esterno responsabile dell'elaborazione dei pagamenti digitali (carte, wallet, bonifici istantanei). Essendo un'entità di terze parti, non è sotto lo stesso controllo amministrativo di A e B.

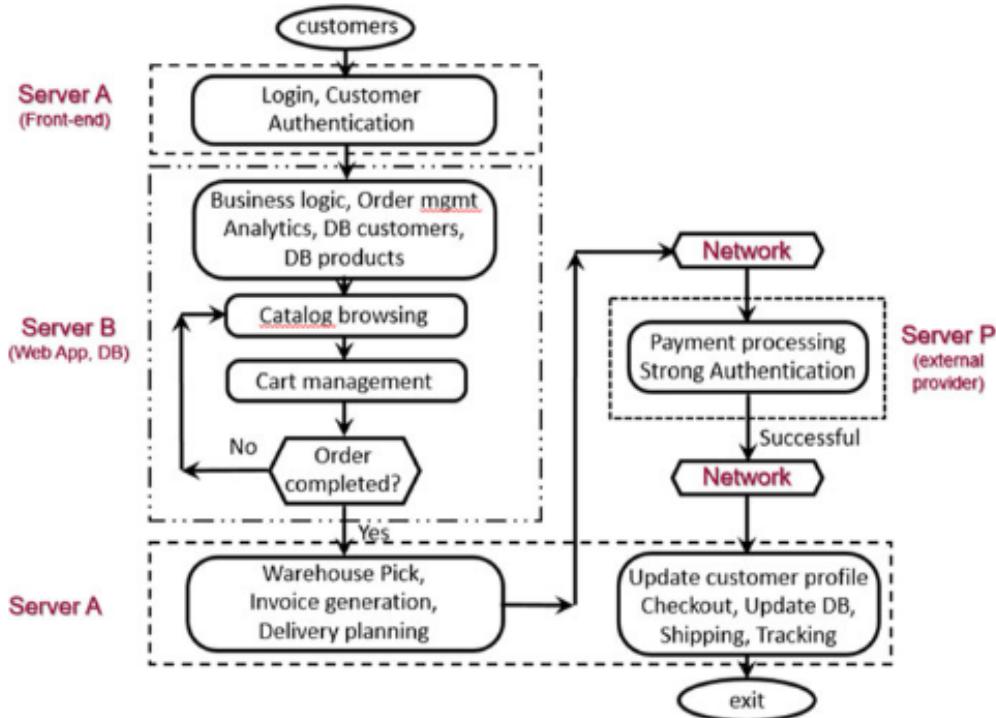


Fig. 2. Per la sezione Performance Analysis of a Web App Workflow (cap. 6.3, pag. 120 del documento)

1.2 Problematiche del sistema

Le principali criticità riguardano: colli di bottiglia (soprattutto A e B), complessità dei class ID, fluttuazioni di carico (flash sale), taratura di meccanismi di autoscaling, e dipendenza dal nodo P, con variabilità non controllabile.

- Colli di bottiglia interni – Le tre visite al nodo A, ciascuna associata a un differente Class ID, e la concentrazione della logica di dominio nel nodo B rendono probabile che uno dei due diventi il fattore limitante all'aumentare del carico. Individuare con precisione quale nodo rivesta questo ruolo costituisce un obiettivo primario dello studio.
- Complessità dei Class ID – Il riutilizzo dello stesso nodo con tempi di servizio differenti semplifica la topologia complessiva, ma richiede una taratura accurata delle stazioni di class-switch. Parametri incoerenti potrebbero falsare i tempi di risposta o alterare i percorsi di routing, complicando la manutenzione del modello ispirato al case study reale.
- Dipendenza da un server esterno (Nodo P) – Il gateway di pagamento, al di fuori del dominio amministrativo, introduce variabilità non controllabile nei tempi di servizio. È quindi necessario valutare in che misura tale incertezza influisca sul tempo di risposta end-to-end, specialmente quando vengono introdotti livelli di sicurezza aggiuntivi (come l'autenticazione a due fattori) che possono aumentare la domanda di servizio.

1.3 Obiettivi

(1) Obiettivo 1 – Modellazione e metriche di base

Implementare un modello in grado di rappresentare fedelmente il sistema oggetto di studio e misurare le principali metriche prestazionali medie: tempo di risposta, popolazione, throughput e utilizzazione. Le metriche vengono valutate sia a livello locale (per singolo nodo) sia a livello globale (per l'intero sistema).

(2) Obiettivo 2 – Impatto del 2FA

Valutare l'effetto dell'introduzione di un sistema di autenticazione a due fattori sul flusso di pagamento, misurando nuovamente tempi di risposta, popolazione, throughput e utilizzazione, confrontandole con lo scenario baseline.

(3) Obiettivo 3 – Comportamento in overload

Osservare il sistema sotto condizioni di carico più intenso, aumentando il tasso di arrivo da 4.320 job/h (1,2 job/s) a circa 5.000 job/h (1,4 job/s), e confrontare le prestazioni con i risultati ottenuti nei primi due obiettivi.

(4) Obiettivo 4 - Routing probabilistico

Osservare le performance del sistema introducendo percorsi alternativi. Esamineremo due scenari specifici:

- (a) Routing A-B-A-P-A o A-B-EXIT: Il sistema offre la possibilità di completare il percorso abituale A-B-A-P-A o di terminare il processo dopo il passaggio per B, uscendo dal sistema (un utente che non effettua un acquisto).
- (b) Routing A-B-A-P-A o Loop A-B: Il sistema offre la possibilità di completare il percorso A-B-A-P-A o di effettuare un loop, ritornando in B dopo il passaggio per A, per poi ripetere la scelta (Modifica ulteriore del carrello).

(5) Obiettivo 5 - scaling verticale server B

Questo obiettivo si basa sulle condizioni di overload dell'Obiettivo 3, ma introduce una variabile fondamentale: la potenza del nodo B. L'obiettivo è determinare come le performance del sistema cambiano quando la potenza di B viene modificata. Partiremo da un tempo di servizio medio di 0.8 s, fino a raddoppiare la potenza riducendo il tempo di servizio a 0.4 s. Monitoreremo l'evoluzione dell'utilizzazione e del throughput del nodo B per identificare il punto di equilibrio ideale, ovvero la combinazione in cui si ottengono le migliori prestazioni al minor costo.

2 MODELLO CONCETTUALE

Il sistema è formalizzato tramite stati e *Class ID* che identificano la fase del job nel workflow A → B → A → P → A:

- **Classe 1:** prima parte del flusso (A₁ e B).
- **Classe 2:** fase intermedia (ritorno ad A₂ e poi P).
- **Classe 3:** fase finale (ritorno ad A₃ prima dell'uscita).

In altre parole: il job entra in A come Classe 1, va a B (Classe 1), torna ad A come Classe 2 e prosegue su P (Classe 2), infine rientra in A come Classe 3 e *sink* (uscita).

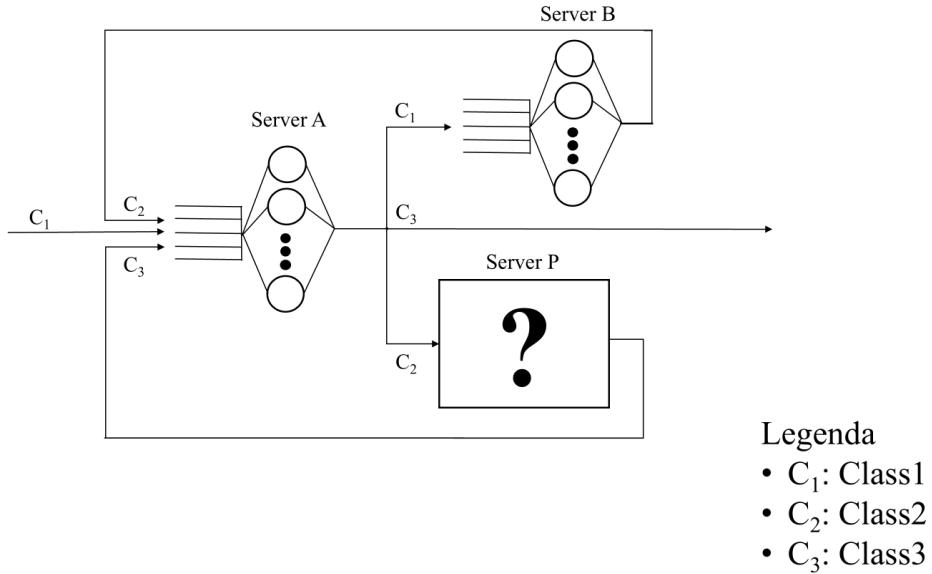


Fig. 3. Modello concettuale iniziale

Il sistema comprende tre server e un insieme di richieste (job) che attraversano il workflow:

- Server A: può trovarsi negli stati:
 - Operativo: il server è disponibile e pronto a elaborare richieste.
 - Occupato: il server sta elaborando una richiesta appartenente a una delle tre classi (Classe 1, Classe 2 o Classe 3).
- Server B: può trovarsi negli stati:
 - Operativo: disponibile per elaborare richieste.
 - Occupato: elaborazione in corso per richieste di Classe 1.
- Server P (Payment Provider): può trovarsi negli stati:
 - Operativo: disponibile per elaborare richieste.
 - Occupato: elaborazione in corso per richieste di Classe 2.
- Richieste (Job) Ogni richiesta può appartenere a una determinata classe, che definisce il suo stato nel workflow:
 - Classe 1: dalla generazione al Source fino al completamento del passaggio al Server B.
 - Classe 2: dopo l'elaborazione al Server B e prima della fase di pagamento sul Server P.
 - Classe 3: dopo il passaggio al Server P e prima dell'uscita definitiva dal sistema.

2.1 Variabili di Stato del Sistema

- $N_A^{(c)}(t)$: numero di job di classe $c \in \{1, 2, 3\}$ in servizio sul nodo A all'istante t .
- $N_B^{(1)}(t)$ (alias $N_B(t)$): numero di job di Classe 1 in B (unica classe servita da B).
- $N_P^{(2)}(t)$ (alias $N_P(t)$): numero di job di Classe 2 in P (unica classe servita da P).
- $C_j(t) \in \{1, 2, 3\}$: classe corrente associata al job j all'istante t .

Per completezza:

$$N_k(t) = \sum_c N_k^{(c)}(t), \quad N_{\text{sys}}(t) = \sum_{k \in \{A, B, P\}} \sum_c N_k^{(c)}(t),$$

e le medie stazionarie sono $E[N_k^{(c)}]$, $E[N_k]$ ed $E[N_{\text{sys}}]$.

3 MODELLO DELLE SPECIFICHE

3.1 Parametri Modello

Obiettivo 1.

- Tassi di Arrivo (λ) $\sim Exp(\lambda)$:
 - $\lambda = 1.2$ richieste/s (4320 richieste/h)
- Tempi di Servizio Medi Server A ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio (0.2 s)
 - Classe 2: tempo medio servizio (0.4 s)
 - Classe 3 (vanilla): tempo medio servizio (0.1 s)
- Tempi di Servizio Medi Server B ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio (0.8 s)
- Tempi di Servizio Medi Server P ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 2 tempo medio servizio (0.4 s)
- Politica di Scheduling: Tutti i server seguono disciplina Processor Sharing (PS).

Obiettivo 2.

- Tassi di Arrivo (λ) $\sim Exp(\lambda)$:
 - $\lambda = 1.2$ richieste/s (4320 richieste/h)
- Tempi di Servizio Medi Server A ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio (0.2 s)
 - Classe 2: tempo medio servizio (0.4 s)
 - Classe 3: tempo medio servizio (0.15 s)
- Tempi di Servizio Medi Server B ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio (0.8 s)
- Tempi di Servizio Medi Server P ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 2 : tempo medio servizio incrementato rispetto al vanilla (0.7 s)
- Politica di Scheduling: Tutti i server seguono disciplina Processor Sharing (PS).

Obiettivo 3.

- Tassi di Arrivo (λ) $\sim Exp(\lambda)$:

- $\lambda = 1.4$ richieste/s (5000 richieste/h)
- Tempi di Servizio Medi Server A ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio (0.2 s)
 - Classe 2: tempo medio servizio (0.4 s)
 - Classe 3: tempo medio servizio (0.1 s)
- Tempi di Servizio Medi Server B ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio (0.8 s)
- Tempi di Servizio Medi Server P ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 2 : tempo medio servizio incrementato rispetto al vanilla (0.4 s)
- Politica di Scheduling: Tutti i server seguono disciplina Processor Sharing (PS).

Obiettivo 4.

- Tassi di Arrivo (λ) $\sim Exp(\lambda)$:
 - $\lambda = 1.2$ richieste/s (4320 richieste/h)
- Tempi di Servizio Medi Server A ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio (0.2 s)
 - Classe 2: tempo medio servizio (0.4 s)
 - Classe 3 (vanilla): tempo medio servizio (0.1 s)
- Tempi di Servizio Medi Server B ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio (0.8 s)
- Tempi di Servizio Medi Server P ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 2 tempo medio servizio (0.4 s)
- Probabilistico con percorsi alternativi
 - Probabilità di uscita variabile $\in [0, 1]$
 - Probabilità di riconfigurazione del carrello $\in [0, 1]$
- Politica di Scheduling: Tutti i server seguono disciplina Processor Sharing (PS).

Obiettivo 5.

- Tassi di Arrivo (λ) $\sim Exp(\lambda)$:
 - $\lambda = 1.4$ richieste/s (5000 richieste/h)
- Tempi di Servizio Medi Server A ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio (0.2 s)
 - Classe 2: tempo medio servizio (0.4 s)
 - Classe 3: tempo medio servizio (0.1 s)
- Tempi di Servizio Medi Server B ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 1: tempo medio servizio variabile $\in [0.4, 0.8]$
- Tempi di Servizio Medi Server P ($E[s]$) $\sim Exp(\frac{1}{E[s]})$:
 - Classe 2 : tempo medio servizio incrementato rispetto al vanilla (0.4 s)
- Politica di Scheduling: Tutti i server seguono disciplina Processor Sharing (PS).

Centro	Tipo di stazione	Politica di servizio	Coda	Note
Server A	M/M/m	PS	Illimitata	Tre visite distinte con Class ID diversi
Server B	M/M/m	PS	Illimitata	Logica di dominio, un solo pool di thread
Server P	M/M/m (Configurazione ignota)	PS	Illimitata	Gestito da terze parti

Table 1. Formalizzazione dei server

3.2 Routing e stazioni di servizio

I nodi sono modellati come:

- **Server A:** M/M/1, Processor Sharing (PS), coda illimitata. Tre visite distinte **Classe 1, Classe 2, Classe 3**.
- **Server B:** M/M/1, PS, coda illimitata. Serve solo richieste di **Classe 1**.
- **Server P:** M/M/1, PS, coda illimitata. Serve solo richieste di **Classe 2**.

Nota sulla modellazione. I nodi fisici sono **M/M/m**, ma per semplicità li approssimiamo con un server **M/M/1 in PS**. L'approssimazione è adeguata per metriche medie.

3.3 Metriche raccolte

- Utilizzazione (ρ).
- Throughput medio (job/s).
- Tempo medio di risposta per job (s).
- Popolazione media del sistema (Numero di job)

4 MODELLO COMPUTAZIONALE

4.1 Dinamica del sistema (Next-Event)

La simulazione adotta un approccio **next-event** con **tre** tipi di evento:

- **BOOTSTRAP** (inizializzazione): definisce lo stato iniziale del sistema e pianifica il primo arrivo esterno con eventuale periodo di warm-up.
- **ARRIVAL** (arrivo): gli inter-arrivi esterni sono Poissoniani con distribuzione $\sim \text{Exp}(\lambda)$ verso A (Classe 1); gli arrivi interni sono indotti dal routing. All'ingresso nel nodo k si incrementa $N_k^{(c)}$ e, con disciplina **PS**, si campiona il servizio $S \sim \text{Exp}(1/E[S_{k,c}])$.
- **DEPARTURE** (partenza/routing): quando un job termina il servizio nel nodo $k \in \{A, B, P\}$, si decrementa $N_k^{(c)}$ e si pianifica l'**ARRIVAL** nel nodo successivo con eventuale *class-switch*.

4.2 Sistema

4.2.1 *Generatore di numeri pseudocasuali.* La simulazione utilizza un generatore congruenziale moltiplicativo di *Lehmer* a 32 bit, con

$$x_{n+1} = (ax_n) \bmod m, \quad u_n = \frac{x_n}{m} \in (0, 1),$$

dove il **modulo** è $m = 2147483647$ e il **moltiplicatore** è $a = 48271$. L'architettura *MultiStream* mantiene uno stato (seed) separato per ciascun processo casuale indipendente, così da preservare indipendenza statistica e riproducibilità:

- arrivi esterni verso il server A ;
- servizi del server A ;
- servizi del server B ;
- servizi del server P .

Ad ogni estrazione, lo stream selezionato avanza lo stato $x \mapsto (ax) \bmod m$ e produce $u = x/m$.

Per evitare sovrapposizioni tra repliche, ogni run consuma un blocco contiguo e disgiunto di lunghezza $L = 10^7$ della sequenza di ciascuno stream. La proprietà del Lehmer

$$x_{n+L} = (a^L x_n) \bmod m$$

consente di calcolare direttamente lo stato iniziale della run successiva senza generare i L numeri intermedi. Definendo il moltiplicatore di salto

$$J \equiv a^L \bmod m,$$

la progressione dei seed tra run è

$$s_{r+1} = (Js_r) \bmod m \quad (\text{equiv. } s_r = (a^{rL} s_0) \bmod m),$$

così che la run r utilizzi esattamente gli indici $[rL, (r+1)L - 1]$ della sequenza u_n per ogni stream. Questo schema garantisce: (i) *non-overlap* tra run, (ii) *coerenza* tra processi (tutti gli stream avanzano su blocchi allineati), (iii) *ripetibilità* controllata fissando s_0 .

Semi adottati. I seed usati per le run sono:

$$[314159265, 1899032171, 1463093617, 779946103, 673620713].$$

Sono ottenuti applicando iterativamente la relazione $s_{r+1} = (Js_r) \bmod m$ a partire da $s_0 = 314159265$ con $J = a^L \bmod m$ e $L = 10^7$. In ciascuna run, ogni stream evolve in modo indipendente a partire dal seed assegnato alla run, preservando separazione e riproducibilità dei processi casuali.

4.2.2 Scheduler. La classe `NextEventScheduler` implementa il meccanismo di scheduling della *Next-Event Simulation*, gestendo gli eventi in ordine cronologico. Gli eventi futuri vengono mantenuti in una coda di priorità, ordinata per tempo di esecuzione (e, in caso di parità, per ordine di inserimento). Il simulatore permette inoltre la registrazione di callback da parte di subscriber, che vengono invocati quando un evento di un certo tipo viene processato.

- **pq (EventQueue<Event>)**: coda di che memorizza tutti gli eventi da eseguire secondo politica PS.
- **subscribers (Map<Event.Type, List<BiConsumer<Event, NextEventScheduler> >)**: mappa che associa a ciascun tipo di evento una lista di callback (funzioni consumatrici) da invocare quando un evento di quel tipo viene processato.

4.2.3 Evento. La classe `Event` rappresenta un evento pianificato nella simulazione *Next-Event*. Ogni evento ha un tempo di occorrenza, un tipo, un server di destinazione, identificatori di job e un numero di sequenza unico per garantire l'ordinamento deterministico. Si ha anche la sottoclasse `BootstrapEvent` che gestisce lo stato iniziale del sistema, permettendo di decidere quanti eventi sono presenti al tempo di simulazione = 0s.

- **Type (enum)**: rappresenta il tipo di evento; può assumere i valori `ARRIVAL`, `DEPARTURE`, oppure `MEASURE_START`.
- **SEQ (AtomicLong)**: contatore atomico globale utilizzato per assegnare ID unici e monotonicamente crescenti agli eventi, utile per rompere i pareggi quando due eventi hanno lo stesso timestamp.

- **id (long)**: identificatore univoco di sequenza dell'evento.
- **time (double)**: tempo di simulazione in cui l'evento si verifica.
- **type (Type)**: tipo dell'evento (arrivo, partenza, o inizio misurazione).
- **server (String)**: nome del server (o nodo) a cui l'evento è associato.
- **jobId (int)**: identificatore del job a cui l'evento si riferisce; il valore -1 indica un arrivo esterno (job che entra nel sistema).
- **jobClass (int)**: classe di appartenenza del job (1,2,3).

4.2.4 *Job*. La classe Job rappresenta un job (richiesta/attività) all'interno della simulazione. Ogni job è caratterizzato da un identificatore univoco, da una classe di appartenenza, da un tempo di arrivo e da un tempo di servizio residuo che può variare durante l'elaborazione.

- **SEQ (AtomicInteger)**: contatore atomico globale utilizzato per assegnare ID unici ai job al momento della loro creazione.
- **id (int)**: identificatore univoco del job.
- **jobClass (int)**: classe corrente del job; può cambiare se il job viene instradato verso una fase di servizio diversa.
- **arrivalTime (double)**: istante di tempo in cui il job è arrivato per la prima volta nel sistema (immutabile).
- **remainingService (double)**: tempo di servizio residuo del job; diminuisce man mano che il job viene servito e può essere reimpostato quando il job passa a una nuova fase di servizio.

4.2.5 *Server*. La classe Node rappresenta un singolo server (nodo) all'interno della rete di simulazione. Ogni nodo mantiene l'elenco dei job in servizio e pianifica gli eventi di partenza in base al tempo di servizio residuo, seguendo una disciplina di tipo *Processor Sharing (PS)*.

- **name (String)**: nome del nodo (ad esempio, "A", "B", "P").
- **serviceMeans (Map<Integer, Double>)**: mappa che associa a ogni classe di job l'atteso tempo medio di servizio $E[S]$.
- **jobs (List<Job>)**: lista dei job attualmente in servizio presso il nodo.
- **lastUpdate (double)**: ultimo istante di tempo in cui i tempi di servizio residui dei job sono stati aggiornati.

4.2.6 *Rete di Server*. La classe Network rappresenta l'insieme dei nodi di servizio della simulazione. Ogni nodo corrisponde a un server (ad esempio "A", "B", "P") ed è caratterizzato dai tempi medi di servizio per le diverse classi di job. La classe utilizza le istanze di Node per modellare il comportamento di ciascun nodo della rete.

- **nodes (Map<String, Node>)**: mappa che associa il nome del nodo (chiave) all'istanza Node corrispondente. In questo modo è possibile accedere e gestire ogni nodo della rete.

Il costruttore riceve una mappa di configurazione dei tassi di servizio, che definisce per ciascun nodo e per ciascuna classe di job il rispettivo tempo medio di servizio $E[S]$. La struttura attesa della configurazione è la seguente:

```
{
    "A": { "1": 0.2, "2": 0.4, "3": 0.1 },
    "B": { "1": 0.8 },
    "P": { "2": 0.4 }
}
```

Dove la chiave esterna rappresenta il nodo e la mappa interna associa le classi di job ai rispettivi tempi medi di servizio.

4.2.7 Simulatore. Il simulatore sviluppato adotta il paradigma della *Next-Event Simulation* (NES), gestendo gli eventi in ordine cronologico tramite una coda di priorità. Gli eventi supportati sono ARRIVAL (arrivo), DEPARTURE (partenza) e MEASURE_START (inizio misurazioni). La disciplina di servizio adottata nei nodi è **Processor Sharing (PS)**, in cui la capacità di servizio viene suddivisa equamente tra tutti i job attivi.

Gestione degli arrivi (ARRIVAL).

- Se l'evento rappresenta un **arrivo esterno** (`jobId = -1`), viene generato un nuovo job di classe specificata, con tempo di servizio estratto da una distribuzione esponenziale. Il job viene registrato nella tabella globale e passato al nodo di destinazione tramite `node.arrival()`.
- Se invece l'evento è un **arrivo interno** (routing), il job già esistente viene recuperato dalla tabella e inserito nel nodo successivo.
- In entrambi i casi, il nodo ridistribuisce i tempi di completamento secondo la politica PS.

Pseudocodice:

```
ON_ARRIVAL(event e, scheduler s):
    node ← network.getNode(e.server)

    IF e.jobId == -1 THEN
        cls ← e.jobClass
        meanService ← node.serviceMeans[cls]
        svc ← Exp(meanService)
        job ← Job(cls, start_time = s.currentTime, service = svc)
        s.jobTable.put(job.id, job)
        totalExternalArrivals++
        node.arrival(job, s)
    ELSE
        job ← s.jobTable.get(e.jobId)
        IF job != null THEN node.arrival(job, s)
```

Gestione delle partenze (DEPARTURE).

- Il nodo completa il servizio di un job e aggiorna lo stato interno.
- Tramite il router si determina la destinazione successiva:
 - se la destinazione è EXIT, il job viene terminato e rimosso;
 - se la destinazione è un altro nodo, si pianifica un nuovo evento di ARRIVAL nello stesso istante temporale, aggiornando la classe del job e il suo tempo di servizio.
- In caso di routing probabilistico, vengono tracciati i percorsi (AB, ABAPA, ABAB).
- Dopo un numero prefissato di completamenti (warm-up), si attiva la raccolta delle statistiche.

Pseudocodice:

```
ON_DEPARTURE(event e, scheduler s):
    node ← network.getNode(e.server)
    job ← s.jobTable.get(e.jobId)
    node.departure(job, s)

    tc ← router.next(e.server, job.class, rng)

    IF tc indica EXIT THEN
        totalCompletedJobs++
        s.jobTable.remove(job.id)
    RETURN

    ELSE
        scheduleTheTarget(s, job, tc)
```

Schedulazione verso il nodo target. Pseudocodice:

```
SCHEDULE_THE_TARGET(s, job, tc):
    nextNode ← tc.serverTarget
    nextClass ← tc.eventClass
    job.class ← nextClass
    meanService ← nextNode.serviceMeans[nextClass]
    svc ← Exp(meanService)
    job.remainingService ← svc
    nextEvent ← Event(s.currentTime, ARRIVAL, nextNode, job.id, nextClass)
    s.schedule(nextEvent)
```

*Nota tc è Target Class quale è il prossimo nodo e quale classe sarà in funzione del routing

Ciclo principale. Pseudocodice:

```
RUN_SIMULATION():
    cnt ← 0
    WHILE scheduler.hasNext():
        scheduler.next()      # esegue evento ARRIVAL/DEPARTURE
        cnt++
        IF cnt >= maxEvents + BIAS THEN stop arrivi esterni
    ENDWHILE
    statsCollector.calculateStats(...)
```

Nota sulla Processor Sharing. Nei metodi `Node.arrival()` e `Node.departure()` viene implementata la disciplina PS: ad ogni arrivo, la capacità di servizio viene ripartita fra tutti i job attivi; ad ogni partenza, i job rimanenti ricevono una quota maggiore, aggiornando i tempi residui.

4.2.8 Generatore di Arrivi esterni. La classe `ArrivalGenerator` modella una sorgente di arrivi esterni al sistema, generati secondo un processo di Poisson. Il generatore inserisce nel `NextEventScheduler` eventi di tipo `ARRIVAL`, mirati a un nodo specifico e appartenenti a una data classe di job. Dopo ogni arrivo esterno, il prossimo arrivo viene pianificato secondo una distribuzione esponenziale degli inter-arrivi, parametrizzata dal tasso λ .

- **rate (double):** tasso di arrivo esterno λ , espresso in job per unità di tempo.
- **targetNode (String):** nome del nodo di destinazione che riceve i job in ingresso (ad esempio, "A").
- **jobClass (int):** identificatore della classe dei job generati (ad esempio, 1).

4.2.9 Raccolta delle statistiche. La raccolta dei dati prestazionali è affidata a un insieme di classi dedicate che stimano in maniera incrementale, sia a livello di singolo nodo che a livello di sistema, le principali metriche di interesse. L'implementazione si basa sullo stimatore di Welford, che permette di aggiornare medie e varianze senza dover memorizzare l'intera sequenza di osservazioni.

Metriche raccolte (Medie).

- **Tempo di ripsosta (Response Time, RT):** stimato per job e mediato a livello di nodo e di sistema. Implementato da `ResponseTimeEstimator`, `ResponseTimeEstimatorNode` e `ResponseTimePerJobCollector`.
- **Popolazione (Population):** numero medio di job presenti nel sistema o in un nodo (`PopulationEstimator`, `PopulationEstimatorNode`).
- **Utilizzazione (Busy Time):** frazione di tempo in cui un nodo è occupato a servire job, stimata da `BusyTimeEstimator` e `BusyTimeEstimatorNode`.
- **Throughput:** tasso di completamenti per unità di tempo, stimato tramite `CompletionsEstimator` e `CompletionsEstimatorNode`.
- **Tempo di osservazione (Observation Time):** finestra temporale effettiva su cui sono state raccolte le misure (`ObservationTimeEstimator`).

Gestione centralizzata. La classe `StatsCollector` coordina la raccolta di tutte le metriche, integrando i contributi dei diversi stimatori e distinguendo fra valori raccolti in fase di *warm-up* (da scartare) e valori effettivamente considerati per le statistiche finali.

Nota metodologica. La scelta dello stimatore di Welford consente una gestione numericamente stabile delle medie e delle varianze, con aggiornamento in tempo reale ad ogni osservazione. In questo modo, il simulatore evita bias e costi di memoria dovuti a grandi quantità di dati grezzi.

5 PROGETTAZIONE DEGLI ESPERIMENTI

5.1 Obiettivo 1 e 2

Abbiamo scelto di progettare il seguente esperimento con i seguenti parametri orizzonte infinito:

- arrival rate {1.2} [job/s]
- Eventi di simulazione: 10_000_000
- seed utilizzati {314159265, 1899032171, 1463093617, 779946103, 673620713}

Abbiamo scelto di progettare il seguente esperimento con i seguenti parametri per lo studio del transitorio:

- arrival rate {1.2} [job/s]
- Eventi di simulazione: 10_000_000
- seed utilizzati {314159265, 1899032171, 1463093617, 779946103, 673620713}
- arrivi iniziali {0, 10, 100}

Abbiamo scelto di progettare il seguente esperimento con i seguenti parametri orizzonte finito:

- arrival rate {1.2} [job/s]
- finestra di osservazione {0, 60} [s]
- seed utilizzati {314159265, 1899032171, 1463093617, 779946103, 673620713}
- arrivi iniziali {0, 10, 100}

5.2 Obiettivo 3

Abbiamo scelto di progettare il seguente esperimento con i seguenti parametri orizzonte infinito:

- arrival rate {1.4} [job/s]
- Eventi di simulazione: 10_000_000
- seed utilizzati {314159265, 1899032171, 1463093617, 779946103, 673620713}
- arrivi iniziali 0

5.3 Obiettivo 4

Abbiamo scelto di progettare il seguente esperimento con i seguenti parametri per la variazione di p nel caso di uscita e nel caso di ripensamento dell'utente:

- arrival rate {0.5, 1.2} [job/s]
- Eventi di simulazione: 10_000_000
- seed utilizzati {314159265, 1899032171, 1463093617, 779946103, 673620713}
- routing probabilistico
- probabilità {0.0, 0.05, 0.1, ..., 0.95, 1.0 }

*Nota. Gli arrivi iniziali indicano quanti job ci sono nel sistema all'avvio della simulazione

5.4 Obiettivo 5

Abbiamo scelto di progettare il seguente esperimento con i seguenti parametri orizzonte infinito:

- arrival rate {1.4}
- Eventi di simulazione: 10_000_000
- seed utilizzati {314159265, 1899032171, 1463093617, 779946103, 673620713}
- tempi di servizio del server B {0.4, 0.45, .., 0.75, 0.8}
- Variazione degli tempi medi del nodo B {0.4, 0.45, .., 0.75, 0.8}
- arrivi iniziali 0

6 ANALISI DEI RISULTATI

Definizioni, unità e assunzioni

- λ [job/s] Tasso di arrivo *esterno* al sistema; nel nostro setting coincide con il throughput esterno (flusso chiuso sull'uscita), per cui nel testo usiamo λ anche come X .
- $E(S_{k,c})$ [s] Tempo medio di servizio alla c -esima visita del nodo $k \in \{A, B, P\}$. Esempi: $E(S_{A,1})$ è la prima visita ad A ; $E(S_{P,2})$ la visita a P quando la classe è 2.
- D_k [s] Domanda di servizio del nodo k per job completato: somma dei tempi medi di servizio attesi su quel nodo lungo l'intero workflow. Nel caso deterministico ABAPA:

$$D_A = E(S_{A,1}) + E(S_{A,2}) + E(S_{A,3}), \quad D_B = E(S_{B,1}), \quad D_P = E(S_{P,2}).$$

$[q, r, p, Z]$ (tutti in $[0, 1]$) Parametri del routing probabilistico:

- $q = P(B, 1 \rightarrow A, 2)$: probabilità che da B si vada in A_2 ;
- $1 - q = P(B, 1 \rightarrow \text{EXIT}, *)$: probabilità di uscire direttamente dal sistema dal nodo B ;
- $r = P(A, 2 \rightarrow P, 2)$: probabilità che da A_2 si vada nel nodo P ;
- $1 - r = P(A, 2 \rightarrow B, 1)$: probabilità di loop da A_2 verso B
- $Z = 1 - q(1 - r) = (1 - q) + qr$: probabilità di non eseguire il loop;
- $p = \frac{qr}{Z}$: probabilità complessiva di visitare P prima.

infine andiamo quindi a definire le visite per il nodo B (V_B) e per A_2 (V_{A_2}):

$$V_B = \frac{1}{Z}, \quad V_{A_2} = \frac{q}{Z}.$$

Con questi aggiustamenti, le domande nel caso probabilistico diventano:

$$D_A = E(S_{A,1}) + V_{A_2}E(S_{A,2}) + pE(S_{A,3}), \quad D_B = V_BE(S_{B,1}), \quad D_P = pE(S_{P,2}).$$

- $[\rho_k]$ Utilizzazione del nodo k : $\rho_k = \lambda D_k$. Condizione di stabilità: $\rho_k < 1$ per tutti i nodi.
- X_{\max} [job/s] Capacità del sistema: $X_{\max} = \frac{1}{\max(D_A, D_B, D_P)}$. La soglia di saturazione è $\lambda^* = X_{\max}$.
- $[U_{\text{sys}}]$ Probabilità che il sistema sia occupato (almeno un nodo in servizio). Usiamo l'approssimazione indipendente:

$$U_{\text{sys}} \approx 1 - \prod_{k \in \{A, B, P\}} (1 - \rho_k).$$

- $E(T_{s,k})$ [s] Tempo medio di risposta “locale” sul nodo k in PS (uguale alla media FCFS in $M/M/1$):

$$E(T_{s,k}) = \frac{E(S_k)}{1 - \rho_k} \quad \text{con } E(S_k) \equiv D_k.$$

- $E(N_{s,k})$ [job] Popolazione media sul nodo k :

$$E(N_{s,k}) = \frac{\rho_k}{1 - \rho_k}.$$

- $[E(T_s), E(N_s)]$ Metriche di sistema come somma dei contributi dei nodi; vale la Legge di Little:

$$E(T_s) = \sum_k E(T_{s,k}), \quad E(N_s) = \sum_k E(N_{s,k}) = \lambda E(T_s).$$

- $[\text{Var}(N_{s,k}), \sigma_{N_s}]$ Dispersioni coerenti con il codice:

$$\text{Var}(N_{s,k}) = \frac{\rho_k}{(1 - \rho_k)^2}, \quad \sigma_{N_s} = \sqrt{\sum_k \text{Var}(N_{s,k}) + 2 \sum_{i < j} \text{Cov}(N_{s,i}, N_{s,j})},$$

con covarianze opzionali (se non specificate: 0). Per il tempo di risposta totale usiamo l'approssimazione d'indipendenza:

$$\sigma_{T_s} \approx \sqrt{E(T_{s,A})^2 + E(T_{s,B})^2 + E(T_{s,P})^2}.$$

Nota sulle probabilità. Il perché della probabilità $p = \frac{qr}{Z}$ e $E_B = \frac{1}{Z}$. Ogni passaggio $B \rightarrow A_2$ avvia un “tentativo” che con probabilità r porta a P (successo) e con probabilità $1 - r$ torna a B (nuovo tentativo). La probabilità di successo prima dell'uscita è la somma di una serie geometrica corretta dal ritorno a B : si ottiene $p = \frac{qr}{1 - q(1 - r)}$. La stessa struttura geometrica fornisce le visite attese: $V_B = \frac{1}{1 - q(1 - r)}$ ed $V_{A_2} = \frac{q}{1 - q(1 - r)}$.

6.1 Scenario OBJ 1 baseline

Obiettivo 1. validiamo il modello del workflow A-B-A-P-A con class-switch e disciplina PS, confrontando simulazione e analitico su *reponse time*, *popolazione*, *utilizzazione* e *throughput* per $\lambda = 1.2$ req/s. *Sintesi.* I grafici mostrano un allineamento stretto; **B** è il collo di bottiglia ($D_B = 0.8$ s $\Rightarrow X_{\max} \approx 1.25$ req/s), con crescita non lineare dei tempi di risposta e della popolazione vicino alla saturazione.

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio	25.14423	25.26909	± 0.846
Popolazione media	30.17308	29.92883	± 1.029
Throughput (job/s)	1.20	1.19920	± 0.001
Utilizzazione	0.99667	0.99665	± 0.000

Table 2. Confronto analitico vs simulazione baseline

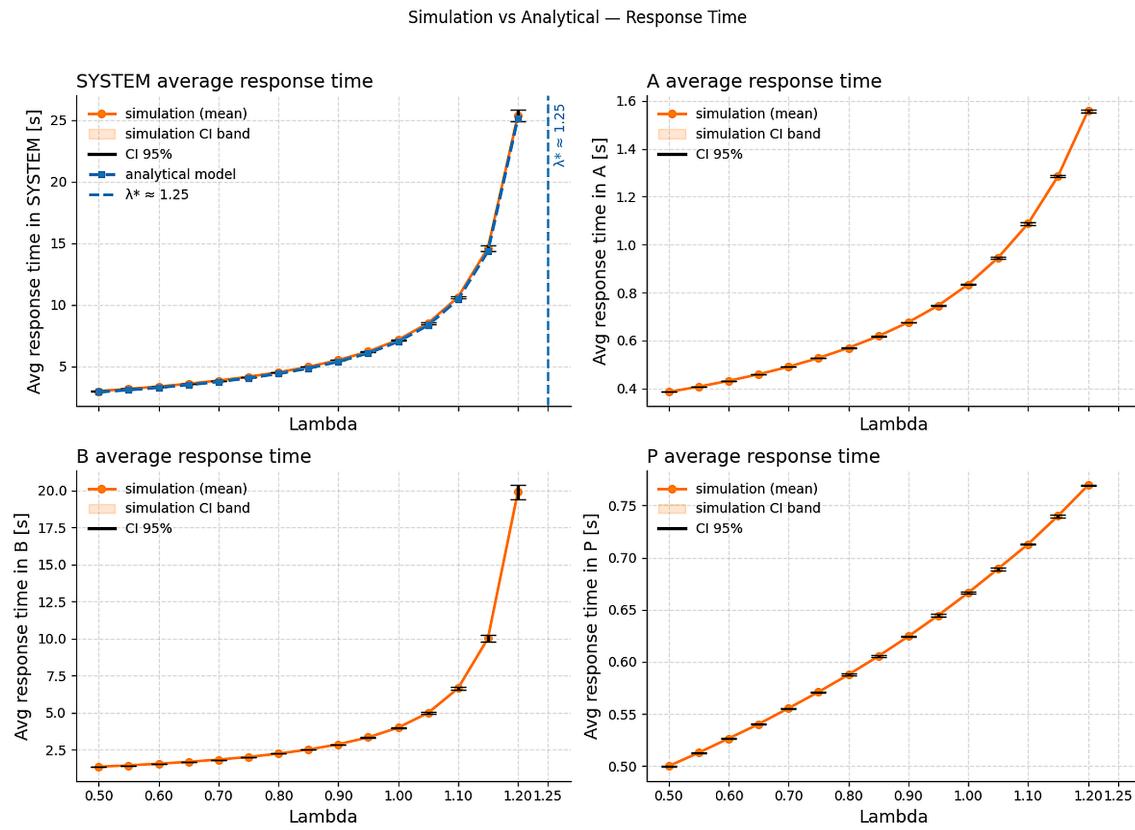


Fig. 4. Response time: sistema e nodi (sim vs analitico, CI 95%).

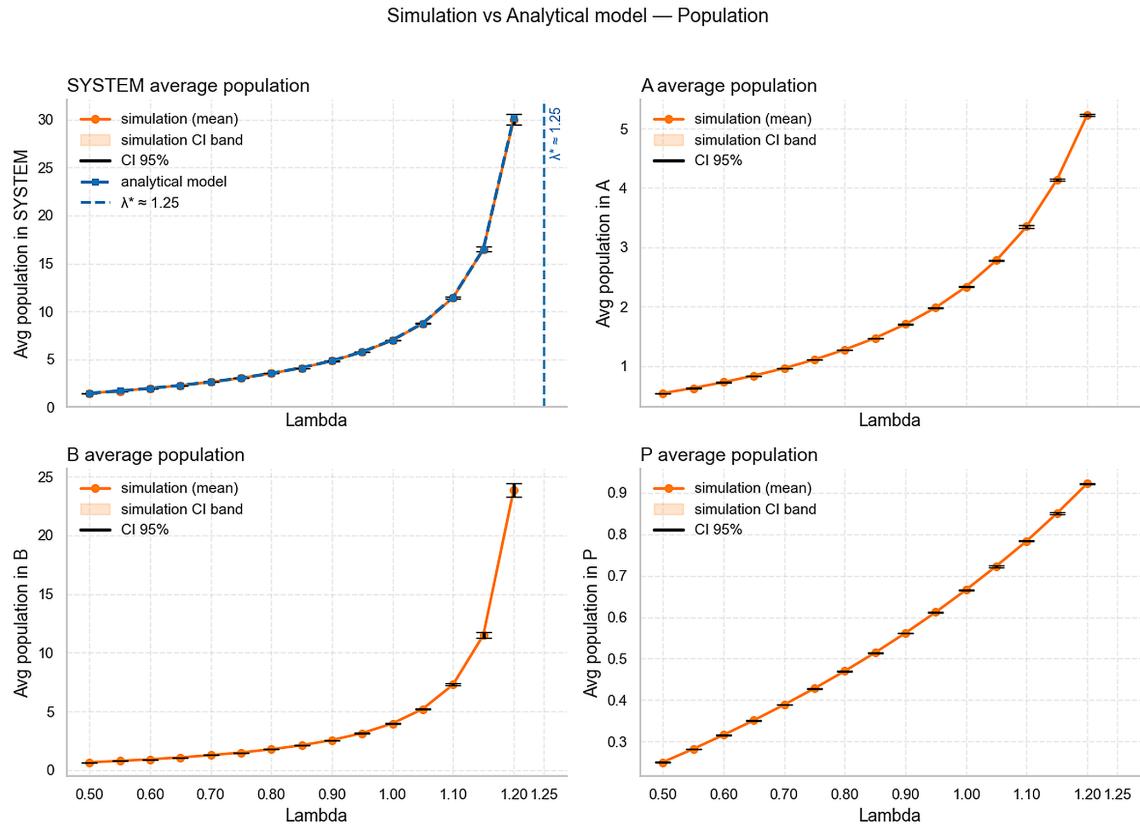


Fig. 5. Popolazione media: crescita non lineare vicino alla saturazione di B.

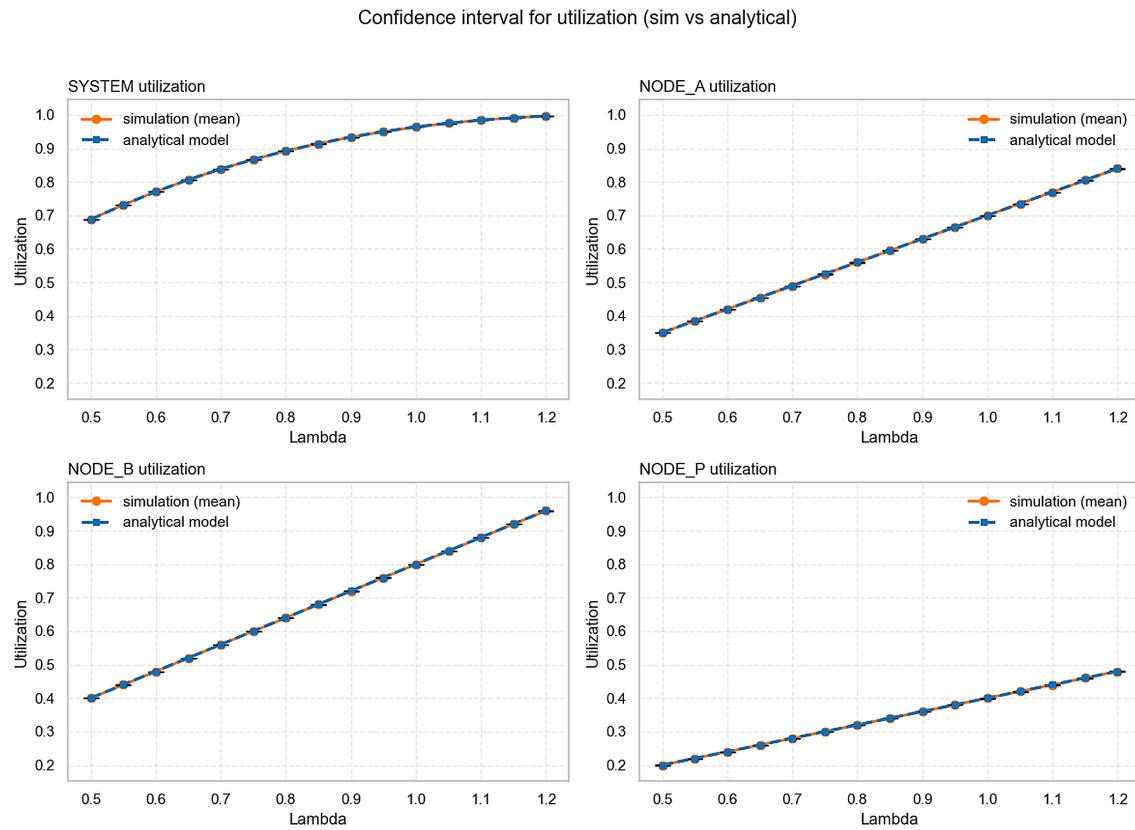


Fig. 6. Utilizzazione: sistema e nodi; \mathbf{B} tende a $\rho \rightarrow 1$ per $\lambda \rightarrow 1.2$.

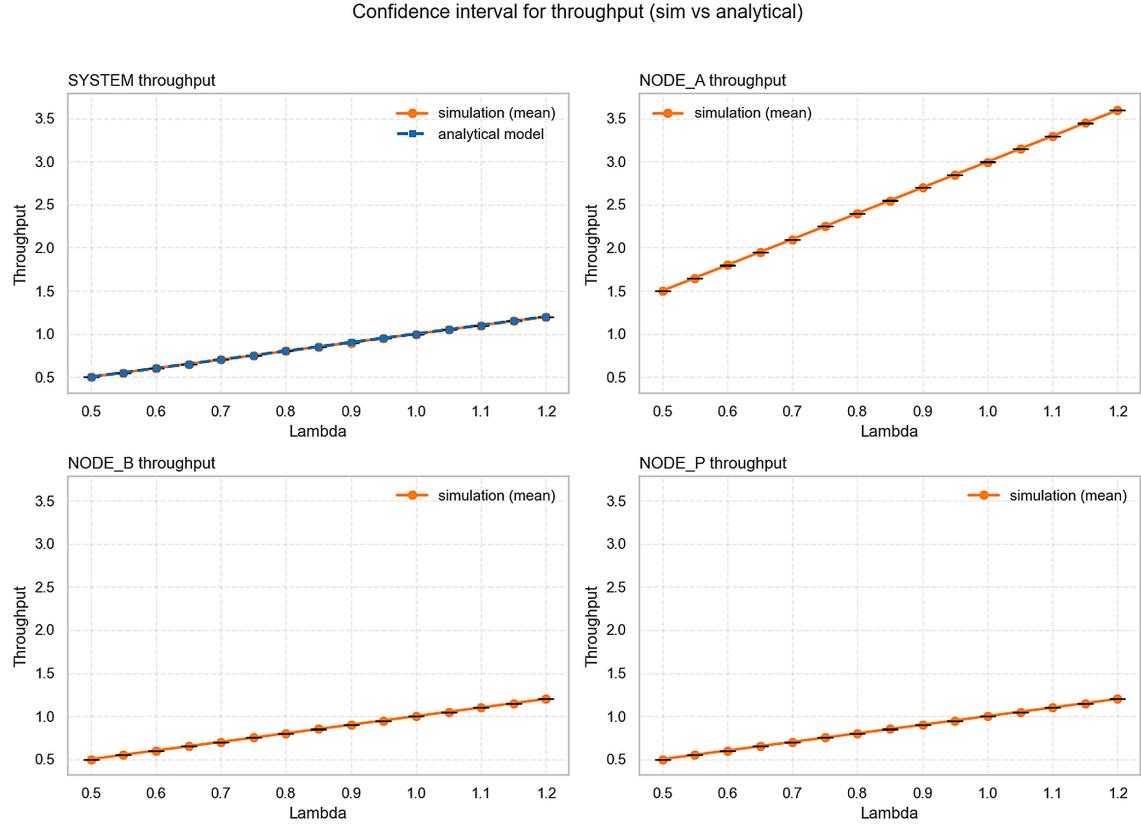


Fig. 7. Throughput: proporzionale a λ finché non si raggiunge la capacità del collo di bottiglia.

Verifica della tabella per quanto riguarda $\lambda = 1.2 \text{ job/s}$ Nodo B emerge come collo di bottiglia con tempi di risposta $\approx 20\text{s}$ e utilizzazione ≈ 0.97 .

6.2 Scenario Obj2 (2FA)

Valutiamo l'impatto dell'autenticazione a due fattori sul workflow. Il 2FA aumenta la domanda su A (terza visita) e P; confrontiamo simulazione e analitico su *response time*, *popolazione*, *utilizzazione* e *throughput*.

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio	31.87500	32.41787	± 0.921
Popolazione media	38.25000	37.97676	± 1.118
Throughput (job/s)	1.20000	1.19918	± 0.001
Utilizzazione	0.99936	0.99934	± 0.000

Table 3. Confronto analitico vs simulazione (2FA)

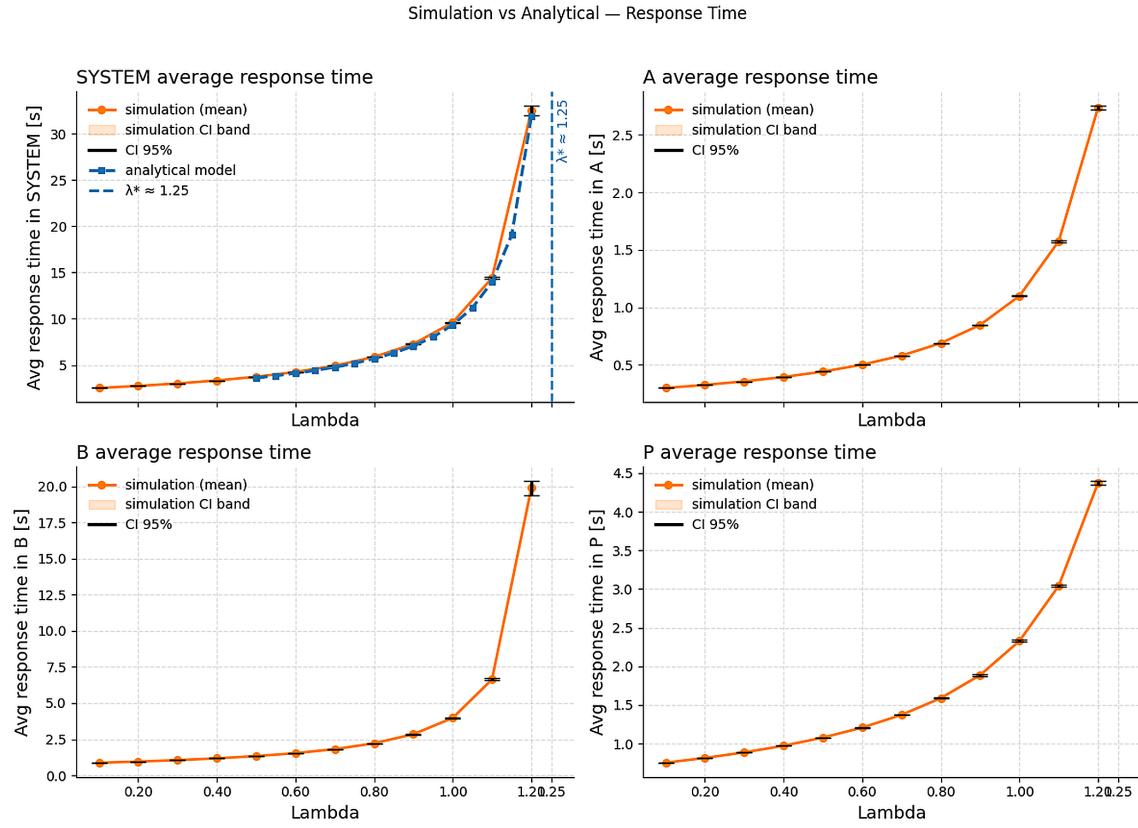


Fig. 8. Obj2 — Response time: sistema e nodi (sim vs analitico, CI 95%).

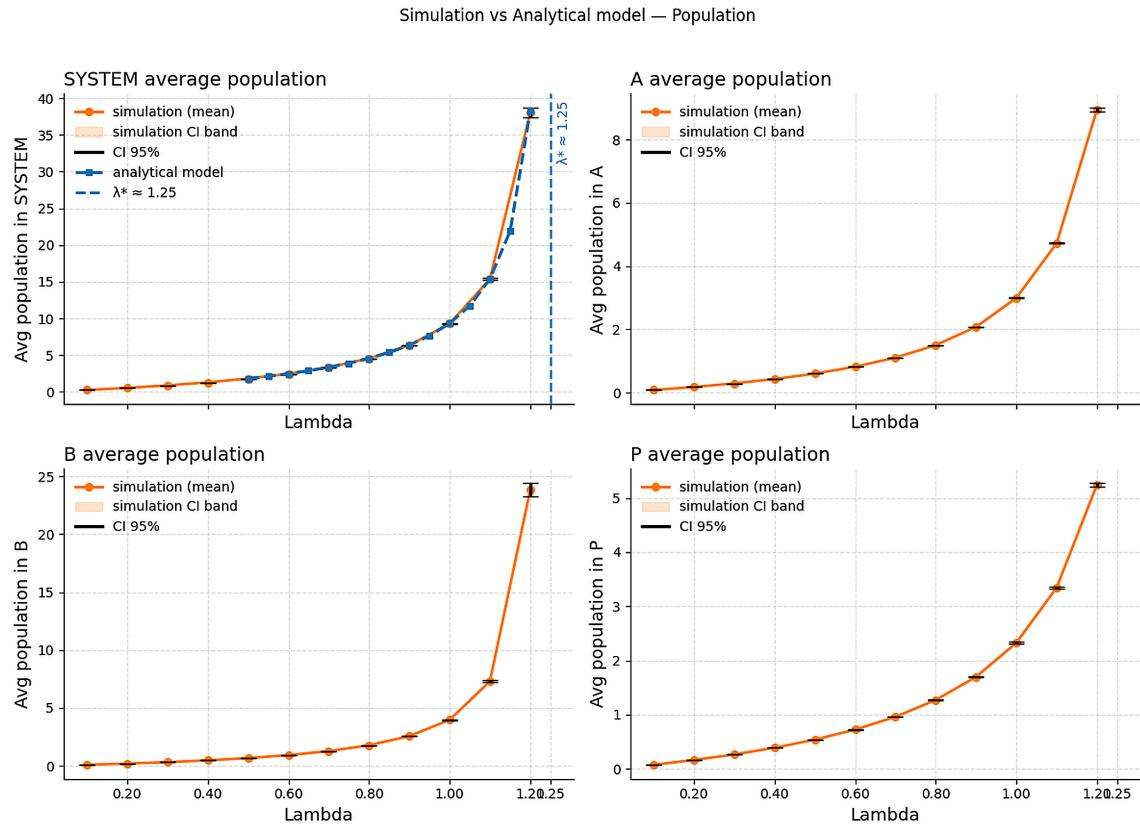


Fig. 9. Obj2 – Popolazione media: crescita più marcata vicino alla saturazione del collo di bottiglia.

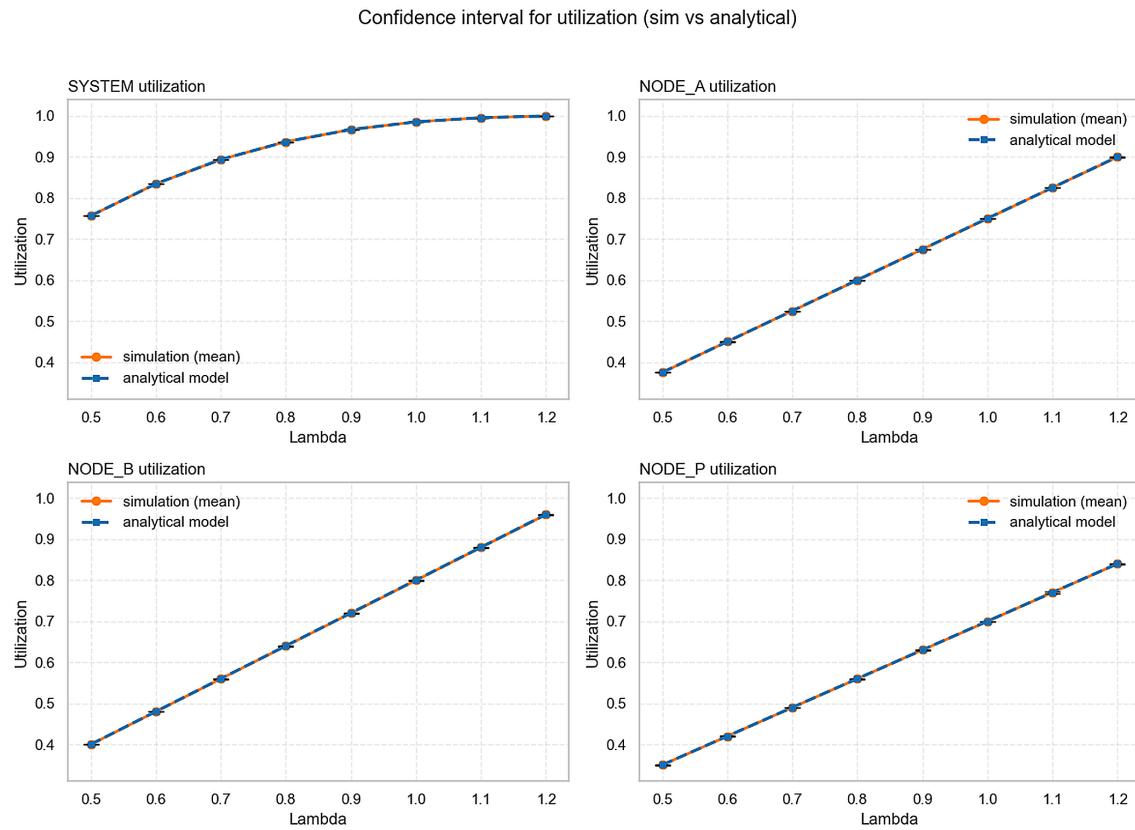


Fig. 10. Obj2 – Utilizzazione: **B** resta il collo di bottiglia; A e P aumentano per l'effetto del 2FA.

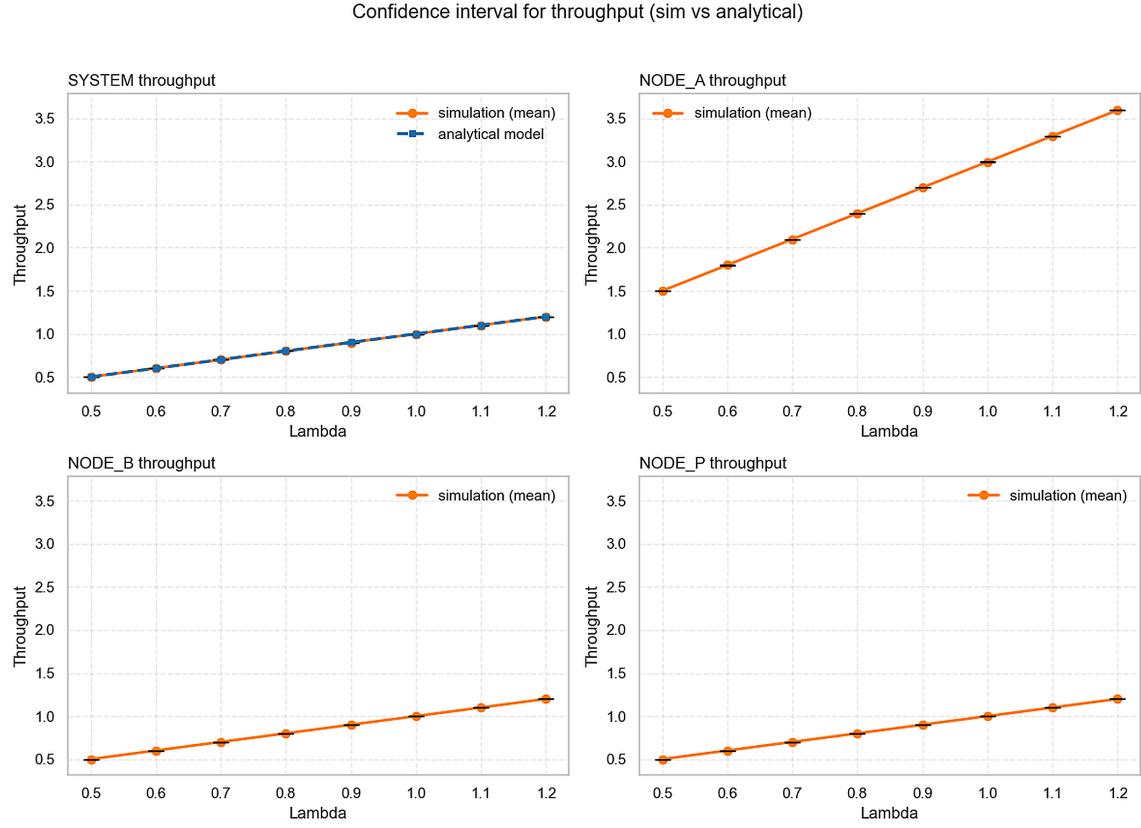


Fig. 11. Obj2 – Throughput: segue λ finché non si raggiunge la capacità del nodo critico.

Anche in questo scenario B è il collo di bottiglia, con throughput ≈ 1.2 job/s e utilizzazione ≈ 0.97 , nonostante l'aumento delle domande in A e in P.

6.3 Obj3 – Comportamento in overload

Analizziamo il sistema in overload ($\lambda \approx 1.4$ req/s). Il nodo **B** raggiunge la saturazione ($\rho_B \rightarrow 1$): i *tempi di risposta* e la *popolazione* crescono rapidamente, mentre il *throughput* di sistema si appiattisce intorno alla capacità del collo di bottiglia ($X_{\max} \approx 1/0.8 \approx 1.25$ req/s).

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio	∞	4602.36674	± 319.817
Popolazione media	∞	5748.59517	± 392.938
Throughput (job/s)	1.25000	1.24920	± 0.003
Utilizzazione	1.00000	0.99998	± 0.000

Table 4. Confronto analitico vs simulazione (overload)

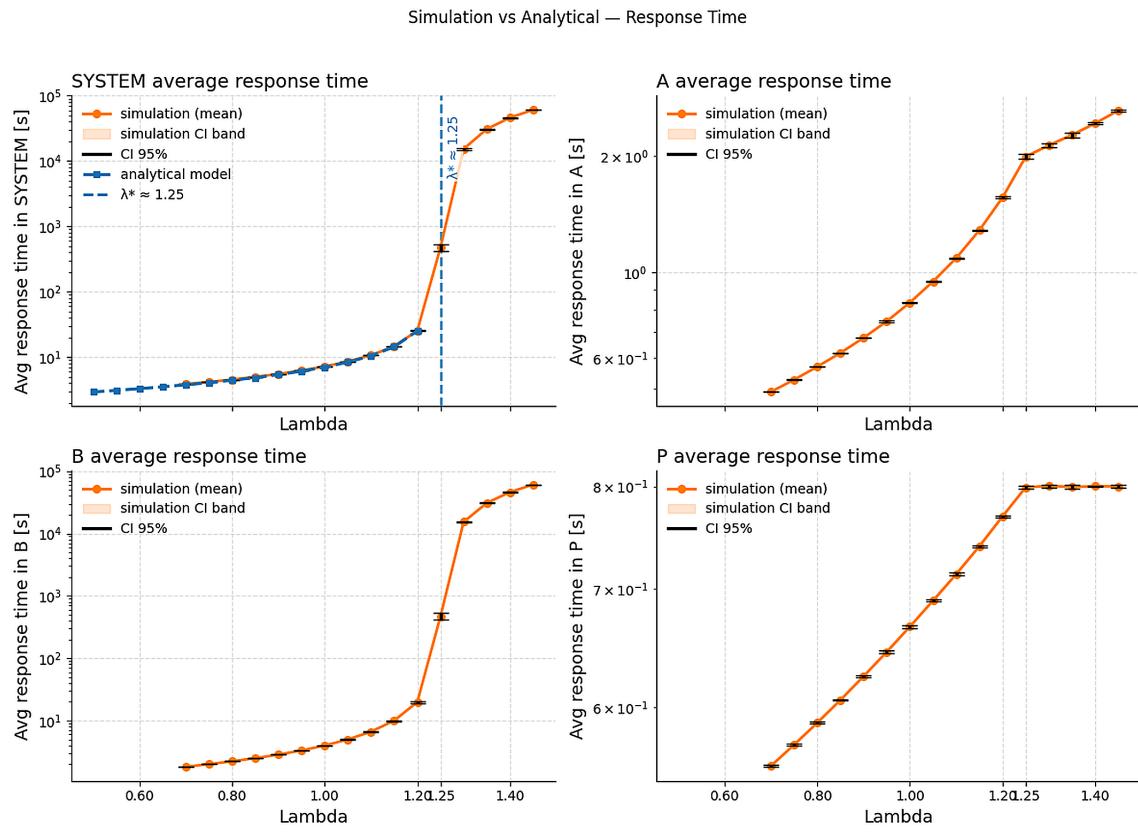


Fig. 12. Obj3 — Response time: crescita ripida in overload, con B saturo.

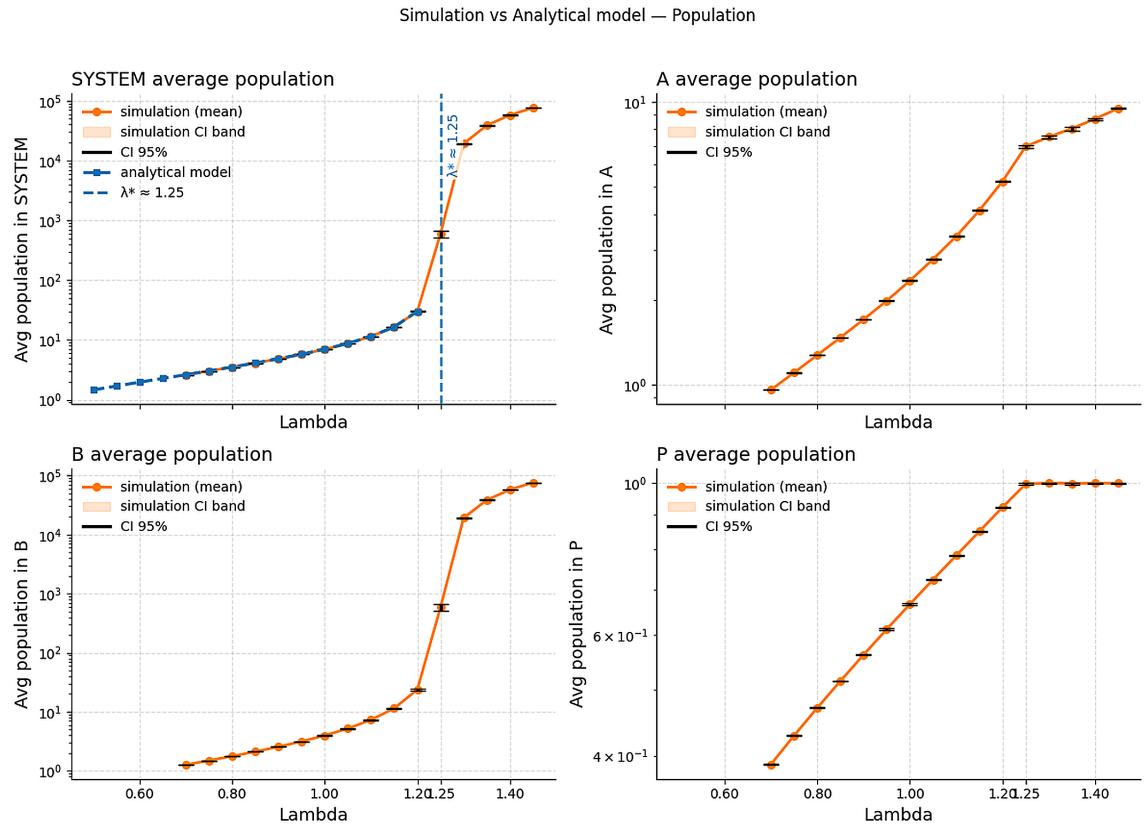


Fig. 13. Obj3 — Popolazione media: aumento non lineare coerente con la saturazione del collo di bottiglia.

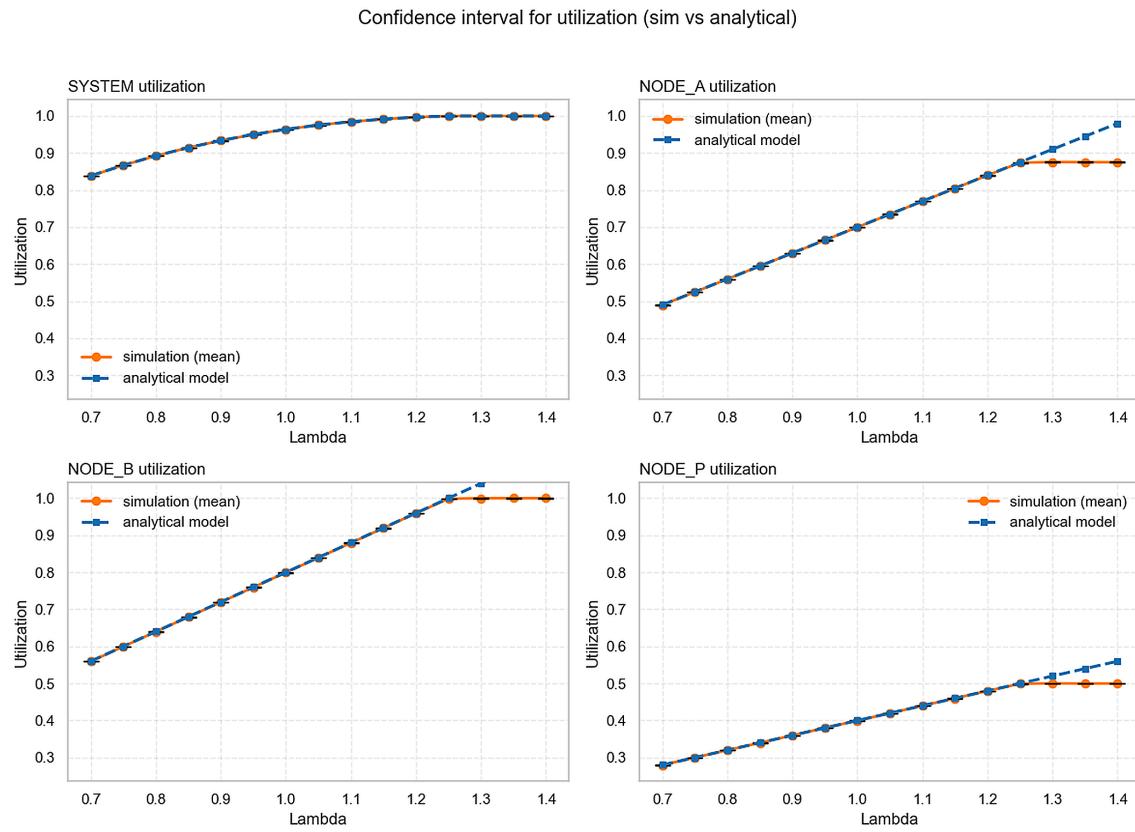


Fig. 14. Obj3 – Utilizzazione: B tende a $\rho \approx 1$; A e P rimangono sub-critici.

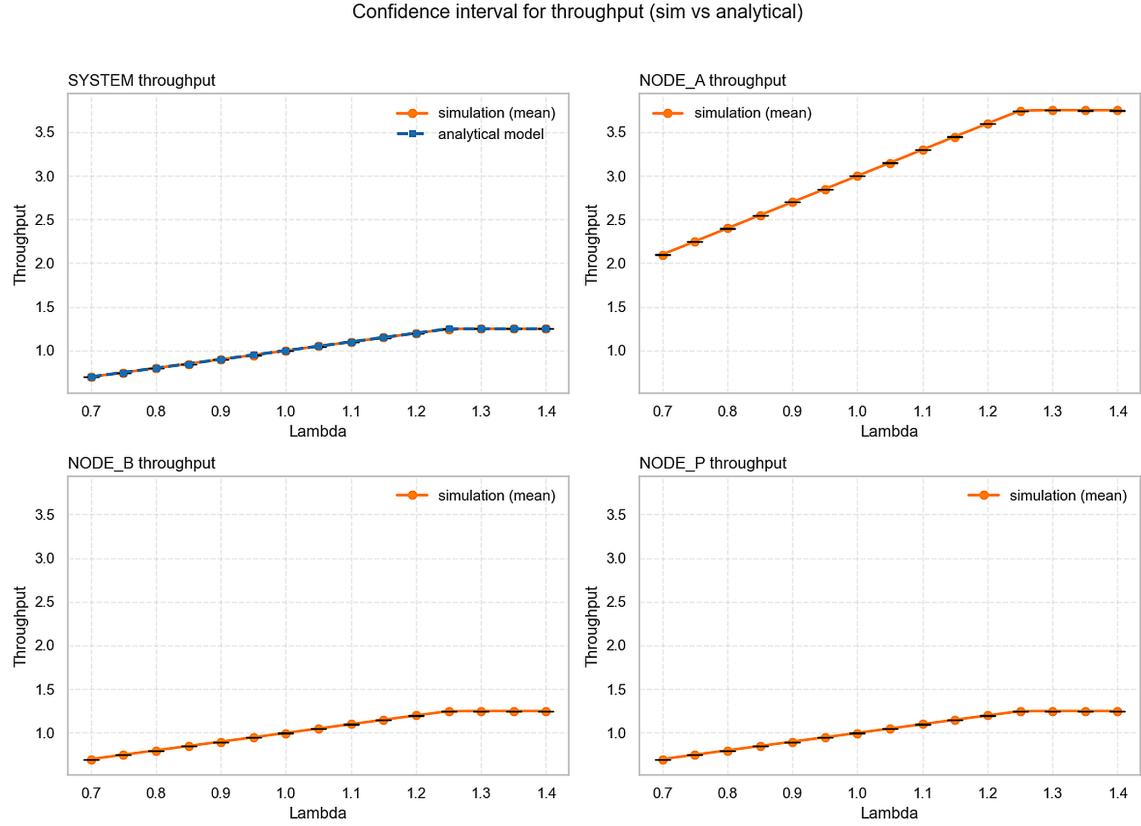


Fig. 15. Obj3 – Throughput: plateau vicino a 1.25 req/s, limitato dal nodo **B**.

Confronto tempi di risposta (Obj1–Obj3). Nei tre scenari la curva di *response time* cresce in modo monotono al crescere di λ , ma con entità diverse. In **Obj1** (baseline) l'aumento è regolare fino a $\lambda \approx 1.2$, poi accelera perché **B** si avvicina alla saturazione; **Obj2** (2FA) mantiene la stessa forma ma le curve si traslano verso l'alto: l'aumento della domanda su A_3 e P spinge $E[T_s]$ a valori maggiori a parità di λ , pur lasciando invariato il collo di bottiglia (**B**). In **Obj3** (overload) la curva “esplode”: il contributo locale di **B** diverge e il tempo di risposta di sistema cresce rapidamente (instabilità), mentre A e P restano relativamente piatti; ciò conferma che la capacità effettiva è limitata da **B** e che, superata $\lambda^* \approx 1.25$, il throughput non cresce più mentre i tempi e popolazioni divergono.

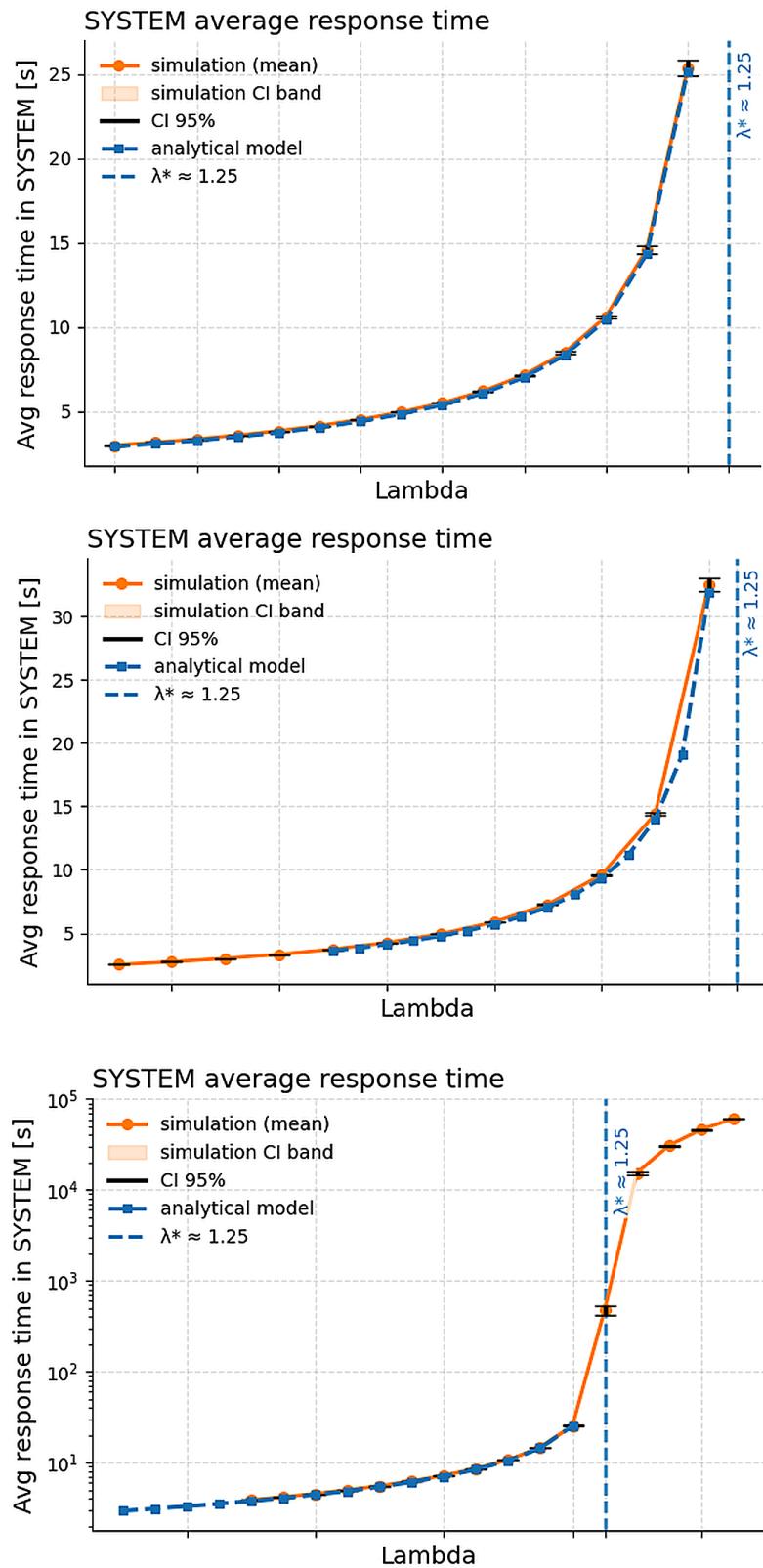


Fig. 16. Confronto **response time** (sim vs analitico, CI 95%): Obj1 (baseline) | Obj2 (2FA) | Obj3 (overload).

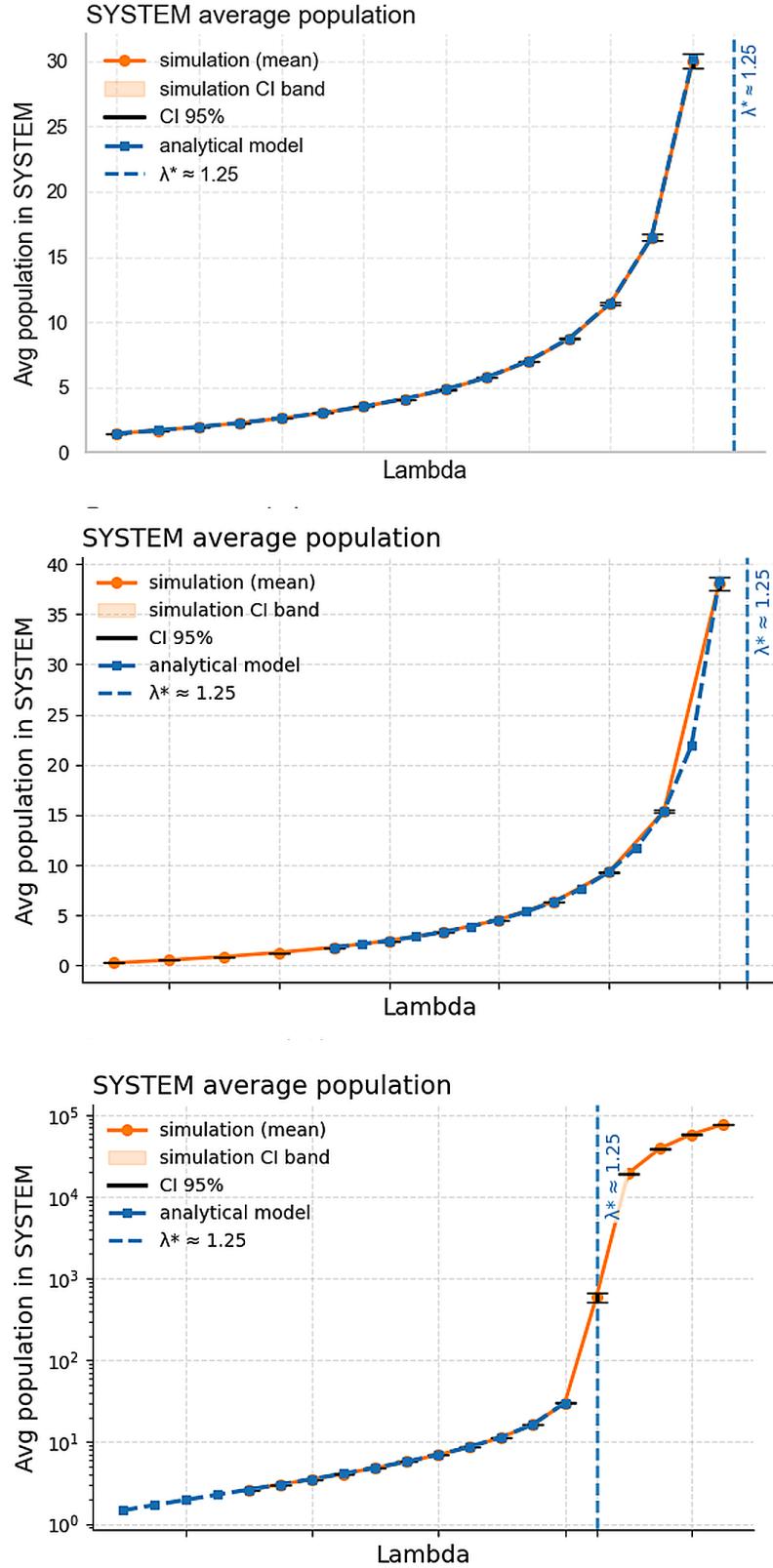


Fig. 17. Confronto popolazione di sistema (mean over seeds, CI 95%): Obj1 (baseline) | Obj2 (2FA) | Obj3 (overload). Coerentemente con $E[N] = \lambda E[T_s]$, le curve di Obj2 sono più alte a parità di λ , mentre in overload non si osserva un plateau.

6.4 Obj4 – Routing con uscita anticipata (effetto della probabilità p)

Riprendiamo l'Obj1 introducendo la probabilità p di *uscita* dopo B . All'aumentare di p si riducono le visite medie ai nodi, calano le domande D_k e dunque le utilizzazioni $\rho_k = \lambda D_k$; di conseguenza il *tempo di risposta* e la *popolazione* di sistema diminuiscono.

p	Analitico [s]	Simulazione [s]	CI_95
0.0	25.14423	25.26909	± 0.846
0.1	23.58835	24.04385	± 1.159
0.2	22.66234	23.26452	± 0.791
0.3	22.03933	22.62872	± 0.878
0.4	21.58708	22.19527	± 0.801
0.5	21.24142	21.87507	± 0.807
0.6	20.96725	21.63109	± 0.917
0.7	20.74364	21.45295	± 1.001
0.8	20.55725	21.37176	± 0.757
0.9	20.39916	21.25205	± 0.596
1.0	20.26316	19.90591	± 0.632

Table 5. Sweep della probabilità p e confronto dei tempi di risposta complessivi (Analitico vs Simulazione)

p = probabilità di uscita dal nodo B ($B \rightarrow \text{EXIT}$).

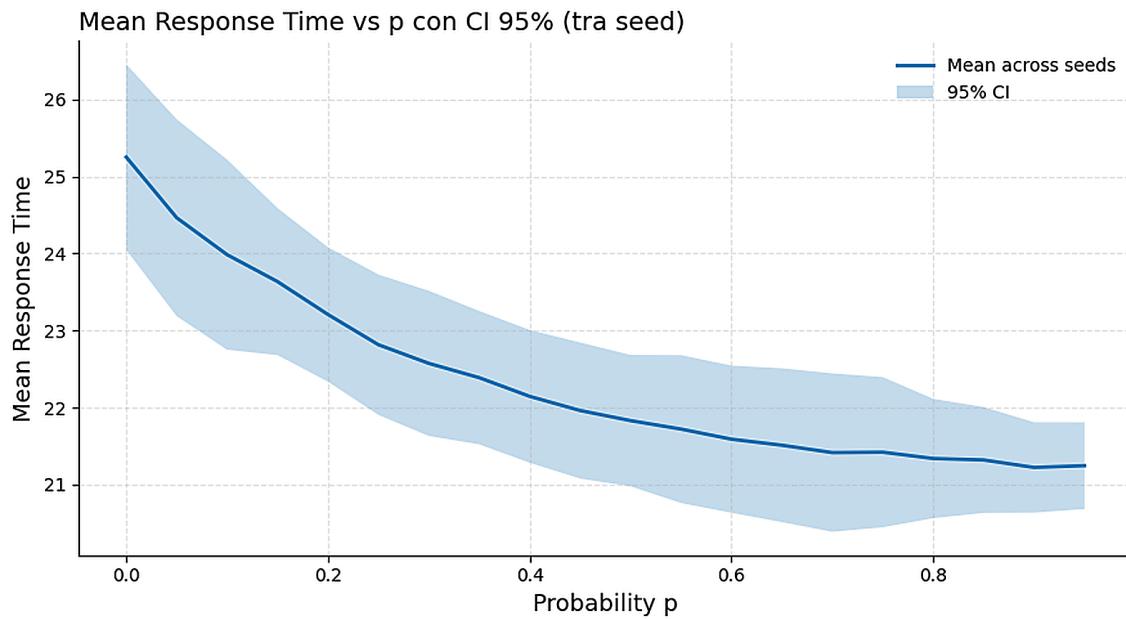


Fig. 18. Obj4 – Mean response time vs probabilità di uscita p (media su seed: il tempo di risposta cala con p).

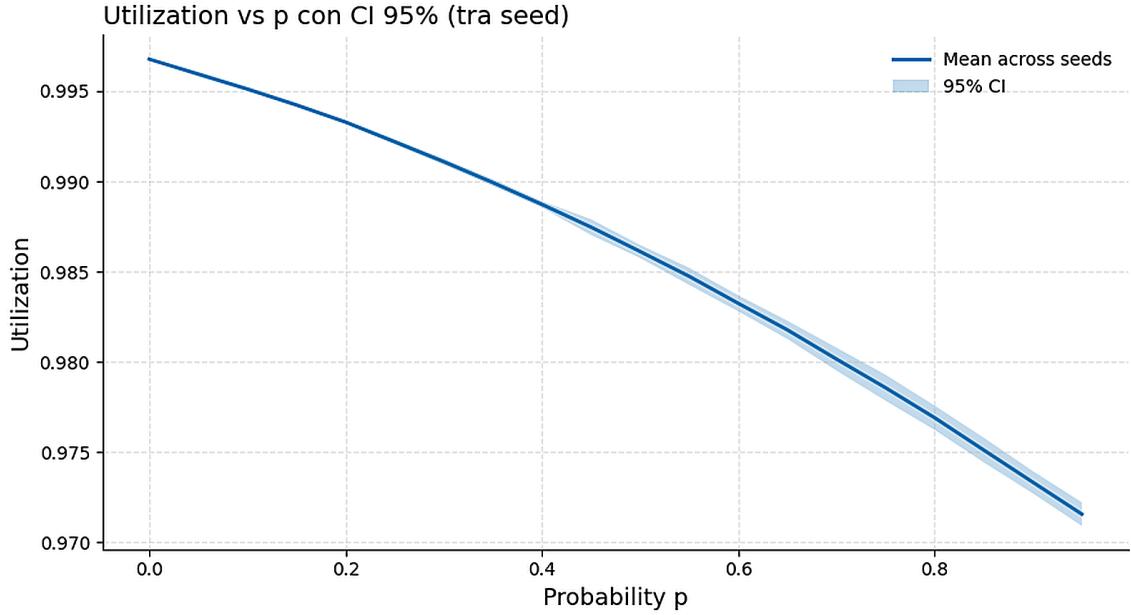


Fig. 19. Obj4 — Utilizzazione vs p : l'aumento di p riduce le domande D_k e quindi ρ , con un calo regolare (qui $\sim 2\text{--}3$ p.p. sull'intervallo).

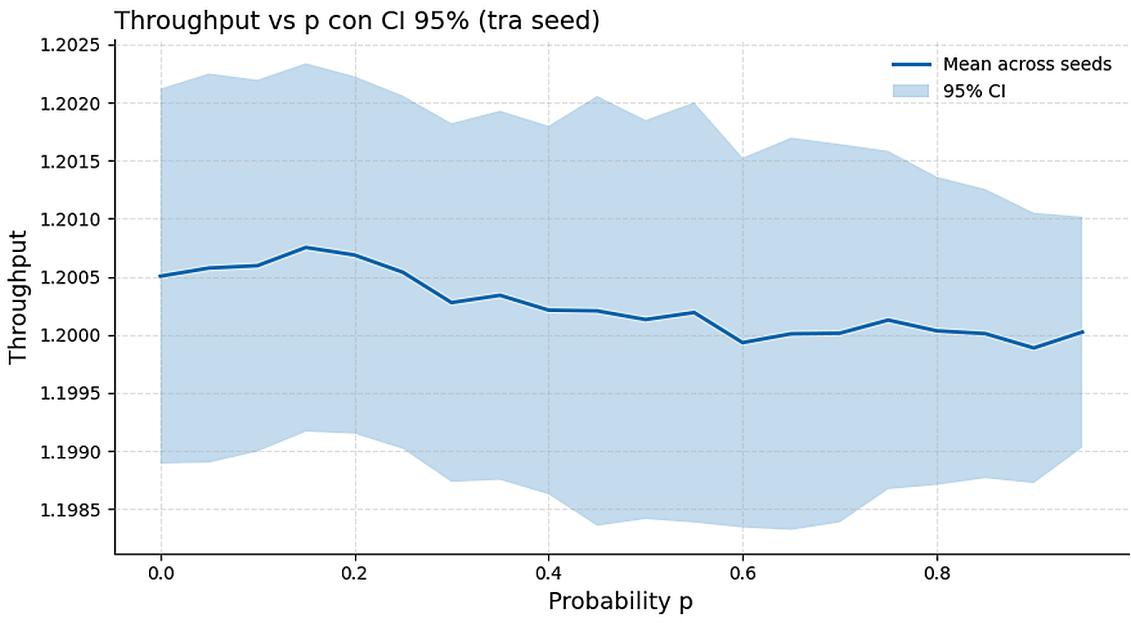


Fig. 20. Obj4 — Throughput vs p : resta pressoché costante vicino a λ finché il sistema rimane stabile.

Nota teorica su p (uscita dopo B, senza loop). Alla fine di B il job:

esce con probabilità $p : P(A_1 \rightarrow B \rightarrow \text{EXIT})$, oppure prosegue con $(1 - p) : P(A_1 \rightarrow B \rightarrow A_2 \rightarrow P \rightarrow A_3)$.

Numero atteso di visite:

$$V_B = 1, \quad V_{A_2} = 1 - p, \quad V_P = 1 - p, \quad V_{A_3} = 1 - p.$$

Le *domande di servizio* corrette diventano

$$\begin{aligned} D_A(p) &= E(S_{A,1}) + (1 - p)[E(S_{A,2}) + E(S_{A,3})], \\ D_B(p) &= E(S_{B,1}), \\ D_P(p) &= (1 - p)E(S_{P,2}). \end{aligned}$$

Quindi le *utilizzazioni* $\rho_k(p) = \lambda D_k(p)$ soddisfano

$$\frac{d\rho_A}{dp} = -\lambda(E(S_{A,2})+E(S_{A,3})) < 0, \quad \frac{d\rho_P}{dp} = -\lambda E(S_{P,2}) < 0, \quad \rho_B(p) = \lambda E(S_{B,1}) \text{ (costante).}$$

All'aumentare di p si riducono linearmente D_A e D_P (e quindi ρ_A, ρ_P), mentre B resta invariato e continua a determinare la capacità $X_{\max} = 1/\max(D_k)$. Ne consegue il calo del *tempo di risposta* e della *popolazione* di sistema, mentre il *throughput esterno* rimane $\approx \lambda$ finché tutti i nodi sono stabili (coerente con i grafici).

6.5 Obj4 – Routing con probabilità di loop

Aumentando la probabilità di rimanere nel ciclo, crescono le visite attese ai nodi (domande D_k), le utilizzazioni tendono alla saturazione e il *tempo di risposta* esplode; i *completamenti* per unità di tempo si riducono.

p	Analitico [s]	Simulazione [s]	CI_95
0.0	25.14423	25.26909	± 0.846
0.01	31.97366	32.36006	± 1.587
0.02	45.48390	45.31261	± 6.874
0.03	85.67690	89.51457	± 27.847
0.04	∞	655.01741	± 287.636
0.05	∞	4535.96379	± 572.666

Table 6. Sweep della probabilità p e confronto dei tempi di risposta complessivi (Analitico vs Simulazione)

p = probabilità di loop con classe 2 ($A_2 \rightarrow B$).

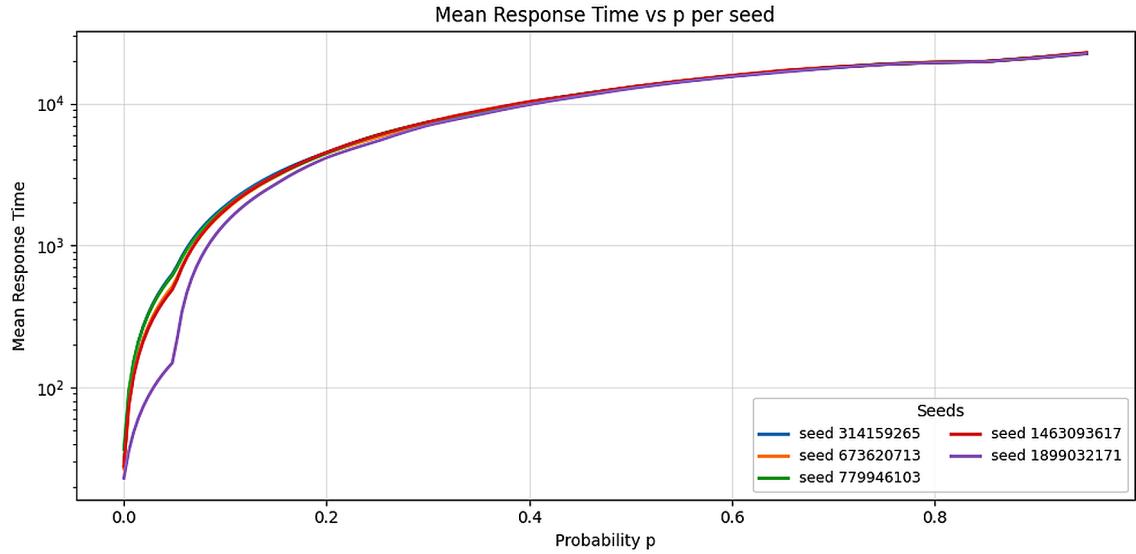


Fig. 21. Obj4-loop — Mean response time vs probabilità di loop (per seed).

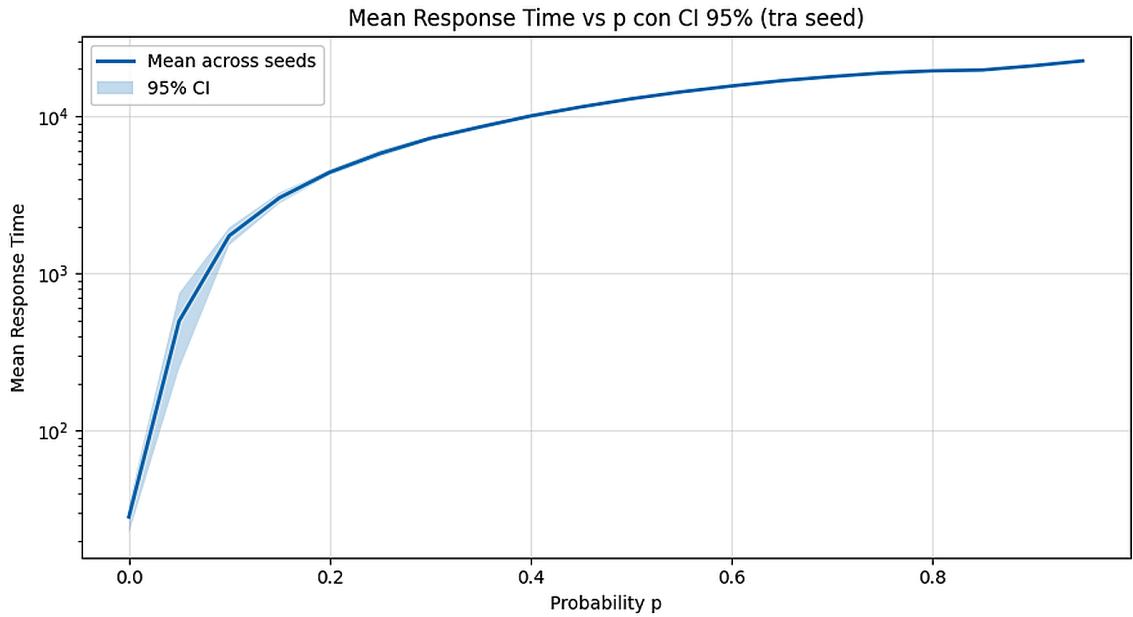


Fig. 22. Obj4-loop — Mean response time con CI 95% (media tra seed): crescita rapida all' aumentare della probabilità di loop.

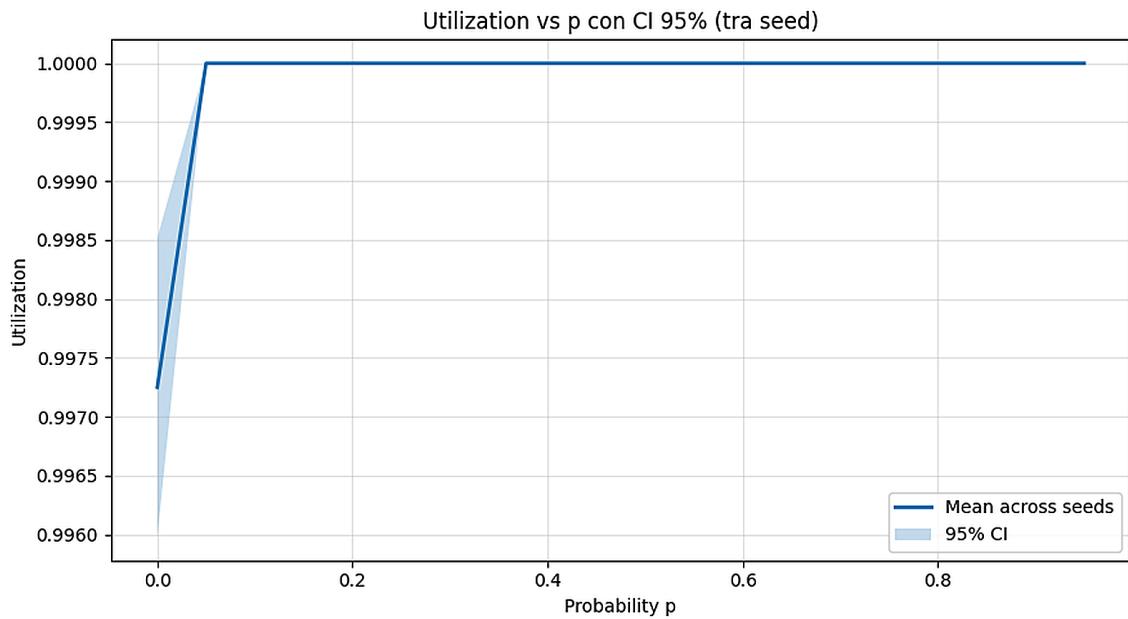


Fig. 23. Obj4-loop — Utilizzazione vs probabilità di loop: ρ tende a 1 per il collo di bottiglia.

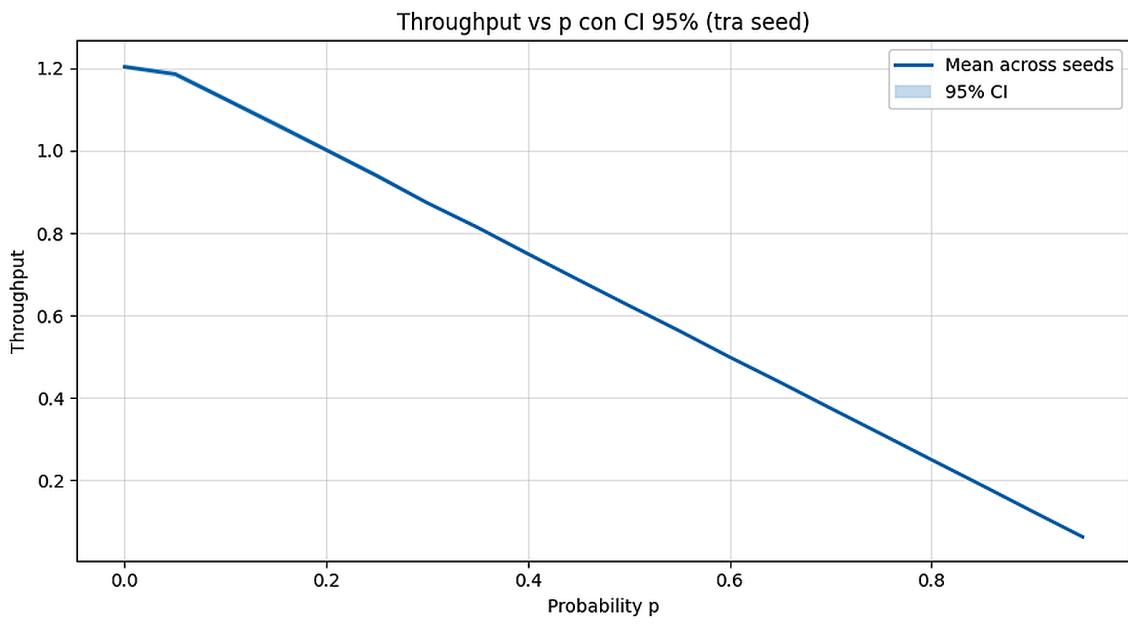


Fig. 24. Obj4-loop — Throughput esterno vs probabilità di loop: diminuisce perché servono più visite per completare.

Nota teorica sul loop (visite attese e domande). Sia $q = P(A_2 \rightarrow B, 1)$ la probabilità di tornare da A_2 a B e $r = P(A_2 \rightarrow P, *)$ la probabilità di proseguire da A_2 verso P . Poniamo

$$Z \equiv 1 - q(1 - r).$$

Allora il numero atteso di visite indotte dal loop è

$$V_B = \frac{1}{Z}, \quad V_{A_2} = \frac{q}{Z}, \quad p = P(\text{visitare } P \text{ prima di uscire}) = \frac{qr}{Z}.$$

Le *domande di servizio* corrette diventano

$$D_A = E(S_{A,1}) + V_{A_2} E(S_{A,2}) + p E(S_{A,3}), \quad D_B = V_B E(S_{B,1}), \quad D_P = p E(S_{P,2}),$$

e le utilizzazioni $\rho_k = \lambda D_k$. Quando la *probabilità di rimanere nel loop* cresce (cioè $q(1 - r) \uparrow$), $Z \downarrow$, V_B e V_{A_2} esplodono $\Rightarrow D_k \uparrow$, $\rho_k \rightarrow 1$, i *tempi di risposta* divergono, e il *throughput esterno* si riduce (poiché occorrono più visite per completare un job, la capacità $X_{\max} = 1/\max_k D_k$ cala).

6.6 Obj4 – Routing con loop probabilistico ($\lambda = 0.5$)

In questo scenario i job possono *loopare* tra B e A_2 prima di raggiungere P . All'aumentare di p cresce il numero atteso di visite interne, con effetto moltiplicativo sulle domande D_k : la coda del collo di bottiglia si satura e le prestazioni degradano rapidamente.

Teoricamente, con probabilità di ritorno $q = \Pr(A_2 \rightarrow B)$ e di proseguire verso P $r = \Pr(A_2 \rightarrow P)$, le visite attese sono $E_B = \frac{1}{1-q(1-r)}$, $E_{A_2} = \frac{q}{1-q(1-r)}$ e la probabilità di visitare P è $p = \frac{qr}{1-q+qr}$.

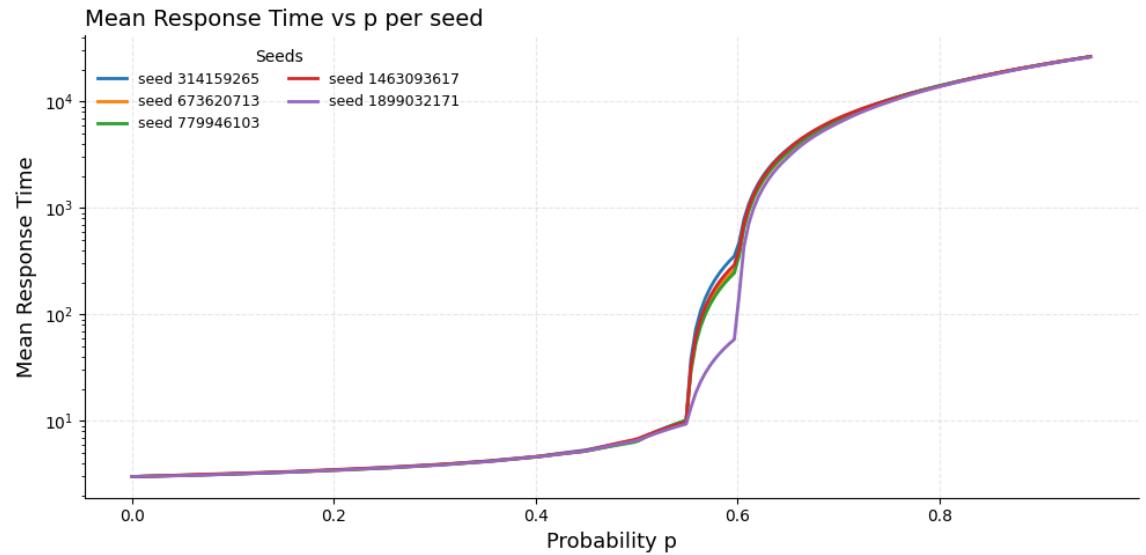


Fig. 25. Obj4-loop ($\lambda = 0.5$) – *Mean response time* vs p per seed. Si osserva crescita marcata dopo la soglia in cui il collo di bottiglia va in saturazione.

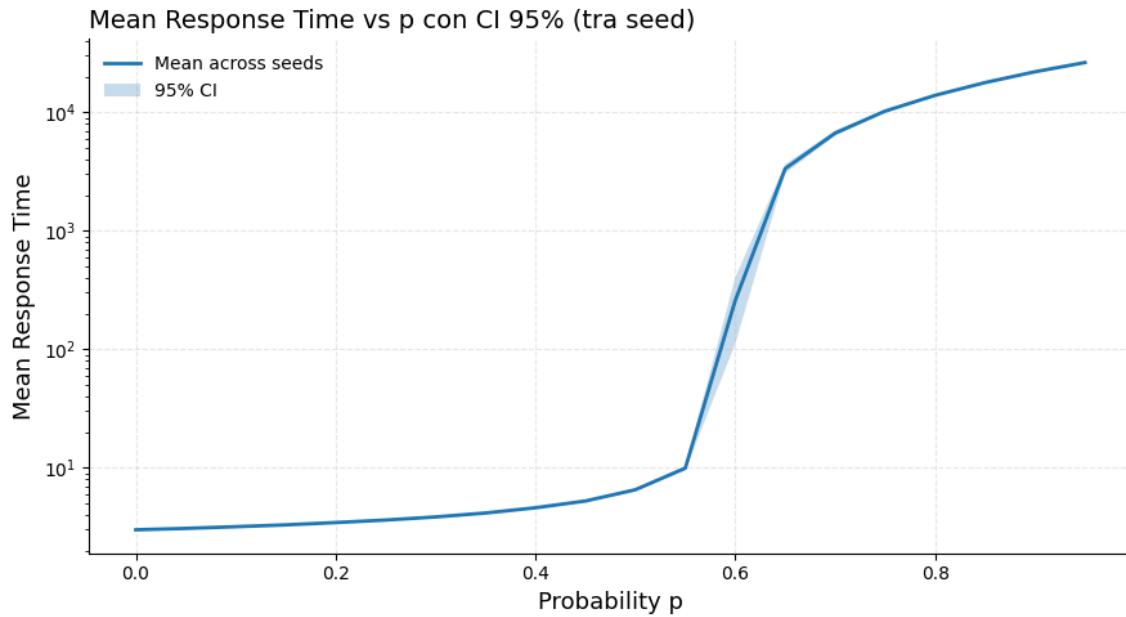
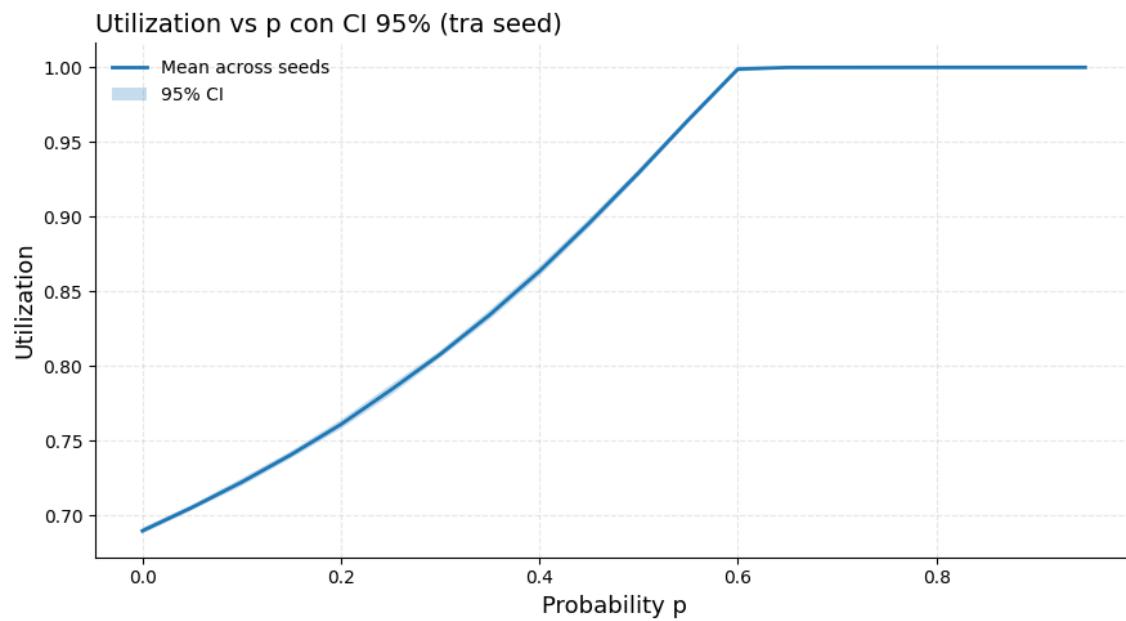


Fig. 26. Obj4-loop — *Mean response time* (media tra seed, CI 95%) vs p : esplosione del RT oltre $p \approx 0.6$ per l'aumento delle visite interne.



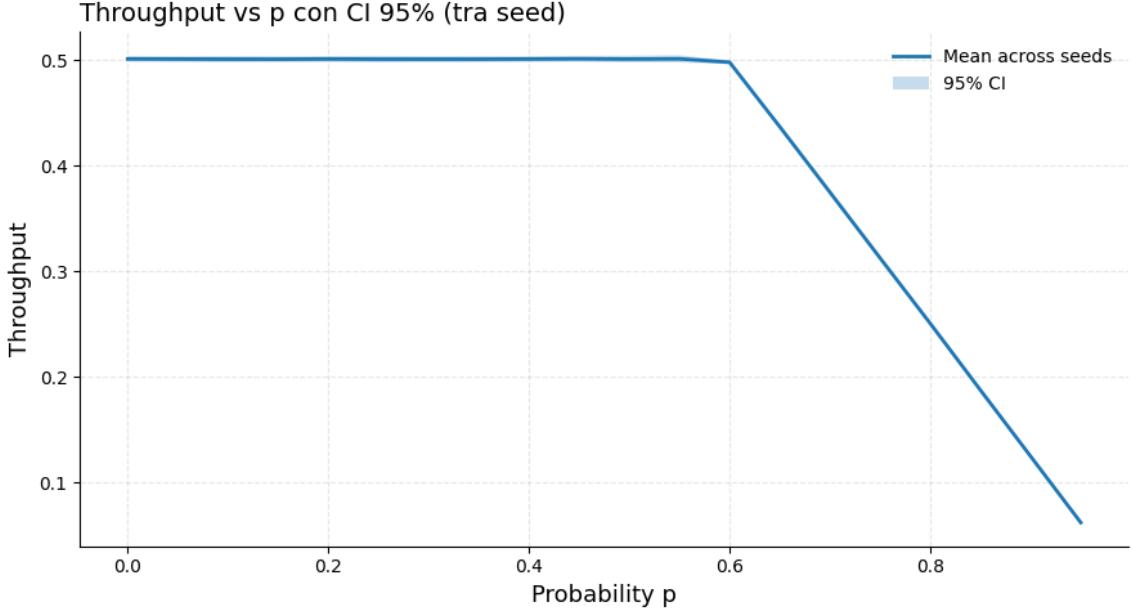


Fig. 27. Obj4-loop – Throughput esterno vs p : inizialmente costante ($\approx \lambda$), poi cala quando il lavoro interno cresce più velocemente della capacità.

6.7 Obj5 – Scaling verticale di B (variazione di $E[S_B]$)

Aumentiamo la capacità del nodo B riducendo il suo tempo medio di servizio $E[S_B]$ (*vertical scaling*). Teoricamente il limite di capacità del nodo è

$$X_{\max,B} = \frac{1}{E[S_B]}.$$

Al diminuire di $E[S_B]$ (servizio più rapido) il collo di bottiglia si attenua: il throughput sostenibile cresce e il punto di saturazione ($\rho_B \rightarrow 1$) si sposta verso destra.

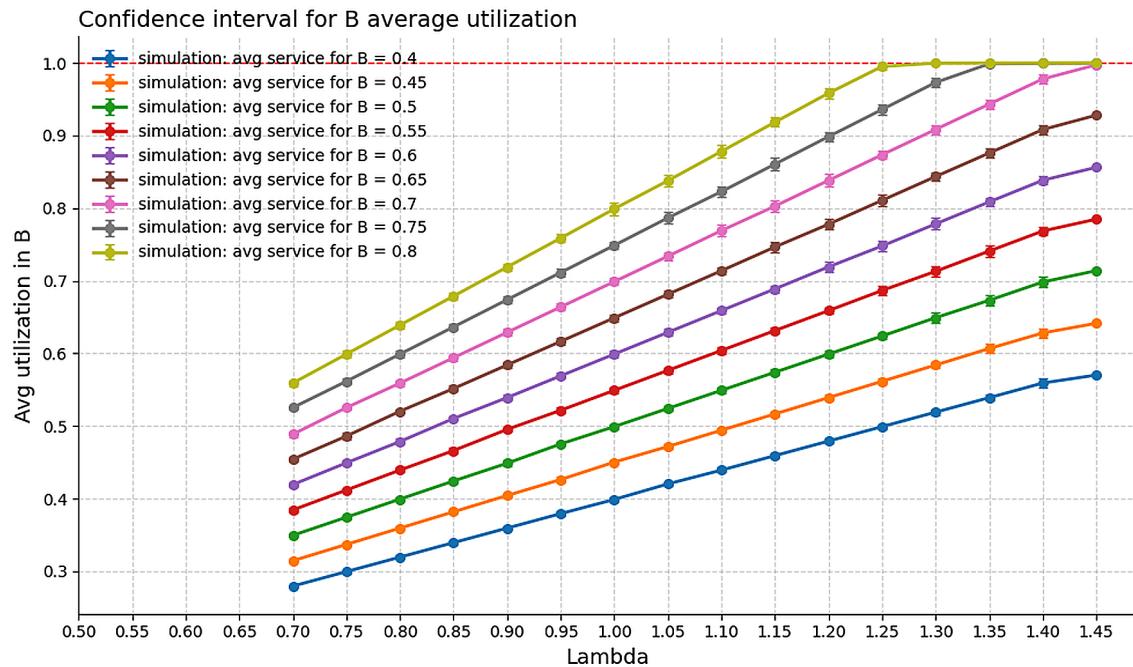


Fig. 28. Obj5 – Utilizzazione media in B al variare di λ e di $E[S_B]$. Le curve con $E[S_B]$ più basso saturano più tardi; la linea tratteggiata rossa indica $\rho_B = 1$.

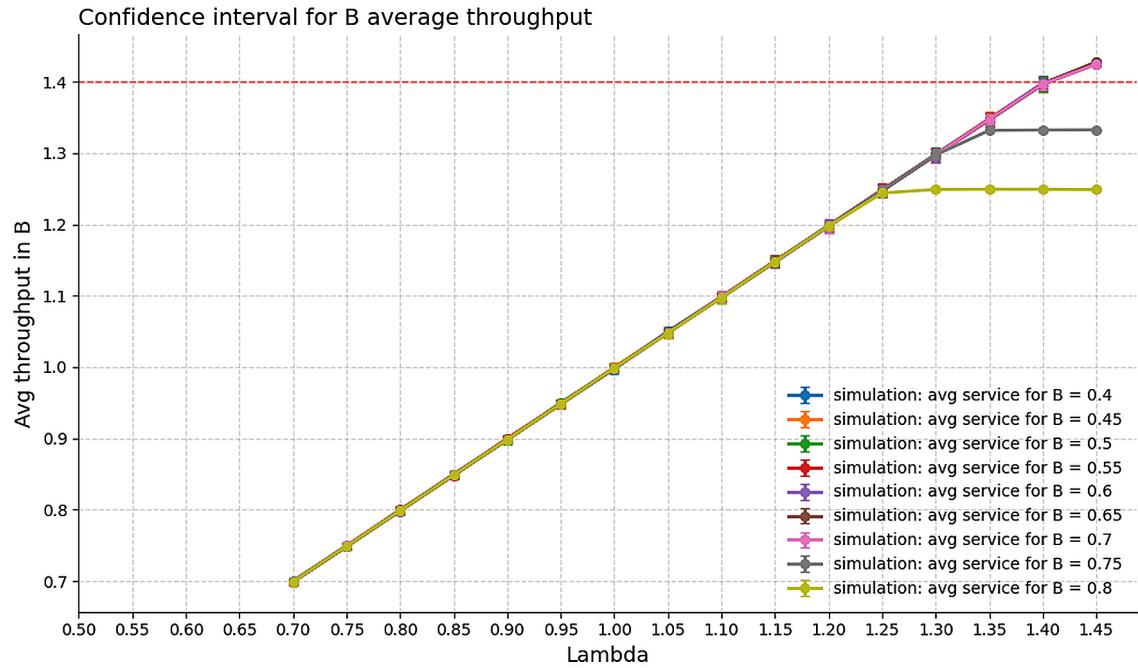


Fig. 29. Obj5 – Throughput medio in B: cresce quasi linearmente con λ fino a $X_{\max,B} = 1/E[S_B]$, poi va in plateau. Riducendo $E[S_B]$ il plateau si sposta a destra, segno di collo di bottiglia mitigato.

Osservando i grafici che il parametro migliore è pari ad $E[S_B] = 0.65$ s, poichè, ha la migliore combinazione tra utilizzazione e throughput.

6.8 Transitori (Obj1): effetto del cold start e del cut sugli arrivi

Nel transitorio senza warm-up, il sistema parte vuoto e le code si formano progressivamente: le metriche (tempo di risposta e popolazione) risultano inizialmente sottostimate e convergono ai valori di regime. Confrontiamo tre configurazioni: *sistema vuoto*, *10 arrivi iniziali* e *100 arrivi iniziali*.

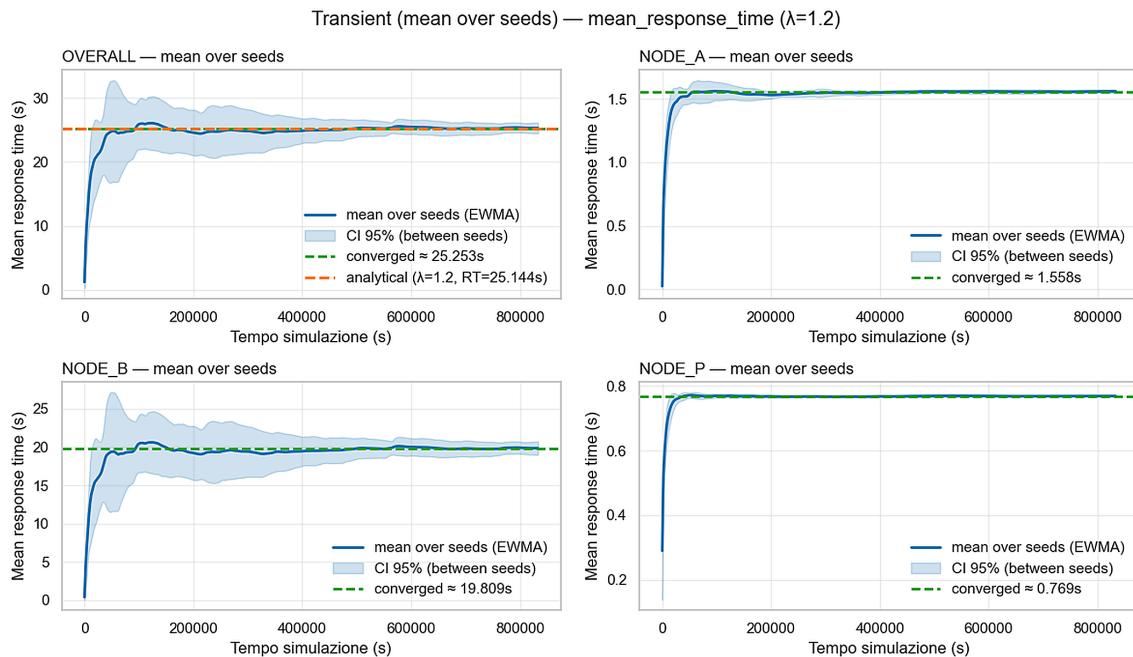


Fig. 30. Transient (simple) — RT medio (mean over seeds, CI 95%): convergenza verso il valore analitico.

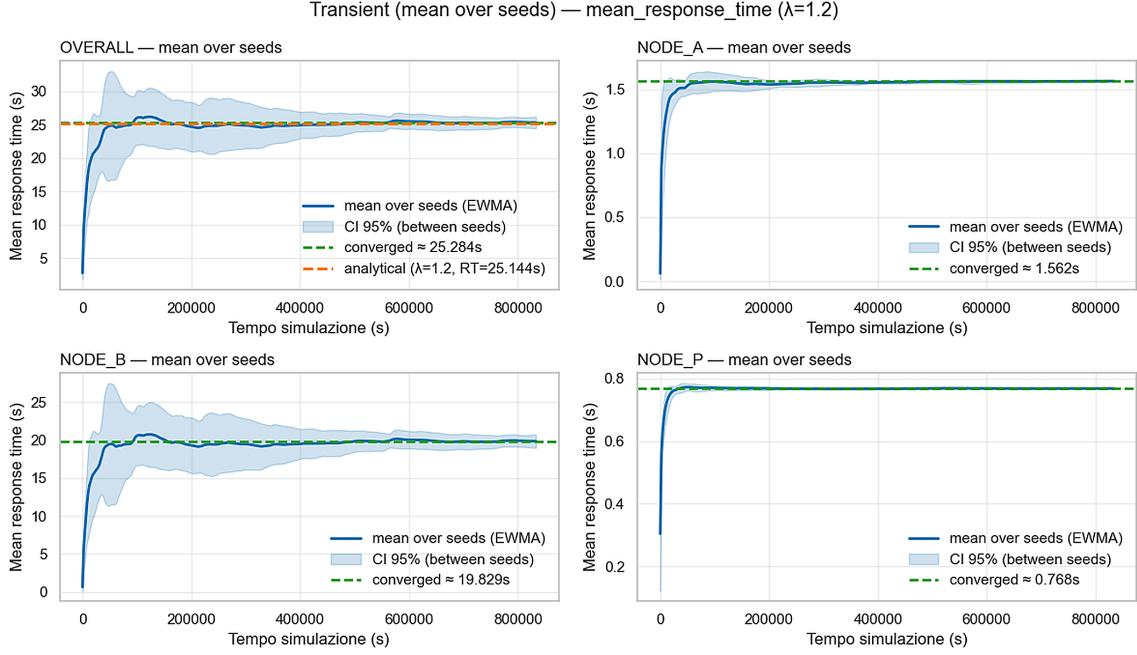


Fig. 31. Transient con 10 arrivi iniziali — RT: il bias iniziale persiste, soprattutto su B.

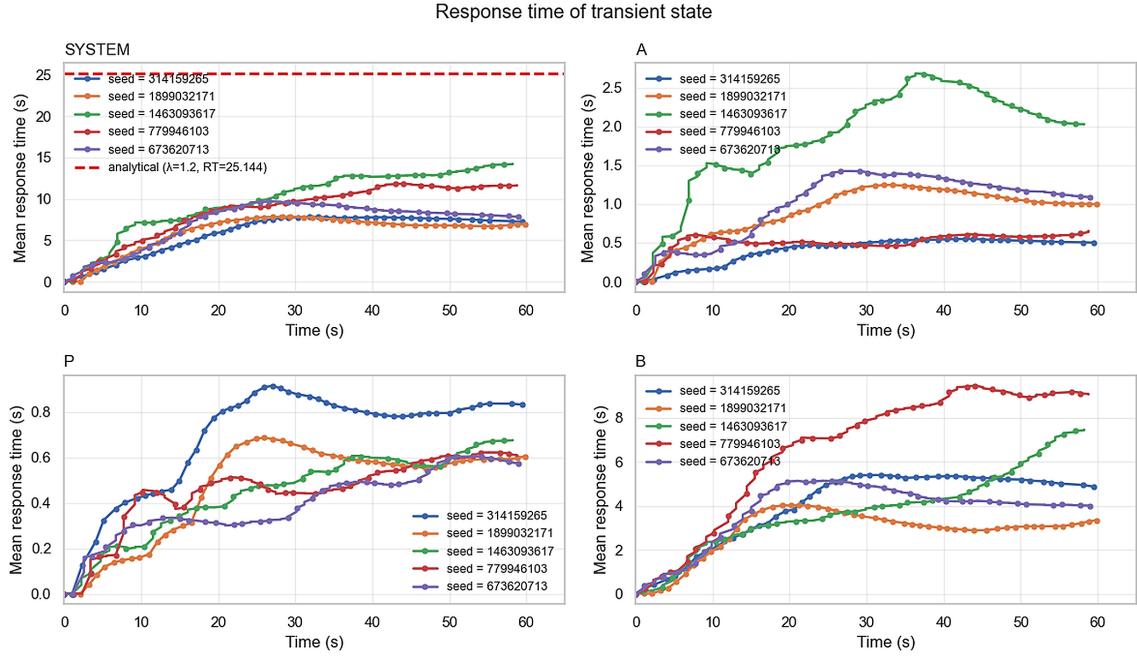


Fig. 32. Orizzonte finito 60 s — confronto per-seed con cut 10: dispersione ancora ampia nel tratto iniziale.

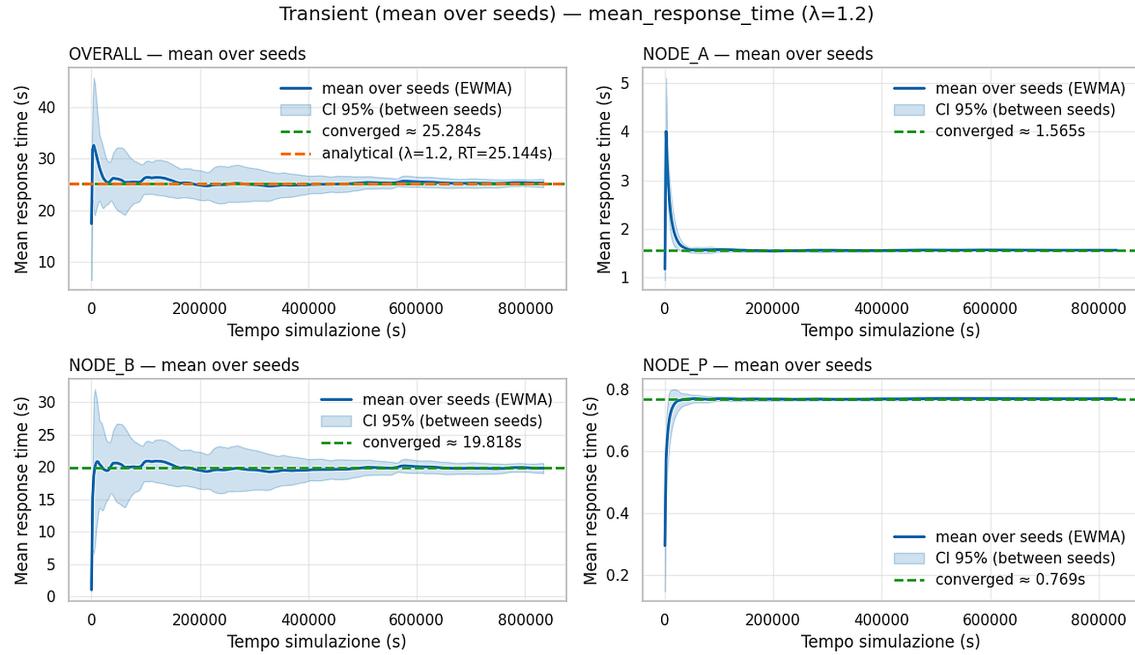


Fig. 33. Transient con 100 arrivi iniziali — RT medio: stime allineate al plateau entro le CI.

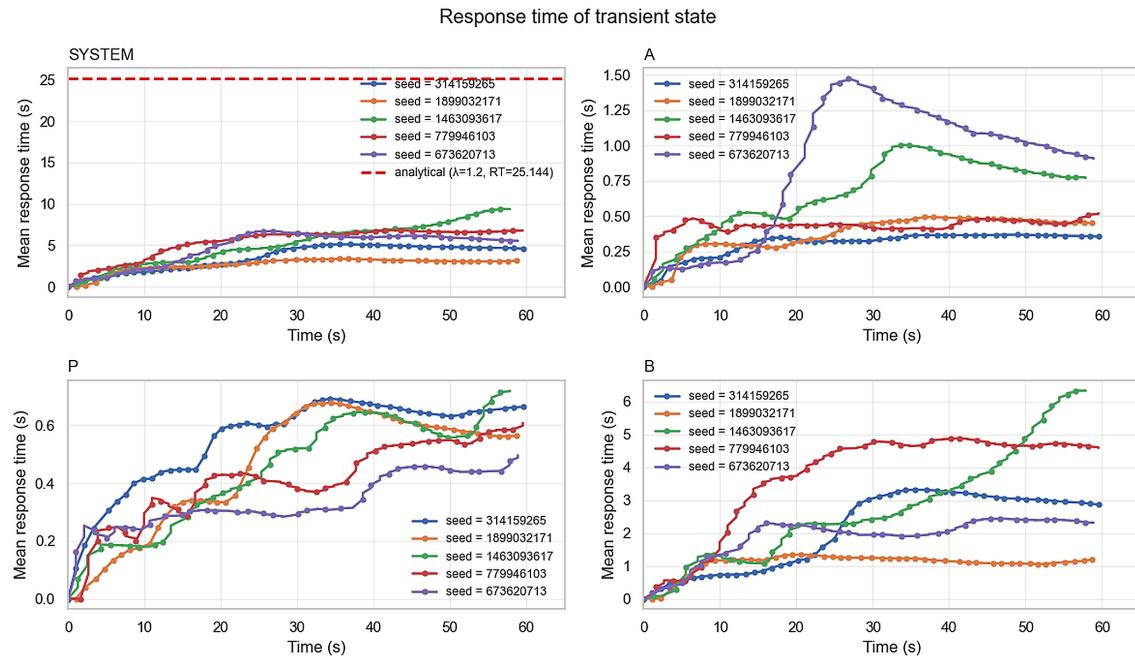


Fig. 34. Orizzonte finito 60 s — per-seed con cut 100: variabilità ridotta e convergenza più rapida.

6.9 Transitori (Obj2 – 2FA)

Con 2FA aumentano i tempi medi su A_3 e P ($E[S_{A,3}] = 0.15$ s, $E[Sp] = 0.7$ s), quindi il plateau del tempo di risposta si porta a ≈ 32 s mentre B resta il collo di bottiglia. Confrontiamo: *sistema vuoto*, *10* e *100* arrivi iniziali.

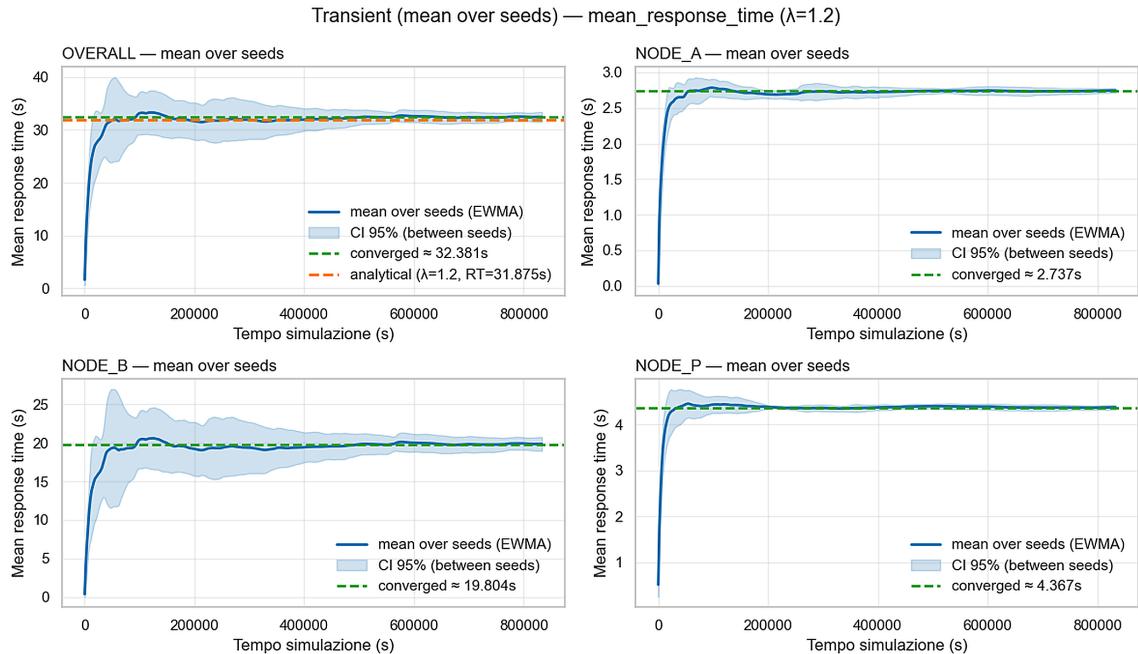


Fig. 35. Obj2/transient (simple) — RT medio (mean over seeds, CI 95%): salita e assestamento a ~ 32 s.

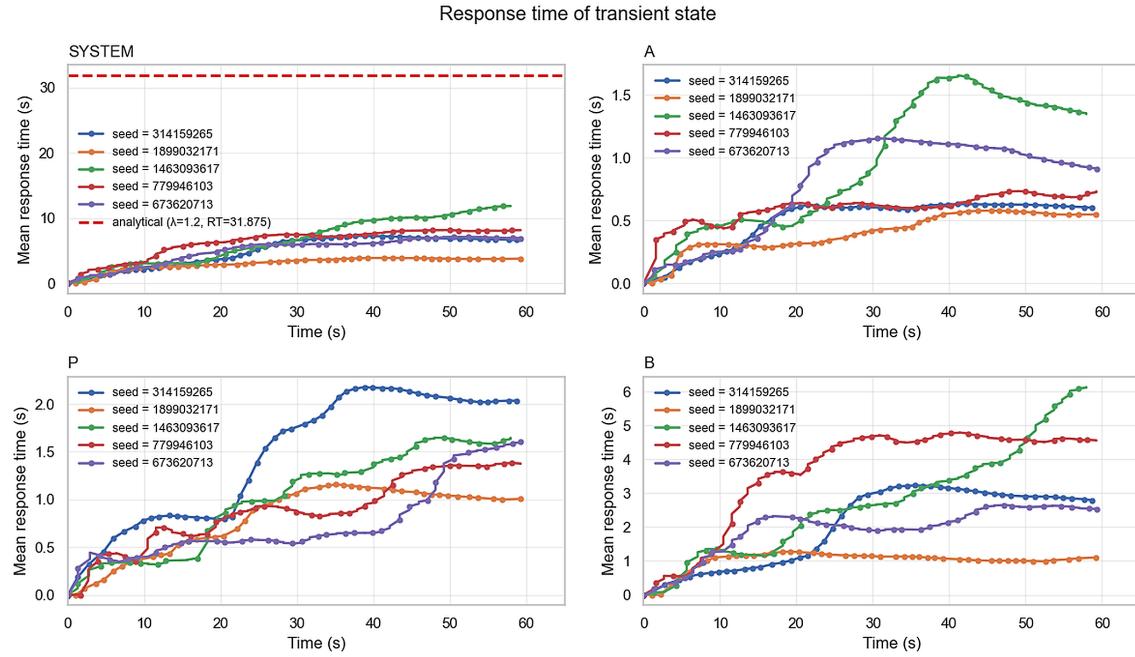
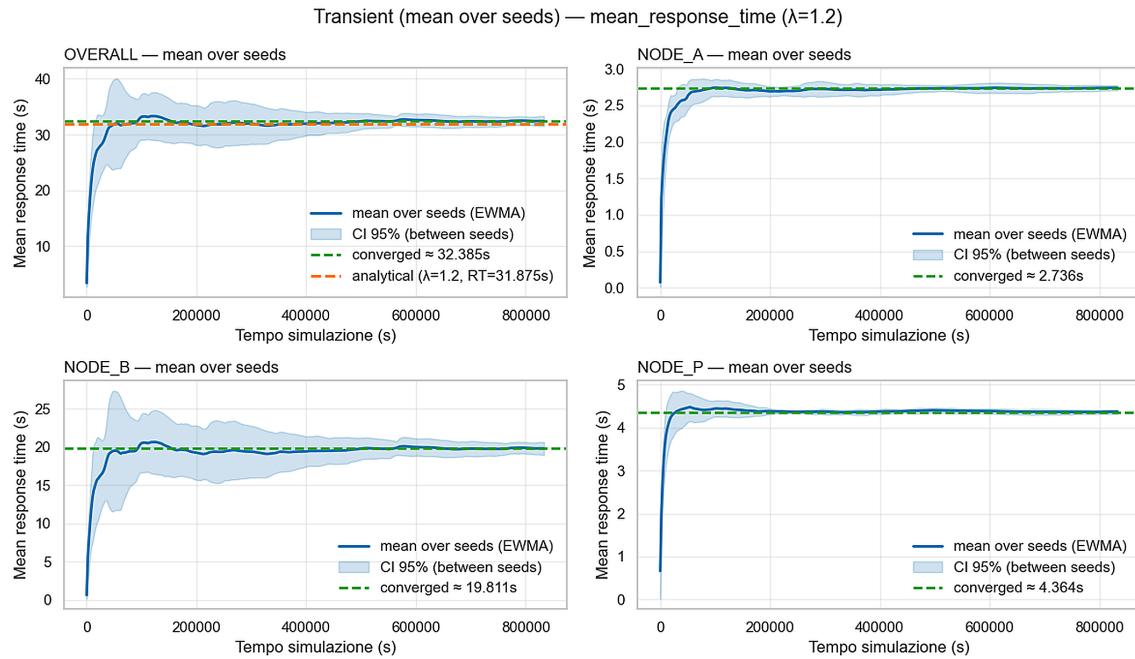


Fig. 36. Obj2 Orizzonte finito 60 s — per-seed

Fig. 37. Obj2/transient — 10 arrivi iniziali: il bias da cold start è ancora presente, RT_{sys} sotto il plateau.

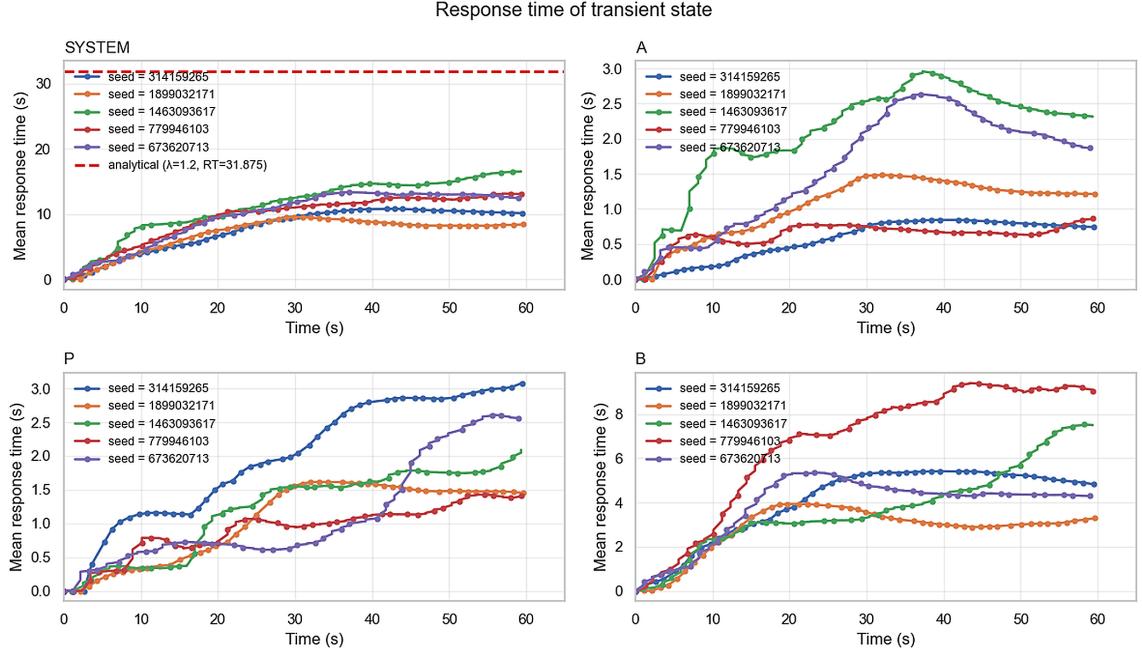


Fig. 38. Orizzonte finito 60 s — diagnostica per-seed con 10 arrivi iniziali

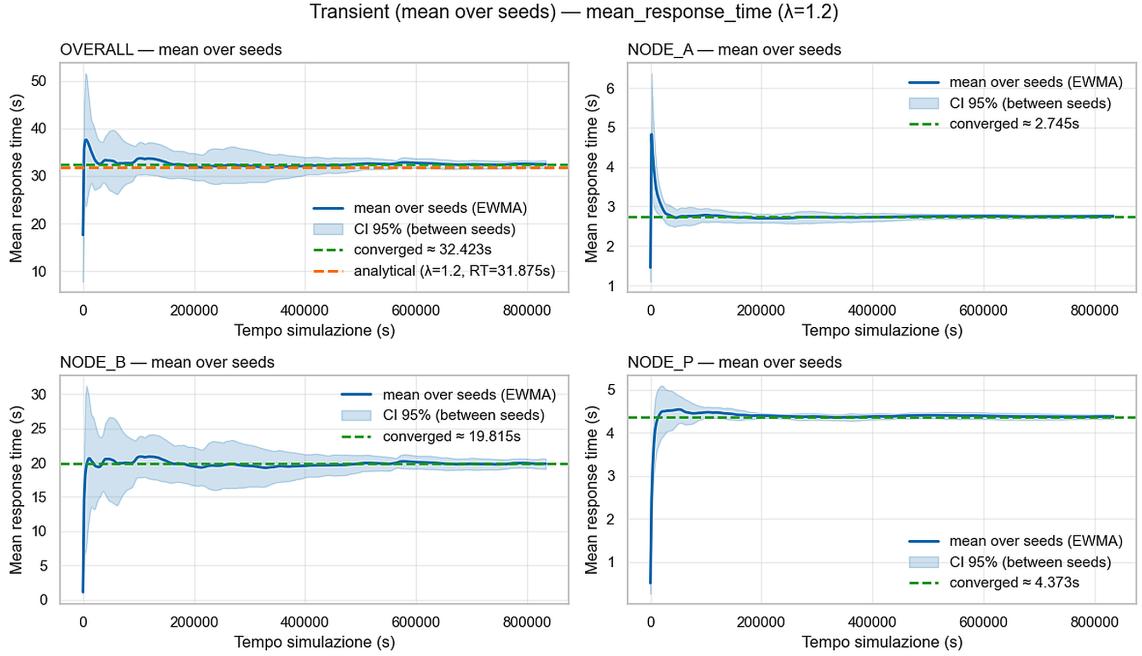


Fig. 39. Obj2/transient — 100 arrivi iniziali: stime allineate al plateau entro le CI, confronto corretto con l'analitico.

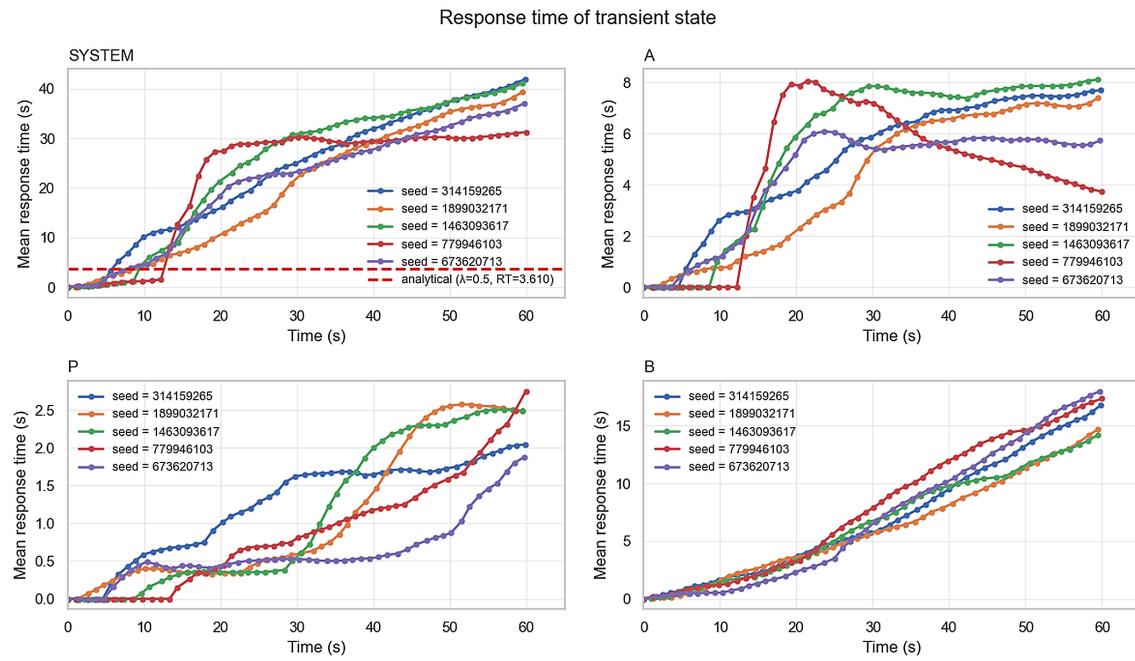


Fig. 40. Orizzonte finito 60 s — per-seed con 100 arrivi iniziali

6.10 Transitori (Obj3 – overload)

Con $\lambda = 1.4 > X_{\max,B} = 1/E[S_B]$ (con $E[S_B] = 0.8$ s), il sistema è instabile: il nodo **B** satura e i tempi di risposta non presentano plateau, crescendo nel tempo.

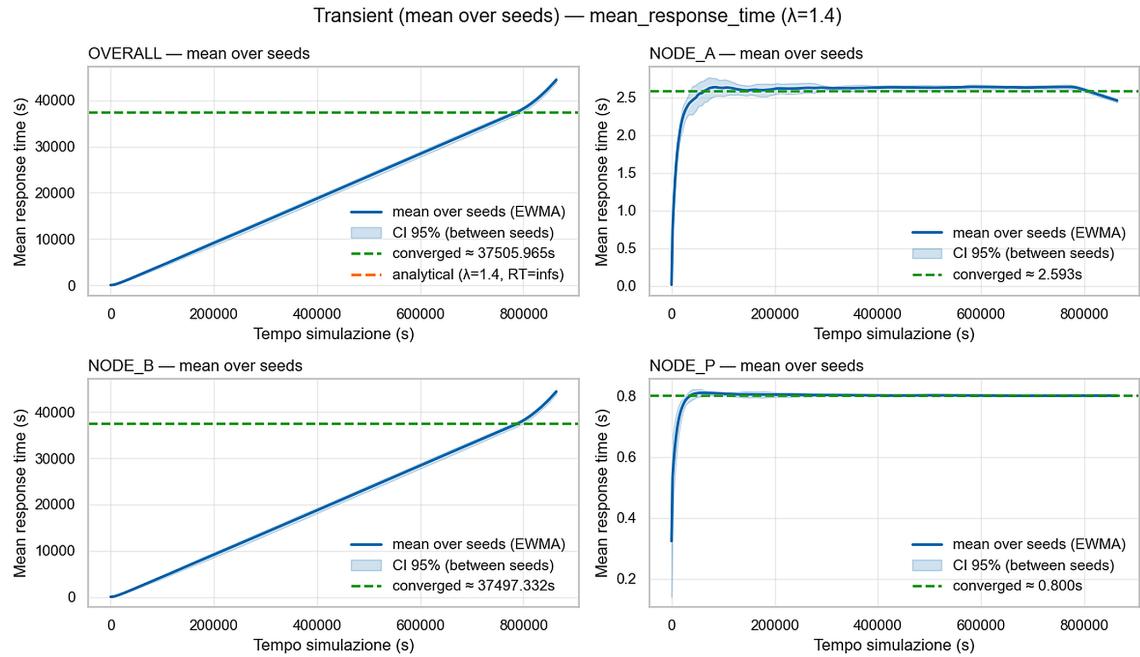


Fig. 41. Obj3/transient (simple) — RT per-seed (SYSTEM, A, B, P): nessuna stabilizzazione, crescita continua in overload.

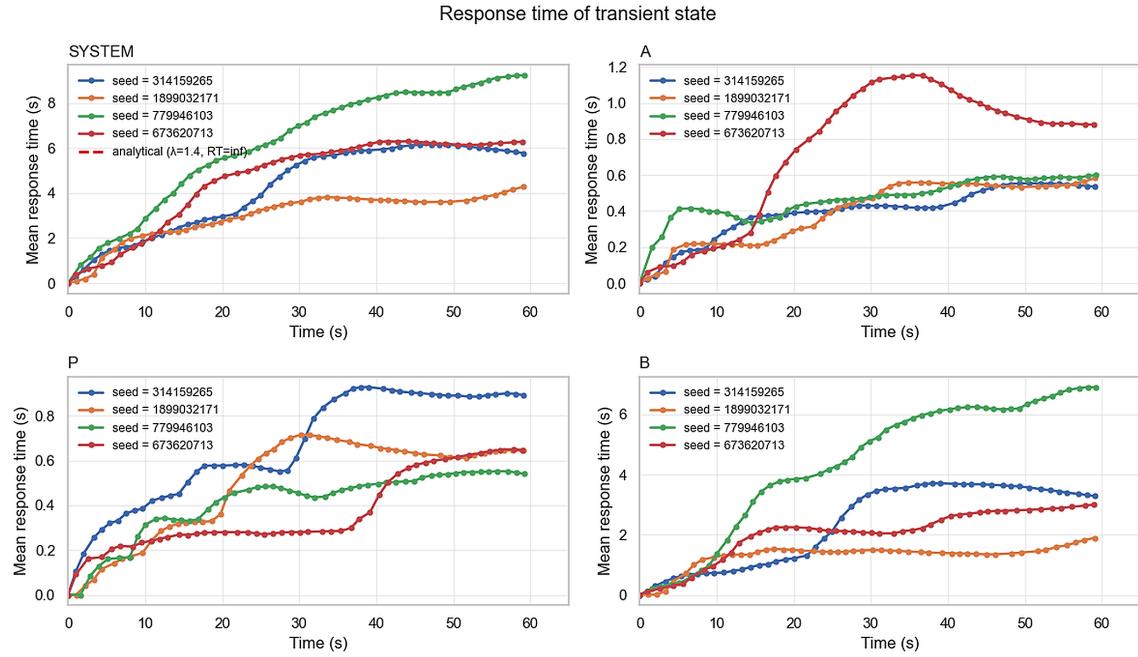
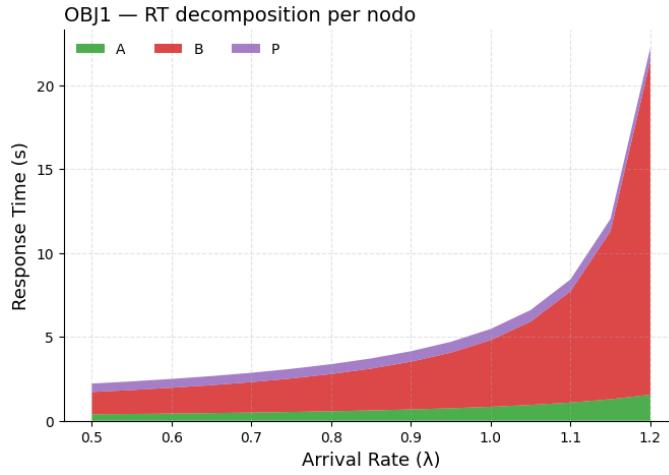


Fig. 42. Obj3/transient — diagnostica le curve non convergono (instabilità).

6.11 Decomposizione del tempo di risposta per nodo (Obj1–Obj3)

La scomposizione per nodo mostra come varia il contributo di A, B e P a $E[T_s]$ al crescere di λ . Nel baseline (Obj1) la parte di B domina e cresce in modo monotono verso la soglia di saturazione; con 2FA (Obj2) aumentano i contributi di A e P ma B resta il collo di bottiglia; in overload (Obj3) la componente di B esplode, segnalando instabilità del sistema.

Fig. 43. Obj1 — Decomposizione $E[T_s]$ per nodo: B cresce fino a dominare il totale.

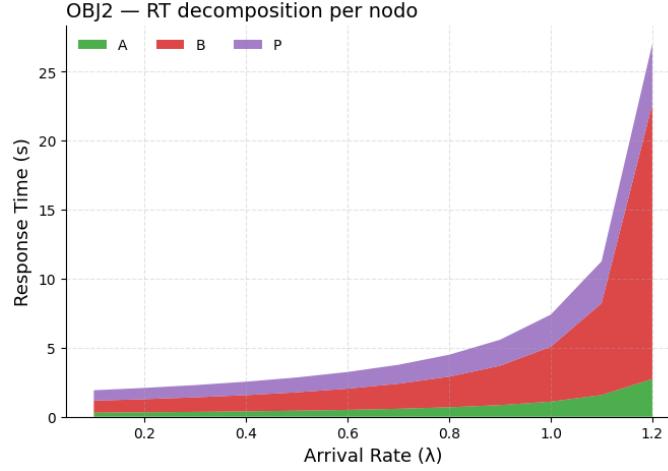


Fig. 44. Obj2 — Con 2FA aumentano le quote di **A** e **P**; **B** resta il limite di capacità.

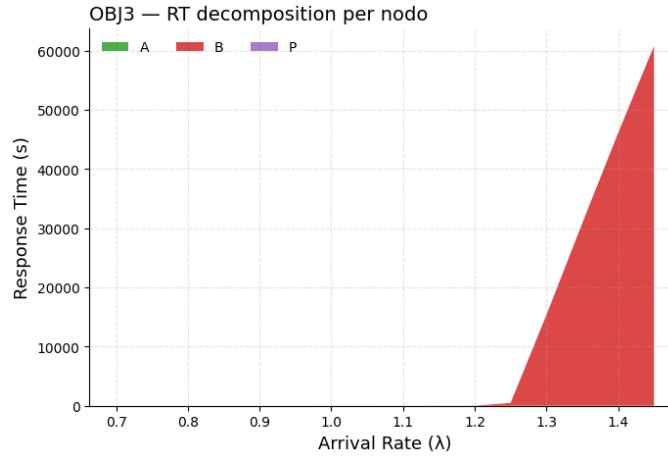


Fig. 45. Obj3 — Overload: la componente di **B** diverge ($\rho_B \geq 1$); sistema instabile.

6.12 Verifica e Validazione

6.12.1 Verifica. La verifica assicura che il modello computazionale implementi correttamente le specifiche e produca indicatori coerenti con i vincoli analitici. Usiamo quattro controlli principali: (i) *bound di capacità* $X_{\max} = 1/\max(D_A, D_B, D_P)$ e individuazione del collo di bottiglia; (ii) *coerenza dei trend* di $E[T_s]$, $E[N_s]$, ρ_k e X al variare di λ ; (iii) *leggi analitiche* (Little e identità $\rho_k \simeq \lambda D_k$ in regime stabile).

Confronto analitico–simulazione per Obiettivi 1–2. In baseline (Obj1) *B* risulta collo di bottiglia ($D_B=0.8$ s, $X_{\max} \simeq 1.25$ job/s): i grafici di sistema e per nodo mostrano allineamento stretto tra analitico e simulazione su tempi di risposta, popolazione, utilizzazione e throughput (Fig. 4–7, Tab. 2). Con 2FA (Obj2) aumentano D_A (terza visita) e D_P , con traslazione verso l’alto di $E[T_s]$ ed $E[N_s]$ a parità di λ ; *B* resta limitante (Fig. 8–11, Tab. 3).

Overload (Obj3). Per $\lambda \approx 1.4$ il nodo B va in saturazione ($\rho_B \rightarrow 1$): $E[T_s]$ ed $E[N_s]$ divergono mentre X si appiattisce al plateau $\simeq X_{\max}$ (Fig. 12–15, Tab. 4). La decomposizione per nodo conferma che il contributo di B domina e diverge in overload (Fig. 41 e 45).

Routing condizionale (Obj4). Con uscita anticipata dopo B (probabilità p) calano linearmente D_A e D_P e quindi ρ_A, ρ_P , con riduzione di $E[T_s]$ e $E[N_s]$ (Fig. 18–20; sweep in Tab. 5). Con probabilità di *loop* (ritorno $A_2 \leftrightarrow B$) crescono le visite attese, le domande D_k e le utilizzazioni fino alla saturazione, con rapida crescita di $E[T_s]$ (Fig. 21–24; sweep in Tab. 6).

Scaling verticale di B (Obj5). Riducendo $E[S_B]$ (es. 0.8→0.4 s) si abbassa D_B , si riduce ρ_B a parità di λ e si sposta a destra $\lambda^* = X_{\max}$: il plateau del throughput arriva più tardi (Fig. 28 e 29).

Transienti e warm-up. Gli esperimenti confermano che il *cold start* sottostima inizialmente $E[T_s]$; una popolazione iniziale più aggressiva (100 arrivi iniziali) caratterizzata da uno spike iniziale maggiore ma che con il tempo si stabilizza (Fig. 30–40). In overload non emerge plateau, a conferma dell’instabilità (Fig. 41 e 42).

6.12.2 Validazione. La validazione attesta che il modello rappresenti correttamente il sistema di riferimento (*did we build the right model?*). A parità di orizzonte ridotto (meno eventi), le curve simulate mantengono forma, pendenze e ordini di grandezza coerenti con il riferimento riportato nel documento: popolazione e tempi di risposta mostrano andamenti qualitativi sovrapponibili per Obj1 e Obj2 (con traslazione verso l’alto in presenza di 2FA) nelle figure di confronto (Fig. 46 e 47). Nel complesso: (i) in baseline B è il collo di bottiglia e determina X_{\max} ; (ii) il 2FA aumenta D_A (A_3) e D_P e quindi $E[T_s], E[N_s]$; (iii) in overload $E[T_s], E[N_s] \rightarrow \infty$ mentre X si satura; (iv) con *exit* ($p \uparrow$) le prestazioni migliorano, con *loop* peggiorano; (v) lo scaling di B ritarda la saturazione e rialza la capacità.

Confronto con i risultati del libro. Possiamo osservare i risultati ottenuti con la simulazione con una simulazione più corta, con 1_000_000 di eventi che i grafici della popolazione media e dei tempi di risposta medi sono simili a quelli riportati nel libro per quanto riguardano gli obiettivi 1 e 2.

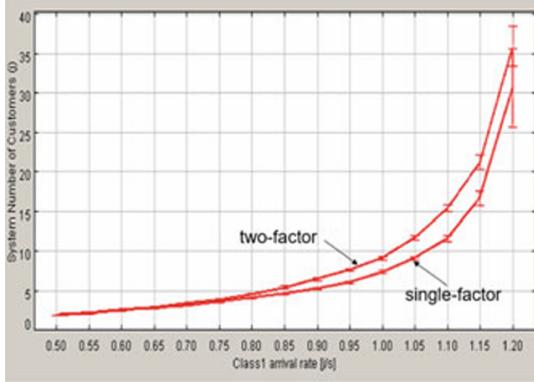


Fig. 46. Popolazione del sistema Serrazzi

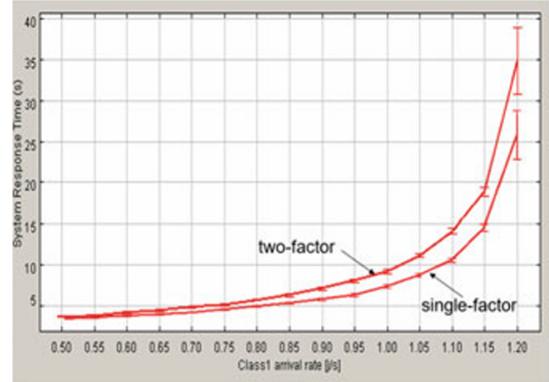


Fig. 47. Tempi di risposta del sistema Serrazzi

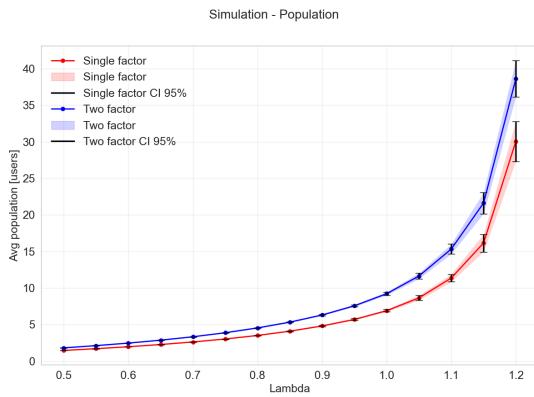


Fig. 48. Popolazione del sistema simulato

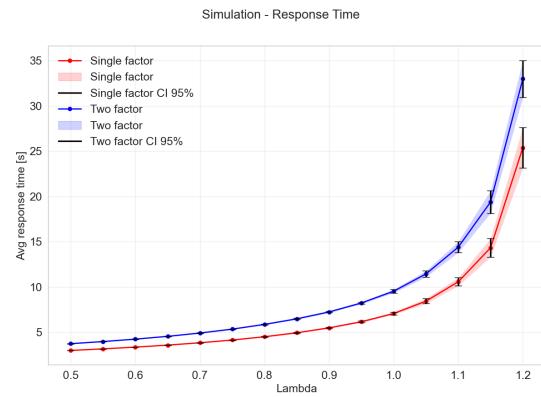


Fig. 49. Tempi di risposta del sistema simulato

7 CONCLUSIONI

Avendo fatto questa analisi What-If con questi parametri, abbiamo notato che il collo di bottiglia del sistema è il nodo B al crescere di λ in tutte e due le configurazioni, baseline e 2FA. Per quanto riguarda la simulazione degli altri obiettivi in funzione del comportamento degli utenti si aprono diversi scenari. Nello scenario in cui un utente non compra dallo shop ma inserisce elementi nel carrello senza fare acquisti si ha un miglioramento del sistema; mentre se un utente sceglie di cambiare la configurazione più volte in uno scenario di alto carico si ha un peggioramento delle performance. Per lo studio del bottleneck di B invece non sempre risulta migliore raddoppiare la potenza ma bastano configurazioni meno performanti per evitare colli di bottiglia nel sistema. Per analisi future si potrebbero usare le tracce di una WebApp per vedere come si comporta il sistema con arrivi reali, oppure si potrebbe analizzare il sistema usando distribuzioni per gli arrivi con varianza più alta rispetto agli arrivi di Poisson. Inserire un autoscaler che gestisce le risorse in funzione del carico.