



TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

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

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 → B → A → P → A.

Authors' Contact Information: 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.

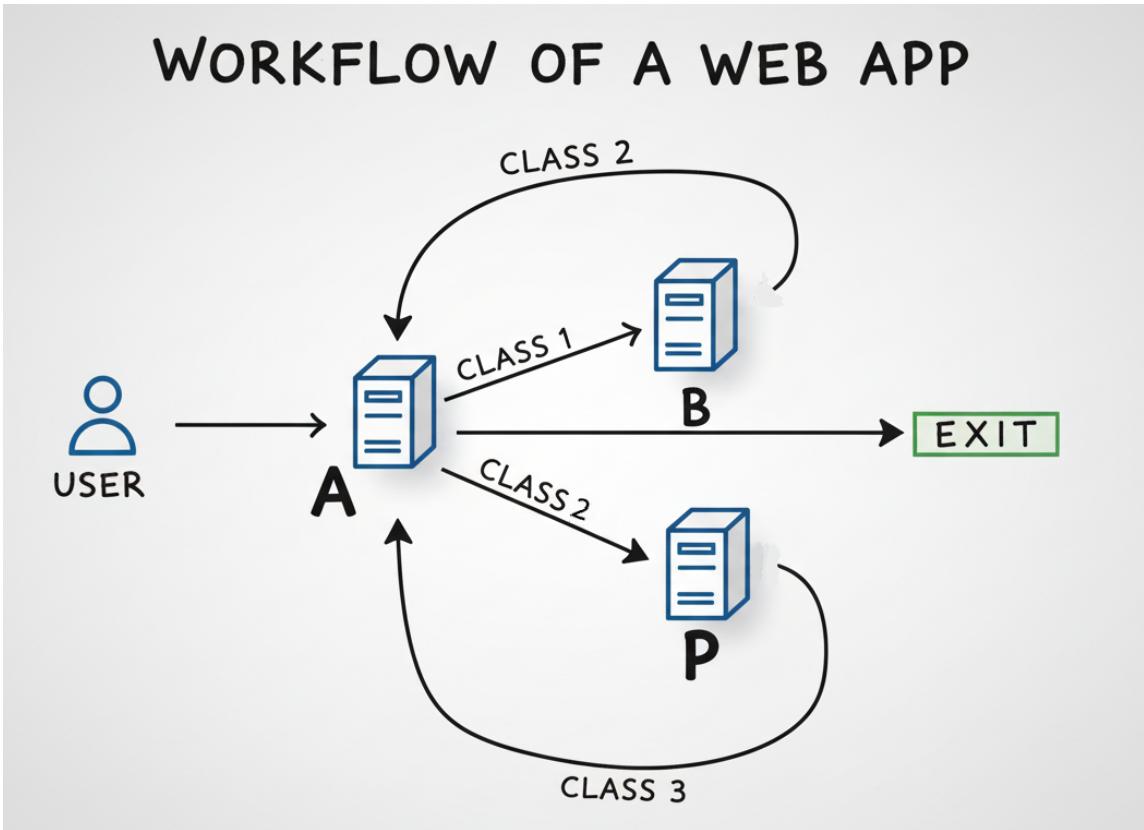


Fig. 1. Introduzione del sistema

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.

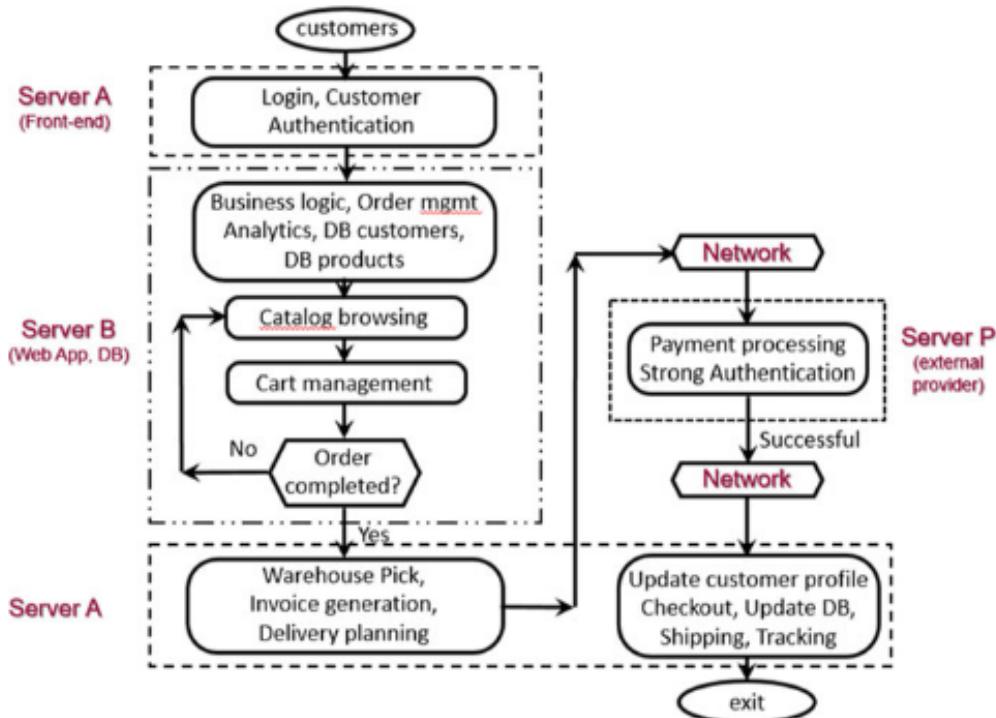


Fig. 2. Per la sezione Performance Analysis of a Web App Workflow (cap. 6.3, pag. 120 del documento)

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 – Baseline

Analisi delle metriche di sistema e dei nodi fissando un tasso di arrivo $\lambda = 4\ 320 \text{ job/h}$ (1,2 job/s), tale che il sistema raggiunga il limite di saturazione.

(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).

(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

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.

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).

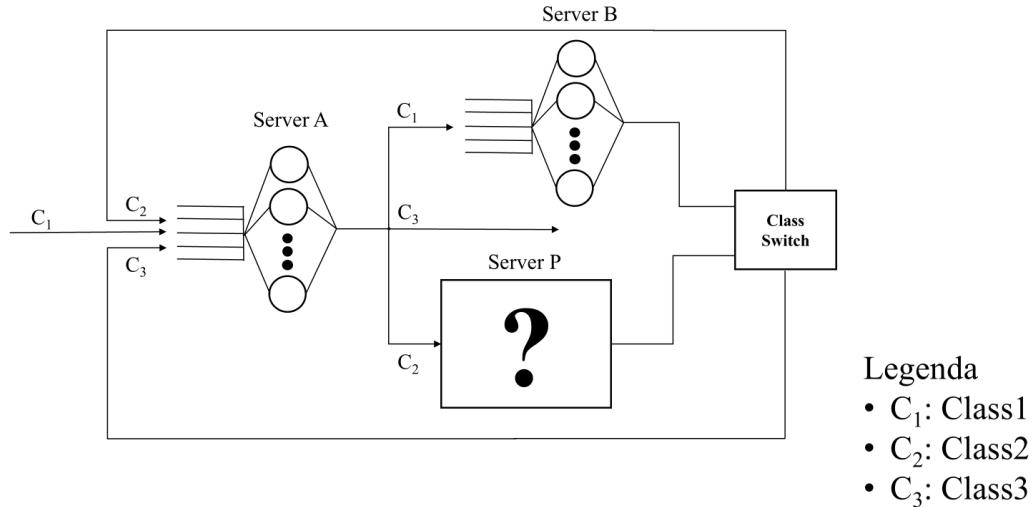


Fig. 3. Modello concettuale iniziale

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 (5040 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:

- **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 *Pseudo-random number generator.* Adottiamo un PRNG di tipo *Lehmer* a 32-bit con parametri $(a, m) = (48271, 2147483647)$ fissati. La scelta dei seed per le varie repliche durante la simulazione avviene nel seguente metodo:

Protocollo operativo:

- Nel file config della simulazione, fissiamo il **seed iniziale** S_1 ed il **numero di repliche** R .
- All'avvio, eseguiamo `plantseed(S_1)` e lanciamo la simulazione.
- **Orizzonte infinito:** poniamo $R = 1$ ed eseguiamo un'unica run lunga con *Batch Means* quindi serve solo il seed iniziale.

- **Orizzonte finito:** poniamo $R > 1$. Per ogni replica $r = 1, \dots, R$ il simulatore consuma esattamente l'orizzonte previsto in eventi. A fine replica il programma:
 - acquisisce lo *stato* del PRNG con `getseed()` ottenendo S_{r+1} ;

Questo schema crea **substream contigui e non sovrapposti** (indipendenti) della stessa sequenza pseudocasuale: ogni replica usa un segmento distinto e immediatamente successivo del flusso, garantendo indipendenza. Risulta quindi:

- **Robusto** a cambiamenti del modello/scheduler poiché l'ampiezza del segmento è viene impostata con l'inizio che coincide con la fine del segmento precedente.
- **Riproducibile:** registrando S_1 e la lista $\{S_r\}_{r=1}^R$ è possibile rigenerare qualsiasi replica eseguendo `plantseed` sul seed corrispondente impostando lo stesso numero di generazioni.

Disciplina di consumo e multistream. Utilizziamo un **multistream**: durante l'inizializzazione della replica fissiamo lo stato base del PRNG con il seed e, per ciascuna sorgente di casualità, assegniamo un substream dedicato mediante `getstream()`. In particolare:

- uno stream quando viene creato l'*arrival generator* per la generazione esponenziale degli arrivi;
- tre stream aggiuntivi quando vengono inizializzati i nodi A , B e P per la generazione esponenziale dei rispettivi tempi di servizio.

Ogni stream è usato esclusivamente dal proprio componente, mentre l'ordine effettivo di consumo resta determinato dal *nextevent scheduler*. La separazione degli stream garantisce indipendenza intrareplica tra le diverse sorgenti; combinata con la delimitazione tra repliche (`getseed/plantseed`), assicura segmenti disgiunti anche interreplica. Nel caso di routing probabilistico viene assegnato uno stream ulteriore per le decisioni di routing.

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 (PriorityQueue<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.

- **Type (enum):** rappresenta il tipo di evento; può assumere i valori `ARRIVAL`, `DEPARTURE`
- **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 (`arrival`, `departure`, o `measure_start`).
- **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 (immutable).
- **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).

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).

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 Node viene implementata la disciplina PS: ad ogni arrivo, viene calcolato l'intervallo di tempo rispetto a quello precedente per poi essere suddiviso rispetto alla lunghezza dei job presenti nel nodo e successivamente ricalcolati i tempi di servizio rimanenti per job.

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).

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 Verifica e Validazione

5.1 Verifica

La verifica risponde alla domanda did we build the model right?

Nel modello di simulazione il server A gestisce tre classi di job, ciascuna con un proprio tasso di servizio (μ_1, μ_2, μ_3). Per il corrispondente modello analitico adottiamo invece una rappresentazione monoclasse, sostituendo i tre tassi con un tasso equivalente dato dalla media pesata dei tassi di classe (nel nostro caso, pesi uniformi), così da preservare la domanda media di servizio del centro. In simulazione i job mantengono il loro tasso “di classe”, mentre in analitico operiamo con il tasso medio equivalente; una prima verifica mostra che le stime di tempo di risposta (e le altre metriche) risultano allineate a quelle analitiche entro l’incertezza statistica, confermando la correttezza dell’approssimazione adottata.

5.1.1 Next-Event scheduler. Il simulatore adotta un next-event scheduler con orologio di simulazione discreto: il clock avanza saltando direttamente all’istante del prossimo evento nella Future Event List (FEL), mantenuta come coda di priorità ordinata per timestamp. Alla generazione di un arrivo sul centro con disciplina Processor Sharing (PS), il job entra immediatamente nel set di servizio; se il server è già occupato, la capacità del server viene ripartita tra i n job presenti. L’inizializzazione avviene a $t = 0$, la simulazione termina al raggiungimento di un contatore di completamenti, e la verifica viene condotta su repliche indipendenti per stimare intervalli di confidenza.

5.1.2 Obj1 - baseline. **Obiettivo:** verificare la correttezza implementativa del workflow A–B–A–P–A con class-switch e disciplina PS in regime stabile. **Procedura:** confronto tra modello analitico e simulazione su *tempo di risposta, popolazione, throughput e utilizzazione*; **Esito:** i risultati in Tabella sono coerenti tra analitico e simulazione, indicando che il *next-event scheduler*, il routing con *class-switch* e l’implementazione *PS* operano correttamente nella configurazione baseline.

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio (s)	25.14423	25.26909	± 0.846
Popolazione media (job)	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

5.1.3 Obj2-2fa. Per obj2-fa c’è un aumento dei tempi ma in generale mantiene l’andamento di obj1

Table 3. Confronto analitico vs simulazione (2FA)

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio (s)	31.87500	32.89309	± 1.197
Popolazione media (job)	38.25000	38.57914	± 1.450
Throughput (job/s)	1.20	1.19988	± 0.001
Utilizzazione	0.99936	0.99937	± 0.000

Confronto con i risultati del libro. I risultati ottenuti dalla simulazione mostrano andamenti della popolazione media e dei tempi di risposta medi simili a quelli riportati nei grafici di Serrazzi per gli obiettivi 1 e 2.

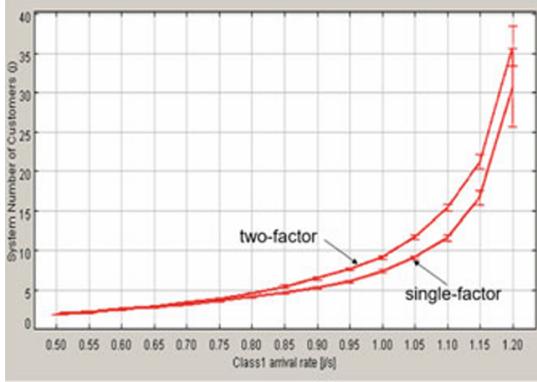


Fig. 4. Popolazione del sistema Serrazzi

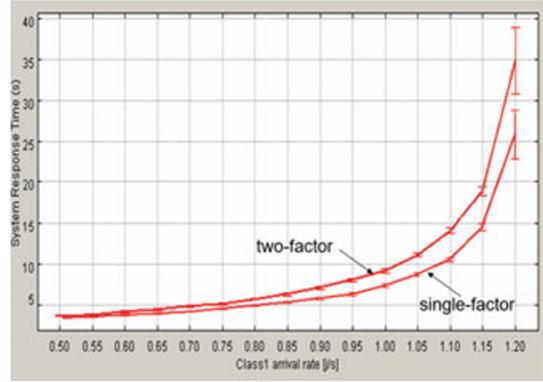


Fig. 5. Tempi di risposta del sistema Serrazzi

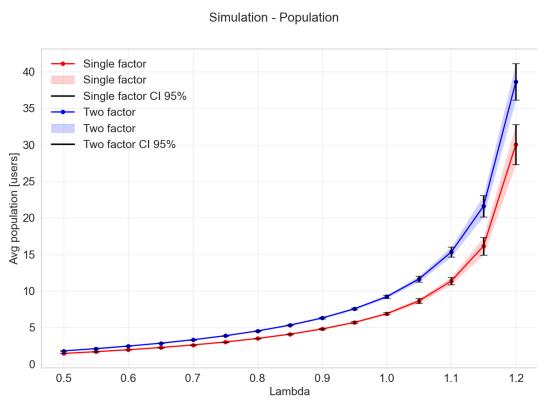


Fig. 6. Popolazione del sistema simulato

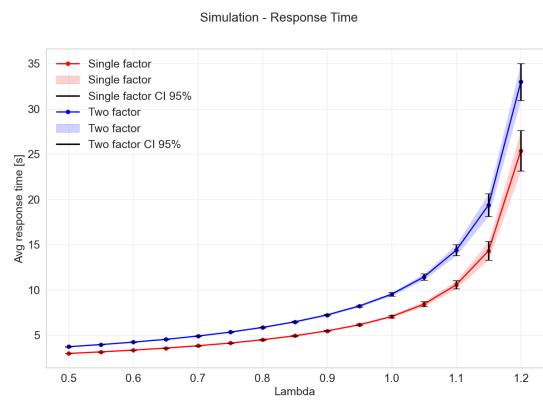


Fig. 7. Tempi di risposta del sistema simulato

5.1.4 Obj3 – overload. In sovraccarico, il modello analitico prevede divergenza di *tempo di risposta* e *popolazione*, saturazione del *throughput* a X_{\max} e *utilizzazione* unitaria sul centro critico. La simulazione riproduce questo comportamento: il throughput si assesta a $\approx X_{\max}$, l'utilizzazione tende a 1, mentre le stime di tempo di risposta e popolazione assumono valori molto elevati, coerenti con un sistema instabile. L'allineamento qualitativo e quantitativo in regime non stazionario conferma che il simulatore gestisce correttamente sia la dinamica PS sia gli stati di instabilità del sistema.

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio (s)	∞	4602.36674	± 319.817
Popolazione media (job)	∞	5748.59517	± 392.938
Throughput (job/s)	1.25	1.24920	± 0.003
Utilizzazione	1.00	0.99998	± 0.000

Table 4. Confronto analitico vs simulazione (overload)

5.1.5 *Obj4 – early exit.* Seguendo i calcoli analitici del routing con *uscita anticipata* dopo B (probabilità p), all'aumentare di p si alleggerisce il carico sul sistema, quindi ci si aspetta una riduzione dei *tempi di risposta* e delle *popolazioni* su A e P , throughput di sistema allineato a $\lambda = 1.2j/s$ in regime stabile e utilizzazioni coerentemente più basse su quei nodi, mentre su B non avendo variazioni sulle visite non ci aspettano cambiamenti. La simulazione riproduce questo comportamento infatti le *tabelle dei tempi* al variare di p mostrano una **decrescita monotona** dei tempi medi sia di sistema; gli intervalli di confidenza (CI 95%) includono i valori attesi dai calcoli analitici per tutti i p testati facendo rimanere, come previsto, il sistema stabile alla sua variazione.

p	Analitico [s]	Simulazione [s]	CI_95
0.1	23.58835	22.59788	± 1.155
0.2	22.66234	21.87211	± 0.793
0.3	22.03933	21.26186	± 0.879
0.4	21.58708	20.83848	± 0.802
0.5	21.24142	20.51865	± 0.806
0.6	20.96725	20.27093	± 0.917
0.7	20.74364	20.08483	± 1.001
0.8	20.55725	19.99813	± 0.758
0.9	20.39916	19.87066	± 0.599
1.0	20.26316	19.70292	± 0.630

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

5.1.6 *Obj4 – loop.* Sulla base dei calcoli analitici del routing con *rientro* $B \rightarrow A_2 \rightarrow B$ (probabilità p), all'aumentare di p crescono le visite su A_2 e su B , con incremento di *tempi* e *popolazioni* e progressivo avvicinamento alla saturazione del nodo critico; già per valori bassi è atteso il passaggio all'*overload* poiché già con $p = 0$ abbiamo una ρ_b già molto vicino all'1. La simulazione conferma questa dinamica; le *tabelle dei tempi* al variare di p evidenziano una **crescita monotona e veloce** dei tempi medi; nelle configurazioni ancora stabili ed ancora di più elevata quando analiticamente il sistema è instabile. In ogni caso almeno per le parti stabili i valori analitici sono compresi nel CI 95%. si conclude quindi che la simulazione è verificata anche per questo obiettivo.

p	Analitico [s]	Simulazione [s]	CI_95
0.0	25.14423	25.26909	± 0.846
0.01	31.97366	31.61378	± 3.194
0.02	45.48390	42.34325	± 6.509
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)

5.1.7 *Obj5 – Scaling verticale.* La verifica dell’obiettivo 5 si concentra sul comportamento del sistema sotto la scalabilità verticale. In particolare, l’obiettivo è garantire che, aumentando la potenza e quindi riducendo $E_B[S]$, il sistema continui a comportarsi correttamente, mantenendo le metriche in utilizzo in accordo con le aspettative teoriche.

Per la verifica, abbiamo confrontato i risultati ottenuti dalla simulazione con quelli derivati dal modello analitico. In particolare dall’analitico osserviamo che mantenendo $E_B[S] = 0,8s$ si riottiene il caso dell’obiettivo già validato in precedenza mentre aumentando la potenza e quindi riducendo $E_B[S]$ otteniamo, oltre una certa soglia, un sistema stabile. I calcoli mettono in maniera particolare in evidenza due casi:

- $E_B[S] = 0,65s$ nel quale osserviamo un sistema stabile ma poco sotto la soglia critica;
- $E_B[S] = 0,4s$ nel quale nonostante la riduzione drastica il sistema è si stabile ma, ancora vicino alla soglia critica di utilizzazione (migrazione del collo di bottiglia su A).

In conclusione, la verifica ha confermato che il sistema risponde correttamente al scaling verticale, mantenendo il tempo di risposta e l’utilizzazione all’interno della stabilità.

Table 7. Confronto analitico vs simulazione ($B=0.65s$)

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio (s)	43.13131	44.61245	± 3.039
Popolazione media (job)	60.38384	59.00502	± 3.920
Throughput (job/s)	1.40000	1.39903	± 0.001
Utilizzazione	0.99921	0.99920	± 0.000

Table 8. Confronto analitico vs simulazione ($B=0.4s$)

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio (s)	36.81818	38.11817	± 2.533
Popolazione media (job)	51.54545	49.95497	± 3.250
Throughput (job/s)	1.40000	1.39904	± 0.001
Utilizzazione	0.99613	0.99610	± 0.000

5.2 Perchè simulare?

L'elevata aderenza tra analitico e simulazione è attesa: abbiamo allineato le ipotesi (arrivi/servizi esponenziali, PS, centro A reso classless con media pesata) e quindi gli indici coincidono entro l'incertezza statistica. La simulazione è comunque necessaria per: (i) analizzare transitori e overload (assenza di soluzione chiusa), (ii) valutare routing probabilistico (early-exit/loop) e mutamenti del collo di bottiglia, (iii) studiare strategie di scaling e (iv) produrre CI operativi su metriche time-weighted.

5.3 Validazione

La validazione attesta che il modello rappresenti correttamente il sistema di riferimento (did we build the right model?).

NOTA BENE Non disponendo di dati reali, **non è stato possibile eseguire una validazione diretta**. Abbiamo quindi adottato un approccio basato su ipotesi analitiche e scenari noti, confrontando tali aspettative con l'andamento delle simulazioni.

5.3.1 *obj1 (baseline) e obj2 (2fa*). In particolare per obj1 e 2, ci si aspetta che al crescere del fattore di utilizzazione ρ i tempi di risposta aumentino sensibilmente: quando ρ tende a 1 ci si attende un incremento marcato dei tempi di risposta rispetto a condizioni di carico più basso, pur senza osservare un'esplosione incontrollata dei valori, dato che il sistema rimane comunque in regime stabile entro i limiti di capacità simulati. Inoltre l'introduzione di 2fa comporta overhead maggiore, quindi ci aspettiamo tempi di processamento più lenti.

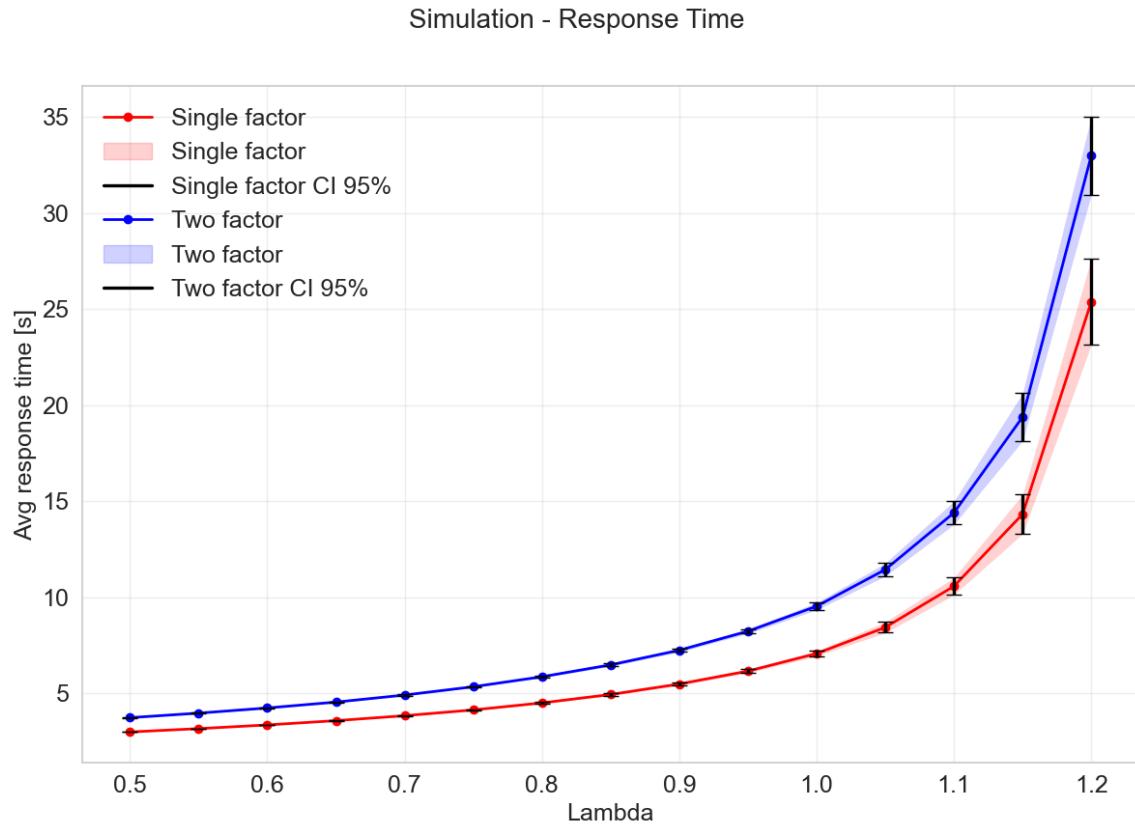


Fig. 8. Tempi di risposta di baseline & 2fa

5.3.2 *Obj3 - overload*. Per la validazione dell'**Obj3** poichè ci troviamo in assenza di dati da confrontare direttamente possiamo solo fare delle ipotesi guidate dall'analitico per poi confrontarle con l'andamento delle simulazioni. Poiché il collo di bottiglia ha domanda di servizio $D_B = 0.8$ s, la capacità massima è $X_{\max} = 1/D_B = 1.25$ job/s. Con tasso d'arrivo $\lambda = 1.4$ job/s si ha $\rho_B = \lambda D_B = 1.12 > 1$, per cui il sistema non ammette regime stazionario e, in particolare, le metriche

di interesse arrivano ad essere $E[T] = +\infty$ ed $E[N] = +\infty$. Quindi ci attendiamo: (i) *throughput* saturato a X_{\max} , (ii) utilizzazione del nodo critico prossima a 1 ,e con essa quella del sistema, (iii) traiettorie di tempo di risposta $T_s(t)$ e popolazione $N(t)$ monotone crescenti senza plateau.

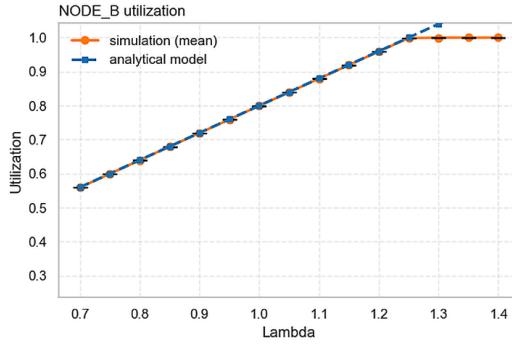


Fig. 9. utilizzazione del nodo in saturazione

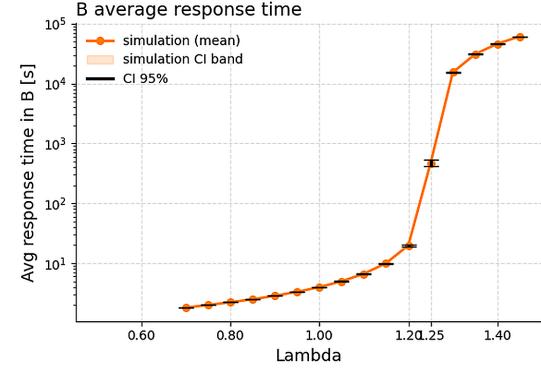


Fig. 10. Tempi di risposta del nodo simulato

5.3.3 Obj4 - early exit. Adesso andiamo a validare che l'implementazione del ramo di *uscita anticipata* dopo il nodo B sia coerente con il riferimento analitico, aumentando la probabilità di uscita immediata dopo B, alleggerisce il carico complessivo; infatti ci aspettiamo che diminuendo le visite attese sui nodi seguenti diminuiscano tempi e popolazioni su A, P, mentre le metriche di B rimangono invariate. Le simulazioni mostrano che al crescere della probabilità di early exit si ottiene una discesa monotona dei tempi medi e delle popolazioni sui nodi A e P. Di conseguenza otteniamo un comportamento simulato allineato con le nostre ipotesi guidate dall'analitico.

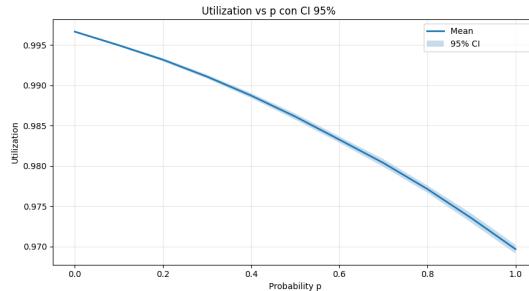


Fig. 11. andamento utilizzazion con early exit in aumento

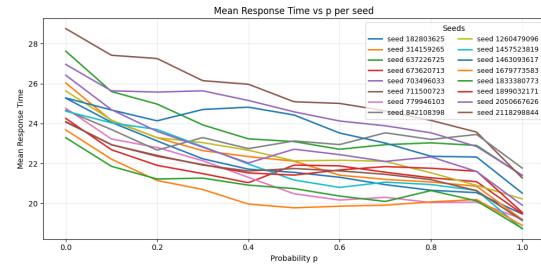


Fig. 12. Tempi di risposta del sistema simulato si abbassano con early exit in aumento

5.3.4 Obj4 - loop. Questa seconda versione di routing probabilistico implementa il loop tra A₂ e B. Seguendo l'analitico, aumentare la probabilità di loop fa aumentare le visite sui nodi A e B, quindi cresce la domanda sul collo di bottiglia (B) e si osserva un progressivo degrado sia dei tempi che delle popolazioni osservando un aumento, fino alla perdita di stabilità del sistema. Le simulazioni confermano l'aumento di tempi e popolazioni e mostrano utilizzazioni > 1 sul nodo critico riportando il sistema al caso dell'obiettivo 3 ovvero in saturazione con assenza di plateau, in linea con le previsioni analitiche. Quindi notiamo che a probabilità di loop basse sia l'analitico che la simulazione mostrano

metriche più elevate ma ancora sistemi stabili e all'aumentare di essa mandando in saturazione il collo di bottiglia il sistema esplode.

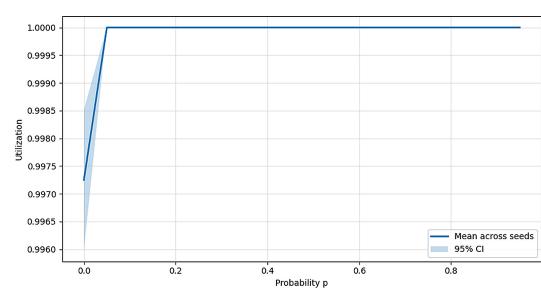


Fig. 13. utilizzazione in saturazione elevata

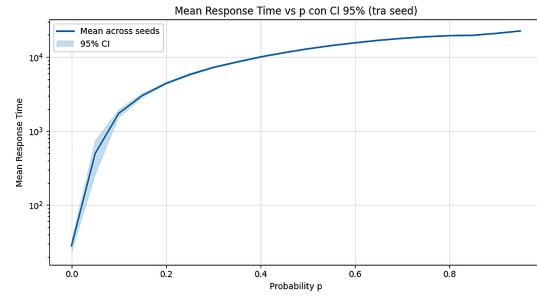


Fig. 14. Tempi di risposta del sistema simulato esplodono con la saturazione

5.3.5 Obj5 - scaling verticale. A carico fisso $\lambda = 1.4$ il nodo B torna stabile quando la sua capacità supera il carico, ossia per $E[S_B] \leq 1/\lambda \approx 0.714$ s. Pertanto, variando $E[S_B]$ da 0.8 s a 0.4 s ci attendiamo: (i) per $E[S_B] > 0.714$ s un regime di *overload* su B ($\rho_B \approx 1$), con throughput in *plateau* e crescite elevate di tempi e popolazioni; (ii) al di sotto della soglia, il *ritorno alla stazionarietà* ($\rho_B < 1$), con throughput sostenibile più elevato e riduzione dei tempi medi; (iii) all'ulteriore riduzione di $E[S_B]$, una discesa monotona di ρ_B e dei tempi, con possibile *spostamento del collo di bottiglia* da B verso un altro nodo. La validazione consiste nel riscontrare in simulazione questa soglia di stabilità e le tendenze qualitative descritte.

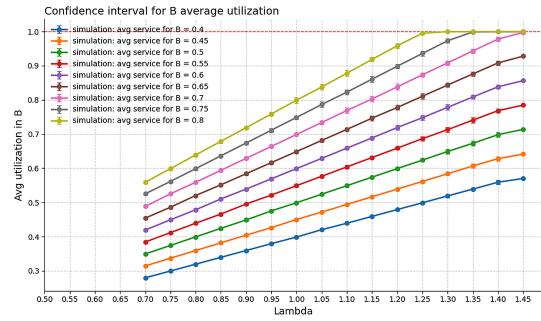


Fig. 15. utilizzazione in discesa con l'aumento della potenza del nodo

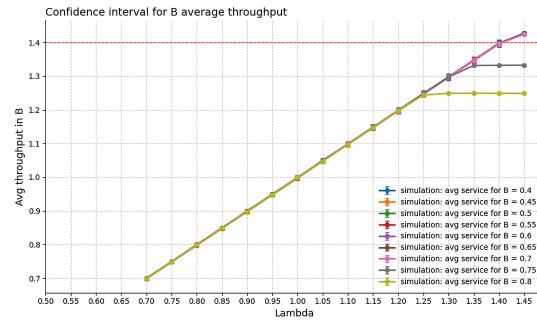


Fig. 16. Throughput in aumento fino a $\lambda = 1.4$

6 Transitori

6.1 Transitori (Obj1–Obj2–Obj3)

In questa sezione vogliamo osservare se il sistema, date le configurazioni ricavate dal libro, converge.

Nel transitorio, il sistema parte vuoto perché ci interessa analizzare il suo comportamento nelle sue prime fasi di simulazione fino ad un eventuale raggiungimento di convergenza.

Osservazione di come nei casi 1 (analogamente anche il 2), il sistema converga. Mentre nel caso 3 notiamo un aumento massivo dei tempi, indice che il sistema non converge.

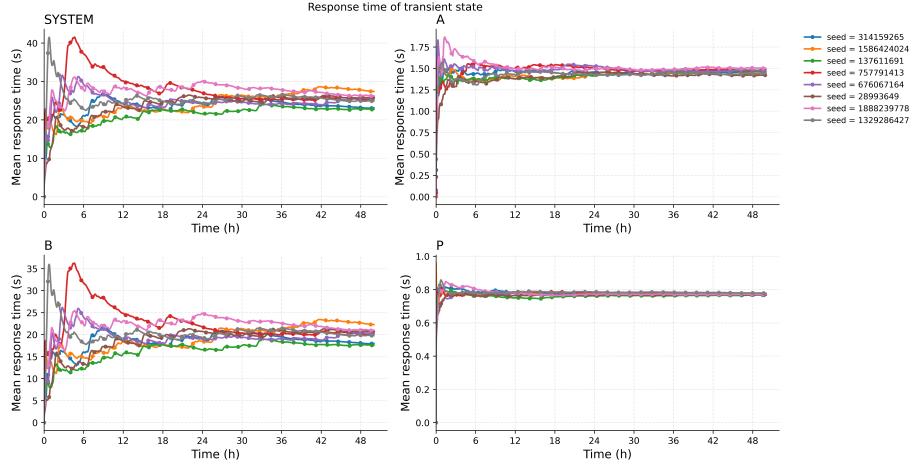


Fig. 17. Transitorio configurazione OBJ-1

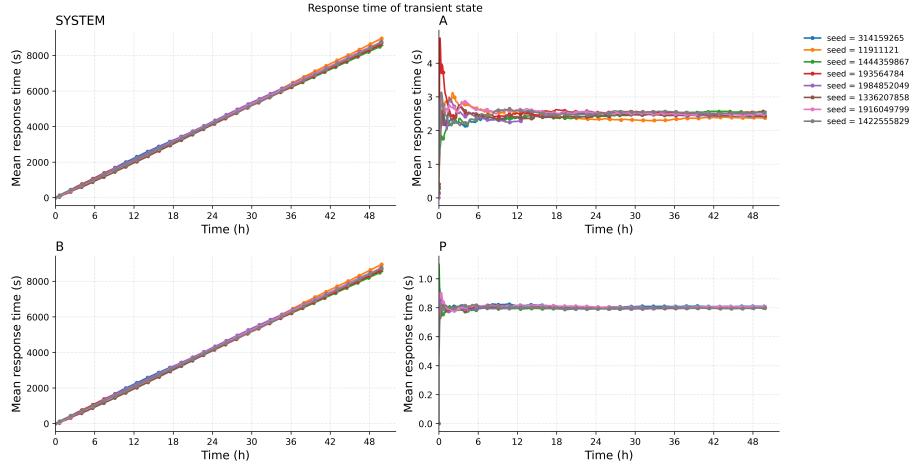


Fig. 18. Transitorio configurazione OBJ-3 non ammette steady-state

6.2 Transitori (Obj4 early-exit)

In Figura 19 mostriamo $E(T_S)$, per $p = 0.9$. L'aumento di p riduce il carico effettivo sulla rete e, come atteso:

- le traiettorie di $E(T_S)$ raggiungono un *plateau* a livelli più bassi;
- il tempo di assestamento si accorcia: la curva si stabilizza sensibilmente prima di $p = 0.00$;

In sintesi, l'uscita anticipata “scarica” i nodi a valle e produce transitori più brevi e più regolari.

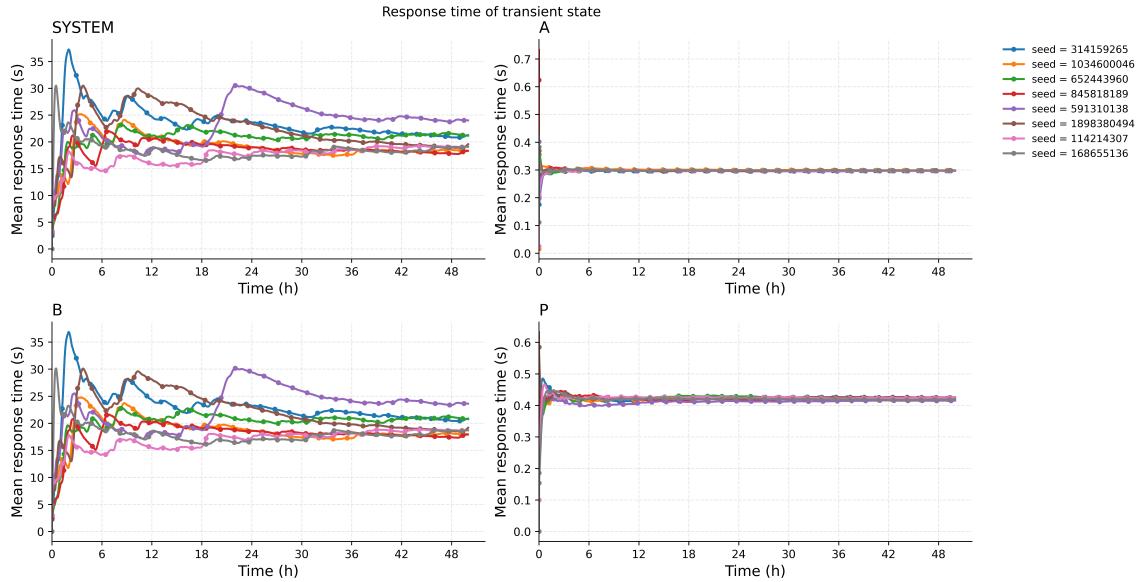


Fig. 19. Obj4 — Exit con probabilità di uscita 0.9.

6.3 Transitori (Obj4 loop)

In Figura 20–22 consideriamo $p \in \{0.02, 0.04, 0.20\}$. In questo scenario le visite attese a B crescono a $V_B = 1/(1-p)$, con utilizzazione $\rho_B \approx \lambda D_B V_B$. Con $D_B = 0.8$ s e $\lambda = 1.2$ job/s si ottiene una soglia

$$p^* \approx 1 - \lambda D_B \approx 1 - 0.96 \approx 0.04,$$

oltre la quale il sistema non ammette steady-state.

I grafici mostrano il comportamento atteso:

- $p = 0.02$ (**quasi-critico ma stabile**): si osserva un transitorio più lungo e una maggiore variabilità prima del plateau;
- $p = 0.04$: si osserva un transitorio instabile ma con una crescita tempi lenta.
- $p = 0.20$ (**instabile, $p > p^*$**): $\bar{T}(t)$ cresce senza evidenziare un plateau.

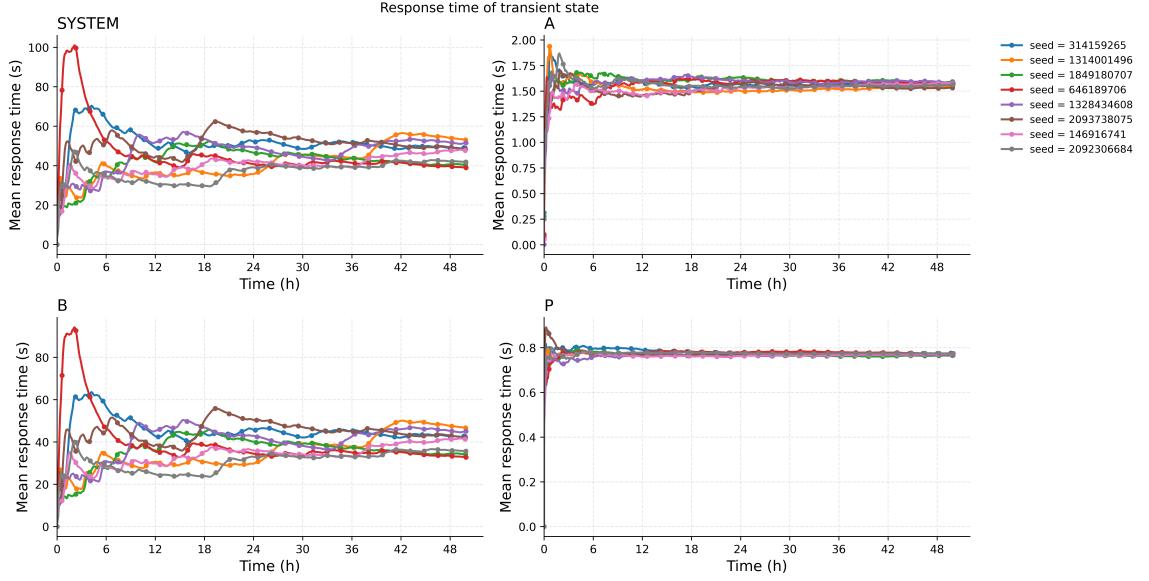


Fig. 20. Obj4/transient – RT per seed (SYSTEM, A, B, P): stabile con transitorio più lungo comportamento convergente.

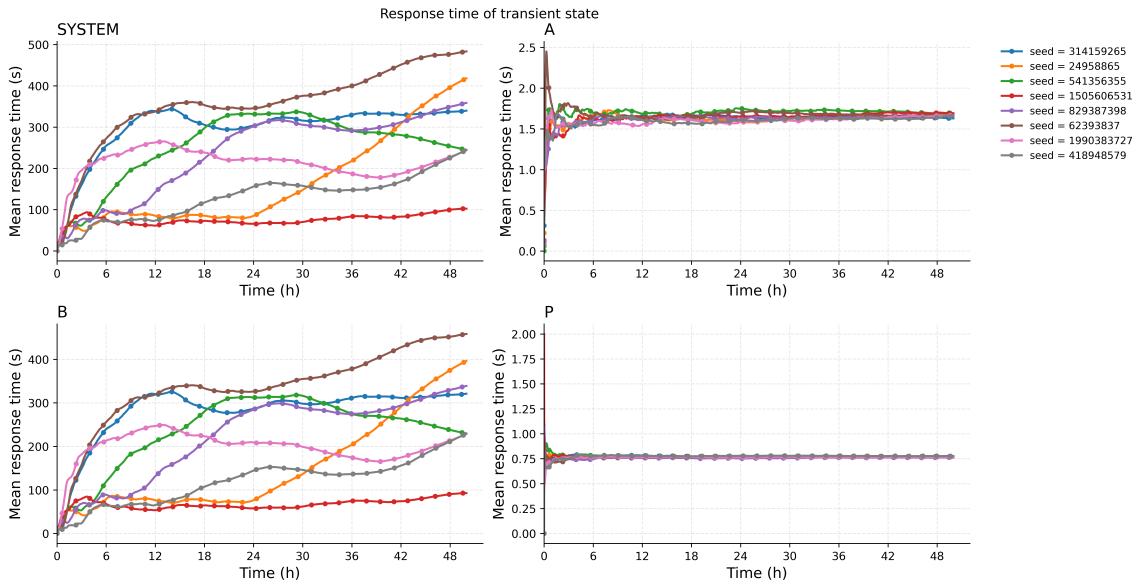


Fig. 21. Obj4/transient – RT per seed: nessuna stabilizzazione (overload) per via dei loop ma lenta.

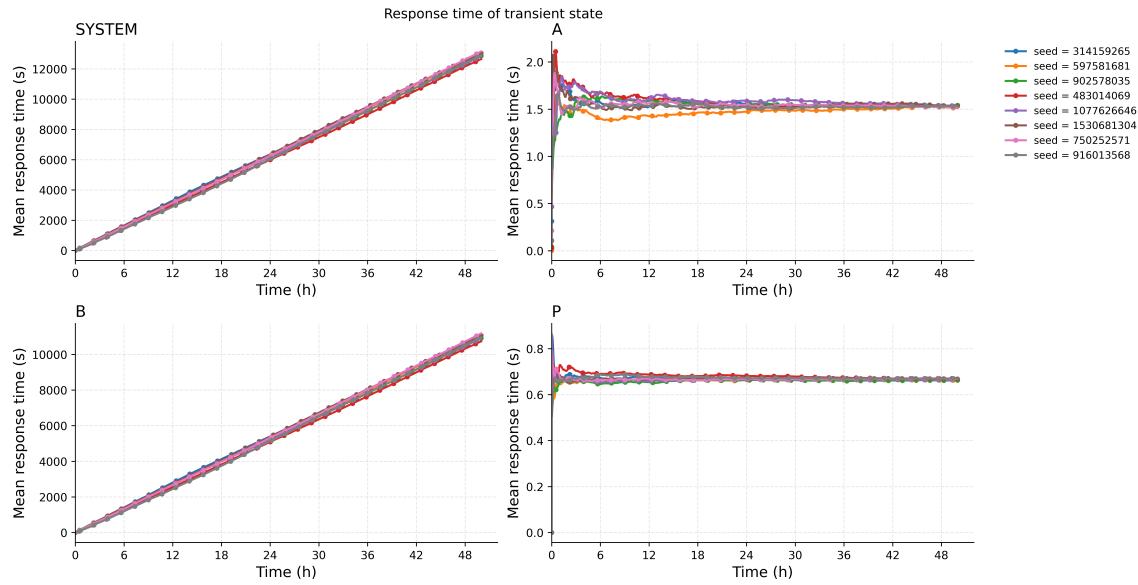


Fig. 22. Obj4/transient — RT per seed: nessuna stabilizzazione (overload) per via dei loop.

6.4 Transitori (Obj-5 scaling verticale)

Con $\lambda = 1.4$ job/s, in Obj3 (Figura 18) il sistema *non* ammette steady-state: il collo di bottiglia è il nodo **B**, per il quale l'utilizzazione tende a $\rho_B \rightarrow 1$ e la traiettoria $E(t_S)$ mostra una crescita persistente (assenza di plateau). In Obj5 interveniamo sul *tempo medio di servizio* di **B**, evidenziando la soglia critica

$$D_B^* = \frac{1}{\lambda} \approx \frac{1}{1.4} \approx 0.714 \text{ s.}$$

Per $D_B > D_B^*$ (ad es. 0.80–0.72 s) si replica il comportamento di Obj3: $\bar{T}(t)$ continua a crescere, $X(t)$ si satura al limite $1/D_B$ e $\rho_B(t) \approx 1$. Viceversa, **riducendo D_B sotto soglia** (0.70–0.60 s) il sistema **recupera la stabilità**: la curva $E(t_S)$ raggiunge un *plateau* visibile, $X(t) \approx \lambda$ e $\rho_B \approx \lambda D_B < 1$. E infine **riducendo D_B** fino a 0.4 (Figura 23) osserviamo un comportamento del sistema che arriva al plateau più velocemente e al suo steady-state rivelando A come nuovo collo di bottiglia.

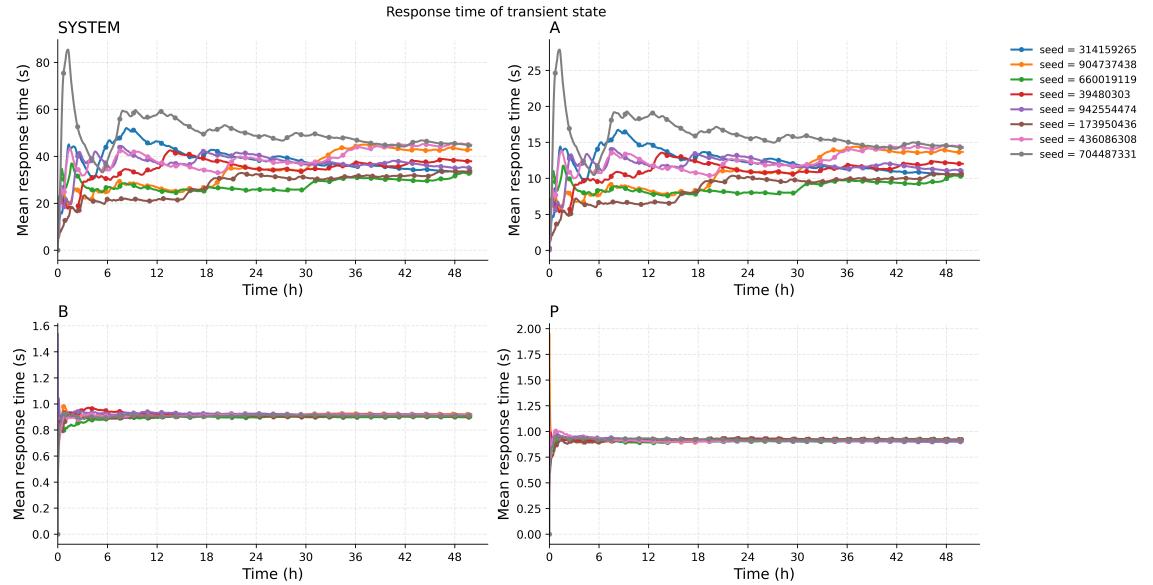


Figure 23. Obj5/transitorio – RT per seed: stabilizzazione possibile grazie a tempo di servizio medio pari a 0.4s.

7 Progettazione degli esperimenti

7.1 Scelta dell'orizzonte – motivazione sperimentale

Orizzonte infinito. Obiettivo: stimare il rendimento medio a lungo termine con carico stazionario (X , $E[T]$, $E[N]$, ρ).
. CI (tra batch). Eseguiamo una long-run con **Batch Means**: la run è suddivisa in k batch contigui di ampiezza b (confini sui completamenti). Le medie di batch Y_1, \dots, Y_k forniscono CI 95%:

$$\bar{Y} \pm t_{k-1, 0.975} \frac{s_Y}{\sqrt{k}},$$

dove s_Y è la deviazione standard campionaria (gradi di libertà $k-1$). La scelta di b è guidata dall'ACF (cutoff) e si sceglie k tale da ottenere IC stabili.

Orizzonte finito (terminating / finestra). Obiettivo: descrivere l'evoluzione su una finestra $[T_0, T_1]$ (profili $T(t)$, $N(t)$, $X(t)$, $\rho(t)$), inclusi l'avvio a freddo e la crescita delle popolazioni (utile anche in near-critical/overload). **CI (tra repliche).** Eseguiamo R repliche indipendenti (semi diversi) con lo stesso criterio terminale (T fissato oppure C completamenti) e stesse condizioni iniziali; CI 95% è:

$$\bar{Y} \pm t_{R-1, 0.975} \frac{s_Y}{\sqrt{R}}.$$

7.2 Metodi di campionamento

7.2.1 *Batch Means (orizzonte infinito).* Per stimare statistiche a orizzonte infinito usiamo i **Batch Means**: eseguiamo una long-run e la dividiamo in k batch contigui di ampiezza b (confini sui completamenti). Agli estremi di ogni batch azzeriamo gli stimatori; le *medie per batch* Y_1, \dots, Y_k (con $Y \in \{X, \bar{T}, \bar{N}, \rho\}$) sono trattate come quasi indipendenti e forniscono l'IC al 95% via t-Student.

Perché BM è adatto all'orizzonte infinito: l'aggregazione in batch riduce l'autocorrelazione tipica delle lunghe tracce, rendendo le medie di batch quasi i.i.d. e idonee alla costruzione di IC affidabili. La scelta di b e k deriva dall'ACF stimata con `acs.py`, seguendo due criteri: (i) $b \gtrsim 10 \times$ il lag massimo ancora significativo; (ii) autocorrelazione a lag 1 tra le *batch means* < 0.2. Nel nostro caso: **BATCH_SIZE = 5070**, **NUM_BATCHES = 256** (per obj1/2) **BATCH_SIZE = 19530**, **NUM_BATCHES = 512** (Per obj5). La media delle medie di batch coincide con la media globale; la coppia (b, k) incide soprattutto sulla *larghezza* delle CI. *Trade-off sintetico:* a lunghezza di run fissata, aumentare b riduce la correlazione ma diminuisce k , quindi le CI tendono ad allargarsi; diminuire b aumenta k ma introduce più dipendenza e può sottostimare la varianza. Se invece si allunga la run mantenendo (b, k) cresce e le CI si stringono senza peggiorare l'indipendenza.

7.2.2 Repliche indipendenti.

Scelta del numero di repliche (RI). Per le simulazione ad orizzonte finito sono state scelte 64 repliche. Questo numero garantisce una stima affidabile e robusta delle metriche e degli intervalli di confidenza calcolati. Secondo il teorema del limite centrale con 64 repliche ci si avvicina molto al comportamento asintotico della distribuzione normale.

7.3 Obiettivi 1–2 (regime stabile)

Avendo visto nel transitorio che il sistema converge, siamo interessati a simulare il sistema a regime per osservare i valori di convergenza e vedere le prestazioni del sistema quando ha raggiunto il suo stato steady-state. Vogliamo osservare l'evoluzione del sistema al variare del tasso degli arrivi esterni per poter confrontare le metriche tra le due configurazioni.

Orizzonte: infinito per $\lambda \in \{0.50, 0.55, \dots, 1.20\}$. **Protocollo: Batch Means** su long-run: $k = 256$ batch contigui, ampiezza $b = 5070$ completamenti (confini su completamenti di *sistema*). Arresto quando raccolti k batch completi ($\approx 1.30 \times 10^6$ completamenti) oppure a *safety cap* sugli eventi (`max_events=5 × 106`).

Output: $X, \mathbb{E}[T], \mathbb{E}[N], \rho$ con **CI 95% tra batch** (t-Student sulle medie di batch).

7.4 Obiettivo 3 (overload)

In overload non esiste un plateau stazionario: le code crescono senza bound. Ci interessa il *profilo entro una finestra operativa* e i *tempi al degrado*; una finestra $[0, 24h]$ contenente una spike di durata 1h e in una seconda run con finestra $[0, 48h]$ con spike di 6h rende confrontabili le configurazioni; questi spike si verificano entrambi dopo 6h di inizio simulazione.

Orizzonte: finito ($\lambda = 1.2 \rightarrow \lambda = 1.4$). **Protocollo:** repliche indipendenti sulla stessa finestra $T = 24$ h e $T = 48$ h.

7.5 Obiettivo 4 (routing probabilistico)

Vogliamo vedere a regime come si evolvono le curve, questo ci permetterà di capire il comportamento del sistema quando gli utenti adottano determinati comportamenti per tempi lunghi nel caso in cui gli utenti non fanno gli acquisti. Vogliamo inoltre visualizzare quando abbiamo stabilità le metriche a regime, quando abbiamo delle riconfigurazioni del carrello, e vedere in una finestra più breve la pendenza delle metriche quando il sistema diventa instabile con un valore di probabilità di riconfigurazione del carrello più grande. **Parametri:** $\lambda = 1.2$; probabilità p (early-exit/loop).

- **Early-exit** → orizzonte finito: switch di probabilità con analisi in intervallo temporale $[20, 68]$ h (2 giorni) orizzonte infinito: analisi variazione delle metriche a regime stazionario con la probabilità che varia $p \in [0.1, 1]$.
- **Loop** → orizzonte finito: switch di probabilità con analisi in intervallo temporale $[24, 48]$ h.

7.6 Obiettivo 5 (scaling di B)

Non sempre raddoppiare la potenza migliora effettivamente il sistema, ricordiamoci che sono presenti altri nodi. Quindi quando viene fatta una miglioria per un determinato nodo, il bottleneck si sposta su un altro. Ovviamente aumentando le prestazioni di tutti i nodi si avrebbe un miglioramento complessivo del sistema; ma quando si progetta non possiamo limitarci a usare sempre nodi più prestanti in quanto nei sistemi reali dobbiamo tener conto anche dei costi delle macchine. In questo caso assumiamo che il limite massimo degli arrivi sia $\lambda = 1.4$ e ci chiediamo come sarebbe la migliore configurazione di B che a regime ha un compromesso migliore tra utilizzazione e tempi di servizio al variare della potenza. **Parametri:** $\lambda = 1.4$, $E[S_B] \in \{0.40, 0.45, \dots, 0.80\}$ s. Se $E[S_B] \leq 1/\lambda$ ($= 0.7142857$ s) ⇒ **stabile**. **Protocollo: Batch Means** su long-run: $k = 512$ batch contigui, ampiezza $b = 19530$ completamenti (confini su completamenti di *sistema*).

8 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.

Caso (i): Early exit al nodo B (parametro p). Assumiamo $p \in [0, 1]$.

$$V_A(p) = p + 3 \cdot (1 - p) = 3 - 2p, \quad V_B(p) = 1, \quad V_P(p) = 1 - p.$$

$$D_A(p) = E(S_{A1}) + (1 - p)(E(S_{A2}) + E(S_{A3})), \quad D_B(p) = E(S_B), \quad D_P(p) = (1 - p)E(S_P).$$

Caso (ii): Loop $B \rightarrow A_2 \rightarrow B$ (parametro q). Assumiamo $q \in [0, 1]$.

Le visite attese ai nodi (per job) sono:

$$V_A(q) = 3 + \frac{q}{1 - q} = \frac{3 - 2q}{1 - q}, \quad V_B(q) = 1 + \frac{q}{1 - q} = \frac{1}{1 - q}, \quad V_P(q) = 1 - q.$$

$$D_A(q) = E(S_{A1}) + E(S_{A3}) + \frac{1}{1 - q}E(S_{A2}), \quad D_B(q) = \frac{1}{1 - q}E(S_B), \quad D_P(q) = E(S_P)(1 - q).$$

- $[\rho_k]$ 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:

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

Orizzonte infinito (Batch Means). Con $\lambda = 1.2$ stimiamo il regime stazionario con *un'unica run lunga*, usando il metodo delle *Batch Means*: $k = 256$ batch di lunghezza $b = 5070$ completamenti. Il sistema mostra un chiaro *plateau* del tempo di risposta, di conseguenza possiamo andare a stimare le medie con un analisi ad orizzonte infinito. La Fig. 24 riporta la media e la CI 95% calcolata tra batches : i nodi convergono rapidamente ($A \approx 1.56$ s, $B \approx 20$ s, $P \approx 0.77$ s) e l'OVERALL si assesta intorno al valore analitico (≈ 25 s).

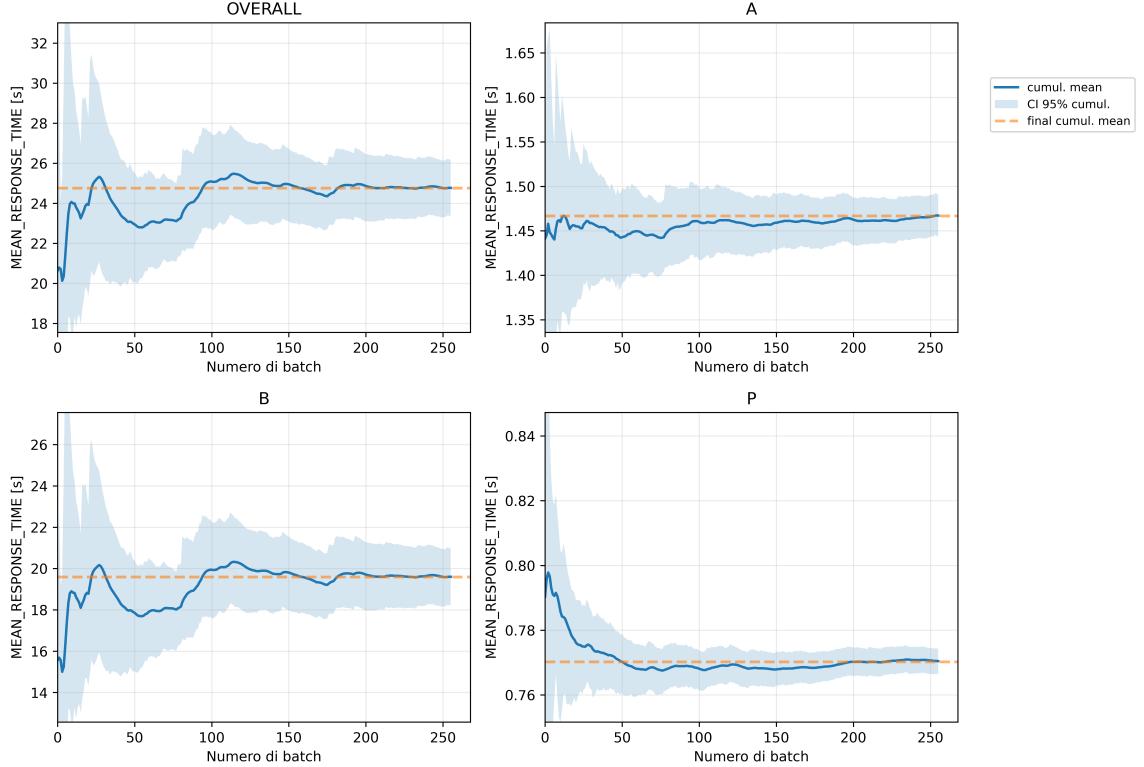


Fig. 24. Orizzonte infinito a $\lambda = 1.2$ (media e CI 95% tra batches). Il plateau evidenzia lo steady-state per sistema e nodi.

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio(s)	25.14423	24.76138	± 1.352
Popolazione media(job)	30.17308	29.81225	± 1.643
Throughput (job/s)	1.20	1.2005	± 0.002
Utilizzazione	0.99667	0.99668	± 0.0002

Table 9. Confronto analitico vs simulazione baseline

8.1 Scenario Obj2 (2FA)

Orizzonte infinito (Batch Means). Con $\lambda = 1.2$ stimiamo il regime stazionario con *un'unica run lunga*, usando il metodo delle *Batch Means*: $k = 256$ batch di lunghezza $b = 5070$ completamenti. Le stime finali e le CI al 95% sono

calcolate *tra batch*; la dipendenza residua (autocorrelazione) è risultata trascurabile inferiore al 0.2. Nei grafici (media cumulativa, EWMA, CI 95%) i nodi B e P si stabilizzano rapidamente; A e OVERALL mostrano rumore iniziale dovuto allo start con sistema vuoto, poi si assestano. Il CI finale tra batch è stabile e si restringe regolarmente.

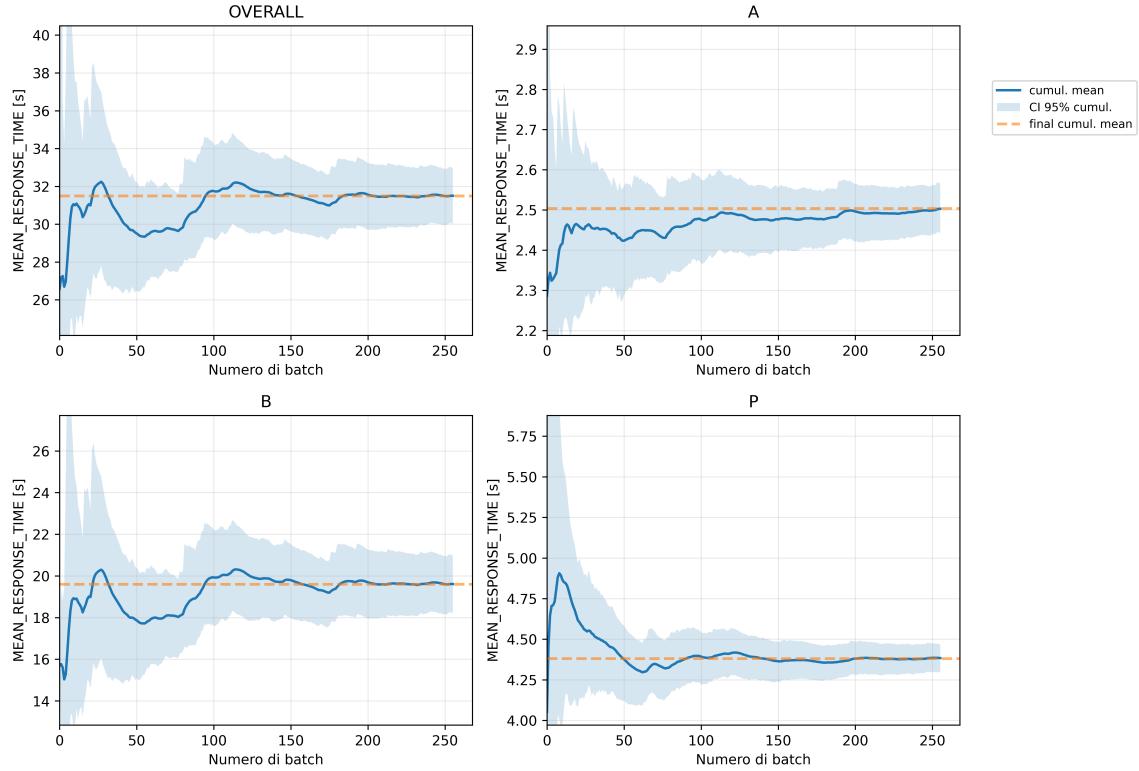


Fig. 25. Obj2 (2FA) — $\lambda = 1.2$: media, CI 95% tra batches.

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*.

Table 10. Confronto analitico vs simulazione (2FA)

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio(s)	31.87500	31.49511	± 1.422
Popolazione media(job)	38.25000	37.90991	± 1.736
Throughput (job/s)	1.20	1.2005	± 0.002
Utilizzazione	0.99936	0.99937	± 0.000

8.1.1 *Impatto del 2FA: confronto tra A, P e OVERALL.* Stimiamo lo steady-state con una singola run lunga e metodo *Batch Means*: per ciascun λ le stime puntuali e i CI al 95% sono calcolati *tra batch*. Con il 2FA (curva rossa tratteggiata,

obj2) il percorso A→P→A aggiunge carico su **P** e, per propagazione, su **A**. Il throughput resta governato da λ , mentre aumentano utilizzazione, popolazione e tempi di risposta — soprattutto su P.

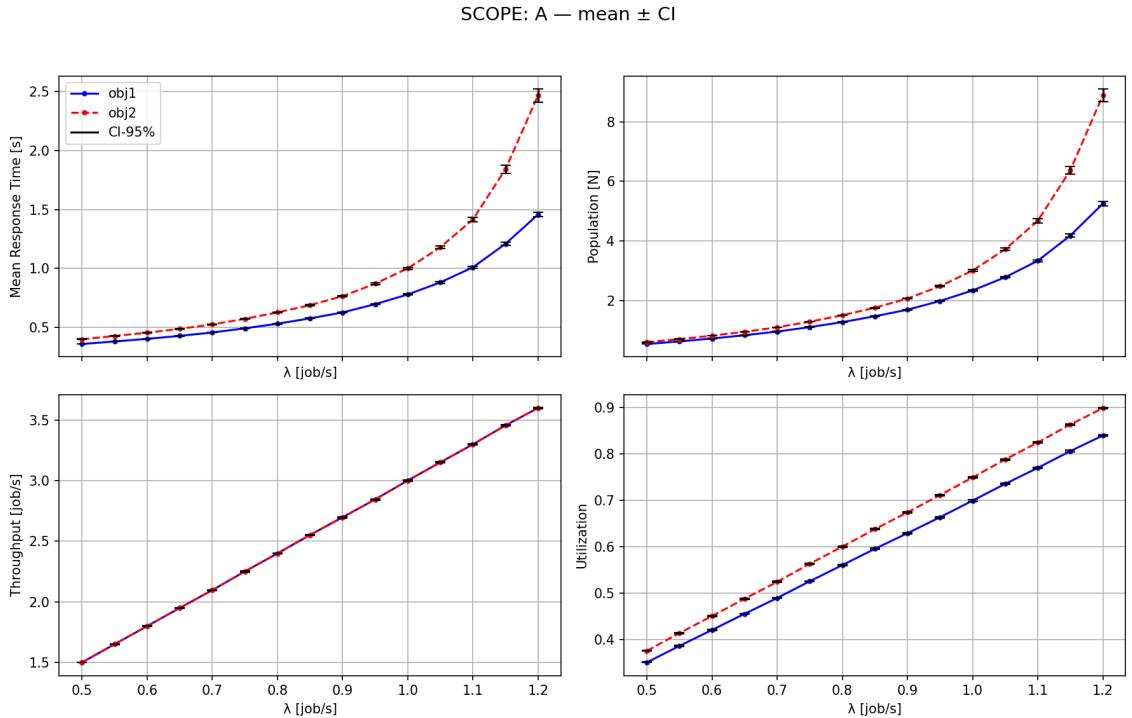


Fig. 26. **Nodo A** — Media \pm CI 95% tra batch per tempo di risposta, popolazione, throughput e utilizzazione al variare di λ . Con 2FA, $E[T]$ e N aumentano progressivamente e l'utilizzazione si avvicina a 1 più rapidamente.

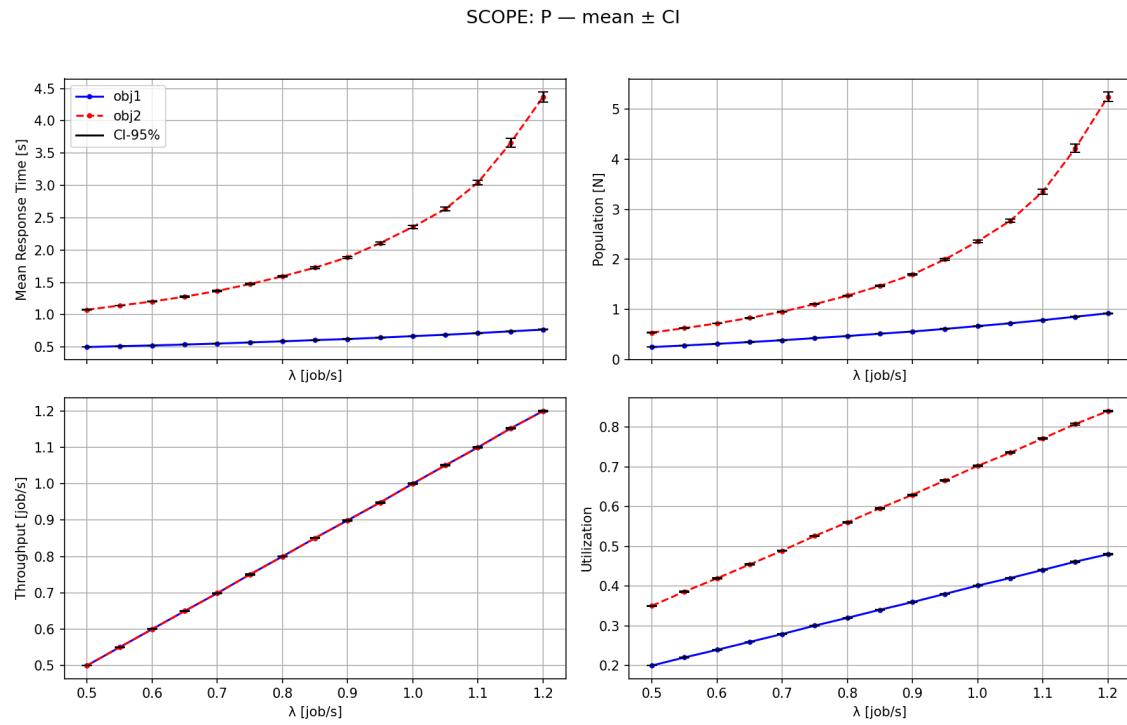


Fig. 27. **Nodo P** — Effetto più marcato del 2FA: forte aumento di $E[T]$ e N con la crescita di λ ; l'utilizzazione sale da valori moderati fino a livelli prossimi alla saturazione.

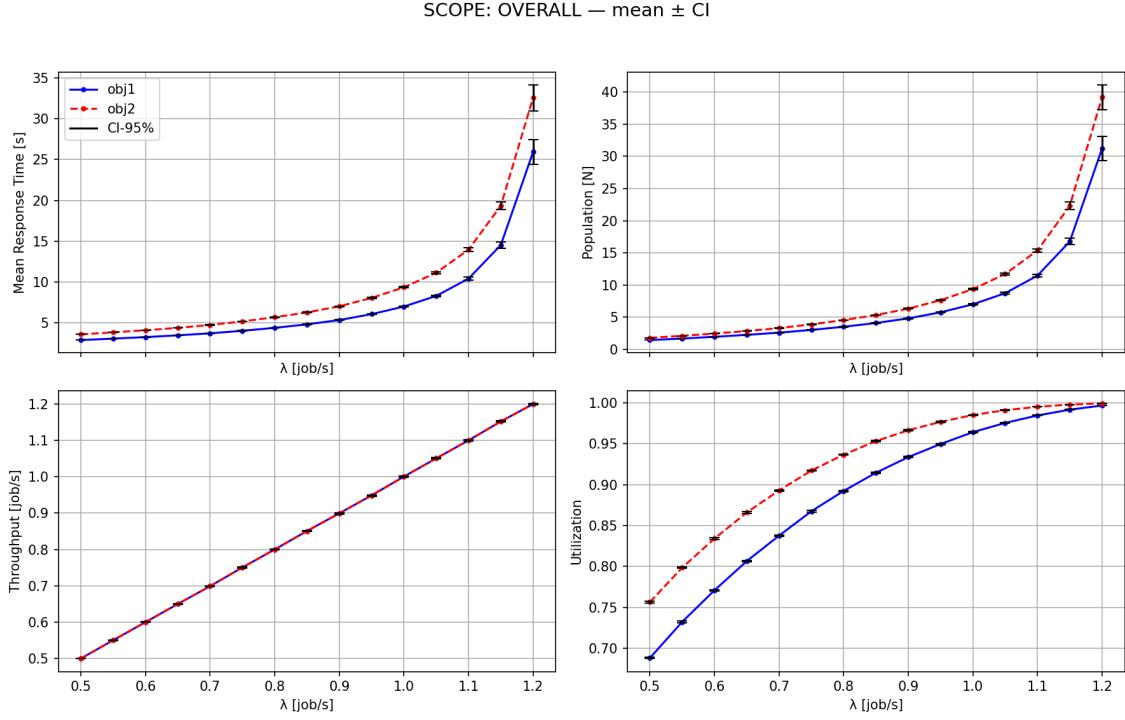


Fig. 28. **OVERALL** — Media \pm CI 95% tra batch aggregando tutti i nodi. L’effetto combinato dell’aumento su P (e su A) porta a un innalzamento chiaro dei tempi di risposta e della popolazione complessiva, mentre il throughput segue λ finché non si raggiunge la saturazione.

Sintesi. 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.

8.2 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.

Metrica	Simulazione	CI 95%
Tempo risposta medio(s)	62.94916	± 3.791
Popolazione media(job)	76.02030	± 4.599
Throughput (job/s)	1.20738	± 0.001
Utilizzazione	0.99718	± 0.000

Table 11. Metriche raccolte nella finestra [0,24h]

8.2.1 Obj3 – Spike di carico ($1.2 \rightarrow 1.4$) su orizzonte finito (24h). All'inizio manteniamo un carico moderato ($\lambda = 1.2$), sotto la capacità del collo di bottiglia: le metriche oscillano attorno a un livello quasi stazionario. In un istante $t_{sw} = 20000s$ aumentiamo l'ingresso a $\lambda = 1.4 (> X_{max})$, il che innesca l'accumulo: code e tempo di risposta crescono. Nella prima fase iniziale il sistema riesce a gestire per poco tempo il nuovo tasso di arrivi, successivamente il sistema si satura. Terminato il burst, il sistema smaltisce il backlog: la curva decresce gradualmente verso i livelli precedenti, restando più alta per un tratto a causa della coda accumulata. Usiamo un orizzonte finito [0, 24] con $R = 64$ e stimiamo la media tra repliche con CI 95% nel tempo.

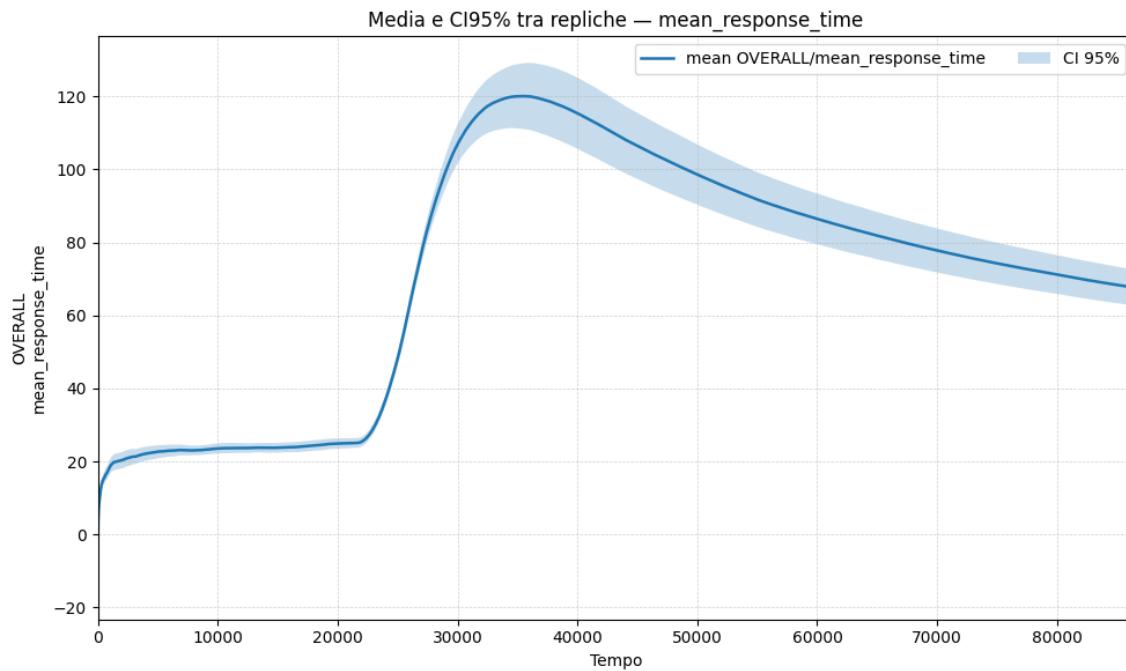


Fig. 29. Obj3 – **tempo di risposta OVERALL nel tempo** (media tra repliche \pm CI 95%) in seguito a uno step di carico da $\lambda = 1.2$ a $\lambda = 1.4$: crescita rapida, picco e successivo *smaltimento* del backlog(24h).

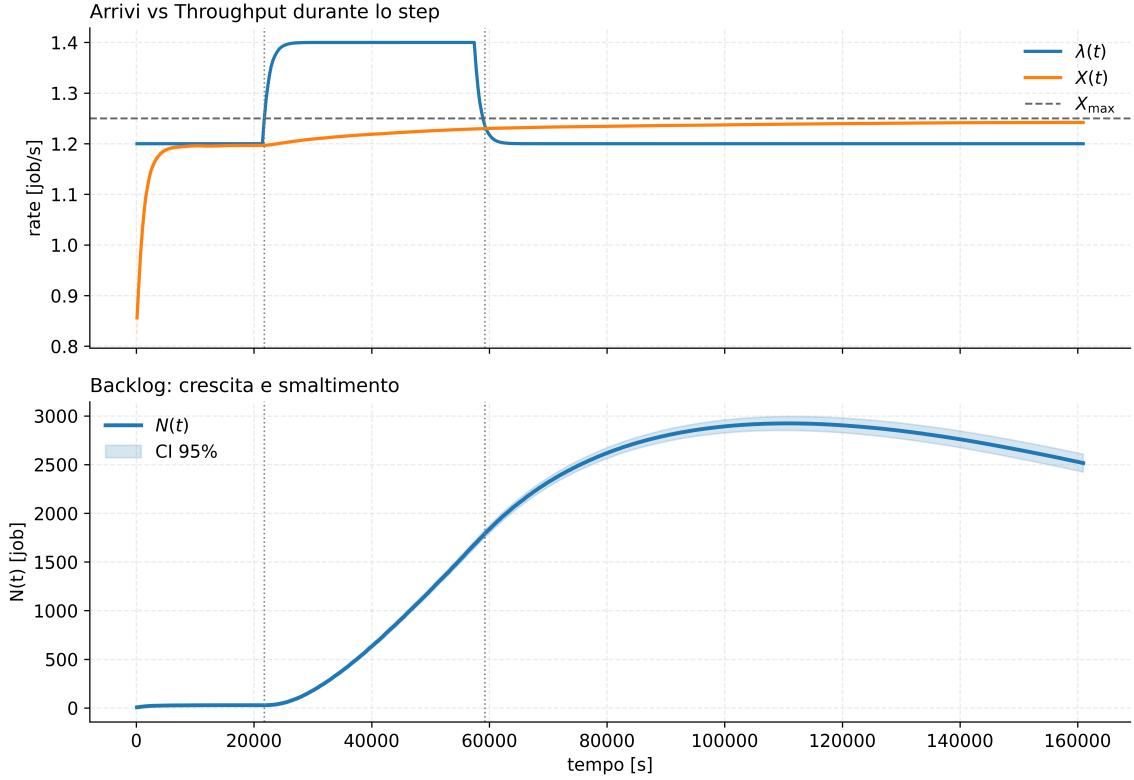


Fig. 30. Obj3 – **step di carico** 1.2 → 1.4 → 1.2. *In alto:* arrivi $\lambda(t)$ e throughput $X(t)$ (linea tratteggiata = capacità X_{\max}). *In basso:* backlog $N(t)$ con banda CI_{95%} (media tra repliche su [0,48h]).

8.3 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.

Orizzonte finito e finestra di osservazione. Per evidenziare la dinamica transiente dopo l'aumento della probabilità di uscita, adottiamo un *orizzonte finito*: fissiamo una finestra temporale [19h, 67h] e, per ogni replica, campioniamo la metrica tempo di risposta medio di sistema. Ripetiamo l'esperimento con $R = 64$ *repliche indipendenti* (semi diversi, stesso scenario e stessi istanti di campionamento) e, per ciascun istante t , calcoliamo la *media tra repliche*.

p	Analitico [s]	Simulazione [s]	CI_95
0.1	23.58835	22.59788	± 1.155
0.2	22.66234	21.87211	± 0.793
0.3	22.03933	21.26186	± 0.879
0.4	21.58708	20.83848	± 0.802
0.5	21.24142	20.51865	± 0.806
0.6	20.96725	20.27093	± 0.917
0.7	20.74364	20.08483	± 1.001
0.8	20.55725	19.99813	± 0.758
0.9	20.39916	19.87066	± 0.599
1.0	20.26316	19.70292	± 0.630

Table 12. Sweep della probabilità p e confronto dei tempi di risposta complessivi (Analitico vs Simulazione)

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

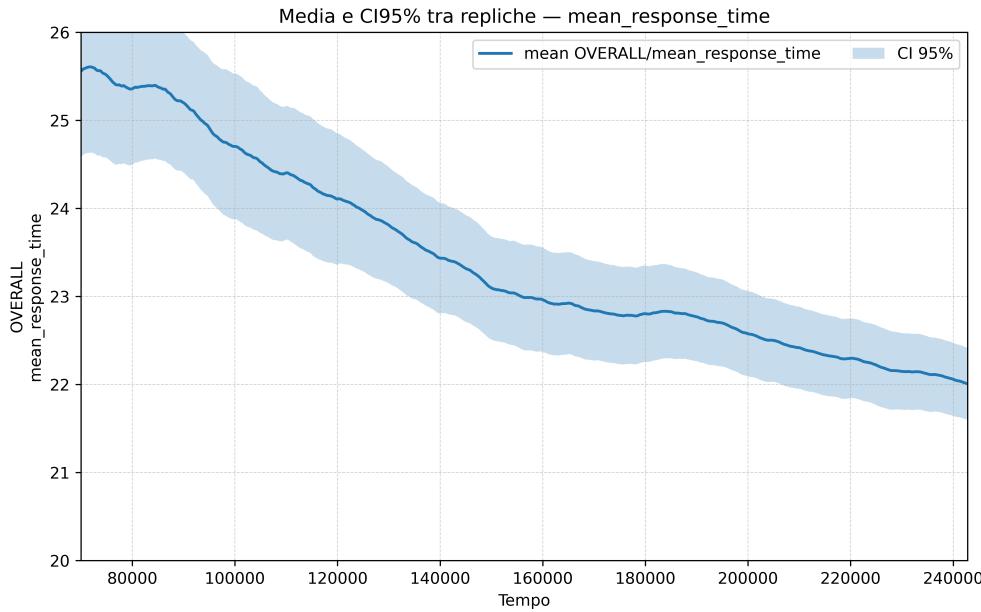


Fig. 31. Obj4 – Orizzonte finito: media tra repliche del tempo di risposta di sistema con CI95% (t–Student). Dopo l'aumento della probabilità di uscita, il tempo di risposta cala gradualmente lungo la finestra.

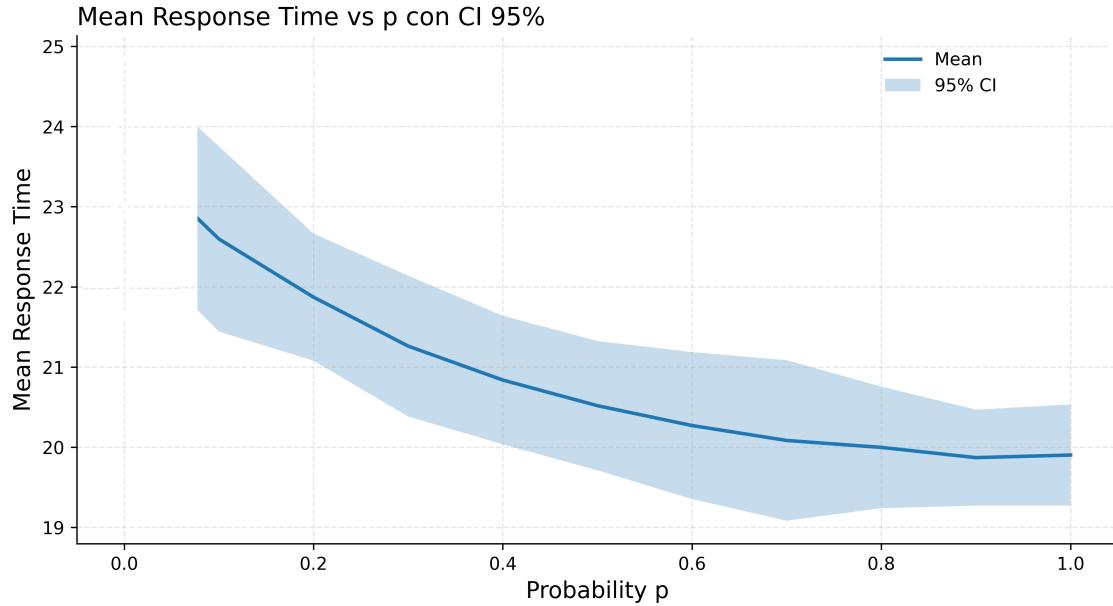


Fig. 32. Obj4 – Mean response time vs probabilità di uscita p (media tra repliche: il tempo di risposta cala con p).

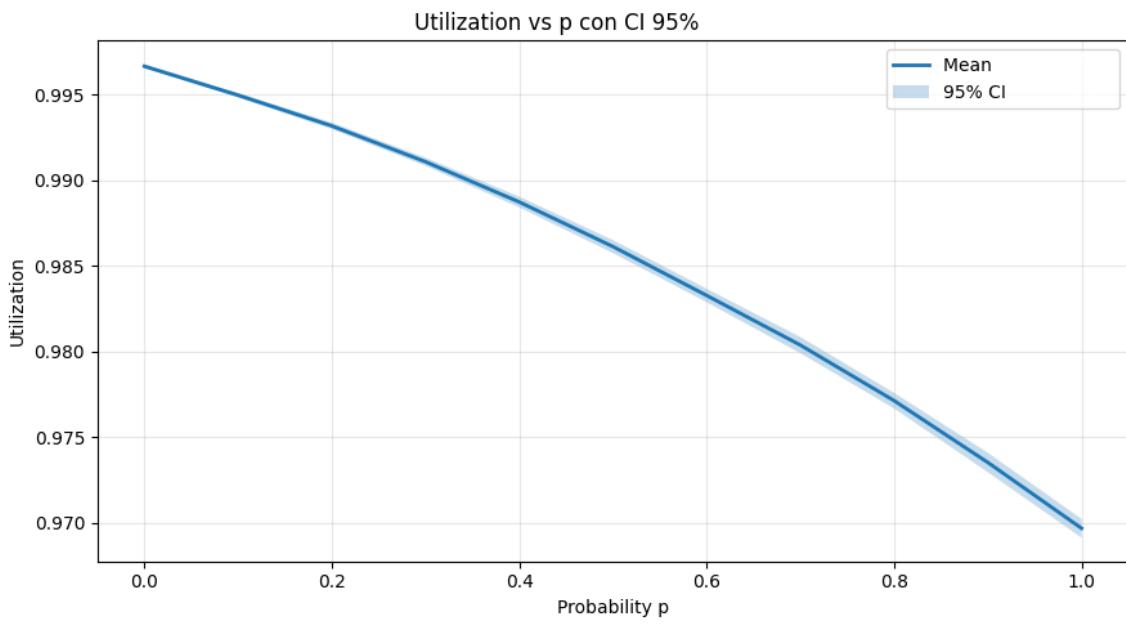


Fig. 33. Obj4 – Utilizzazione vs p l'aumento di p riduce le domande D_k e quindi ρ , con un calo regolare (qui $\sim 2\text{--}3$ p.p. sull'intervallo).

8.4 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	31.61378	± 3.194
0.02	45.48390	42.34325	± 6.509
0.03	85.67690	89.51457	± 27.847
0.04	∞	655.01741	± 287.636
0.05	∞	4535.96379	± 572.666

Table 13. Sweep della probabilità p e confronto dei tempi di risposta complessivi (Analitico vs Simulazione)

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

Nell'orizzonte *finito* consideriamo la finestra temporale [22, 44] h. La curva blu riporta la *media tra repliche indipendenti* ($R = 64$) del tempo di risposta di SYSTEM, mentre la banda azzurra è la CI95% (t-Student) calcolata puntualmente nel tempo. L'asse y è in *scala logaritmica* per evidenziare la crescita accelerata. Dopo l'introduzione del *loop* di rientro, il carico effettivo supera la capacità del sistema: il throughput non riesce a inseguire gli arrivi e il tempo di risposta *aumenta notevolmente* (instabilità). La linea tratteggiata indica la baseline pre-cambio; la banda di confidenza appare relativamente stretta perché la scala log comprime la variabilità relativa.

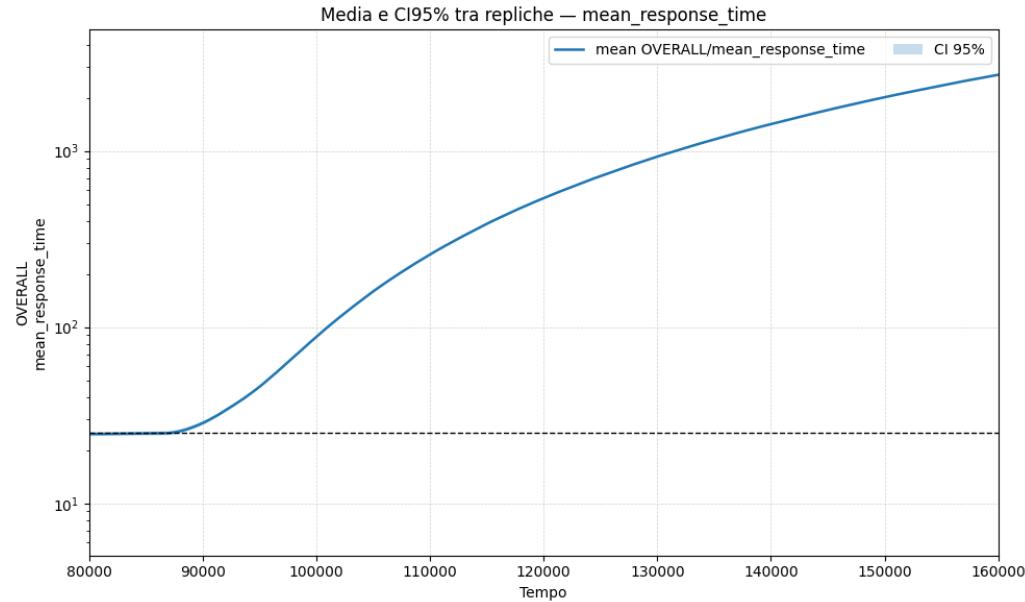


Fig. 34. Obj4 – Orizzonte finito con *loop*: media tra repliche del tempo di risposta di SYSTEM (asse log) con CI95% (t-Student). Dopo il cambio il sistema entra in regime instabile e il tempo di risposta diverge; la linea tratteggiata segna la baseline pre-cambio.

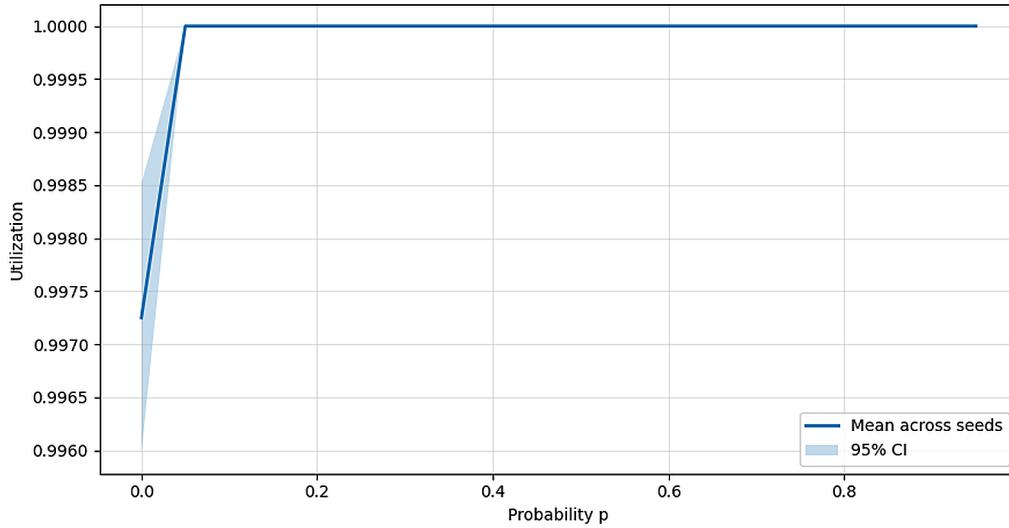


Fig. 35. Obj4-loop – Utilizzazione vs probabilità di loop: ρ tende a 1 per il collo di bottiglia.

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

Proseguiamo adesso con l'analisi di due configurazioni chiave a $\lambda = 1.4$. Abbiamo testato tre valori mirati: (i) $E[S_B] = 0.4$ s ovvero il raddoppio della potenza del server B. (ii) $E[S_B] \approx 0.713$ s, immediatamente sotto la soglia analitica, (iii) $E[S_B] = 0.65$ s, che garantisce maggiore margine.

Orizzonte infinito (Batch Means). Con $\lambda = 1.4$ stimiamo il regime stazionario con *un'unica run lunga*, usando il metodo delle *Batch Means*: $k = 512$ batch di lunghezza $b = 19530$ completamenti. Le stime finali e le CI al 95% sono calcolate *tra batch*; la dipendenza residua è risultata trascurabile. Andiamo ad analizzare Mean response time e Utilizzazione.

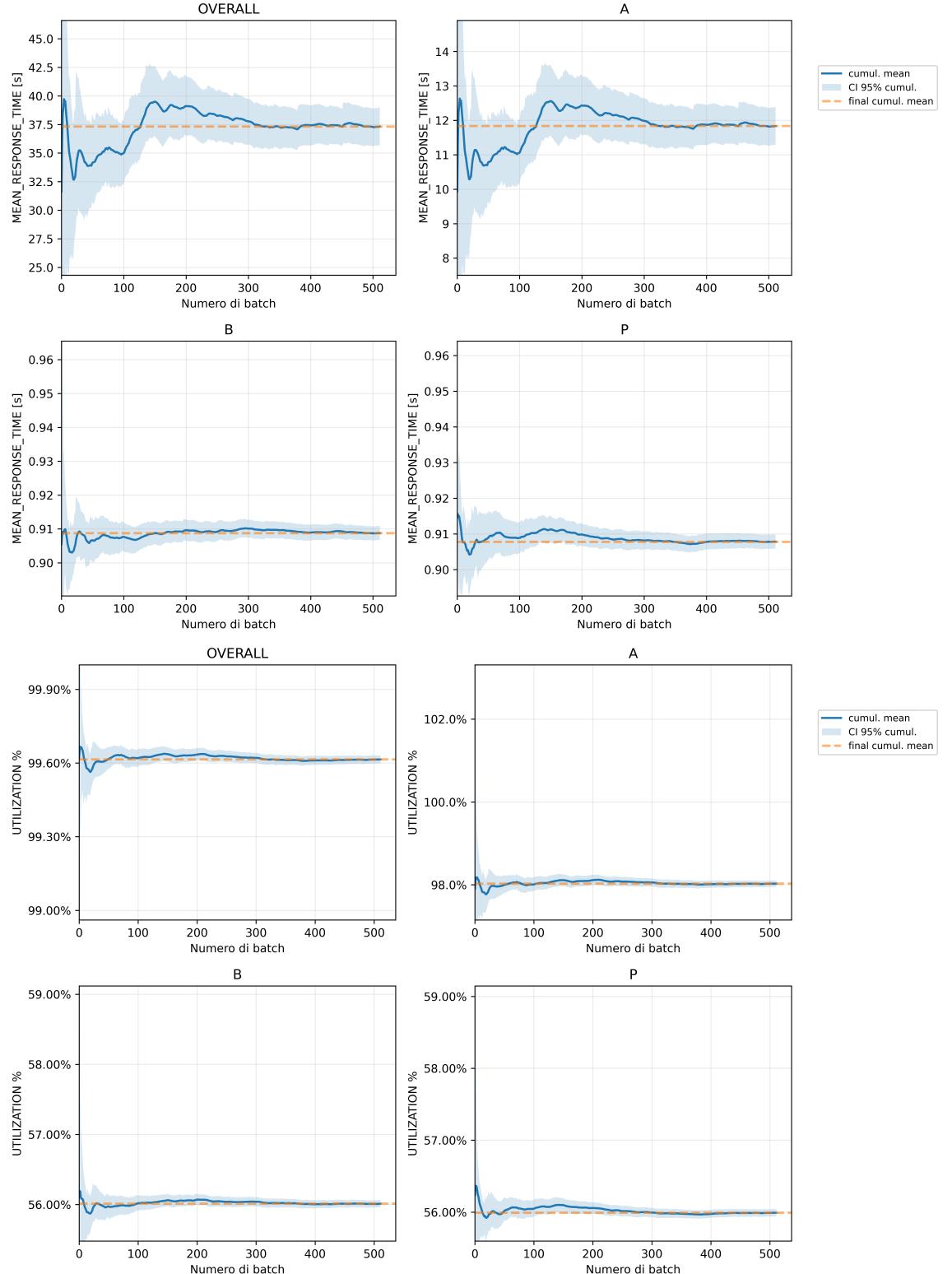


Fig. 36. Scenario di scaling 0.4 ($\lambda = 1.4$). Pannello superiore: evoluzione del **tempo di risposta** (media cumulativa con CI 95% tra batch) per OVERALL, A, B e P. B e P si stabilizzano rapidamente; A e OVERALL mostrano una sola onda di avvio e poi un *plateau* stabile. Pannello inferiore: **utilizzazione** (media cumulativa con CI 95%). Le utilizzazioni risultano molto elevate: OVERALL ≈ 1 , A ≈ 0.98 , B ≈ 0.91 , P ≈ 0.56 . A è il nodo più sollecitato (quasi collo di bottiglia), mentre P rimane lontano dalla saturazione.

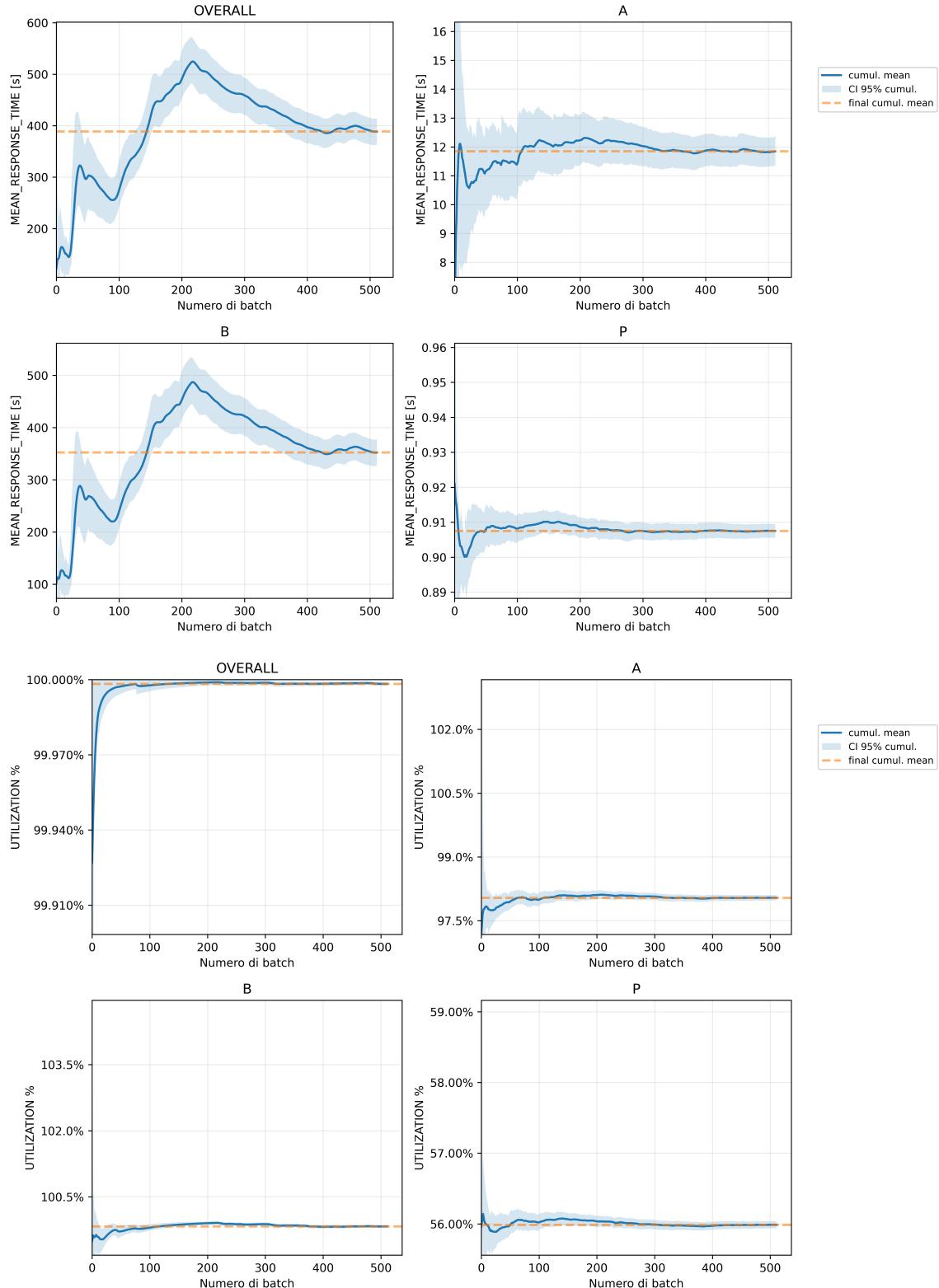


Fig. 37. Scenario con $\lambda = 1.4$ e $E[S_B] \approx 0.713$ s. **Pannello superiore:** evoluzione del **tempo di risposta** (media cumulativa con CI 95% tra batch) per OVERALL, A, B e P. **Pannello inferiore:** **utilizzazione** (media cumulativa con CI 95%). Le utilizzazioni sono molto elevate: OVERALL ≈ 1 , A ≈ 0.98 , B ≈ 0.91 , P ≈ 0.56 . A risulta il nodo più sollecitato.

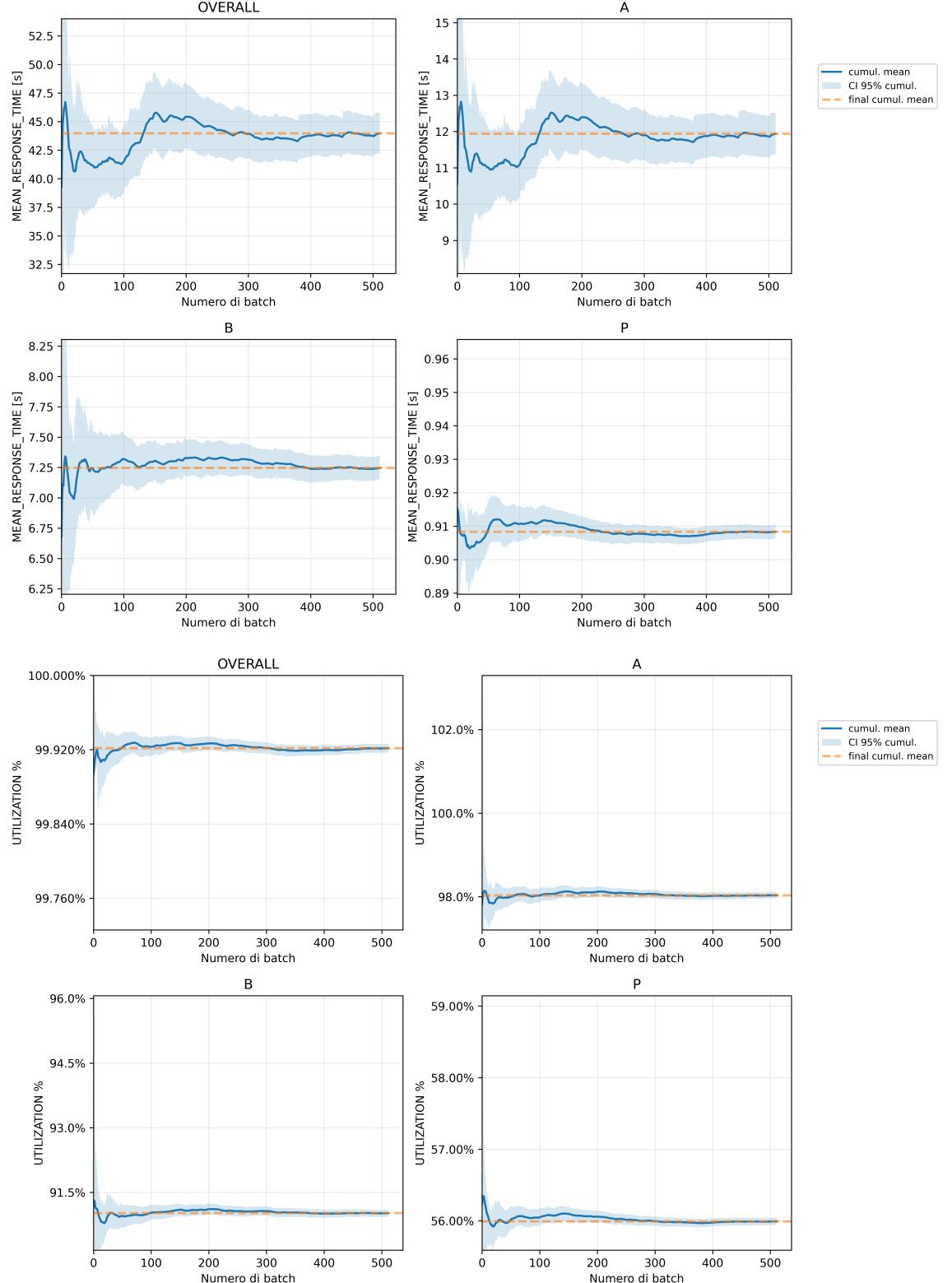


Fig. 38. Scenario con $\lambda = 1.4$ e $E[S_B] = 0.65$ s. **Pannello superiore:** evoluzione del **tempo di risposta** (media cumulativa con CI 95% tra batch) per OVERALL, A, B e P. B e P si stabilizzano rapidamente; A e OVERALL mostrano una lieve onda iniziale e poi un *plateau* stabile. **Pannello inferiore:** **utilizzazione** (media cumulativa con CI 95%). Le utilizzazioni restano elevate ma sotto saturazione: OVERALL ≈ 1 , A ≈ 0.98 , B ≈ 0.91 , P ≈ 0.56 . L'incremento di capacità di B ($E[S_B] = 0.65$ s) allontana la sua saturazione; il collo di bottiglia rimane A.

Aumentiamo la capacità del nodo B riducendo il suo tempo medio di servizio $E[S_B]$ (*vertical scaling*). In analitico, il limite di capacità del nodo è $X_{\max,B} = 1/E[S_B]$; pertanto, per non saturare B a $\lambda = 1.4$ è necessario porsi a $E[S_B] \leq 1/\lambda \approx 0.714$ s.

La Fig. 39 mostra l'utilizzazione media di B al variare di λ per diversi $E[S_B]$: curve con $E[S_B]$ più basso raggiungono $\rho_B \approx 1$ (linea tratteggiata) più tardi, ovvero *spostano a destra* la saturazione. La Fig. 40 riporta il throughput medio in B : cresce quasi linearmente con λ fino al limite $X_{\max,B}$, oltre il quale compare un plateau. Riducendo $E[S_B]$, il plateau trasla verso destra, segno di collo di bottiglia mitigato.

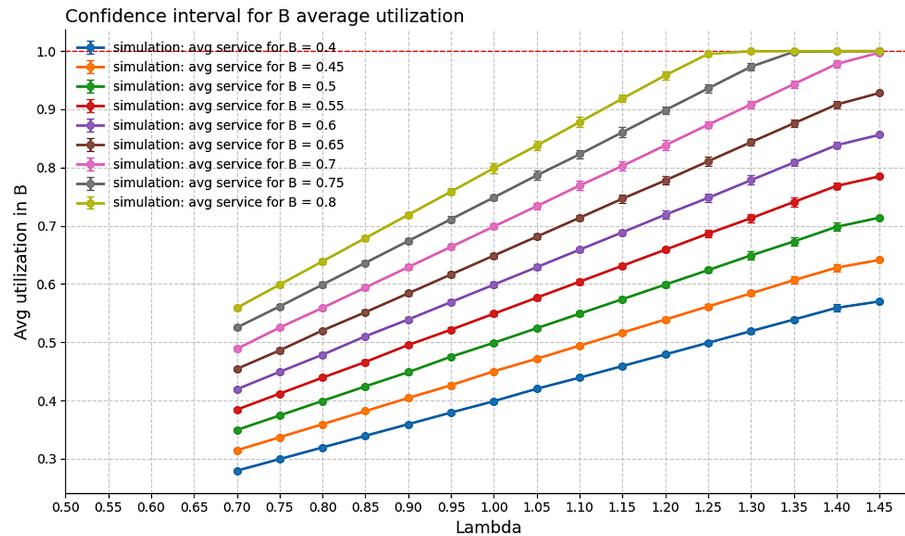


Fig. 39. Obj5 – Utilizzazione media in B al variare di λ e di $E[S_B]$. Le curve con $E[S_B]$ più basso saturano più tardi; la linea tratteggiata rossa indica $\rho_B = 1$.

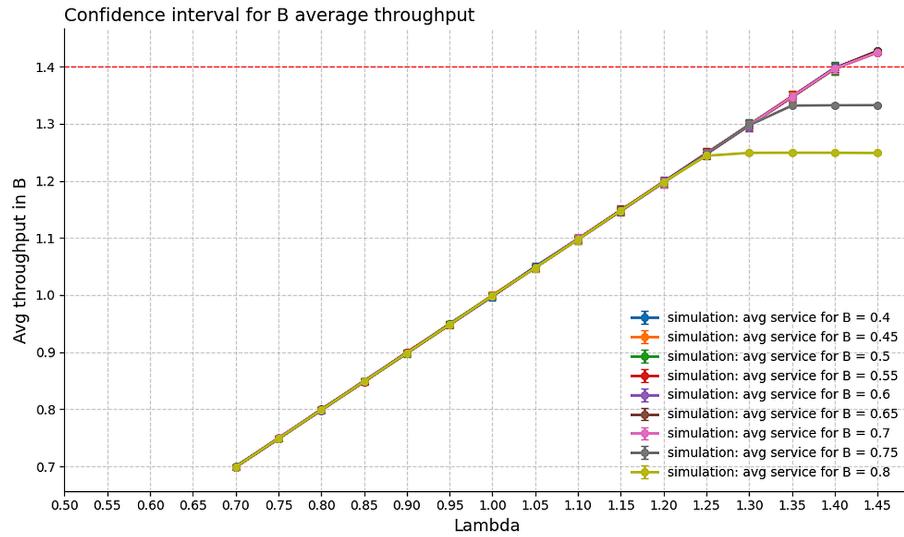


Fig. 40. Obj5 – Throughput medio in B: crescita quasi lineare con λ fino a $X_{\max,B} = 1/E[S_B]$, poi plateau. Riducendo $E[S_B]$ il plateau si sposta a destra, sintomo di collo di bottiglia mitigato.

Osservazione analisi metriche con $B = 0.4$ e $B = 0.65$. Nell'analisi abbiamo un miglioramento significativo quando abbiamo un B migliore.

Table 14. Confronto analitico vs simulazione ($B=0.65s$)

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio(s)	43.13131	43.9825	± 1.769
Popolazione media(job)	60.38384	61.6806	± 2.520
Throughput (job/s)	1.40000	1.39996	± 0.001
Utilizzazione	0.99921	0.99921	± 0.000

Table 15. Confronto analitico vs simulazione ($B=0.4s$)

Metrica	Analitico	Simulazione	CI 95%
Tempo risposta medio(s)	36.81818	37.3268	± 1.634
Popolazione media(job)	51.54545	52.361	± 2.325
Throughput (job/s)	1.40000	1.39996	± 0.001
Utilizzazione	0.99613	0.9961	± 0.000

8.6 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.

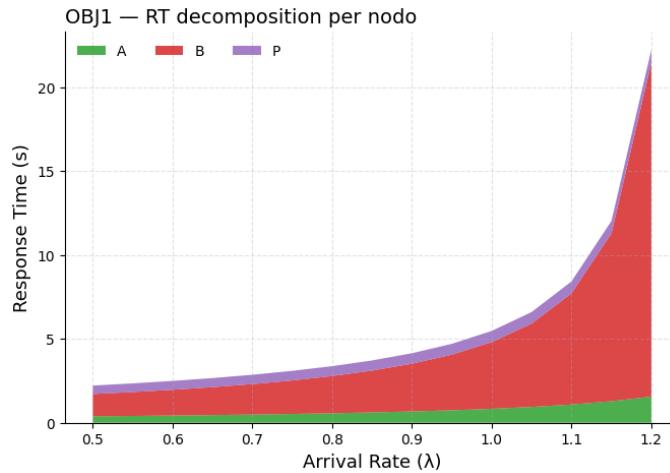


Fig. 41. Obj1 – Decomposizione $E[T_s]$ per nodo: **B** cresce fino a dominare il totale.

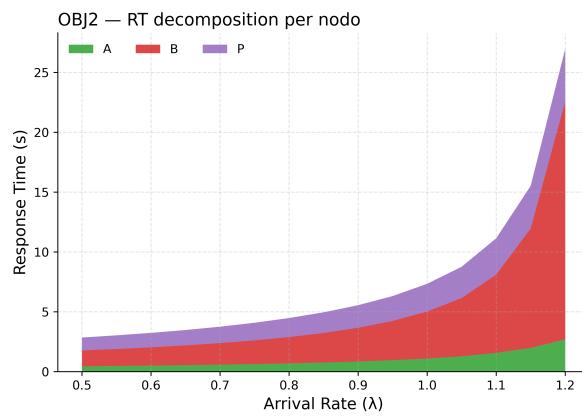


Fig. 42. Obj2 – Con 2FA aumentano le quote di **A** e **P**; **B** resta il limite di capacità.

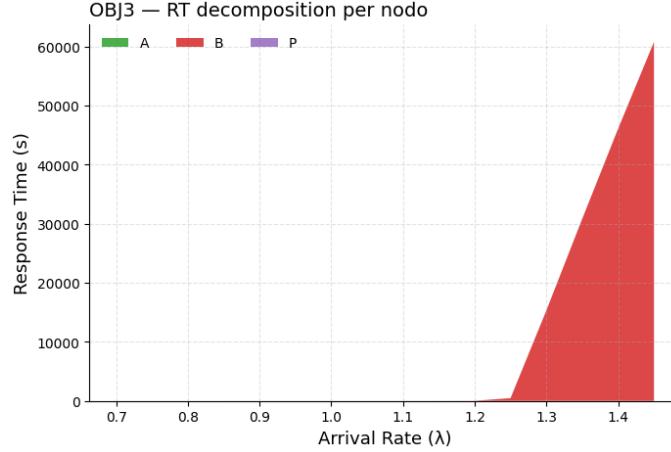
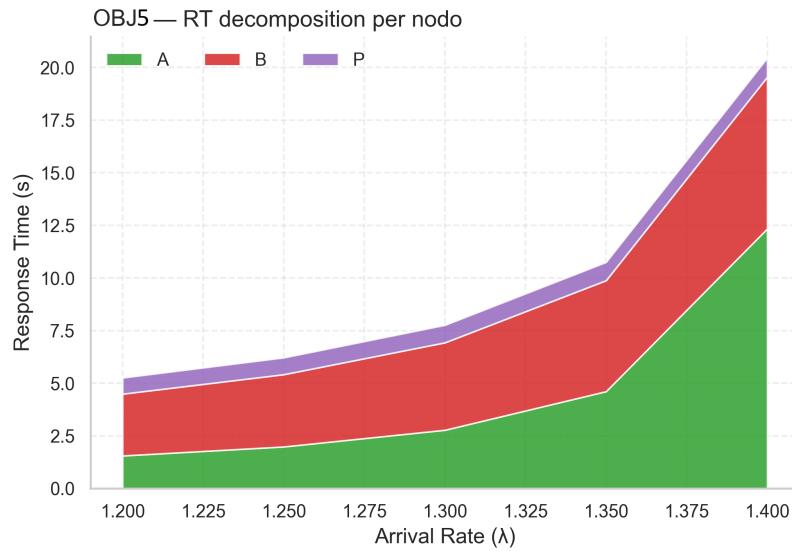
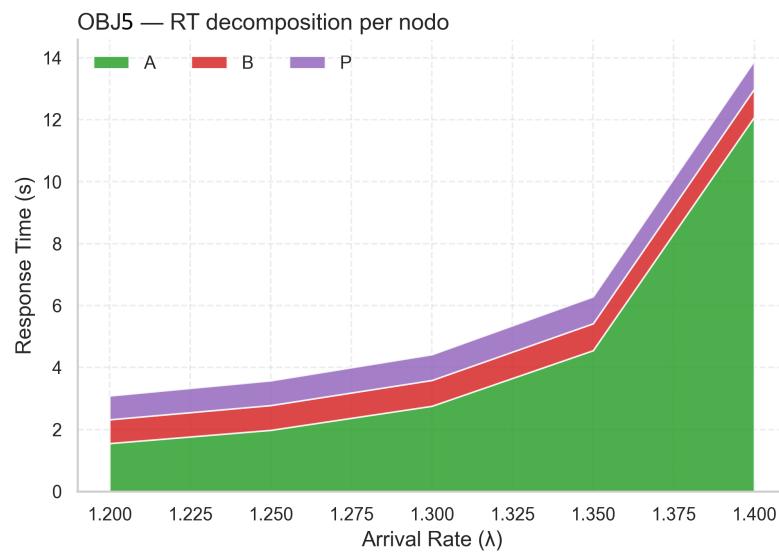


Fig. 43. Obj3 — Overload: la componente di **B** diverge ($\rho_B \geq 1$); sistema instabile.

8.6.1 Analisi del collo di bottiglia. Con $E[S_B] \approx 0.713$ s il collo di bottiglia rimane in *B* infatti le simulazioni di *popolazione e tempo di risposta* su *B* si posizionano più in alto delle altre componenti e convergono vicino al punto previsto dall'analitico (Fig. 37). Portando *B* a $E[S_B] = 0.65$ s, invece, il collo di bottiglia *si sposta su A*. I grafici ad orizzonte infinito (Fig. 38) mostrano *popolazione e tempi di risposta* di *A* nettamente superiori a quelli di *B* e *P*, con *A* che opera vicino al proprio punto di saturazione(utilizzazione elevata) pur rimanendo stabile. Questo shift è coerente con la riduzione della domanda su *B* e conferma che l'upgrade di *B* scarica il nodo ma rende *A* il nuovo elemento critico del sistema.

Scelta del parametro. Selezioniamo $E[S_B] = 0.65$ s come configurazione finale perché garantisce *headroom* sufficiente con il tasso di arrivo a $\lambda = 1.4$; evita condizioni di *borderline stability* (tipiche di $E[S_B] \approx 0.713$ s), riduce sensibilmente rumore, e sposta il plateau oltre l'intervallo operativo, lasciando margine per eventuali ottimizzazioni su *A*, ora nodo dominante per le metriche. Decomposizione tempo totale tra i nodi evidenzia la salita di *A* come collo di bottiglia.

Fig. 44. Decomposizione $E[S_B] = 0.65$ sFig. 45. Decomposizione $E[S_B] = 0.4$ s

9 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.