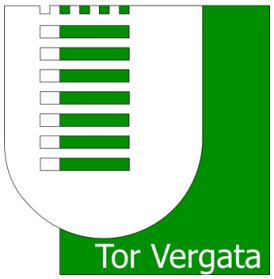

Università degli Studi di Roma "Tor Vergata"

Dipartimento di Ingegneria Civile e Ingegneria Informatica

Corso di

Performance Modeling of Computer Systems and Networks

Workflow di una Web App

Progetto di modellistica, simulazione e valutazione delle prestazioni

Matteo Basili 0342020

Anno Accademico 2024/2025

Indice

1 Introduzione

Il caso di studio analizza la simulazione e la valutazione delle prestazioni di un sistema di e-commerce per la vendita online di generi alimentari, ispirato al modello descritto nel Capitolo 6 di Serazzi [1]. Il sistema rappresenta una tipica architettura web multi-tier in cui più server collaborano per elaborare le richieste secondo un workflow deterministico.

L'obiettivo del progetto è sviluppare un modello di simulazione a eventi discreti capace di riprodurre il comportamento dinamico dell'applicazione e di valutare le principali metriche prestazionali, con particolare attenzione ai colli di bottiglia che limitano il throughput.

Il caso di studio è rilevante poiché consente di:

- analizzare l'impatto di modifiche architetturali o funzionali, come l'introduzione dell'autenticazione a due fattori (Strong Customer Authentication, SCA);
- osservare l'interazione tra componenti eterogenei in una configurazione web realistica;
- condurre esperimenti di *capacity planning* e *performance tuning*, in linea con gli obiettivi del corso.

La simulazione prende in considerazione tre scenari di carico:

1. **Light workload:** sistema attuale, autenticazione a un fattore;
2. **SCA workload:** stesso carico, ma con autenticazione a due fattori;
3. **Heavy workload:** incremento del 15% del tasso di arrivo.

Tra i benefici attesi dall'analisi vi sono:

- una maggiore comprensione del comportamento del sistema sotto differenti livelli di carico;
- la quantificazione del throughput massimo e l'identificazione del server critico;
- la valutazione degli effetti di strategie migliorative.

2 Descrizione del Sistema

Il sistema simulato è un data center che ospita una piattaforma di e-commerce. È composto da tre nodi principali, che cooperano nell'elaborazione delle richieste utente secondo un workflow deterministico. Le richieste cambiano *Class ID* in funzione della fase applicativa (Fig. 1).

- **Server A — Front-End Web Services**

  Server multicore che esegue servizi di autenticazione, interazione con il pagamento, aggiornamento database, fatturazione e altre funzioni di front-end. Ogni richiesta vi accede tre volte, ma con tempi di servizio differenti a seconda della classe.

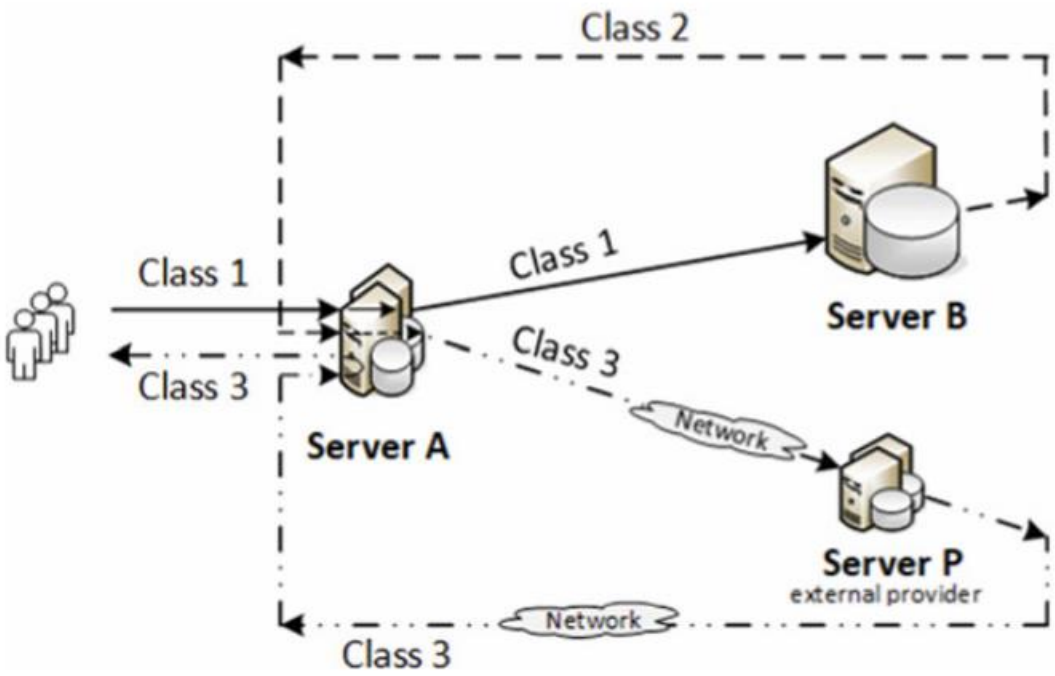

Fig. 1 Il data center con il percorso seguito dalle richieste e i loro *Class ID*

- **Server B — Database Server**

  Sistema multiprocessore blade dedicato alla navigazione del catalogo e alla gestione dei database di prodotti e clienti.
  È potenzialmente il nodo più critico del sistema, avendo il tempo di servizio medio più elevato.

- **Server P — Payment Server**

  Server esterno, gestito da un fornitore terzo, utilizzato per le procedure di pagamento. Ogni richiesta vi transita una sola volta.
  Con l'introduzione del protocollo SCA, i suoi tempi di servizio aumentano sensibilmente.

Le richieste seguono il seguente workflow (Fig. 2):

$$\mathbf{A \rightarrow B \rightarrow A \rightarrow P \rightarrow A}$$

Durante queste transizioni avviene il cambio del *Class ID*, che consente di modellare servizi differenti pur visitando gli stessi nodi.

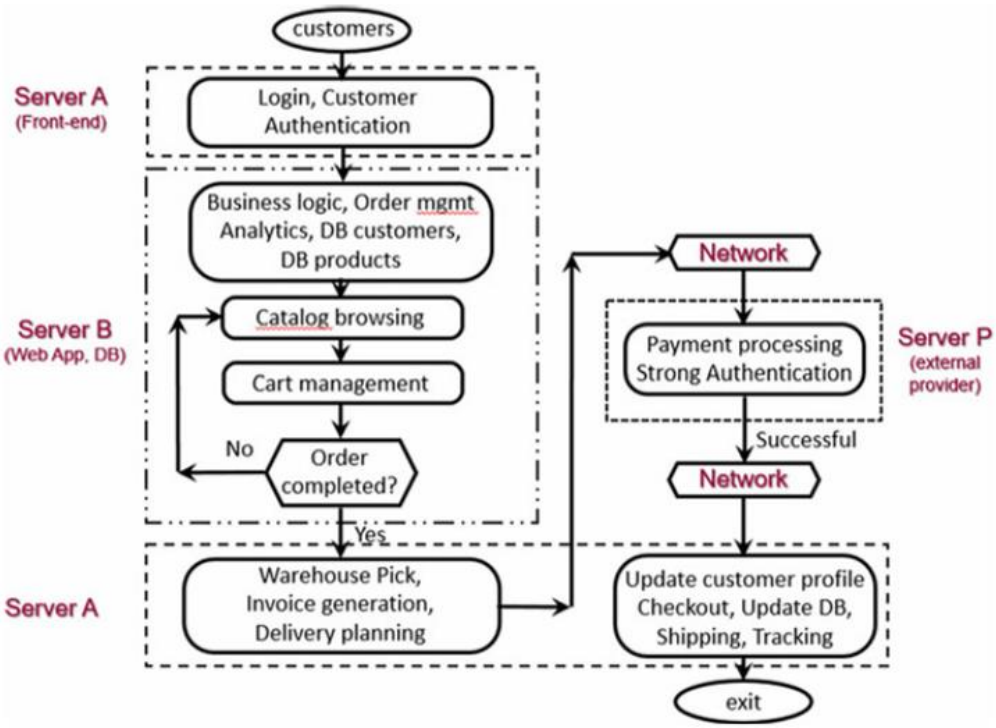

Fig. 2 Versione breve del workflow per l'invio di un ordine a un negozio di generi alimentari online

---

3 Obiettivi e Scopi dell'Analisi

L'analisi mira a valutare le prestazioni del sistema e a pianificarne la capacità operativa nei tre scenari considerati.
Gli obiettivi sono organizzati in quattro aree principali.

1. **Valutazione del sistema attuale**

   Si verifica se la configurazione corrente (autenticazione a un fattore) è in grado di sostenere il carico attuale di circa 4300 richieste/ora, mantenendo un tempo di risposta inferiore a 30 secondi.
   In questa fase si identificano il collo di bottiglia (atteso: Server B) e il *Throughput bound* del sistema.

2. **Impatto dell'autenticazione a due fattori (SCA)**

   Si analizza il degrado prestazionale dovuto all'aumento dei *Service Demand* richiesti dal nuovo protocollo SCA, osservando variazioni in throughput, utilizzazioni e tempi di risposta. L'obiettivo è determinare se il tempo di risposta cresce di oltre il 20% rispetto alla configurazione attuale.

3. **Analisi di capacità con *heavy workload***

   Si studia il comportamento del sistema con un incremento di circa il 15% nel tasso di arrivo (≈5000 richieste/ora), valutando se l'infrastruttura attuale sia sufficiente oppure se sia necessario potenziare il Database Server (Server B).

4. **Effetti del potenziamento del Database Server**

   Si valutano le prestazioni del sistema nel caso in cui il Server B venga sostituito con una versione due volte più veloce (Service Demand dimezzato).
   L'analisi consente di verificare:

   - il collo di bottiglia in questa configurazione (atteso: Server A);
   - l'incremento percentuale del throughput rispetto al sistema originale;
   - la capacità del nuovo sistema di sostenere il tasso di 1,4 richieste/s richiesto dallo scenario *heavy workload*.

Per raggiungere tali obiettivi, il sistema viene analizzato sia in **fase transitoria** (per valutare la convergenza verso lo stato stazionario) sia in **regime permanente** (per misurare le prestazioni stabili).

4 Modello Concettuale

Il sistema di e-commerce è stato modellato come una rete aperta di code, rappresentata in Fig. 3.
La rete comprende tre sottosistemi principali, corrispondenti ai server coinvolti nel workflow applicativo:

- **Blocco A – Server A**
- **Blocco B – Server B**
- **Blocco P – Server P**

4.1 Struttura del Modello

Ogni blocco viene modellato come una coda **M/M/1 con disciplina Processor Sharing (PS)** e coda illimitata.
Questa scelta consente di rappresentare in modo realistico sistemi web single-server con elevata variabilità nei tempi di servizio, senza introdurre vincoli artificiali sulla capacità di accettazione delle richieste.

Poiché il Server A viene visitato tre volte da ogni richiesta ed esegue funzioni differenti a seconda della fase, il modello distingue tre **classi** di richieste:

- **Classe 1:** prima visita a Server A, seguita da Server B;
- **Classe 2:** seconda visita a Server A, seguita da Server P;
- **Classe 3:** terza visita a Server A, al termine della quale avviene l'uscita dal sistema (Sink).

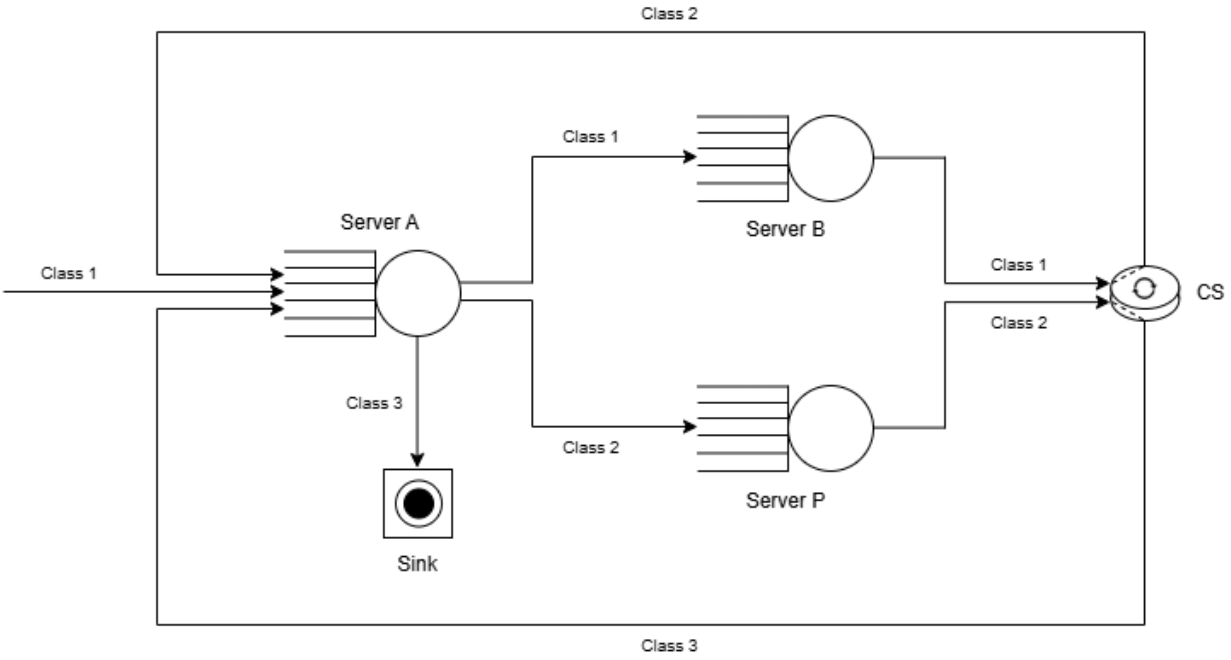

Fig. 3 Modello concettuale del sistema

Per modellare la progressione tra le tre fasi applicative, viene introdotta una stazione di **Class-Switch (CS)**.
Tale nodo non esegue servizio: si limita a modificare il Class ID e reindirizzare la richiesta al blocco successivo.
In questo modo il workflow applicativo è rappresentato fedelmente senza introdurre ritardi non necessari.

4.2 Eventi del Sistema

Il modello prevede tre eventi principali:

1. **Arrivo di una nuova richiesta**.
2. **Completamento del servizio** presso uno dei nodi A, B o P.
3. **Cambio di classe**, effettuato dalla stazione CS in modo istantaneo.

Questi eventi costituiscono l'insieme minimo necessario per la simulazione a eventi discreti.

4.3 Variabili di Stato

Il comportamento del sistema è descritto mediante le seguenti variabili di stato:

- numero di richieste in servizio:
  - $N_A(t)$, $N_B(t)$, $N_P(t)$;

- **Class ID** associato a ciascuna richiesta attiva;
- **tempo di risposta individuale**, aggiornato lungo il percorso della richiesta;
- **utilizzazione dei server**:
  - $U_A(t)$, $U_B(t)$, $U_P(t)$.

4.4 Dinamica delle Richieste

Il routing è completamente deterministico:

- **Classe 1**: A → B → CS → Classe 2
- **Classe 2**: A → P → CS → Classe 3
- **Classe 3**: A → Sink

Ogni richiesta attraversa dunque l'intero percorso:

$$A \rightarrow B \rightarrow A \rightarrow P \rightarrow A \rightarrow \text{Sink}$$

La modellazione multi-classe permette di descrivere le tre funzioni applicative del Server A mantenendo un workflow semplice e leggibile.

Il modello trascura esplicitamente:

- i ritardi di rete verso Server P, considerati trascurabili rispetto ai tempi di servizio;
- i tempi di riflessione degli utenti (**think times**), fortemente variabili e non significativi ai fini del carico massimo;
- eventuali fallimenti o rifiuti di servizio: ogni richiesta percorre sempre l'intera sequenza.

Queste assunzioni permettono di simulare uno scenario di carico massimo, focalizzandosi sulle prestazioni dei server e sull'identificazione del collo di bottiglia.

5 Modello di Specifica

In questo capitolo il modello concettuale viene tradotto in un **modello di specifica**, definendo parametri numerici e distribuzioni stocastiche necessarie al simulatore a eventi discreti.

Tutti i parametri sono stati ricavati dal caso di studio di Serazzi [1, Cap. 6] e completati tramite assunzioni standard per sistemi web multi-tier.

5.1 Tasso di Arrivo delle Richieste

Gli arrivi sono modellati come un processo di Poisson.
La **Tabella 1** riassume i tassi utilizzati nei tre scenari. La distribuzione esponenziale è stata adottata in quanto ipotesi largamente utilizzata nella modellazione del traffico web.

Scenario	λ (richieste/ora)	λ (richieste/secondo)
Light workload	4320	1.2
SCA workload	4320	1.2
Heavy workload	5040	1.4

Tabella 1 Tassi di arrivo delle richieste nei tre scenari

5.2 Tempi di Servizio

Tutti i nodi sono modellati come code **M/M/1–PS**, quindi i tempi di servizio sono esponenziali. I valori medi corrispondono ai **Service Demands** forniti dal caso di studio.

La **Figura 4** mostra i Service Demands per Server A, B e P nei due scenari:

- **autenticazione a un fattore** (configurazione attuale),
- **autenticazione a due fattori (SCA)**.

Questi valori rappresentano il servizio necessario per ciascuna visita dei tre blocchi.

5.3 Probabilità di Routing

Essendo il workflow deterministico, tutte le probabilità di routing sono pari a 1, come segue:

- **Uscite dal Server A**
  - Classe 1 → Server B:
    - $p_{A,1,B} = 1$

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

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

Fig. 4 *Richieste di servizio* [s] ai server per il carico di lavoro attuale con autenticazione a un fattore (sinistra) e con autenticazione a due fattori (destra) per la sicurezza dei pagamenti

    - Classe 2 → Server P:
        - $p_{A,2,P} = 1$
    - Classe 3 → uscita dal sistema:
        - $p_{A,3,exit} = 1$
- **Uscite dal Server B e Server P**
    - Server B → CS:
        - $p_{B,1,CS} = 1$
    - Server P → CS:
        - $p_{P,1,CS} = 1$
- **Uscita dalla stazione CS**
    - Classe 1 → Class2 → Server A:
        - $p_{CS,1,A} = 1$
    - Classe 2 → Class3 → Server A:
        - $p_{CS,2,A} = 1$

Questa definizione garantisce un flusso completamente deterministico, conforme alle specifiche del workflow applicativo.

6 Modello Computazionale

Il modello di specifica è stato tradotto in un **modello computazionale** mediante una simulazione a eventi discreti basata sull'approccio **Next-Event Simulation**, in cui il tempo avanza processando di volta in volta l'evento successivo nella lista degli eventi futuri.
Per l'implementazione è stato utilizzato **Python**, scelto come linguaggio di programmazione generico per la sua semplicità, flessibilità e facilità di gestione delle strutture dati necessarie alla simulazione.

Il codice è disponibile nel **repository GitHub** del progetto [2]. La struttura del repository è organizzata in modo da facilitare la consultazione e l'esecuzione della simulazione:

- nella cartella *src* si trovano il codice principale della simulazione (**simulator.py**), le classi principali del modello (**entities.py**) e le diverse funzioni ausiliarie utilizzate per l'implementazione (**utils.py**);
- in **sim_config.py** è possibile modificare i parametri di input e di configurazione del sistema;
- **run_simulation.py** fornisce un'interfaccia a menu per avviare la simulazione (ad ***orizzonte finito*** o ad ***orizzonte infinito***).

Nella realizzazione del modello è stata utilizzata la libreria fornita dagli autori del libro *Discrete Event Simulation: A First Course* [3], [4], reperibile sul sito del **Prof. Larry Leemis** [5].

6.1 Strutture Dati

Descriviamo le principali strutture dati utilizzate nel nostro software.

- **Classe *Clock***
  Rappresenta l'orologio della simulazione.
  - *current*: tempo attuale di simulazione.
  - *arrival*: tempo del prossimo arrivo da processare.
  - *next*: tempo del prossimo evento da processare.

- **Classe *Job***
  Rappresenta una singola richiesta nel sistema.
  - *current_class*: class ID attuale del job.
  - *server*: server attuale in cui si trova il job.
  - *remaining*: tempo di servizio rimanente nel server.

- **Classe *CompletionEvent***
  Rappresenta un evento di completamento.
  - *time*: istante in cui l'evento avviene.
  - *payload*: dati aggiuntivi.

- **Classe *PSServer***
  Rappresenta un server a condivisione di processore (Processor-Sharing).
  - *jobs*: lista dei job attualmente nel server.
  - *version*: versione del server.
  - I metodi principali consentono di aggiornare il progresso dei job, gestire arrivi, completamenti e calcolare il tempo del prossimo evento di partenza.

- ***Heap compl_q* (Event Queue delle Completions)**
  Mantiene tutti gli eventi di completamento ordinati per tempo di occorrenza. In questo modo, per ottenere il prossimo completamento è sufficiente accedere a **compl_q[0]**.
  - La struttura è una heap binaria (**heapq**), che permette inserimenti e estrazioni del prossimo evento in **O(log n)**.
  - *heapq.heappush()* inserisce un evento mantenendo l'ordinamento.
  - *heapq.heappop()* rimuove l'evento con tempo minimo (prossimo completamento).

6.2 Gestione degli Eventi

Le tipologie di evento gestite sono due:

- **arrival**: arrivo di un nuovo job nel sistema;
- **departure**: completamento del servizio di un job su un server PS.

La logica generale della simulazione è la seguente:

1. Il clock di simulazione avanza all'istante del prossimo evento *clock.next*, che è il minimo tra:
   - il tempo del prossimo completamento **compl_q[0].time**,
   - il tempo del prossimo arrivo **clock.arrival**.
2. Tutti i server aggiornano lo stato del servizio tramite **update_progress()**.
3. In base al tipo di evento:
   - se **clock.current == clock.arrival**, viene gestito un **arrivo** tramite la funzione *handle_arrival()*;

- altrimenti viene gestito un **completamento** tramite *handle_departure()*.

4. Dopo ogni evento, *clock.next* viene aggiornato all'istante del prossimo evento.

6.2.1 Arrivo

Ogni arrivo viene generato tramite le funzioni di sampling (*interarrival_time()* per gli arrivi esterni, che vengono gestiti dal Server A, ed *exp_sample()* per i servizi), che sfruttano i generatori di numeri casuali presenti nella libreria di riferimento utilizzata [4], in particolare la funzione *Exponential()* di **rvgs.py**.

Quando un arrivo viene processato:

1. **Schedulazione dell'arrivo successivo**
   Nel caso di arrivi dall'esterno, viene generato il prossimo tempo di arrivo **t_next_arr** e memorizzato nel clock tramite *clock.update_arrival(t_next_arr)*.

2. **Inserimento del job nel server**
   Il job viene registrato nel server tramite *process_arrival()*, che:
   - assegna il tempo di servizio rimanente (*job.remaining = service_time*);
   - aggiunge il job alla lista *jobs*;
   - aggiorna le statistiche locali.

3. **Schedulazione del prossimo completamento**
   Dopo l'inserimento, viene calcolato il nuovo istante del prossimo completamento possibile nel server tramite *server.next_departure_time()* e viene inserito questo nuovo evento di **departure** nella heap degli eventi di completamento.
   Questo evento include la **versione del server**, necessaria per evitare il processamento di eventi obsoleti.

6.2.2 Completamento

Quando viene estratto un evento di completamento, prima di essere processato è necessario verificare che sia **ancora valido**.
Questo avviene confrontando:

- *event.payload["version"]*
- la versione corrente del server *server.version*

Se le versioni non coincidono, significa che la struttura del server è cambiata dopo la schedulazione dell'evento (ad esempio a causa di ulteriori arrivi al server oppure job che lasciano il server: tutti eventi che modificano i tempi di completamento delle richieste) e quindi viene scartato.

Se invece il completamento è valido, il processo prosegue:

1. **Identificazione del job da completare**
   Nel modello PS, il job con il tempo di servizio rimanente minimo è quello che completa. Il server lo individua tramite *_job_to_complete()* e lo rimuove tramite *_remove_job()*.

2. **Ricalcolo del prossimo completamento del server**
   Se nel server sono ancora presenti job, viene calcolato il prossimo istante di completamento e inserito un nuovo evento *departure* nella heap.

3. **Routing e aggiornamento della classe**
   Viene determinato il nodo successivo nel workflow del job tramite *next_node_after()*:

   - Se la destinazione è il Class-Switch (**CS**), l'attributo *current_class* del job viene aggiornato tramite *do_class_switch()*, e viene immediatamente processato (allo stesso istante), come descritto nel paragrafo **6.2.1**, un nuovo evento di **arrivo** di questo job sul server **A**.
   - Se la destinazione è un altro server (**B** o **P**), viene immediatamente processato un nuovo evento di **arrivo** del job sul server di destinazione come descritto sempre in **6.2.1**.
   - Se la destinazione è **SINK**, il job viene contrassegnato come completato (esce dal sistema).

6.3 Simulazione ad Orizzonte Finito

La simulazione ad **orizzonte finito** è stata implementata per analizzare il comportamento dinamico del sistema di e-commerce lungo una finestra temporale predefinita. Il flusso degli arrivi e l'evoluzione interna del sistema vengono osservati a partire da condizioni iniziali non stazionarie, permettendo di valutare l'intero transiente e la successiva convergenza verso il regime permanente.

La durata della finestra è stata fissata, in ogni scenario di carico considerato, a un intervallo **T** (in secondi) sufficiente a osservare sia la crescita iniziale delle code sia il raggiungimento (o meno) dello stato stazionario.

Il sistema è inizialmente **idle**, ovvero privo di richieste in servizio. Per ottenere statistiche affidabili nel dominio del tempo, la simulazione è stata replicata **128 volte**, generando così un ensemble di dimensione 128. Ogni replica costituisce un'osservazione indipendente del processo stocastico e consente di stimare correttamente la media delle variabili prestazionali. Come da linee guida [3, Sec. 8.3], le repliche condividono l'inizializzazione dei seed tramite **plantSeeds()** effettuata *prima* del ciclo di replicazione, mentre per ogni replica successiva alla prima vengono utilizzati come stati iniziali degli RNG gli stati finali della replica precedente, evitando sovrapposizioni indesiderate tra gli stream utilizzati nelle diverse runs.

La condizione di terminazione della simulazione è il raggiungimento del tempo t = T. A ogni replica, le principali metriche di interesse — in particolare **tempo di risposta**, **numero di job in esecuzione** e **utilizzazioni dei server** — vengono campionate a intervalli regolari di **300 secondi (5 minuti)** e registrate in file DAT.
Pertanto, per ogni istante di osservazione si ottengono 128 misure della stessa statistica.

Nel contesto di questo progetto, l'analisi ad orizzonte finito ha, quindi, i seguenti obiettivi principali:

- **Analizzare la fase di avviamento del sistema**, osservando la (eventuale) velocità di convergenza verso il regime stazionario;

- **Studiare l'evoluzione temporale del tempo di risposta e delle utilizzazioni dei server**, con particolare attenzione al comportamento del Database Server (Server B), atteso come possibile collo di bottiglia;
- **Valutare l'impatto dell'introduzione del protocollo SCA nella fase transitoria**, verificando se il degrado prestazionale emerge immediatamente;
- **Confrontare il comportamento del sistema nei tre scenari di carico** (Light, SCA e Heavy workload) in condizioni dinamiche realistiche.

6.4 Simulazione ad Orizzonte Infinito

Nella simulazione ad **orizzonte infinito** il sistema viene simulato per un tempo di simulazione sufficientemente lungo da poter essere considerato "infinito" rispetto alla dinamica del sistema reale. In questo modo è possibile ottenere le **statistiche a regime stazionario**, ovvero indipendenti dalle condizioni iniziali.

In ciascuna simulazione si assume che il sistema sia **statico**, ovvero che il **tasso di arrivo λ rimanga costante** per tutta la durata della run. Tale ipotesi vale per tutti e tre gli scenari di carico analizzati, ognuno dei quali rappresenta una diversa configurazione operativa del sistema.

Per ridurre l'influenza delle condizioni iniziali, la simulazione viene avviata con tutte le code vuote, ma la raccolta delle statistiche avviene solo dopo un periodo sufficientemente lungo, attraverso l'utilizzo del **metodo delle Batch Means**:

- Per stimare la media campionaria del tempo di risposta e delle altre metriche prestazionali, la run di simulazione viene suddivisa in **k batch**, ciascuno contenente **b job completati**.
- Da ogni batch vengono estratte le statistiche di interesse tramite la funzione *compute_metrics_infinite()*.
- Al termine di ogni batch:
    - le **statistiche dei server vengono azzerate** tramite *reset_statistics()*;
    - lo **stato del sistema non viene resettato**, quindi i job ancora presenti nelle code proseguono nel batch successivo.

In questo modo si ottiene un campione di k osservazioni statisticamente indipendenti, sul quale è possibile calcolare la media campionaria e l'intervallo di confidenza al 95% (tramite, ad esempio, il programma **estimate.py** della libreria [4], che internamente utilizza la distribuzione *Student*).

Le scelte dei parametri **b** e **k** influenzano la qualità del campione:

- un valore elevato di **b** riduce l'autocorrelazione tra i batch consecutivi;
- un valore elevato di **k** migliora la precisione dell'intervallo di confidenza.

Nel presente studio sono stati utilizzati **k = 128 batch**. Seguendo le linee guida proposte da *Banks*, *Carson*, *Nelson* e *Nicol* [3, Sec. 8.4], è stato individuato, per ogni scenario, il valore di **b** tale per cui l'autocorrelazione del campione risultasse inferiore a 0.2 al lag j = 1.

A tal fine è stata definita la funzione *find_batch_b()*, che consente di generare campioni statistici del tempo di risposta medio del sistema al variare di diversi valori di **b** (un file DAT per ogni **b**). Per ciascun campione/file è stata quindi valutata l'autocorrelazione mediante il programma **acs.py** della libreria [4].

Dai risultati ottenuti si è osservato che:

- nello scenario *Light*, per **b = 8192** l'autocorrelazione al primo lag risulta pari a **0.107**;
- nello scenario *SCA*, per **b = 8192** l'autocorrelazione al primo lag risulta pari a **0.131**;
- nello scenario *Heavy*, per **b = 32768** l'autocorrelazione al primo lag risulta pari a **0.065**.

Pertanto, **b = 8192** è stato scelto come valore definitivo per la dimensione dei batch nei primi due scenari, mentre nel terzo è stato scelto **b = 32768**.
Di conseguenza, per ciascuna simulazione ad orizzonte infinito vengono complessivamente processati **b · k = 1048576 job completati** nei primi due scenari e **b · k = 4194304 job completati** nel terzo scenario.

Gli obiettivi principali della simulazione ad orizzonte infinito, nel contesto del presente progetto, sono quindi i seguenti:

- **Analisi dei tempi di risposta a regime stazionario**, per valutare il rispetto del vincolo QoS dei 30 secondi;
- **Analisi delle utilizzazioni dei server A, B e P**, per individuare il collo di bottiglia del sistema nei diversi scenari di carico;
- **Stima del throughput massimo del sistema** e verifica della sostenibilità del carico imposto in ogni scenario;
- **Valutazione dell'impatto dell'autenticazione a due fattori (SCA)** sul degrado prestazionale a regime;
- **Analisi degli effetti del potenziamento del Server B**, verificando il nuovo collo di bottiglia e l'incremento percentuale di throughput ottenibile.

7 Verifica

La fase di **verifica** è finalizzata ad accertare che il modello computazionale implementato sia **coerente con il modello di specifica**, ovvero che tutti i meccanismi stocastici, di servizio, di routing e di sincronizzazione siano stati correttamente tradotti nel simulatore a eventi discreti.

In particolare, la verifica è stata condotta con riferimento ai seguenti aspetti fondamentali del modello:

- il **processo di generazione degli arrivi**;
- la **generazione dei tempi di servizio**;
- la **corretta implementazione della disciplina Processor Sharing**;
- la **correttezza del workflow del sistema e del meccanismo di cambio classe (Class Switch)**.

Le verifiche sono state condotte mediante l'esecuzione di simulazioni sia **ad orizzonte finito che ad**

orizzonte infinito, affiancate da **test deterministici mirati** sui principali componenti logici del modello, al fine di validarne il comportamento in modo isolato.

7.1 Processo di Arrivo

Per verificare il corretto funzionamento del generatore di arrivi, abbiamo effettuato una simulazione a **orizzonte finito** come descritto in **6.3**, con **T = 3600 (un'ora)**. In ciascuna replica della simulazione, abbiamo tracciato il numero totale di richieste arrivate al sistema, sfruttando la variabile **total_system_arrivals** aggiornata all'interno della funzione *simulate_finite* ogni volta che viene chiamata *handle_arrival*.

Al termine di ogni replica:

1. Il numero totale di arrivi è salvato in un file .dat tramite la funzione *save_finite_total_arrivals*.
2. Successivamente, calcoliamo la **media** e l'**intervallo di confidenza al 95%** sui 128 valori raccolti (tramite il programma **estimate.py**).

Questo ci permette di confrontare, in ogni scenario, il numero osservato di arrivi con quello atteso, basato sul **tasso di arrivo specificato (ARRIVAL_RATE)**.
I risultati di questo processo sono mostrati nella **Tabella 2**. Per lo scenario *SCA* è stato utilizzato un seed pari a **234567891**, diverso dal seed utilizzato negli altri due scenari, ovvero **123456789**: questo è stato fatto per distinguere il risultato dal primo caso (*Light*) e per avere un'ulteriore conferma della correttezza del nostro modello.

La verifica ha confermato che, su scala oraria, il numero medio di arrivi osservati è coerente con il tasso teorico, garantendo così la correttezza del processo di generazione degli arrivi.

7.2 Tempi di Elaborazione

I tempi di servizio sono generati tramite la funzione *exp_sample()*, che campiona una variabile casuale

---

Scenario	Numero Atteso di Arrivi all'ora	Numero Osservato di Arrivi in un'ora (α = 0.05)
Light	4320	4316.945312 +/- 10.851996
SCA	4320	4325.140625 +/- 11.858194
Heavy	5040	5036.296875 +/- 12.166733

Tabella 2 Confronto tra il numero di richieste attese in un'ora e il numero effettivamente osservato nelle simulazioni (per ogni scenario)

esponenziale con media pari al Service Demand specificato per ciascun server e per ciascuna classe, in ogni scenario. La loro verifica è stata condotta mediante **simulazioni ad orizzonte infinito**, secondo il metodo delle **Batch Means**, come descritto nella Sezione 6.4.
I risultati delle simulazioni, confrontati con quelli attesi, sono riportati in **Tabella 3**. Da notare che è stata considerata la domanda totale per ogni server (come si può vedere per il Server A) e non sono stati calcolati i tempi relativi allo scenario di carico "Heavy con Server B NON potenziato", perché, come vedremo più avanti, il sistema, in questo caso, risulta instabile (e inoltre i tempi sarebbero uguali a quelli del *Light Workload*, perché le varie domande sono uguali).

Il confronto tra i tempi medi di servizio stimati tramite simulazione e i valori teorici impostati nei

---

LIGHT WORKLOAD		
Server	**Tempo di Servizio Medio Atteso [s]**	**Tempo di servizio Medio Osservato [s] ($\alpha = 0.05$)**
A	0.7	0.700753 +/- 0.001009
B	0.8	0.799627 +/- 0.001573
P	0.4	0.399290 +/- 0.000726
SCA WORKLOAD		
A	0.75	0.750349 +/- 0.000963
B	0.8	0.799627 +/- 0.001578
P	0.7	0.698759 +/- 0.001279
HEAVY WORKLOAD (CON SERVER B POTENZIATO)		
A	0.7	0.700135 +/- 0.000431
B	0.4	0.400102 +/- 0.000366
P	0.4	0.399637 +/- 0.000375

Tabella 3 Confronto tra i tempi di servizio medi attesi e i tempi di servizio medi osservati per ciascun server nei tre scenari di carico: *Light*, *SCA* e *Heavy* (con server B potenziato)

Service Demands ha mostrato una piena coerenza entro l'errore statistico, confermando la **corretta implementazione delle distribuzioni esponenziali dei tempi di servizio** per tutti i server del sistema.

7.3 Disciplina Processor Sharing

La correttezza della disciplina di servizio **Processor Sharing** è stata verificata mediante un **test deterministico controllato** (vedi *test/ps_test.py* nel repository [2]), privo di componenti stocastiche, al fine di validare separatamente la logica di avanzamento del servizio e il calcolo dei completamenti. La verifica è stata articolata in tre casi significativi. Nel primo caso è stato inserito un solo job nel server, osservando che il tempo residuo di servizio decresce linearmente con pendenza unitaria, come nel caso di un server dedicato. Nel secondo caso sono stati inseriti due job simultaneamente nello stesso server, verificando che ciascun job riceve metà della capacità di servizio e che il completamento avviene in base al minimo tempo residuo pesato per il numero di job presenti. Nel terzo caso sono stati considerati tre job con istanti di arrivo differenti, al fine di verificare il corretto aggiornamento del progresso dei job al variare dinamico del numero di richieste nel server.

Per ciascun caso sono stati confrontati i **tempi residui** dei vari job, l'**istante del prossimo completamento** nel server e il **job selezionato per il completamento** con i valori teorici attesi. In tutti gli scenari analizzati i risultati prodotti dal simulatore sono risultati pienamente coerenti con il comportamento teorico atteso, confermando la **corretta implementazione della disciplina PS** in tutti i casi analizzati.

In **Figura 5** sono riportati i risultati del test nel terzo caso (3 job in esecuzione sullo stesso server).

```
>>> CASO 3: 3 job nel server P
-   Primo job (ID = 3), con tempo di servizio pari a 7.6 secondi, arriva all'istante 0.0 (s)
-   Secondo job (ID = 4) con tempo di servizio pari a 8.0 secondi, arriva all'istante 0.8 (s)
-   Terzo job (ID = 5) con tempo di servizio pari a 6.5 secondi, arriva all'istante 1.3 (s)

[Avanziamo nel tempo di 13 secondi]

--> Valori attesi:
-   Remaining (Job 1, Job 2, Job 3) = (2.65, 3.85, 2.6)
-   Next departure time = 20.8
-   Job to complete = 5

--> Risultati:
-   Remaining (Job 1, Job 2, Job 3): (2.65, 3.85, 2.6)
-   Next departure time at: 20.8
-   Job to complete: 5
```

Fig. 5 Risultati del test di verifica della disciplina *Processor Sharing* nel caso di tre job in esecuzione contemporaneamente sullo stesso server

7.4 Workflow e Class Switch

La correttezza del **workflow deterministico** del sistema e del **meccanismo di cambio classe (Class Switch)** è stata verificata mediante una **simulazione controllata a parametri deterministici**, attivabile tramite la variabile *PLOT_VISITS*.

In tale modalità, gli **intervalli di arrivo** sono fissati costanti a 3 s e anche i **tempi di servizio** assumono valori deterministici, così da eliminare ogni componente stocastica e rendere tracciabile in modo esatto l'evoluzione di ciascuna richiesta. In particolare, il tempo di servizio di ciascuna fase viene posto **uguale al valore medio del Service Demand previsto dallo scenario di simulazione considerato**.
Il routing deterministico $A \rightarrow B \rightarrow A \rightarrow P \rightarrow A$ e il meccanismo di avanzamento delle classi sono quindi verificati tracciando la **history temporale di ciascun job** e visualizzando la relativa **sequenza delle visite ai server** tramite il diagramma temporale.

Il diagramma riportato in **Figura 6** mostra in modo esplicito (nel caso dello scenario *Light*) che ogni richiesta attraversa correttamente tutte le fasi previste dal modello, confermando la **corretta implementazione sia del workflow che del Class Switch**.

8 Validazione

La fase di validazione ha lo scopo di accertare che il modello computazionale sia in grado di riprodurre in modo affidabile il comportamento prestazionale atteso del sistema reale, almeno entro i

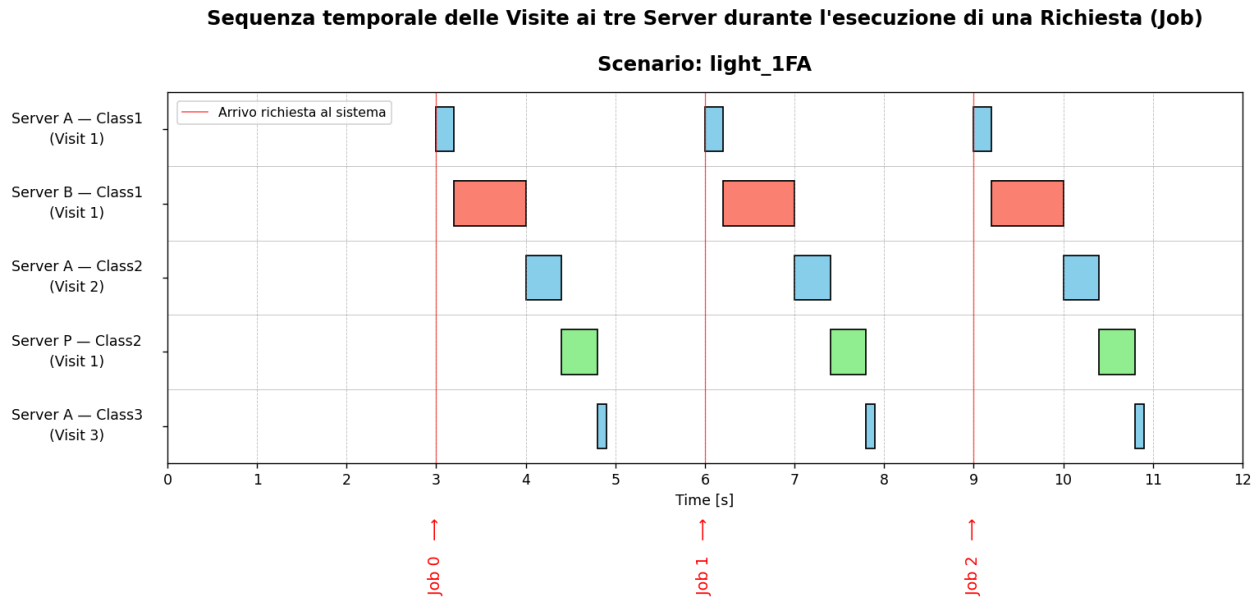

Fig. 6 Diagramma temporale delle visite ai tre server durante l'esecuzione di una richiesta

limiti delle assunzioni modellistiche adottate.

Poiché non era disponibile un dataset di misure reali provenienti da un sistema di e-commerce operativo, la validazione è stata condotta mediante **confronto sistematico con il modello analitico**, sfruttando le leggi teoriche delle reti di code aperte M/M/1 con disciplina Processor Sharing [6].
In particolare, sono state effettuate **simulazioni ad orizzonte infinito**, al fine di confrontare i risultati ottenuti dal modello computazionale con quelli previsti dal modello analitico di riferimento.

Per ottimizzare questa fase, è stato sviluppato un file Excel, disponibile nel repository [7], in cui sono espresse le leggi teoriche, in modo che, modificando i parametri di input e di configurazione, vengano automaticamente calcolati tutti i risultati attesi.
Questo ha permesso di avere, durante ogni fase di simulazione, un immediato riscontro teorico, potendo quindi intervenire prontamente quando sono state rilevate delle imprecisioni.

Le leggi teoriche controllate sono descritte in **Tabella 4**, dove:

- $i \in \{A, B, P\}$;
- λ_i è il tasso di arrivo al Server i;
- $E(S_i)$ è il tempo medio di servizio (totale) richiesto al Server i se questo avesse l'intera capacità disponibile.

Essendo il workflow delle richieste deterministico, abbiamo che:

Legge	Significato
$\rho_i = \lambda_i * E(S_i)$	Utilizzazione media del Server i
$E(T_i) = \frac{E(S_i)}{1-\rho_i}$	Tempo medio di risposta del Server i
$E(N_i) = \frac{\rho_i}{1-\rho_i}$	Numero medio di richieste nel Server i
$E(T_{TOT}) = \sum_{i \in \{A,B,P\}} E(T_i)$	Tempo medio di risposta complessivo del sistema
$E(N_{TOT}) = \sum_{i \in \{A,B,P\}} E(N_i)$	Numero medio di richieste nel sistema
$\lambda_{MAX} = \frac{1}{\max_i E(S_i)}$	Throughput massimo del sistema

Tabella 4 Leggi teoriche controllate durante la fase di *Validazione*

$$E(S_i) = \sum_{j=1}^{3} E(S_{ij})$$

dove $E(S_{ij})$ è il tempo medio di servizio richiesto al Server i da un job di classe j, con $j = 1, 2, 3$.
In particolare, quindi:

- $E(S_A) = E(S_{A1}) + E(S_{A2}) + E(S_{A3})$;
- $E(S_B) = E(S_{B1})$;
- $E(S_P) = E(S_{P2})$.

Inoltre, se indichiamo con λ il tasso di arrivo esterno, si ha che $\lambda_A = \lambda_B = \lambda_P = \lambda$.

A titolo di esempio, in **Tabella 5** sono riportati i risultati della simulazione ad orizzonte infinito nel caso dello scenario *Light* e il confronto con i relativi risultati analitici.
Dai numeri ottenuti possiamo allora vedere come il modello simulativo rispetti le leggi del modello analitico: tutti i valori sono molto vicini ai valori analitici, i quali sono sempre contenuti negli intervalli di confidenza (solo il valore di $E(T_P)$ risulta esterno all'intervallo, ma lo scarto è inferiore a 2×10^{-4}, una differenza assolutamente trascurabile e compatibile con l'errore stocastico della simulazione).

In conclusione, possiamo confermare che, anche non avendo dati reali a disposizione, il nostro modello rappresenta bene il sistema reale, e possiamo quindi proseguire con la fase finale: quella di **analisi dei risultati**.

9 Progettazione ed Esecuzione degli Esperimenti

L'approccio adottato nella simulazione prevede l'esecuzione di simulazioni distinte (sia ad orizzonte finito che ad orizzonte infinito), ciascuna caratterizzata da una specifica configurazione dei parametri di input, in particolare:

- il **tasso di arrivo delle richieste** (ARRIVAL_RATE);
- i **Service Demands dei server** (SERVICE_DEMANDS);
- l'eventuale **introduzione dell'autenticazione a due fattori (SCA)**;
- l'eventuale **potenziamento del Server B**.

Tali parametri sono contenuti e gestiti centralmente nel file **sim_config.py**, che svolge il ruolo di **struttura di configurazione degli esperimenti**.
Attraverso la variabile SCENARIO, è possibile selezionare una delle seguenti configurazioni operative:

- **"light_1FA"**: carico Light con autenticazione a un fattore;
- **"light_2FA"**: carico Light con autenticazione a due fattori (SCA);
- **"heavy_1FA"**: carico Heavy con infrastruttura invariata;

LIGHT WORKLOAD			
Componente	**Statistica**	**Risultato Analitico**	**Risultato Sperimentale (α = 0.05)**
Server A	ρ_A	0.84	0.841675 +/- 0.001978
	$E(T_A)$	4.375	4.392341 +/- 0.079006
	$E(N_A)$	5.25	5.279162 +/- 0.101607
Server B	ρ_B	0.96	0.960461 +/- 0.002670
	$E(T_B)$	20	20.218496 +/- 2.333550
	$E(N_B)$	24	24.366349 +/- 2.877108
Server P	ρ_P	0.48	0.479590 +/- 0.001312
	$E(T_P)$	0.769231	0.765374 +/- 0.003687
	$E(N_P)$	0.923077	0.919405 +/- 0.005429
Sistema Complessivo	$E(T_{TOT})$	25.144231	25.376211 +/- 2.347516
	$E(N_{TOT})$	30.173077	30.564916 +/- 2.899990
	λ_{MAX}	1.25	1.250741 +/- 0.002469

Tabella 5 Confronto tra i risultati del modello computazionale e i risultati del modello analitico di riferimento (nel caso dello scenario *Light*)

- **"heavy_1FA_newServerB"**: carico Heavy con Server B potenziato (Service Demand dimezzato).

Per ciascuno scenario, il simulatore imposta automaticamente:

- il valore corretto di ARRIVAL_RATE;
- i **Service Demands per classe e per server**, applicando, se necessario, le modifiche dovute al protocollo SCA o al potenziamento del Database Server.

Tutti i risultati delle simulazioni vengono salvati automaticamente nella cartella **results/**, organizzata in due sottodirectory:

- *results/finite/* per le simulazioni ad orizzonte finito;
- *results/infinite/* per le simulazioni ad orizzonte infinito.

Ogni metrica viene memorizzata in un file **.dat** separato, facilmente analizzabile tramite **estimate.py** per ottenere i valori di *media* ed *intervallo di confidenza* (al 95%).
La separazione dei file per metrica e per scenario consente una gestione rigorosa, riproducibile e confrontabile dei risultati sperimentali.

10 Analisi dei Risultati

10.1 Analisi ad Orizzonte Finito

L'analisi ad orizzonte finito consente di:

- verificare la **stabilità dinamica del sistema**;
- individuare precocemente fenomeni di **accumulo delle code**;
- osservare l'**impatto immediato delle modifiche strutturali** introdotte dallo scenario SCA e dal potenziamento del Server B;
- fornire una **base qualitativa e quantitativa** per l'interpretazione dei risultati a regime che verranno discussi nell'analisi ad orizzonte infinito.

Nei paragrafi successivi verranno quindi discussi in dettaglio gli andamenti temporali delle principali metriche per ciascuno scenario considerato, mettendo in evidenza le differenze di comportamento e le criticità emergenti già nella fase transitoria.

10.1.1 Carico Light con Autenticazione a un Fattore

In questa configurazione, simulando il sistema per un totale di **5 ore (T = 3600 * 5)**, vediamo che il **tempo di risposta medio** (vedi Fig.7) mostra una crescita iniziale nella fase di avviamento, seguita da una stabilizzazione su un valore costante nella parte finale. Questo è un segnale che ci indica che il sistema riesce a raggiungere il regime stazionario (dopo circa 10000 secondi, ovvero ~ 3 ore) senza manifestare comportamenti instabili nel periodo osservato.
Dato che in un server Processor Sharing il tempo di risposta dipende dal numero di job in coda, ci aspettiamo allora che anche il **numero medio di richieste presenti nel sistema** mostri, a partire da un certo momento in poi, un andamento costante. Infatti, dalla Fig. 8 possiamo osservare che, dopo una fase iniziale di crescita, questo numero oscilla continuamente tra circa le 25 e le 35 richieste, indicando che il sistema mantiene un livello di occupazione relativamente stabile, con leggere fluttuazioni naturali dovute alla natura stocastica degli arrivi e delle durate di servizio.

Per quanto riguarda le **utilizzazioni dei server** (Fig. 9), dopo una breve fase iniziale di assestamento, i valori di utilizzazione dei Server A, B e P si stabilizzano rapidamente e rimangono pressoché costanti per l'intera durata della simulazione, evidenziando il raggiungimento di un equilibrio dinamico. In particolare, il **Server B opera in condizioni di quasi saturazione**, con un'utilizzazione prossima a 1, mentre i Server A e P (soprattutto quest'ultimo) presentano valori più bassi.

Nel complesso, quindi, lo scenario *light_1FA* mostra che il sistema:

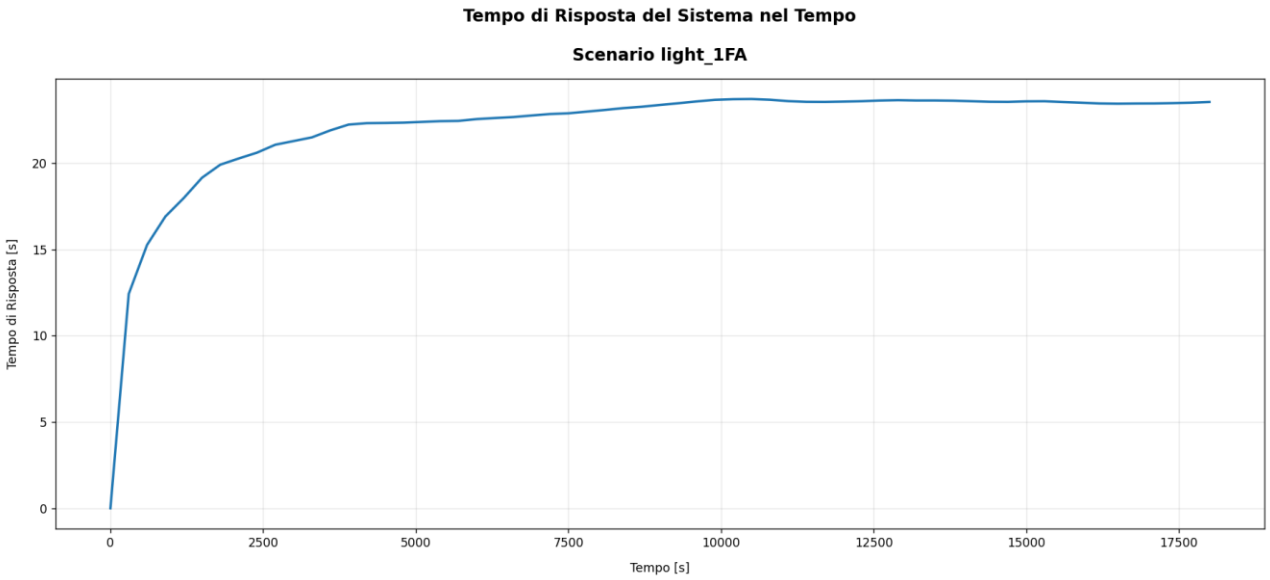

Fig. 7 Andamento temporale del tempo di risposta medio del sistema (Scenario *light_1FA*)

- riesce a **raggiungere il regime stazionario** nel periodo di osservazione;
- non manifesta fenomeni di **instabilità dinamica**;
- opera tuttavia con un **margine di sicurezza ridotto**, a causa della forte sollecitazione del Server B.

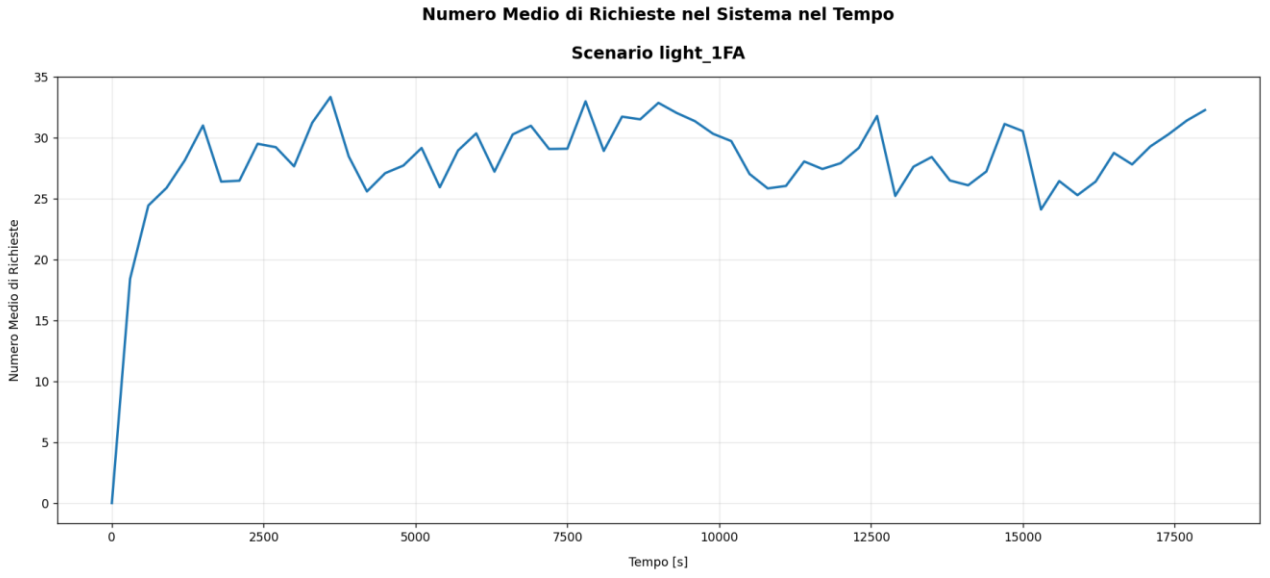

Fig. 8 Andamento temporale del numero medio di richieste nel sistema (Scenario *light_1FA*)

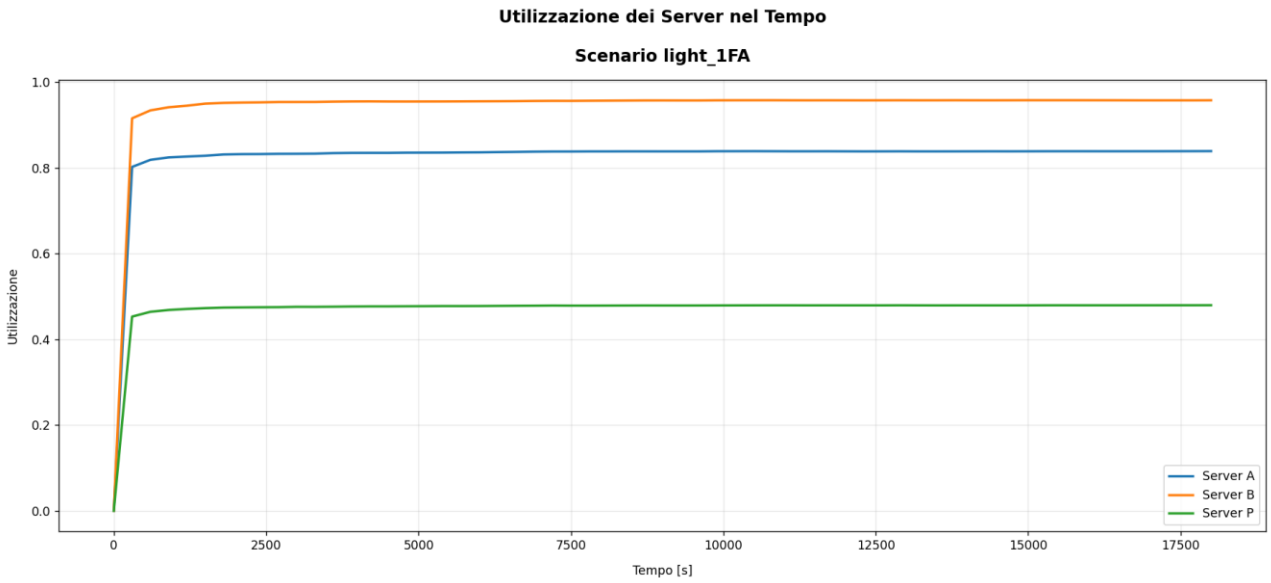

Fig. 9 Andamento temporale delle utilizzazioni dei server (Scenario *light_1FA*)

10.1.2 Carico Light con Autenticazione a Due Fattori

Anche qui, simulando il sistema sempre per 5 ore, otteniamo dei risultati del tutto analoghi allo scenario precedente. Le **Figure 10 e 11** mostrano un confronto temporale, dal punto di vista del tempo di risposta (Fig. 10) e dal punto di vista del numero di richieste in esecuzione (Fig. 11), tra le due configurazioni (***1FA*** e ***2FA***).

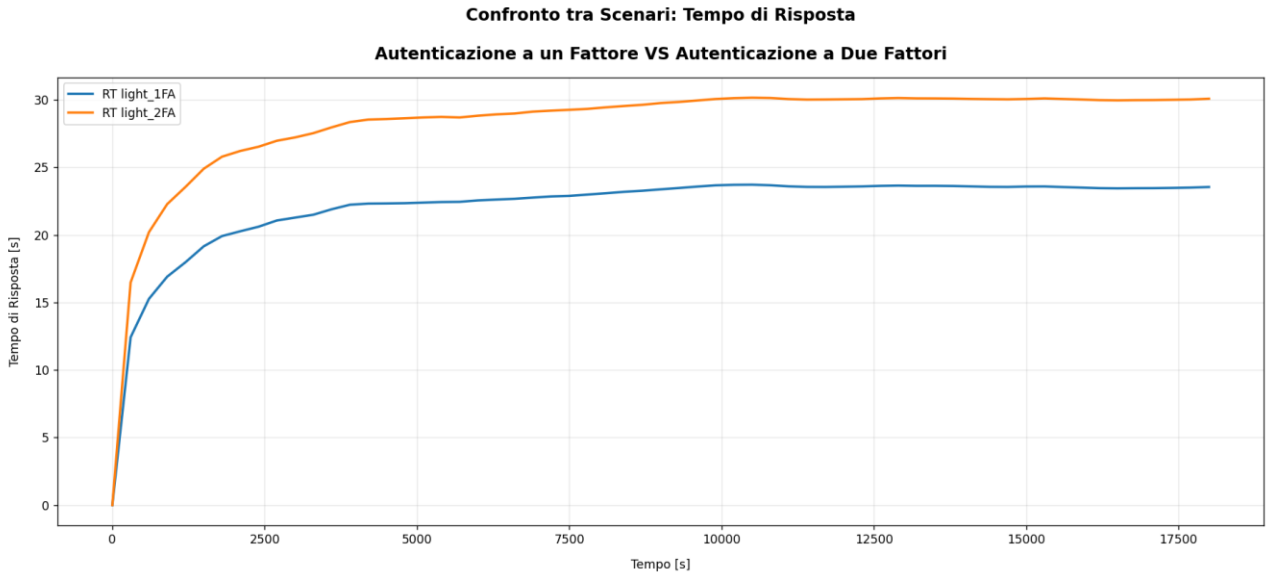

Fig. 10 Andamento temporale del tempo di risposta medio del sistema (*1FA* vs. *2FA*)

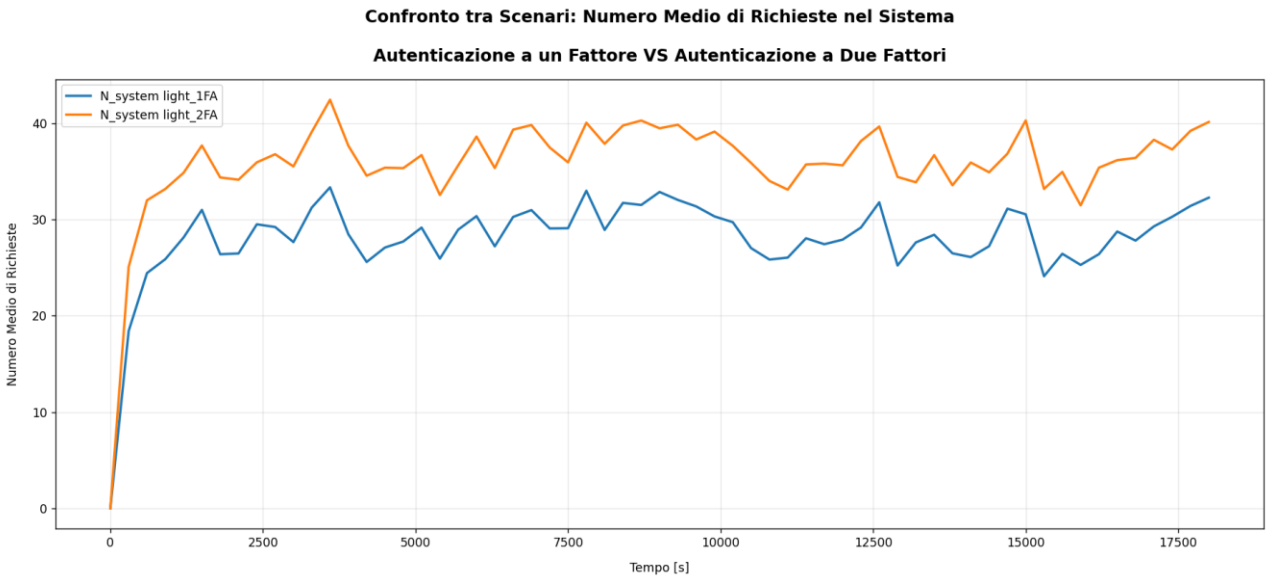

Fig. 11 Andamento temporale del numero medio di richieste nel sistema (*1FA* vs. *2FA*)

Dai risultati ottenuti emerge innanzitutto che **entrambi i sistemi convergono verso il regime stazionario dopo un intervallo temporale pressoché identico** (~ 3 ore). Questo risultato indica che l'introduzione del protocollo SCA non altera in modo significativo la velocità di avviamento del sistema, né la dinamica di convergenza nella fase transitoria.

Tuttavia, in entrambe le rappresentazioni temporali (tempo di risposta e numero medio di richieste nel sistema), la curva relativa allo scenario *2FA* si colloca sistematicamente al di sopra di quella dello scenario *1FA*. Ciò evidenzia che l'aumento dei Service Demand introdotto dall'autenticazione a due fattori produce un **degrado prestazionale immediato**, già visibile nella fase transitoria, pur senza compromettere la stabilità dinamica del sistema.

Per quanto riguarda le utilizzazioni dei server (Fig. 12), il comportamento temporale risulta del tutto analogo a quello osservato nello scenario precedente: dopo una brevissima fase iniziale, le utilizzazioni si stabilizzano rapidamente e rimangono pressoché costanti per tutta la durata della simulazione. Ciò che cambia in modo significativo sono però i livelli medi di utilizzo, che nello scenario *light_2FA* risultano elevati per tutti e tre i server, con valori superiori allo 0.8. Questo indica che l'introduzione del protocollo SCA comporta un **incremento generalizzato del carico su tutta l'infrastruttura**, riducendo sensibilmente il margine operativo disponibile.

10.1.3 Carico Heavy con Infrastruttura Invariata

A differenza degli scenari precedenti, in questo caso la simulazione ad orizzonte finito, condotta su una finestra temporale di **2 ore**, **evidenzia chiaramente l'assenza di convergenza verso il regime stazionario**.

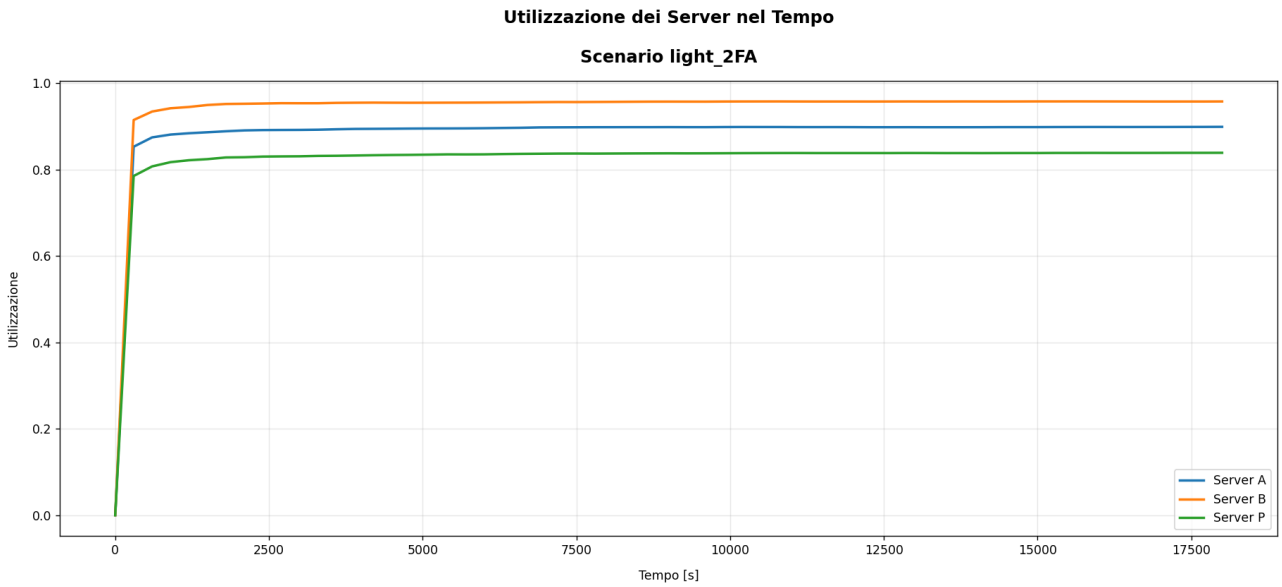

Fig. 12 Andamento temporale delle utilizzazioni dei server (Scenario *light_2FA*)

L'andamento temporale del tempo di risposta medio, riportato in **Figura 13**, mostra una **crescita continua e non limitata nel tempo**, senza alcun segnale di stabilizzazione. Questo comportamento indica che il sistema attuale, sotto carico *heavy*, non è in grado di smaltire il flusso di richieste in ingresso, dando luogo a un progressivo accumulo.

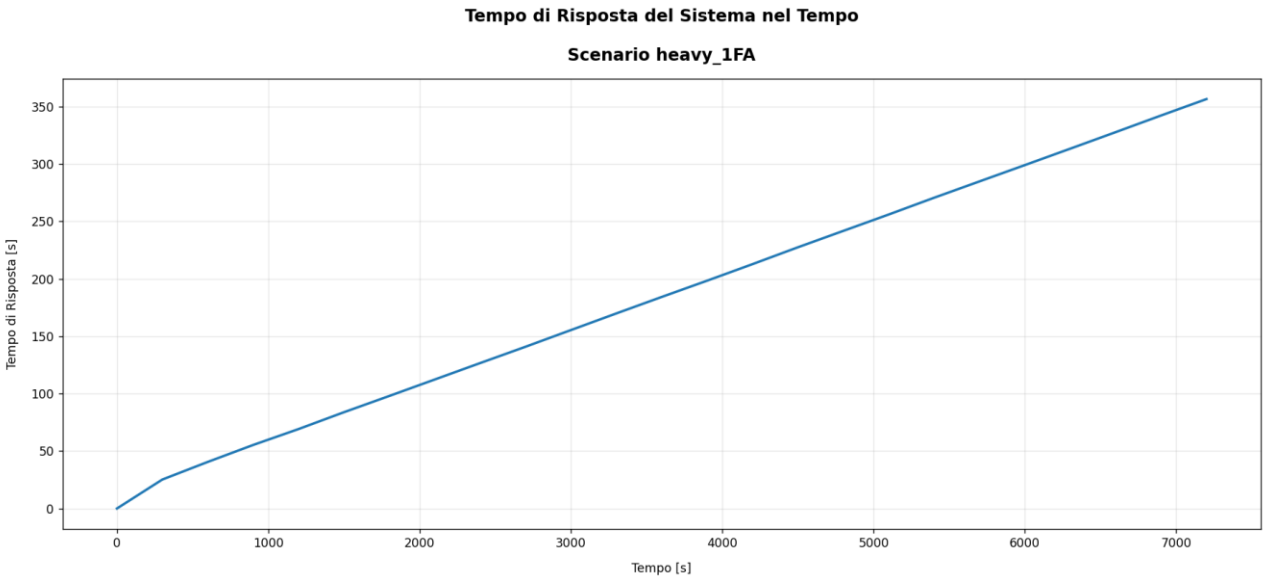

Fig. 13 Andamento temporale del tempo di risposta medio del sistema (Scenario *heavy_1FA*)

La causa del fenomeno emerge chiaramente analizzando l'andamento del numero medio di richieste in ciascun server, riportato in un unico grafico comparativo (Fig. 14). Si osserva che l'incremento indefinito è **interamente imputabile al Server B**, nel quale la coda cresce senza stabilizzarsi.

A ulteriore conferma di questo comportamento, la Fig. 15 mostra che, dopo pochi minuti, l'utilizzazione del Server B segnala una condizione di **saturazione permanente**: il server opera perennemente al massimo della propria capacità, senza possibilità di recupero, generando l'instabilità osservata a livello di sistema.

Poiché, allora, questo scenario non consente il raggiungimento del regime stazionario, non risulta significativo eseguire l'analisi ad orizzonte infinito in questa configurazione. In assenza di condizioni di stabilità, infatti, le grandezze prestazionali medie non convergono a valori finiti e qualsiasi stima a regime risulterebbe priva di significato fisico e statistico.

10.1.4 Carico Heavy con Server B Potenziato

Lo scenario *heavy_1FA_newServerB* rappresenta l'evoluzione dello scenario critico precedente, in cui il Database Server viene sostituito con una versione due volte più veloce.
In questa configurazione, per osservare correttamente il comportamento del sistema, la simulazione ad orizzonte finito è stata condotta su una finestra temporale di 12 ore.

A differenza dello scenario *heavy_1FA*, sia il tempo di risposta medio (Fig. 16) sia il numero medio di richieste presenti nel sistema (Fig. 17) mostrano un andamento analogo a quello osservato nei primi due scenari a carico leggero. Tuttavia, la fase transitoria risulta in questo caso più lunga: la convergenza verso il regime stazionario avviene infatti solo dopo circa 25 000 secondi. Superato tale intervallo, vediamo che il tempo di risposta si comincia a stabilizzare su un valore costante, indicando

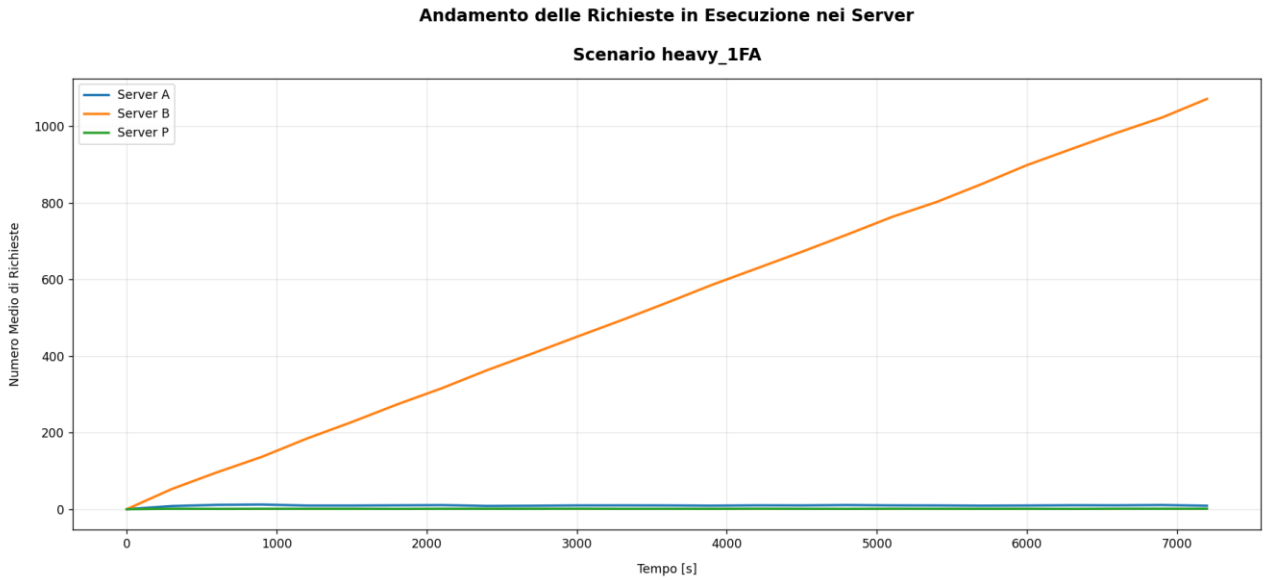

Fig. 14 Andamento temporale del numero medio di richieste nei server (Scenario *heavy_1FA*)

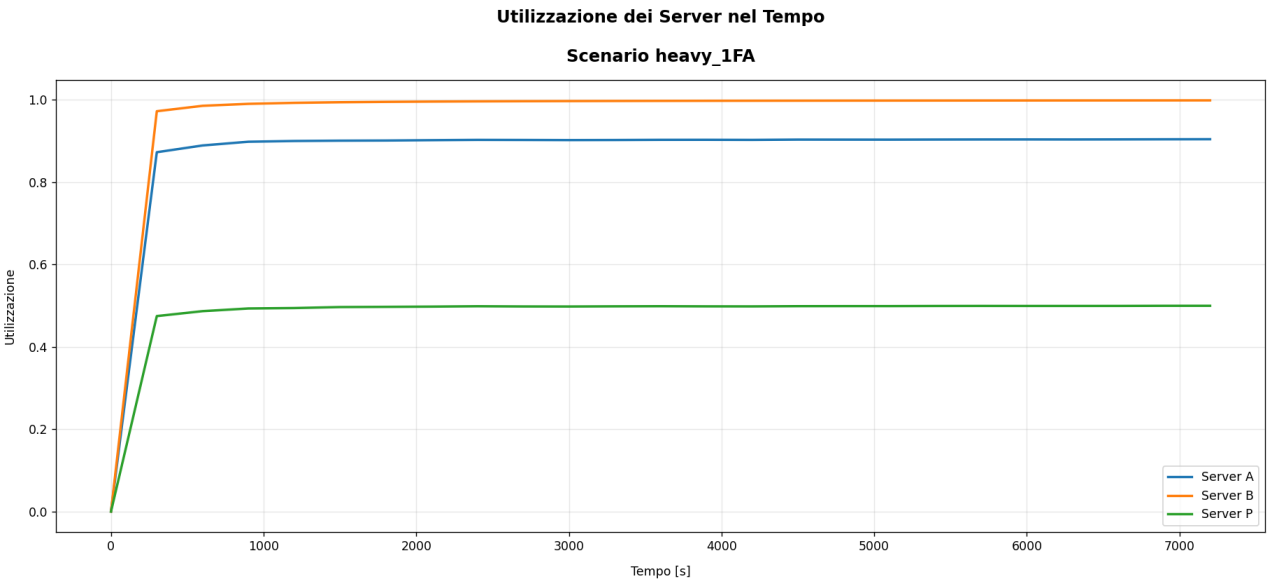

Fig. 15 Andamento temporale delle utilizzazioni dei server (Scenario *heavy_1FA*)

---

il **ripristino della stabilità dinamica del sistema anche in condizioni di carico elevato**.

Per quanto riguarda le utilizzazioni dei server (Fig. 18), si osserva una dinamica significativamente diversa rispetto allo scenario precedente. Dopo poco tempo, le utilizzazioni del Server B e del Server P convergono verso lo stesso valore medio (~ 55%). Il **Server A**, invece, raggiunge quasi immediatamente un'**utilizzazione prossima alla saturazione**, diventando il nuovo componente

---

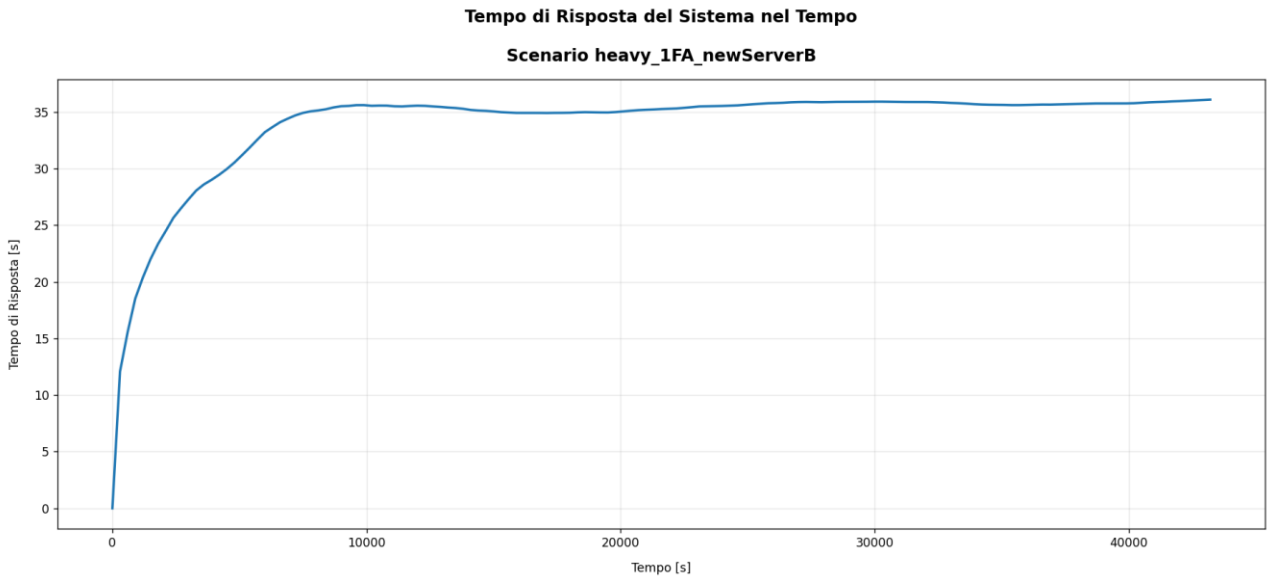

Fig. 16 Andamento temporale del tempo di risposta medio del sistema (Scenario *heavy_1FA_newServerB*)

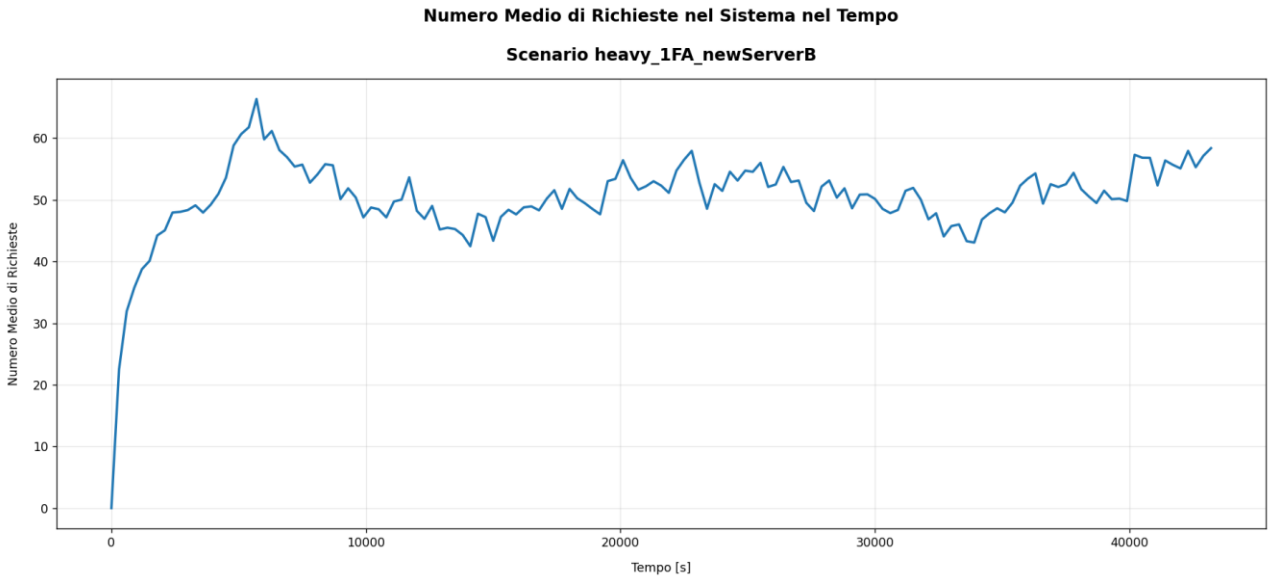

Fig. 17 Andamento temporale del numero medio di richieste nel sistema (Scenario *heavy_1FA_newServerB*)

---

maggiormente sollecitato dell'architettura.

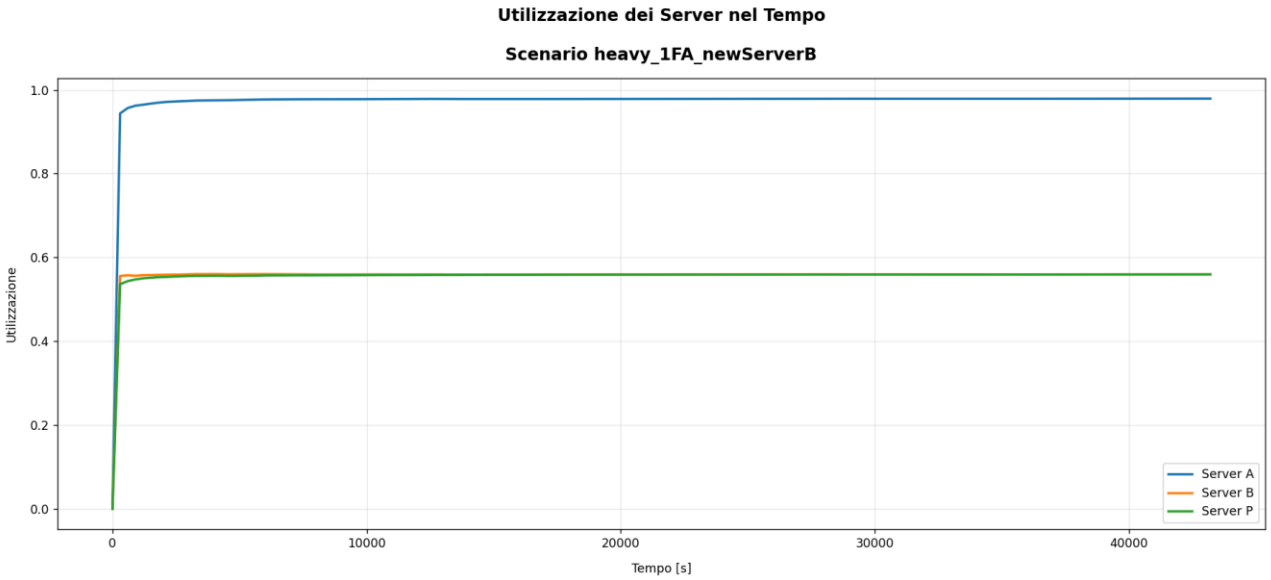

Fig. 18 Andamento temporale delle utilizzazioni dei server (Scenario *heavy_1FA_newServerB*)

10.2 Analisi ad Orizzonte Infinito

L'analisi ad orizzonte infinito ha l'obiettivo di studiare il comportamento del sistema **in regime stazionario**, una volta esauriti gli effetti del transitorio iniziale.

In particolare, ricapitolando, l'analisi si concentra su:

- **tempo di risposta medio a regime**, per la verifica del vincolo di QoS dei 30 secondi;
- **utilizzazioni dei server A, B e P**, per l'individuazione del collo di bottiglia;
- **throughput del sistema**, per verificare la sostenibilità del carico imposto;
- **valutazione del degrado prestazionale dovuto allo SCA**;
- **valutazione dell'efficacia del potenziamento del Server B**.

10.2.1 Sistema Attuale

Dai risultati ottenuti con la simulazione (Tabella 5) emerge che il **tempo di risposta medio a regime** (vedi anche Fig. 19) risulta **inferiore al vincolo di 30 secondi** ($E(T_{TOT}) = 25.376211$ +/- 2.347516), indicando che il sistema, nella configurazione attuale, **è in grado di soddisfare i requisiti di servizio previsti** per il carico operativo di circa 4300 richieste/ora.

Le **utilizzazioni dei server** confermano quanto già osservato nell'analisi transitoria:

- il **Server B** presenta l'utilizzazione più elevata (0.960461 +/- 0.002670),
- i **Server A e P** risultano meno sollecitati (rispettivamente 0.841675 +/- 0.001978 e 0.479590 +/- 0.001312).

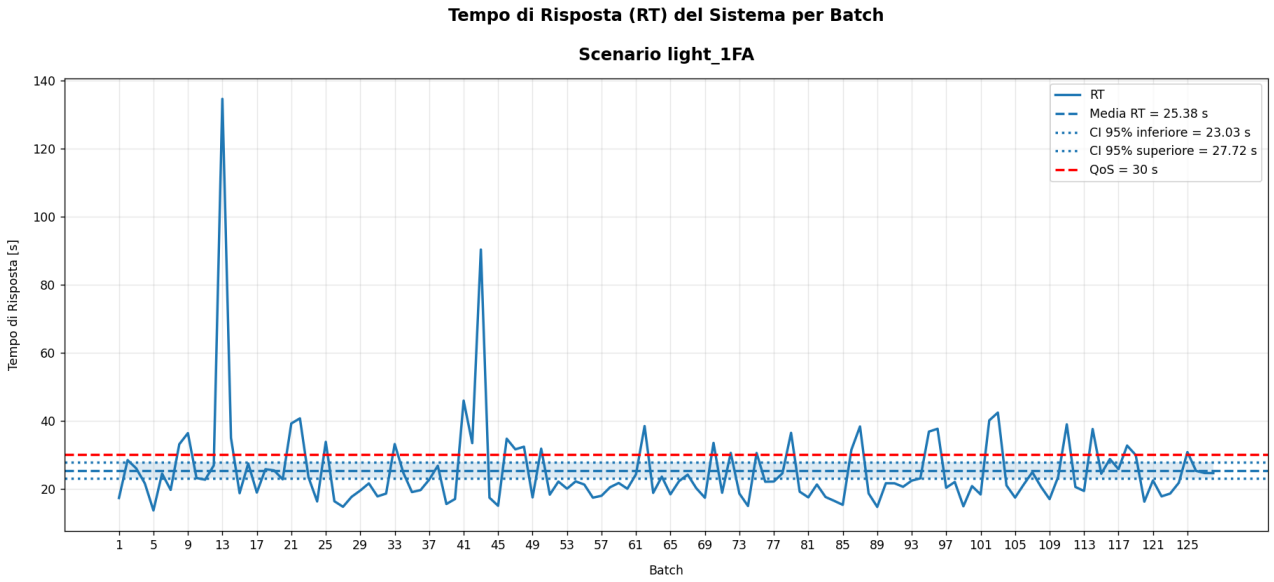

Fig. 19 Andamento del tempo di risposta del sistema per batch nello scenario *light_1FA*, con indicazione della media, dell'intervallo di confidenza al 95% e del vincolo di QoS

Questa distribuzione delle utilizzazioni identifica chiaramente il **Server B come collo di bottiglia del sistema**.

Il **throughput a regime** (1.201101 +/- 0.002434) risulta allineato con il tasso di arrivo imposto (1.2 req/s), confermando che il sistema **riesce a sostenere stabilmente il carico attuale** senza fenomeni di instabilità. Inoltre, il valore osservato risulta coerente anche **con il throughput bound** (1.250741 +/- 0.002469).
La validità del limite ottenuto è stata verificata mediante un'analisi simulativa del throughput al variare del tasso di arrivo λ. A tale scopo, sono state eseguite diverse simulazioni a orizzonte infinito, facendo variare λ da 0.5 a 1.35 con incrementi di 0.05; per ciascun valore è stato calcolato il throughput medio sui batch.
Il grafico risultante (Fig. 20) mostra chiaramente che, al crescere del carico, il throughput si stabilizza su un plateau, corrispondente al valore del throughput massimo. Questo ci dimostra, quindi, che questo limite è tale anche nello scenario *heavy_1FA*, il quale non analizziamo ulteriormente.

10.2.2 Impatto dell'Autenticazione a Due Fattori

Nello scenario **2FA**, il tempo di risposta medio a regime (vedi Fig. 21) risulta **superiore** (31.960176 +/- 2.397176) **rispetto allo scenario 1FA**, con un incremento pari a circa il **25.94%**. Tale aumento risulta significativamente superiore alla soglia del 20% fissata come obiettivo progettuale, confermando che l'introduzione del protocollo SCA comporta un degrado prestazionale rilevante anche in condizioni di carico light.

Le utilizzazioni dei server A, B e P (rispettivamente 0.901250 +/- 0.002112, 0.960460 +/- 0.002673

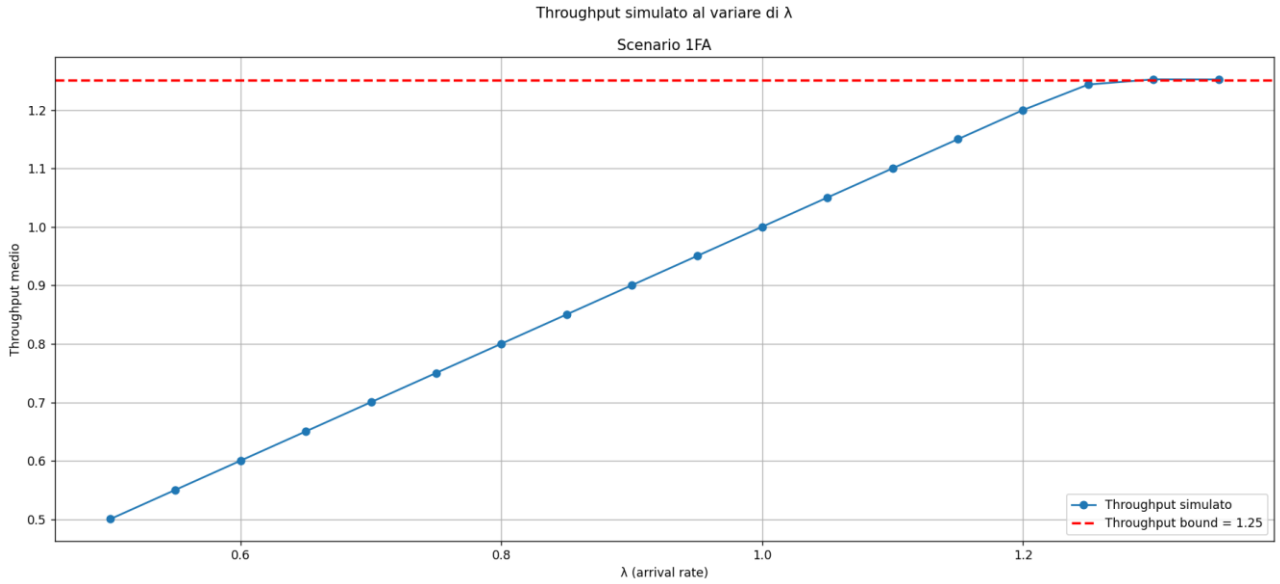

Fig. 20 Andamento del throughput simulato al variare del tasso di arrivo λ, variato da 0.5 a 1.35 con step di 0.05. Il plateau evidenzia il valore massimo raggiungibile dal sistema, in accordo con il throughput bound analitico

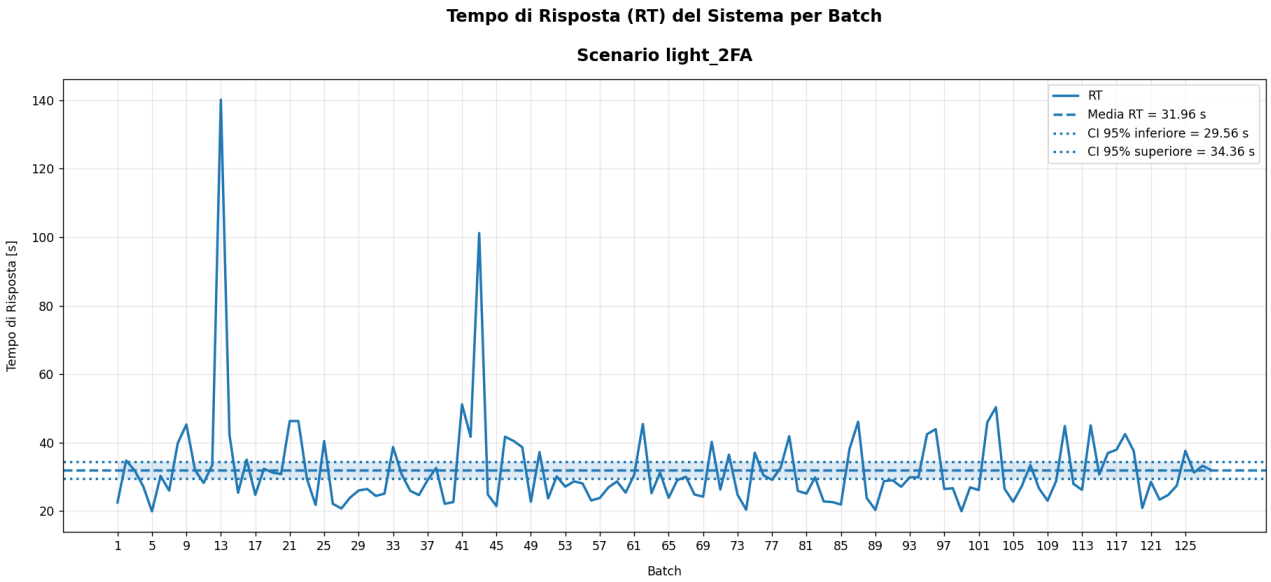

Fig. 21 Andamento del tempo di risposta del sistema per batch nello scenario *light_2FA*, con indicazione della media e dell'intervallo di confidenza al 95%

---

e 0.839282 +/- 0.002310) risultano tutte molto elevate, indicando che l'aumento dei Service Demand introdotto dallo SCA produce un incremento generalizzato del carico su gran parte dell'infrastruttura. In particolare, il Server B continua a presentare l'utilizzazione più elevata, confermando il suo ruolo di collo di bottiglia del sistema anche in presenza dell'autenticazione a due fattori.

Il throughput a regime (1.201102 +/- 0.002468) rimane compatibile con il tasso di arrivo, segnalando che, pur in presenza di un degrado prestazionale, il sistema **rimane stabile sotto carico light anche con SCA**.
Infine, il throughput bound (1.250742 +/- 0.002479) rimane invariato rispetto allo scenario 1FA, confermando che l'introduzione del protocollo SCA, pur aumentando i tempi di risposta e le utilizzazioni dei server, non modifica la capacità massima teorica del sistema.

10.2.3 Effetti del Potenziamento del Database Server

Lo scenario *heavy_1FA_newServerB* rappresenta l'unico caso, per quanto riguarda il carico elevato, in cui risulta sensato condurre un'analisi ad orizzonte infinito, poiché, come evidenziato nell'analisi transitoria, il sistema risulta ora stabile e raggiunge il regime stazionario grazie al potenziamento necessario del Database Server.

Il tempo di risposta medio a regime (Fig. 22) assume un valore finito e stabile (39.270252 +/- 4.395315), confermando che l'intervento sul Server B è stato sufficiente a eliminare l'instabilità strutturale presente nella configurazione heavy senza potenziamento.

L'analisi delle utilizzazioni dei server evidenzia un cambiamento netto nella struttura dei carichi: il

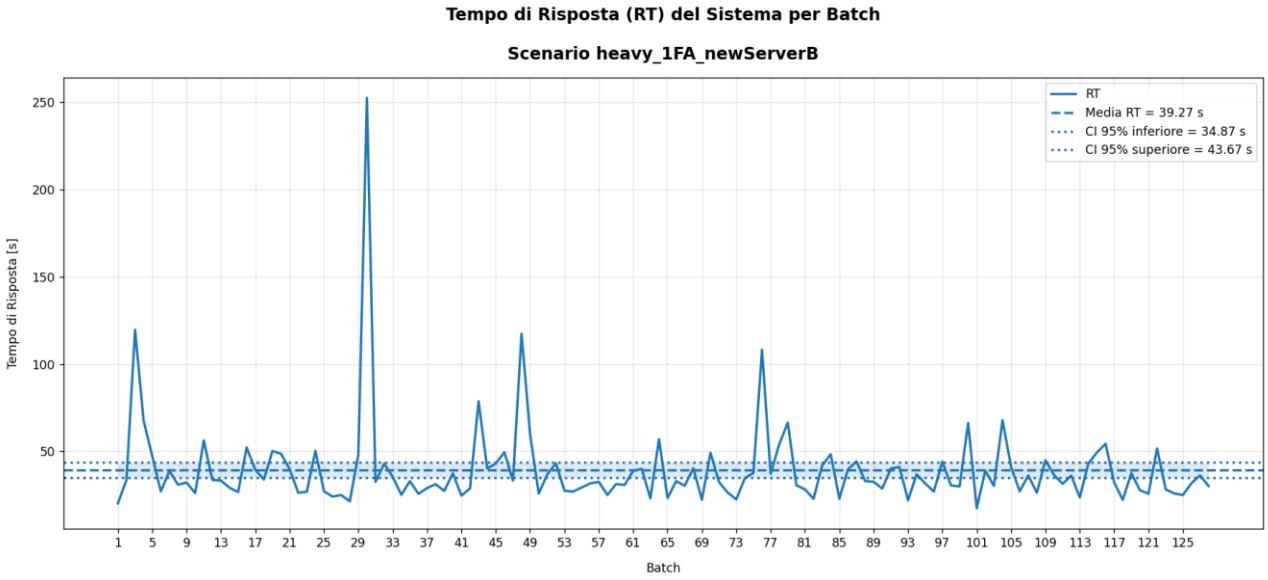

Fig. 22 Andamento del tempo di risposta del sistema per batch nello scenario *heavy_1FA_newServerB*, con indicazione della media e dell'intervallo di confidenza al 95%

Server B non risulta più il nodo maggiormente sollecitato, mantenendo un livello di utilizzazione più contenuto (0.560266 +/- 0.000753), simile a quello del Server P (0.559616 +/- 0.000762); il Server A, invece, presenta ora l'utilizzazione più elevata (0.980406 +/- 0.001136), assumendo il ruolo di **nuovo collo di bottiglia** del sistema, in pieno accordo con le previsioni teoriche.

Per quanto riguarda il throughput a regime, i risultati (1.400311 +/- 0.001409) mostrano che il sistema è ora in grado di raggiungere e sostenere stabilmente il valore richiesto di circa 1.4 richieste/s dallo scenario heavy workload. Inoltre, il throughput effettivamente ottenuto risulta coerente con il nuovo throughput bound dell'architettura (1.428314 +/- 0.000878), il quale, adesso, ha ricevuto un incrementato di circa il **14.19%**.

11 Conclusioni

In questo progetto è stato sviluppato e validato un modello di simulazione a eventi discreti per l'analisi delle prestazioni di un sistema di e-commerce per la vendita online di generi alimentari, ispirato al caso di studio proposto da Serazzi [1, Cap. 6]. L'obiettivo principale era valutare il comportamento del sistema in diversi scenari di carico e configurazione, individuare i colli di bottiglia e studiare l'impatto di modifiche architetturali e funzionali, in particolare l'introduzione dell'autenticazione a due fattori (SCA) e il potenziamento del Database Server.

I risultati ottenuti mostrano che, nello scenario **Light con autenticazione a un fattore**, il sistema è in grado di sostenere stabilmente il carico di circa 4300 richieste/ora, rispettando il vincolo QoS sul tempo di risposta medio (inferiore ai 30 secondi). In tale configurazione, il **Server B (Database**

Server) risulta il collo di bottiglia dell'architettura, operando con un'utilizzazione prossima alla saturazione.

L'introduzione dell'**autenticazione a due fattori (SCA)** genera un **degrado prestazionale significativo**, con un incremento del tempo di risposta medio superiore al 20%, pur mantenendo la stabilità del sistema sotto carico light. Anche in questo scenario il Server B continua a rappresentare il nodo critico. Questo risultato evidenzia come l'aumento dei requisiti di sicurezza possa avere un impatto rilevante sulle prestazioni complessive del sistema.

Nello scenario di **carico Heavy con infrastruttura invariata**, la simulazione ha mostrato chiaramente che il sistema non è in grado di raggiungere il regime stazionario: il Database Server entra in saturazione permanente, generando un accumulo indefinito delle richieste e un tempo di risposta divergente. Tale comportamento conferma l'inadeguatezza dell'architettura originale per sostenere un aumento significativo del tasso di arrivo.

Il **potenziamento del Server B**, mediante il dimezzamento del suo Service Demand, consente invece di **ripristinare la stabilità del sistema anche sotto carico Heavy**. In questa nuova configurazione il collo di bottiglia si sposta sul **Server A**, come previsto teoricamente, e il throughput massimo del sistema aumenta di circa il 14%. Il sistema risulta quindi in grado di sostenere il tasso di 1.4 richieste/s previsto dallo scenario heavy.

Nel complesso, il lavoro ha dimostrato come la simulazione a eventi discreti rappresenti uno strumento efficace per:

- individuare i colli di bottiglia di un sistema complesso;
- valutare l'impatto di modifiche architetturali e funzionali;
- supportare decisioni di **capacity planning** e **performance tuning**.

11.1 Limitazioni del Modello

Nonostante i risultati ottenuti siano coerenti e significativi, il modello presenta alcune limitazioni:

- **Caratterizzazione semplificata del carico di lavoro**: nel modello è stata considerata una sola classe di utenti. Una caratterizzazione più realistica potrebbe prevedere più classi di clienti, ad esempio differenziate in base al numero di prodotti acquistati o al tipo di operazioni effettuate.
- **Modello degli arrivi semplificato**: gli arrivi sono stati modellati tramite un processo di Poisson. Sebbene tale scelta sia ampiamente utilizzata in letteratura e adeguata per lo studio dei fenomeni medi di funzionamento del sistema, essa non consente di catturare fenomeni di burst, autocorrelazione e variazioni temporali del traffico tipiche dei sistemi reali.
- **Assenza di ritardi di rete, fallimenti e retry**: il modello trascura completamente gli effetti della rete, le possibili perdite di pacchetti, i timeout, i rifiuti di servizio e i tentativi di ritrasmissione, che in scenari reali possono influenzare sensibilmente le prestazioni percepite dall'utente.

11.2 Possibili Miglioramenti

A partire dalle limitazioni individuate, sono possibili diversi sviluppi futuri del lavoro:

- **Introduzione di più classi di utenti**, per modellare diverse tipologie di clienti, ottenendo una stima più realistica dei tempi di risposta segmentata per profilo.
- **Adozione di modelli di arrivo più avanzati**, come distribuzioni iper-esponenziali, Coxiane, Phase-Type o modelli bursty, al fine di rappresentare in modo più accurato le fluttuazioni del traffico web.
- **Introduzione di meccanismi di load balancing**, fondamentali nelle moderne architetture web. Una migliore allocazione dei compiti tra i diversi server e servizi potrebbe ridurre l'impatto dei colli di bottiglia e migliorare significativamente la scalabilità del sistema.
- **Estensione del modello verso architetture multi-server o cloud**, includendo repliche del Server A o del Server B, politiche di scaling automatico e meccanismi di failover.

Riferimenti

[1] G. Serazzi, *Performance Engineering*, Springer Open Access.

[2] M. Basili, “Simulazione del Workflow di una Web App,” Github repository, 2025. Disponibile online: *https://github.com/MatteoBasili/pmcsn-progetto-2024_25/tree/master/WebAppWorkflow/simulation* (ultimo accesso: 10 dicembre 2025).

[3] Lawrence M. Leemis, Sthephen K. Park, *Discrete-Event Simulation - A first course*, Pearson Education Prentice Hall, 2006.

[4] P. Steele, "DES-Python: Python code for Discrete-Event Simulation – A First Course." Disponibile online: *https://github.com/pdsteele/DES-Python* (ultimo accesso: 18 novembre 2025).

[5] Lawrence M. Leemis, "Materiale di supporto per Discrete-Event Simulation – A First Course." Disponibile online: *https://www.math.wm.edu/~leemis/* (ultimo accesso: 18 novembre 2025).

[6] J. Virtamo, “Teletraffic Theory – Processor Sharing Queue (PS Queue),” corso 38.3141, Aalto University, Helsinki, Finlandia, s.d. Disponibile online: *https://www.netlab.tkk.fi/opetus/s38314-1/kalvot/E_psjono.pdf* (ultimo accesso: 10 dicembre 2025).

[7] M. Basili, “Modello Analitico del Workflow di una Web App,” Github repository, 2025. Disponibile online: *https://github.com/MatteoBasili/pmcsn-progetto2024_25/tree/master/WebAppWorkflow/analytical* (ultimo accesso: 10 dicembre 2025).