

Università degli Studi di Roma “**Tor Vergata**”

Dipartimento di Ingegneria
Corso di Laurea in Ingegneria Informatica

Sistema degli Scanner Facciali negli e-Gate dell’Aeroporto di Ciampino

Progetto PMCSN



Leonardo Beccarini – 0360299 Luca Capone – 0351439 Pierfrancesco Lijoi – 0350092

Anno Accademico 2024–2025

Indice

1 Introduzione	3
1.1 Livello IoT	3
1.2 Livello Edge	3
1.3 Livello Cloud	4
2 Scopi e Obiettivi	4
2.1 Dettagli sul vincolo di non prioritizzazione	5
3 Modello concettuale	5
3.1 Classi di pacchetti e flusso logico	5
3.2 Eventi	5
3.3 Meccanismi operativi specifici	6
3.4 Schema del modello	7
4 Modello delle specifiche	7
4.1 Modellazione dei tempi di arrivo	8
4.2 Fasce orarie e probabilità giornaliere	8
4.3 Formula generale per il tasso di arrivo	9
4.4 Lambda per fascia oraria	9
4.5 Media ponderata giornaliera	9
4.6 Equazioni di traffico e matrice di routing	10
4.6.1 Probabilità dei job di Classe E nel Coordinator Server Edge	10
4.7 Modellazione dei centri	11
5 Modello computazionale	13
5.1 Gestione degli eventi	13
6 Verifica	14
6.1 Controlli di consistenza	17
7 Validazione	17
8 Design degli esperimenti	18
9 Calcolo del seed indipendente tra repliche	18
10 Analisi del transitorio	18
11 Analisi del collo di bottiglia	22
12 Intervallo di confidenza	24
13 Simulazione Orizzonte Finito	24
14 Simulazione ad Orizzonte Infinito	31
15 Scalabilità	38
15.1 Implementazione	38
15.2 Risultati	40
16 Valutazione dei Risultati	43
17 Modello migliorativo	43

18 Obiettivi Modello migliorativo	43
19 Modello Concettuale	44
20 Modello delle Specifiche	44
20.1 Modellazione dei centri	45
21 Modello Computazionale	46
22 Verifica	47
23 Validazione	48
24 Design degli esperimenti	49
24.1 Calcolo del seed indipendente tra repliche	49
24.2 Analisi del transitorio	49
24.3 Analisi del collo di bottiglia	51
24.4 Simulazione ad Orizzonte Finito	53
24.4.1 Implementazione	53
24.4.2 Risultati	55
25 Conclusioni finali	59
25.1 Applicabilità operativa	60
26 Bibliografia	61

1 Introduzione

Il caso di studio descritto nel volume *Performance Engineering: Learning Through Applications Using JMT* [ref1] è stato preso come punto di partenza e adattato al contesto reale dell'aeroporto di Ciampino. A differenza del libro, abbiamo utilizzato dati e informazioni ufficiali e, di conseguenza, apportato modifiche ad alcune semplificazioni presenti nel modello originale, che trascuravano aspetti che, invece, abbiamo considerato, come la gestione dei server coordinatori.

Inoltre, sulla base dei dati ufficiali, abbiamo ampliato il caso di studio modellando tutti gli eGate presenti alle partenze [ref3] e includendo anche i *Coordinator Server Edge* — non semplificato come nella traccia originale del volume — il quale ha il compito di elaborare le richieste ricevute dal rispettivo nodo Edge, in modo da poter sincronizzare le operazioni e reagire tempestivamente in base alla classificazione dei pacchetti elaborati.

L'architettura del sistema è organizzata su tre livelli principali: **IoT**, **Edge** e **Cloud**.

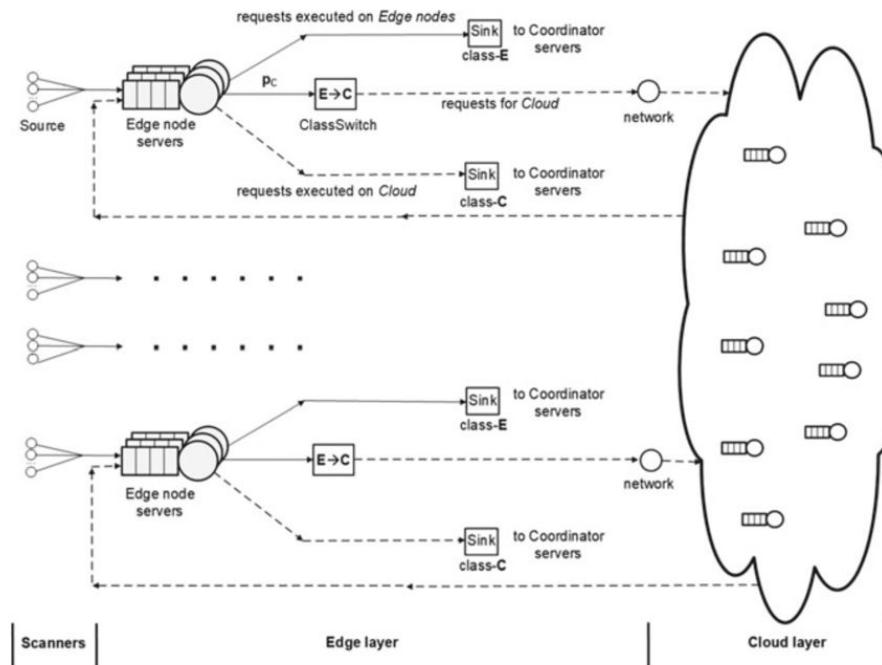


Figura 1: Schema a tre livelli del sistema IoT–Edge–Cloud

1.1 Livello IoT

Il primo livello comprende le videocamere distribuite lungo tutta la struttura; nel nostro caso, i dati provengono in particolare da quelle installate ai gate di partenza, identificati nei dati ufficiali come eGate [ref3], le quali effettuano le scansioni facciali che vengono poi inviate al livello edge.

1.2 Livello Edge

Il secondo livello è costituito dagli **Edge node**, ossia rack che ospitano uno o più server fisici. In questo livello avviene la prima analisi: i server confrontano le scansioni con i database locali per determinarne la categoria. Se la corrispondenza è già stata trovata a livello Edge, lo scan viene inviato al *Coordinator Server Edge* per eseguire le operazioni di reazione.

Le possibili tipologie di scansioni sono cinque¹:

¹come definito a pagina 92 nel libro *Performance Engineering: Learning Through Applications Using JMT*

- **Qualità scadente**, corrispondenti al pacchetto **P1**;
- **Regolari**, corrispondenti al pacchetto **P2**;
- **Sospette**, corrispondenti al pacchetto **P3**;
- **Pericolose**, corrispondenti al pacchetto **P4**;
- **Sconosciute**, corrispondenti al pacchetto **P5**;

Dopo la classificazione, le scansioni appartenenti alle tipologie da P1 a P4 vengono inoltrate dal nodo Edge al *Coordinator Server Edge*. Quest'ultimo si occupa di sincronizzare rapidamente le telecamere per monitorare i passeggeri: in particolare, i passeggeri pericolosi (P4) e sospetti (P3) vengono seguiti in tempo reale, mentre per i pacchetti relativi a passeggeri regolari (P2) o a immagini di scarsa qualità (P1) sono previste altre operazioni, non specificate, finalizzate comunque a garantire la sicurezza all'interno dell'aeroporto. Invece, i pacchetti di tipo sconosciuto (P5) dal nodo Edge, vengono inoltrati al livello successivo dell'architettura, il cloud.

1.3 Livello Cloud

Il terzo livello viene attivato solo per le scansioni classificate come pacchetti di tipo **P5**. In questo caso, l'immagine viene inoltrata al cloud, modellato come un server con capacità infinita, in grado di processare tutte le richieste senza generare colli di bottiglia. Abbiamo assunto che l'aeroporto di Ciampino operi su cloud fornito da un provider esterno che offre un servizio di tipo SaaS (Software as a Service), per cui non è stato possibile reperire informazioni ufficiali sull'architettura, compatibili con il modello realistico del nostro progetto.

Qui, l'analisi sfrutta basi di dati molto più estese rispetto a quelle locali, incluse fonti come social network, siti web e archivi biometrici. Una volta completata l'elaborazione, i risultati vengono trasmessi nuovamente all'Edge node, che provvede ad aggiornare il database locale.

2 Scopi e Obiettivi

Questo studio, come già accennato, si basa su dati ufficiali dell'aeroporto di Ciampino [ref2], consentendoci di modellare in modo realistico il sistema aeroportuale. Per questo motivo, abbiamo modificato alcune semplificazioni adottate nel libro: in particolare, abbiamo incluso la gestione dei server coordinatori, rendendo il progetto più aderente al nostro contesto operativo. A tal proposito, abbiamo esteso gli obiettivi proposti ottenendo una visione precisa della modellazione delle code di rete relative agli eGate dell'aeroporto.

Innanzitutto, è richiesto il QoS di un tempo di risposta medio inferiore ai 3 secondi per tutte le tipologie di scan gestite a livello Edge, ad eccezione di quelle sconosciute (P5), ovvero i pacchetti di classe C provenienti dal nodo cloud². Tale requisito deve essere rispettato **senza introdurre priorità tra le diverse classi di job**. Inoltre, il modello proposto si pone lo scopo di mantenere il QoS tenendo conto anche di altri aspetti, quali la riduzione dei costi e la variabilità del carico di lavoro.

Più nello specifico:

- Ridurre il numero di Edge node da istanziare inizialmente in tutta la struttura;
- Mantenere il QoS in funzione di diverse frazioni di scan di categoria sconosciuta e delle possibili fluttuazioni del carico di lavoro;
- Garantire la scalabilità dinamica dei server in funzione di picchi di carico tramite valori di guardia.

²come definito a pagina 93 nel libro *Performance Engineering: Learning Through Applications Using JMT*

2.1 Dettagli sul vincolo di non prioritizzazione

Il QoS relativo al tempo di risposta medio per le scansioni gestite a livello Edge mira a garantire che ogni elaborazione rimanga inferiore a 3 secondi, escludendo i pacchetti di feedback relativi a “persone sconosciute” (P5). La scelta di non attribuire priorità a una classe di job rispetto ad un’altra è motivata dalla necessità di trattare ogni aggiornamento nel sistema con la stessa importanza, inclusi i pacchetti di update.

Un aggiornamento nel database effettuato da un pacchetto di feedback potrebbe infatti modificare lo stato di un pacchetto nei successivi arrivi al centro, facendolo passare da “sconosciuto” a “pericoloso” o “sospetto”. Per questo motivo, al fine di garantire la sicurezza aeroportuale, è fondamentale che tutte le scansioni siano trattate con pari priorità. In questo modo si evitano ritardi o la trascuratezza di pacchetti potenzialmente critici per la sicurezza, che potrebbero rappresentare una minaccia per i passeggeri.

3 Modello concettuale

Secondo fonti ufficiali, a Ciampino sono presenti complessivamente 11 e-gate, di cui 6 per gli arrivi e 5 per le partenze [ref3]. Nel nostro modello abbiamo scelto di soffermarci sui 5 scanner dedicati alle partenze, che abbiamo considerato come sorgenti di immagini.

Definito il numero di e-gate, il passo successivo è stato stabilire il numero di nodi edge. Sebbene un nodo edge possa essere composto da più server, per l’analisi di base è sufficiente³ considerarlo come unità singola. Individuare la configurazione ottimale di questo nodo rappresenta pertanto uno degli obiettivi dello studio.

Per quanto riguarda il livello cloud, abbiamo scelto di rappresentarlo come un *infinite server*, poiché si tratta di una risorsa esterna, non gestita direttamente dall’aeroporto, e che non impatta sugli obiettivi principali del modello. Abbiamo quindi considerato il cloud come una risorsa con capacità illimitata.

3.1 Classi di pacchetti e flusso logico

Inizialmente tutti i pacchetti provenienti dagli e-gate arrivano all’Edge node con etichetta di *classe E*, poiché il loro tipo non è ancora stato identificato.

All’arrivo su un nodo Edge, per identificarne il tipo, viene effettuata una ricerca su un database locale⁴:

- se la ricerca produce una corrispondenza, lo scanner viene classificato come un pacchetto di tipo P1, P2, P3 o P4; in tal caso il pacchetto mantiene l’etichetta di *classe E* e viene inviato al *Coordinator Server Edge* per eseguire le operazioni specifiche definite in precedenza;
- se non viene trovata corrispondenza, il pacchetto risulta sconosciuto (P5) e viene inviato al cloud tramite un *class switch*, che ne cambia l’etichetta in *classe C*. Il Cloud potrà quindi eseguire operazioni più elaborate e complesse per identificarne l’identità del passeggero sconosciuto, interrogando database NoSQL distribuiti, integrando documenti, profili social, dati biometrici e tracce vocali per un’analisi completa. Una volta completata la ricerca, il pacchetto viene reinoltrato al nodo Edge come feedback per aggiornare il database, in modo che il passeggero non risulti più sconosciuto nelle successive scansioni.

3.2 Eventi

Lo stato del sistema è definito dal numero dei job presenti nel sistema. Nel nostro modello consideriamo i seguenti eventi:

³come definito a pagina 97 nel volume Performance Engineering: Learning Through Applications Using JMT

⁴come definito a pagina 94 nel libro Performance Engineering: Learning Through Applications Using JMT

Sistema:

- Evento di Arrivo: l'arrivo di un pacchetto dagli e-gate coincide con un evento di arrivo al sistema;
- Evento di completamento: Il completamento di un job si verifica all'uscita dall'Edge Node. Per i pacchetti di *classe C*, tale completamento coincide con l'uscita dal sistema, che avviene in maniera deterministica. Per i pacchetti di *classe E*, invece, il completamento all'interno dell'Edge Node ha valore solo locale: i pacchetti vengono infatti inoltrati al *coordinator server edge*. Solo il completamento di questi pacchetti presso il *coordinator server edge* rappresenta la loro effettiva uscita dal sistema.

Edge node:

- Evento di Arrivo:
 - Un pacchetto di *classe E* dagli e-gate;
 - Un pacchetto di *classe C* di ritorno dal cloud (non considerato come nuovo arrivo nel sistema);
- Evento di Completamento:
 - Un pacchetto di *classe E* inoltrato al nodo *Coordinator Server Edge*, rappresenta un completamento locale per il nodo edge, ma non l'uscita dal sistema.
 - Dopo l'aggiornamento locale dal nodo Edge, un pacchetto di *classe C* lascia il sistema in maniera deterministica.
 - Un pacchetto di *classe C* inoltrato al Cloud, tramite switch, rappresenta un completamento locale per il nodo edge, ma non l'uscita dal sistema.

Cloud server:

- Evento di Arrivo: quando un pacchetto di *classe E* viene riclassificato come *classe C*;
- Evento di Completamento Locale : quando il pacchetto di *classe C* verrà inviato al nodo edge nuovamente;

Coordinator Server Edge:

- Evento di Arrivo: i pacchetti di *classe E* provenienti dal nodo edge;
- Evento di Completamento : i pacchetti una volta elaborati usciranno dal sistema ;

3.3 Meccanismi operativi specifici

Per rendere il modello più aderente al contesto, abbiamo etichettato i pacchetti come segue:

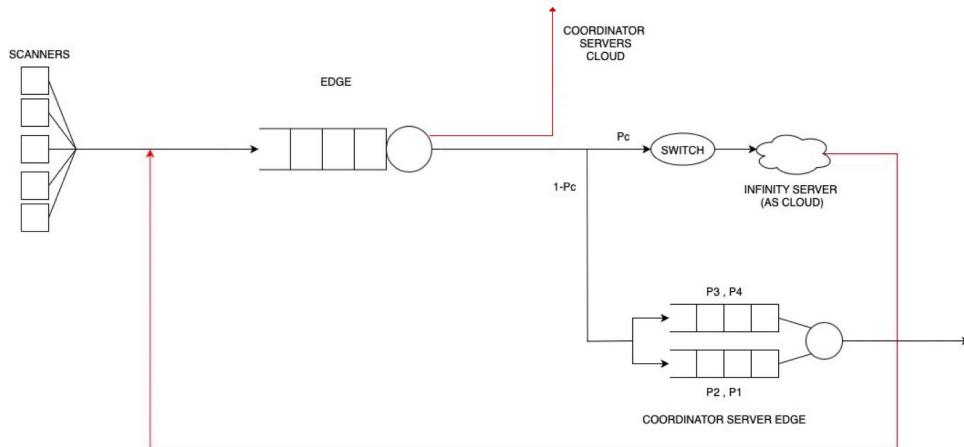
- Tutti i nuovi pacchetti che arrivano al nodo *Edge* sono inizialmente etichettati come *classe E*. Al termine del servizio all'Edge, ciascun pacchetto viene rietichettato con il *tipo effettivo* P_i (con $i \in \{1, \dots, 5\}$) in base all'esito dell'identificazione, e viene quindi assegnato alla *classe di destinazione*: resta di *classe E* se gestito localmente dal *Coordinator Server Edge* oppure passa a *classe C* se inoltrato al *Cloud*.
- i pacchetti di *classe E*, che restano per l'elaborazione locale, nel *Coordinator Server Edge*, sono di tipo **P1, P2, P3 e P4** ;

- il pacchetto **P5** è di *classe C* e viene sempre processato dal cloud; dopo l'elaborazione, ritorna al nodo edge con etichetta **P5** sempre di *classe C*, diventando un pacchetto di aggiornamento che, dopo l'update sul database locale, esce direttamente dal sistema senza entrare in ulteriori centri, poiché non considerati negli obiettivi.
- i pacchetti **P1**, **P2**, **P3** e **P4** vengono processati nel *coordinator server Edge* e messi in coda in base al tipo:
 - pacchetti **P3** e **P4**: hanno priorità elevata, in quanto richiedono un'azione immediata e tempestiva⁵. Oltre alla scansione facciale, devono infatti attivare ulteriori operazioni, come la sincronizzazione dei dispositivi di sorveglianza negli e-gate.
 - pacchetti **P1** e **P2**: non vengono specificate le ulteriori operazioni.

Il nodo Edge ha il compito di classificare i nuovi pacchetti ricevuti (da **P1** a **P5**), effettuando interrogazioni su database.

3.4 Schema del modello

Il modello concettuale appena descritto può essere visualizzato per un singolo edge node come segue:



Dove P_c è la probabilità che uno scan elaborato dall'Edge Node venga classificato come pacchetto di classe 'C'. Tutti i pacchetti in arrivo sono inizialmente di classe 'E' e possono cambiare classe solo in seguito alla fase di classificazione presso l'Edge Node. I pacchetti che diventano di classe 'C' vengono poi elaborati dal cloud e successivamente ritornano all'Edge Node per operazioni di aggiornamento, al termine delle quali lasciano il sistema con probabilità 1 (percorso in rosso).

4 Modello delle specifiche

In questa fase, lo stato del sistema è stato definito come collezione di variabili e sono state applicate a quest'ultime una serie di equazioni aventi lo scopo di descriverne le relazioni. Non essendo disponibili tutte le informazioni pubbliche relative all'aeroporto di Ciampino, per le specifiche mancanti ci siamo basati sul caso di studio riportato nel libro di testo o su esperienze personali. In questo modo, abbiamo potuto ottenere dei valori iniziali da utilizzare come punto di partenza per la nostra modellazione.

⁵come definito a pagina 92 nel libro *Performance Engineering: Learning Through Applications Using JMT*

4.1 Modellazione dei tempi di arrivo

Per quanto riguarda i tempi di arrivo nel sistema, abbiamo scelto un processo di Poisson, in cui il tempo di inter-arrivo segue una distribuzione esponenziale. Questa scelta è motivata da:

- **Validità teorica:** la letteratura sui sistemi di attesa riconosce che i flussi di arrivo possono essere modellati come *processo di Poisson non stazionario (NHPP)*, caratterizzato da tassi di arrivo $\lambda(t)$ variabili nel tempo. Dividendo la giornata in finestre temporali, in ciascuna fascia oraria gli inter-arrivi possono essere approssimati da una distribuzione esponenziale con parametro costante λ_i [ref6].
- **Proprietà di memorylessness:** la distribuzione esponenziale è l'unica che gode della proprietà di assenza di memoria, rendendo il tempo fino al prossimo arrivo indipendente dal tempo già trascorso. Inoltre, la somma di processi di Poisson indipendenti rimane un processo di Poisson, caratteristica utile quando i flussi derivano da più fonti [ref12].
- **Validità empirica:** studi applicativi hanno dimostrato che l'utilizzo di modelli basati su NHPP ed esponenziali ponderate per fasce orarie fornisce buone performance predittive nella simulazione del traffico aeroportuale [ref4] [ref5].

Per individuare i tassi medi di arrivo per le fasce orarie abbiamo utilizzato i dati ufficiali di traffico forniti da Aeroporti di Roma per l'aeroporto di Roma Ciampino [ref2].

Mese e anno	Totale passeggeri in Partenza
Gennaio 2024	309106
Febbraio 2024	299836
Marzo 2024	330900
Aprile 2024	318644
Maggio 2024	330631
Giugno 2024	325146
Luglio 2024	338978
Agosto 2024	342599
Settembre 2024	319241
Ottobre 2024	332077
Novembre 2024	312542
Dicembre 2024	302106

Tabella 1: Passeggeri aeroporto di Ciampino per mese nel 2024

4.2 Fasce orarie e probabilità giornaliere

Abbiamo diviso la giornata in 6 fasce di 4 ore ciascuna. Poiché né il libro né i dati ufficiali forniscono la distribuzione delle probabilità per fascia oraria, i valori sono stati definiti sulla base di esperienze personali e osservazioni pratiche.

Fascia	Ore	Probabilità p_i
F1	00:00–03:59	0.10
F2	04:00–07:59	0.20
F3	08:00–11:59	0.14
F4	12:00–15:59	0.23
F5	16:00–19:59	0.18
F6	20:00–23:59	0.15

Tabella 2: Distribuzione delle probabilità per fascia oraria

4.3 Formula generale per il tasso di arrivo

Il tasso di arrivo medio per ogni fascia oraria si calcola con:

$$\lambda_i = \frac{P}{H_i \cdot 3600 \cdot G} \cdot p_i \cdot k$$

dove:

- P = numero totale di passeggeri nel mese
- H_i = lunghezza in ore della fascia oraria i
- G = numero di giorni nel mese
- p_i = probabilità della fascia oraria i
- k = tempo medio di transizione nel eGate in secondi per passeggero

Il valore di $k = 10.79$ secondi è stato scelto in maniera ragionevole sulla base di controlli online e osservazioni pratiche: il tempo medio di transito effettivo si attesta intorno a questo valore, consentendo di rendere più rapide le operazioni di check-in e migliorare la simulazione dei flussi di passeggeri. Inoltre, supponendo che uno scanner invia circa un frame al secondo, k sarà il numero medio di scansioni inviate per passeggero.

4.4 Lambda per fascia oraria

Per l'analisi, sono stati presi in considerazione i valori registrati nel mese più critico in termini di passeggeri in partenza, cioè Agosto 2024:

$$P = 342599, \quad G = 31, \quad k = 10.79$$

Fascia	Probabilità p_i	λ_i (arrivi/s)
F1	0.10	0.83
F2	0.20	1.66
F3	0.14	1.16
F4	0.23	1.90
F5	0.18	1.49
F6	0.15	1.24

Tabella 3: Tassi di arrivo λ_i per fascia oraria ad agosto 2024 , mese peggiore nel 2024

4.5 Media ponderata giornaliera

La media ponderata dei tassi di arrivo sull'intera giornata si calcola come:

$$\lambda_{\text{globale}} = \sum_{i=1}^6 \frac{H_i}{\sum_{j=1}^6 H_j} \cdot \lambda_i$$

Poiché tutte le fasce hanno durata uguale $H_i = 4$ ore, la media ponderata diventa semplicemente la media aritmetica:

$$\lambda_{\text{globale}} = \frac{1}{6} \sum_{i=1}^6 \lambda_i = 1.38 \text{ j/s}$$

In questo modo, abbiamo ottenuto la media giornaliera del tasso di arrivo (λ_{globale}) nell'aeroporto di Ciampino, per poter studiare il suo comportamento in un contesto complessivo e le fluttuazioni del carico di lavoro nelle diverse fasce orarie della giornata.

4.6 Equazioni di traffico e matrice di routing

Di seguito viene riportato il sistema, con aggiunte le rispettive equazioni di traffico e la matrice di routing.

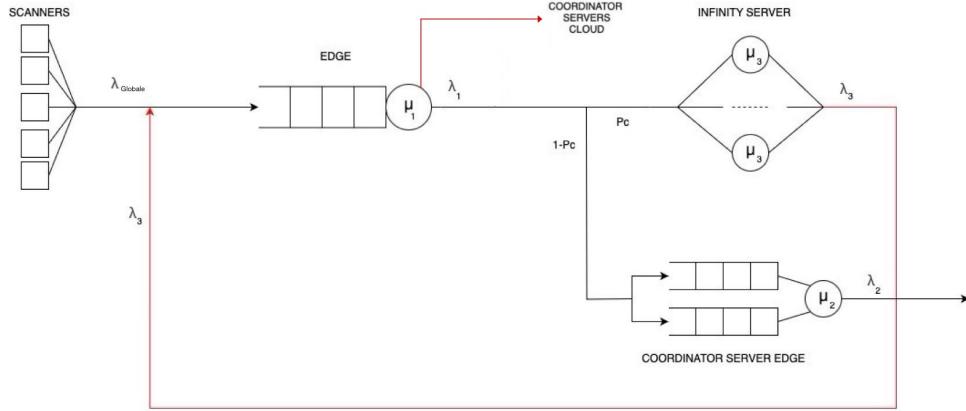


Figura 2: Modello delle specifiche

	Coordinator Server Edge	Edge	Cloud	Esterno
Coordinator Server Edge	0	0	0	1
Edge	$1 - p_c$	0	p_c	0
Cloud	0	1	0	0
Esterno	0	1	0	0

Tabella 4: Matrice di routing

Come definito in precedenza, viene modellato in questo momento con λ_{globale} , poiché lo studio in questa prima fase è inerente ad uno studio del carico medio complessivo durante la giornata. Qui di seguito l'equazioni di traffico:

$$\begin{cases} \lambda_1 = \lambda_{\text{globale}} + \lambda_3 \\ \lambda_2 = \lambda_1(1 - p_c) \\ \lambda_3 = \lambda_1 p_c \end{cases}$$

4.6.1 Probabilità dei job di Classe E nel Coordinator Server Edge

Le seguenti probabilità si riferiscono al 60% dei job di Classe E che, dopo la fase di classificazione svolta dall'Edge, con probabilità $1 - P_c = 0.6$ vengono instradati verso il nodo Coordinator Server Edge. Poiché non sono disponibili dati ufficiali, la distribuzione di tali job tra i pacchetti P1 a P4 è stata stimata sulla base di opportune considerazioni e assunzioni.

- $P1\text{-prob}$: probabilità del 20% che il job sia poor-quality;
- $P2\text{-prob}$: probabilità del 25% che il job sia regular;
- $P3\text{-prob}$: probabilità del 10% che il job sia suspect;
- $P4\text{-prob}$: probabilità del 5% che il job sia dangerous.

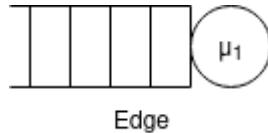
Il primo tasso è coerente con le osservazioni condotte in aeroporti secondari, dove circa un quinto delle immagini richiede un ritocco o una nuova acquisizione per via di inquadrature non ottimali [ref8] Il secondo tasso rispecchia le statistiche interne di gate a traffico moderato, in cui circa un quarto dei passeggeri segue correttamente le istruzioni di posizionamento. Il terzo tasso costituisce un tasso di allerta moderato, in linea con studi [ref8] di settore su aeroporti di medio picco, che riportano tra l'8% e il 12%. L'ultimo tasso risulta coerente con report [ref9] interni di sicurezza per checkpoint secondari, dove si registra mediamente un 3–7% di allarmi “high-risk” da parte degli operatori.

4.7 Modellazione dei centri

Questa sezione ha lo scopo di chiarire le scelte effettuate per la modellazione dei centri, specificando le distribuzioni e i tempi medi di servizio adottati. Poiché non sono state reperite informazioni ufficiali relative a tali parametri, si è scelto di utilizzare i valori riportati nel libro di testo come punto di partenza, oppure da assunzioni personali in caso di assenza anche da quest ultimo. Per quanto riguarda i processi di arrivo, si è assunto un tasso con distribuzione esponenziale: nonostante la presenza di feedback e di un flusso logico governato da probabilità, il teorema (di Burke o di Little, a seconda del contesto) continua a essere valido.

Qui di seguito viene affrontata la descrizione delle scelte di modellazione effettuate:

- Edge



Il nodo Edge è stato inizialmente modellato come un singolo server con coda FIFO infinita, così da garantire l'elaborazione di tutti i pacchetti in arrivo e preservare la sicurezza aeroportuale. Questa rappresenta la configurazione base del sistema. Uno degli obiettivi dello studio è individuare la configurazione ottimale del centro Edge, intesa come il bilanciamento tra numero di server, rispetto dei requisiti di QoS e contenimento dei costi, anche in presenza di fluttuazioni del carico di lavoro.

Non avendo trovato informazioni ufficiali inerenti alla distribuzione del tempo di servizio con i relativi valori medi per tipologia di classe, sono stati utilizzati quelli indicati dal libro di testo. In particolare, si è scelto di modellare i tempi di servizio con una distribuzione esponenziale, poiché:

- riflette la natura regolare e ripetitiva delle operazioni svolte all'Edge;
- consente di rappresentare in modo semplice la variabilità dei tempi di servizio attorno a un valore medio;
- gode della proprietà di memoria nulla (*memoryless*), che rende il modello coerente con l'ipotesi di indipendenza tra i job.

Nello specifico, la distribuzione del tempo di servizio è esponenziale con tempo medio:

$$E[S] = 0.5 \text{ s} \quad \text{per i job di classe E}$$

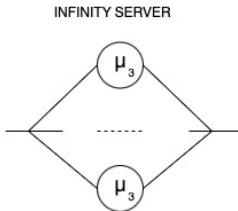
$$E[S] = 0.1 \text{ s} \quad \text{per i job di classe C.}$$

Inoltre, il tempo tra due arrivi successivi è modellato, per via del teorema di Burke ,con una distribuzione esponenziale di parametro :

$$\frac{1}{\lambda_{\text{globale}} + \lambda_3}$$

Dove λ_{globale} rappresenta il tasso complessivo di arrivo e λ_3 eventuali pacchetti aggiuntivi di feedback.

- **Cloud**



Il cloud è stato modellato come un server con capacità infinita, avente un tempo di servizio descritto da una distribuzione esponenziale con valore medio del tempo di servizio pari a:

$$E[S] = 0.8 \text{ s}$$

La scelta della distribuzione esponenziale è stata adottata in coerenza con le specifiche riportate nel testo di riferimento ⁶, che fornisce valori e modelli standardizzati per i sistemi cloud.

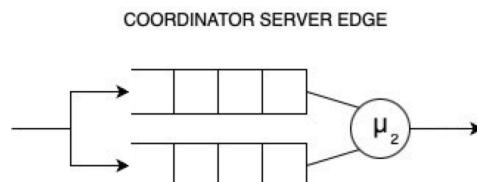
Abbiamo scelto inoltre di rappresentare il cloud server come *infinite server*, poiché nei dati ufficiali disponibili non viene fornita alcuna informazione riguardo al numero effettivo di server utilizzati. A tal proposito, si è ipotizzato che l'aeroporto faccia uso di una tecnologia *SaaS (Software as a Service)* erogata da un provider esterno. Questa scelta riflette la grande capacità computazionale comunemente associata ai servizi cloud, come ad esempio quelli offerti da Amazon, i cui data center sono in grado di processare tutte le richieste senza generare colli di bottiglia.

Inoltre, il tempo tra due arrivi successivi è modellato, per via del teorema di Burke ,con una distribuzione esponenziale di parametro :

$$\frac{1}{\lambda_1 \cdot P_c}$$

Dove λ_1 è il tasso di arrivo dei pacchetti principali e P_c la probabilità di instradamento verso il cloud..

- **Coordinator Server Edge**



⁶Come definito a pagina 96 nel libro *Performance Engineering: Learning Through Applications Using JMT*.

Questo centro è stato modellato come un centro a servente singolo con due code FIFO. La prima coda, a priorità più alta, è destinata ai pacchetti di tipo suspect e dangerous, mentre la seconda gestisce i pacchetti di tipo low-quality e regular⁷.

Per la coda a priorità minore, che contiene i pacchetti P1 e P2, è stata adottata una distribuzione esponenziale con tempo medio di servizio $E[S] = 0.40, s$. Questi pacchetti richiedono infatti un insieme di operazioni relativamente semplici e rapide, che comportano un'elaborazione meno onerosa rispetto alle altre classi.

Per la coda a priorità maggiore, riservata ai pacchetti P3 e P4 che richiedono operazioni tempestive, è stata invece adottata una distribuzione esponenziale con tempo medio di servizio $E[S] = 0.60, s$. In questo caso, oltre alle stesse operazioni richieste dai pacchetti P1 e P2, è prevista un'ulteriore fase di elaborazione e sincronizzazione con componenti esterni, che rende il servizio più complesso e mediamente più lungo.

La scelta della distribuzione esponenziale per entrambe le code è stata dettata dalla sua capacità di modellare correttamente la variabilità del carico di lavoro e di rappresentare in modo realistico processi di servizio che possono presentare tempi anche molto diversi tra loro. Inoltre, l'esponenziale possiede la proprietà di (*memoryless*), che riflette bene il comportamento di sistemi in cui la durata residua di un servizio non dipende dal tempo già trascorso. Questa caratteristica consente di rappresentare in maniera semplice e aderente alla teoria le operazioni dei pacchetti in elaborazione per questo centro, sia nei casi più semplici che in quelli più complessi.

Inoltre, il tempo tra due arrivi successivi è modellato, per via del teorema di Burke ,con una distribuzione esponenziale di parametro :

$$\frac{1}{\lambda_1 \cdot (1 - P_c)}$$

Il quale è corrispondente ai pacchetti che rimangono da elaborare localmente presso l'Edge.

5 Modello computazionale

Il progetto è organizzato in quattro principali directory:

- **Libraries:** include alcuni script sviluppati da Steve Park e Dave Geyer, forniti durante il corso. Questi sono stati riutilizzati per semplicità. Nello specifico, il file `rngs.py` è stato impiegato per la generazione di numeri casuali multi-stream; `rvms.py` è stato utilizzato per il calcolo di densità di probabilità, funzioni cumulative e delle loro inverse; infine, `acs.py` ha permesso di stimare l'autocorrelazione delle variabili statistiche di interesse.
- **Output:** questa cartella raccoglie i file `.csv` contenenti le statistiche risultanti dalle diverse simulazioni, oltre ai grafici generati.
- **Simulation:** rappresenta il nucleo del progetto e contiene il codice relativo alle simulazioni principali.
- **Utils:** contiene tutte le funzioni di supporto e le costanti utilizzate nel sistema, come ad esempio il valore del parametro p .

5.1 Gestione degli eventi

La simulazione effettuata è di tipo *next-event*, che comincia al tempo **START** e termina al tempo **STOP**, e può essere distinta in due casi principali:

⁷come definito a pagina 92 nel libro Performance Engineering: Learning Through Applications Using JMT

- **Orizzonte finito:** il tempo della simulazione corrisponde a quello di una giornata (24 ore), con lo STOP fissato a 86400 secondi.
- **Orizzonte infinito:** il tempo di simulazione è posto a `float("inf")` e viene utilizzato per osservare il comportamento asintotico del sistema.

Nel caso di queste due simulazioni, è stato impiegato il valore medio giornaliero del tasso di arrivo, λ_{globale} , al fine di studiare il comportamento complessivo del modello su un'intera giornata. In questo modo è possibile ottenere una visione d'insieme delle prestazioni del sistema sotto un carico medio rappresentativo.

Per una prima simulazione con serventi singoli, gli eventi sono stati gestiti confrontando il tempo della simulazione corrente con i tempi dei possibili eventi:

- arrivo dall'esterno,
- completamento all'edge node,
- completamento al coordinator server edge node,
- completamento al cloud server.

Nella simulazione con **scalabilità** (descritta nel paragrafo 15), sono state considerate tutte le fasce orarie e i relativi picchi di carico, così da condurre un'analisi più dettagliata delle prestazioni del modello in funzione della variabilità giornaliera del traffico.

6 Verifica

In questa fase abbiamo verificato la correttezza del modello computazionale implementato tramite la simulazione ad orizzonte infinito. Il seed base utilizzato è 123456789 con 100 repliche, con l'intervallo di confidenza del 95%.

Edge Node

$$P_c = 0.0$$

$$\lambda = \lambda_{\text{globale}} + \lambda_3 = \lambda_{\text{globale}} + \lambda_1 \cdot P_c = \lambda_{\text{globale}}$$

$$E[S] = 0.5 \text{ s}$$

$$\mu = \frac{1}{E[S]} = \frac{1}{0.5} = 2 \text{ j/s}$$

Formule:

$$\rho = \frac{\lambda}{\mu}$$

$$E(T_S) = \frac{1}{\mu - \lambda}$$

$$E(T_Q) = \frac{\rho \cdot E[S]}{1 - \rho}$$

$$E(N_Q) = \frac{\rho^2}{1 - \rho}$$

$$E(N_S) = \frac{\lambda}{\mu - \lambda}$$

Risultati analitici:

λ (j/s)	ρ	$E(T_S)$ (s)	$E(T_Q)$ (s)	$E(N_Q)$	$E(N_S)$
1.38	0.69	1.6129	1.1129	1.5365	2.2258

Tabella 5: Metriche del sistema M/M/1 con $\lambda_{globale} = 1.38$ j/s.

Ai fini di questa verifica, data l'impossibilità di validare i risultati in modo analitico a causa del complesso meccanismo di feedback, abbiamo simulato il sistema assumendo $P_c = 0$.

Risultati di simulazione:

λ (j/s)	ρ	$E(T_S)$ (s)	$E(T_Q)$ (s)	$E(N_Q)$	$E(N_S)$
1.38	0.690598	1.616963	1.116830	1.542214	2.232812
± 0.000577	± 0.005178	± 0.005031	± 0.007564	± 0.008049	

Tabella 6: Metriche del sistema M/M/1 stimate tramite simulazione con intervallo di confidenza al 95%.

Cloud

$$P_c = 0.4$$

$$\lambda_3 = \lambda_1 \cdot P_c$$

$$E[S] = 0.8 \text{ s}$$

$$\mu = \frac{1}{E[S]} = \frac{1}{0.8} = 1.25 \text{ j/s}$$

Formule:

$$E(T_S) = E[S]$$

$$E(T_Q) = 0$$

$$E(N_S) = \lambda_3 \cdot E[S]$$

dove

$$\lambda_3 = \lambda_1 \cdot P_c = 1.38 \cdot 0.4 = 0.552 \text{ j/s}$$

Risultati analitici:

λ_1 (j/s)	λ_3 (j/s)	ρ	$E(T_S)$ (s)	$E(N_S)$
1.38	0.552	-	0.8	0.4416

Tabella 7: Metriche del nodo Cloud modello M/M/ ∞ con $\lambda_{globale} = 1.38$ j/s.

Il nodo Cloud è modellato come un sistema con un numero infinito di server (M/M/ ∞), pertanto non si forma mai una coda di attesa.

Risultati di simulazione con $\lambda_{globale} = 1.38$ j/s e $P_c = 0.4$:

λ_1 (j/s)	λ_3 (j/s)	ρ	$E(T_S)$ (s)	$E(N_S)$
1.38	0.552	-	0.799755	0.441728
			± 0.000702	± 0.000600

Tabella 8: Metriche del nodo Cloud stimate tramite simulazione con intervallo di confidenza al 95%.

Coordinator Server Edge

$$\begin{aligned}
 P_c &= 0.4 \\
 p_1 &= 0.25 \quad (\text{probabilità pacchetto alta priorità}) \\
 p_2 &= 0.75 \quad (\text{probabilità pacchetto bassa priorità}) \\
 \lambda_2 &= \lambda_1 \cdot (1 - P_c) = 1.38 \cdot 0.6 = 0.828 \text{ job/s} \\
 E(S_1) &= 0.60 \text{ s}, \quad E(S_2) = 0.40 \text{ s}
 \end{aligned}$$

Formule:

$$\begin{aligned}
 E(S) &= p_1 E(S_1) + p_2 E(S_2) = 0.45 \text{ s} \\
 \rho_1 &= \lambda_2 \cdot p_1 \cdot E(S_1) \\
 \rho_2 &= \lambda_2 \cdot p_2 \cdot E(S_2) \\
 \rho &= \rho_1 + \rho_2
 \end{aligned}$$

Dal teorema di KP, sapendo che i tempi di servizio seguono una distribuzione esponenziale e che lo scheduling è astratto e non preemptive, risulta che le prestazioni globali del sistema non dipendono dalla specifica suddivisione in code. Per tale ragione:

$$E[T_Q] = \frac{\rho E(S)}{1 - \rho}$$

$$E[T_S] = E[T_Q] + E(S)$$

$$\begin{aligned}
 E(N_Q) &= \lambda_2 E(T_Q) \\
 E(N_S) &= \lambda_2 E(T_S)
 \end{aligned}$$

Risultati analitici con $\lambda_{globale} = 1.38$ j/s e $P_c = 0.4$:

λ_2	ρ	$E(T_S)$ (s)	$E(T_Q)$ (s)	$E(N_Q)$	$E(N_S)$
0.828	0.3726	0.718	0.268	0.222	0.595

Tabella 9: Metriche del Coordinator Server Edge con $\lambda_{globale} = 1.38$ j/s.

Risultati di simulazione con $\lambda_{globale} = 1.38$ j/s e $P_c = 0.4$:

λ_2	ρ	$E(T_S)$ (s)	$E(T_Q)$ (s)	$E(N_Q)$	$E(N_S)$
0.828	0.362415	0.683407	0.245970	0.203789	0.566205
	± 0.000379	± 0.001046	± 0.000821	± 0.000734	± 0.001032

Tabella 10: Metriche del Coordinator Server Edge stimate tramite simulazione con intervallo di confidenza al 95%.

6.1 Controlli di consistenza

Dai risultati ottenuti è possibile verificare che sono rispettati i seguenti controlli di consistenza:

- $E(T_S) = E(T_Q) + E(S)$
- $0 \leq \rho \leq 1$
- $E(N_S) = E(N_Q) + \rho$

7 Validazione

In questa fase verifichiamo se il modello implementato è coerente con il sistema reale di riferimento e se riproduce correttamente i comportamenti attesi.

Edge Node Con $P_c = 0$, non abbiamo il feedback, quindi ci aspettiamo dei tempi di risposta inferiori. Impostando $P_c = 0.4$ introduciamo il feedback dal Cloud verso l'Edge. Ci si aspetta quindi un peggioramento della saturazione e dei tempi medi, poiché il tasso di arrivo aumenta ulteriormente a causa del feedback introdotto. La simulazione conferma queste previsioni, riproducendo il comportamento atteso confermando la correttezza del modello. Di conseguenza, aumentando ulteriormente il valore di P_c ci aspettiamo un peggioramento ulteriore della saturazione e dei tempi medi, tant'è che la simulazione conferma questa previsione.

Cloud L'ipotesi iniziale è che, poiché il Cloud è un sistema M/M/ ∞ (con un numero infinito di server), non si formano code e la capacità del sistema rimanga sempre disponibile. Pertanto, ci si aspetta che l'aumento del carico non influisca sui tempi di risposta. Con $P_c = 0.4$, la simulazione ha confermato questa ipotesi, poiché i tempi di risposta sono rimasti invariati nonostante l'aumento del carico. Questo comportamento è completamente coerente con le aspettative per un sistema M/M/ ∞ , dove il sistema è in grado di gestire qualsiasi incremento di carico senza influire sui tempi di risposta. Aumentando ulteriormente il valore di P_c ci si aspetta che l'aumento del carico non influisca sui tempi di risposta, tant'è che la simulazione ha confermato questa previsione. Di conseguenza, i tempi di risposta dell'edge peggioreranno, poiché il feedback aumenterà il carico complessivo.

Coordinator Con $P_c = 0.4$, ci si aspettano dei miglioramenti nelle prestazioni rispetto al caso senza feedback. L'aumento di P_c riduce il flusso che arriva al Coordinator, poiché $1 - P_c$ rappresenta la percentuale del traffico diretto al centro. Questo comporta una riduzione dell'utilizzazione e dei tempi di risposta, con un miglioramento delle prestazioni, manifestato da tempi di risposta più bassi, in linea con le aspettative teoriche. La simulazione conferma questa relazione inversamente proporzionale tra P_c e le prestazioni. Analogamente per $P_c = 0$ aumenta il flusso che arriva al Coordinator, quindi ci si aspetta un peggioramento dei tempi di risposta, tant'è che la simulazione ha confermato questa previsione.

8 Design degli esperimenti

La fase di progettazione degli esperimenti si articola nei seguenti tre passaggi principali:

- **Analisi del collo di bottiglia:** questa fase ha l'obiettivo di individuare il centro con la domanda più elevata, utilizzando strumenti di analisi operazionale.
- **Simulazione a orizzonte finito:** in questa fase il sistema viene simulato per un periodo di 24 ore impiegando il metodo delle repliche. Nel nostro caso, sono state effettuate 100 repliche.
- **Simulazione a orizzonte infinito:** questa simulazione prevede l'analisi del sistema su un periodo molto più lungo rispetto a quello reale, utilizzando il metodo dei *Batch Means*. Il run viene suddiviso in gruppi chiamati *batch*, ciascuno composto da un numero definito di job. Nel presente studio sono stati impiegati 128 batch, ognuno contenente 4080 job. Nel relativo capitolo verrà spiegato il motivo del numero e size dei batch utilizzati.

9 Calcolo del seed indipendente tra repliche

Per garantire l'**indipendenza dei seed tra le repliche** abbiamo adottato la tecnica dei jump multipliers descritta durante il corso: partendo da un seme radice, il seme della replica r viene calcolato applicando un salto di $r * J$ passi sul generatore di Lehmer, secondo la formula $x_r = a^{r*j} * x_0 \text{ mod } m$ con parametri standard $m = 2^{31} - 1$ e $a = 48271$

Nel codice questa logica è implementata con una funzione dedicata che utilizza l'esponenziazione modulare per calcolare il moltiplicatore di salto:

```
M = 2**31 - 1
A = 48271 # moltiplicatore standard Lehmer

def lehmer_replica_seed(root_seed: int, J: int, r: int) -> int:
    """Genera un seme indipendente per la replica r."""
    AJ = pow(A, r * J, M) # calcola a^(r*J) mod m
    s = (AJ * root_seed) % M
    return s if s != 0 else 1 # evita lo stato nullo
```

10 Analisi del transitorio

Prima di analizzare nel dettaglio le fasi sopra menzionate, è opportuno condurre uno studio del transitorio per verificare la convergenza del sistema allo stato stazionario. A tal fine, sono state effettuate 10 simulazioni tramite il metodo delle repliche, utilizzando seed differenti e indipendenti tra loro, fissando $P_c = 0.4$ e adottando il tempo di risposta medio come metrica di riferimento per valutare la convergenza del sistema. Inoltre i grafici fanno riferimento ad un periodo di circa 3 giorni ($\approx 300000s.$).

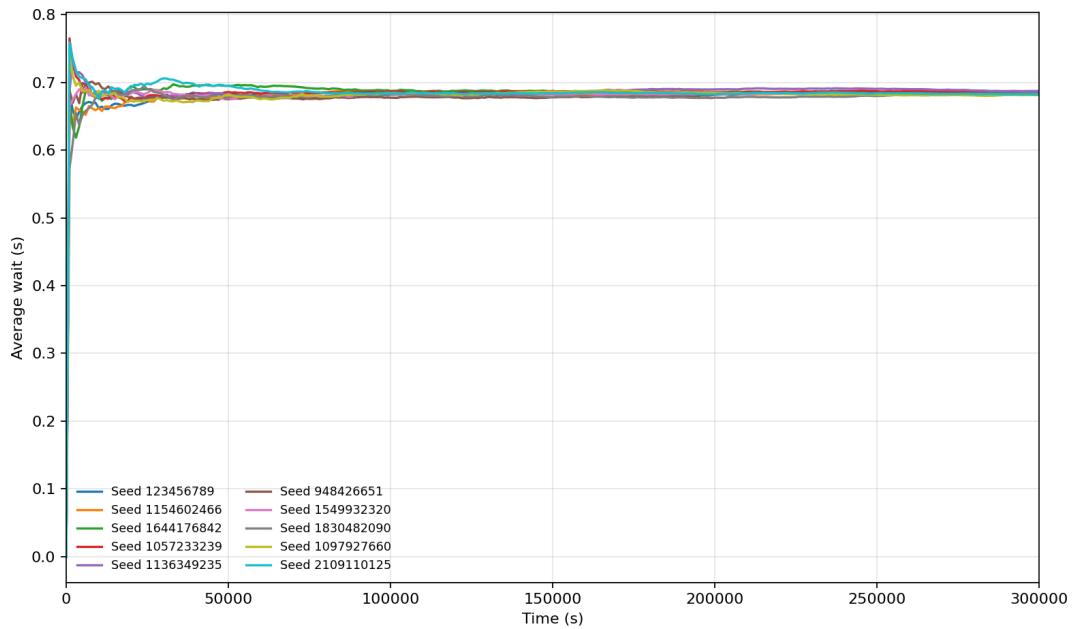


Figura 3: Tempo di risposta medio presso il Coordinator Server.

Il sistema supera la fase transitoria e converge allo stato stazionario in un tempo brevissimo, inferiore ai 50.000 secondi simulati. Le repliche mostrano un andamento stabile e coerente per tutta la durata della simulazione.

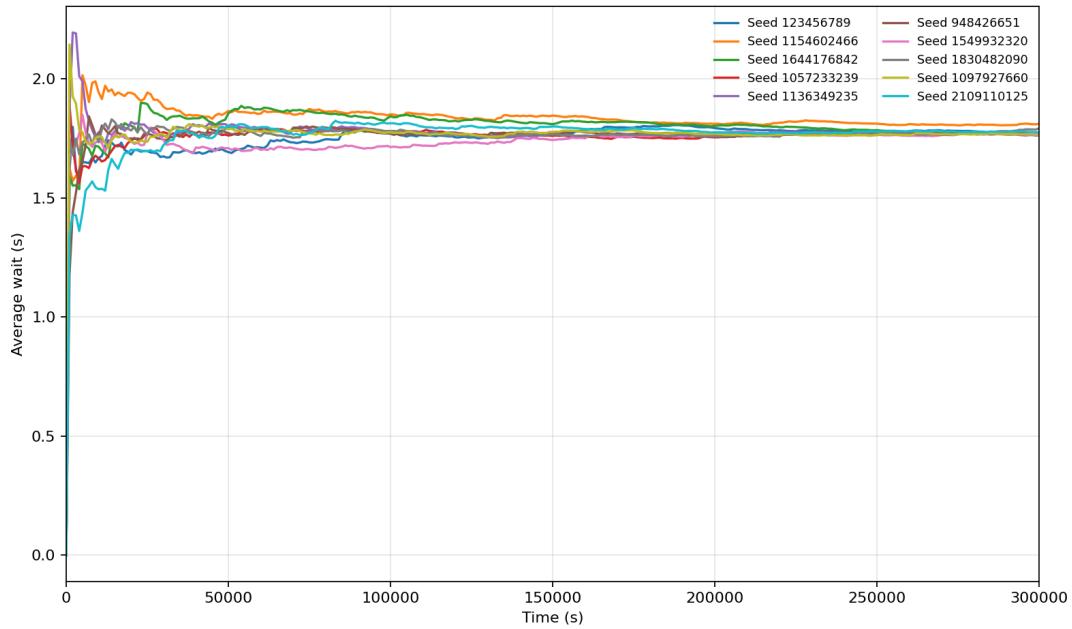


Figura 4: Tempo di risposta medio complessivo presso il nodo Edge

La metrica presenta un transitorio iniziale con una leggera oscillazione fino a circa 150.000 secondi, per poi assestarsi definitivamente su un valore di regime per il rimanente periodo di osservazione.

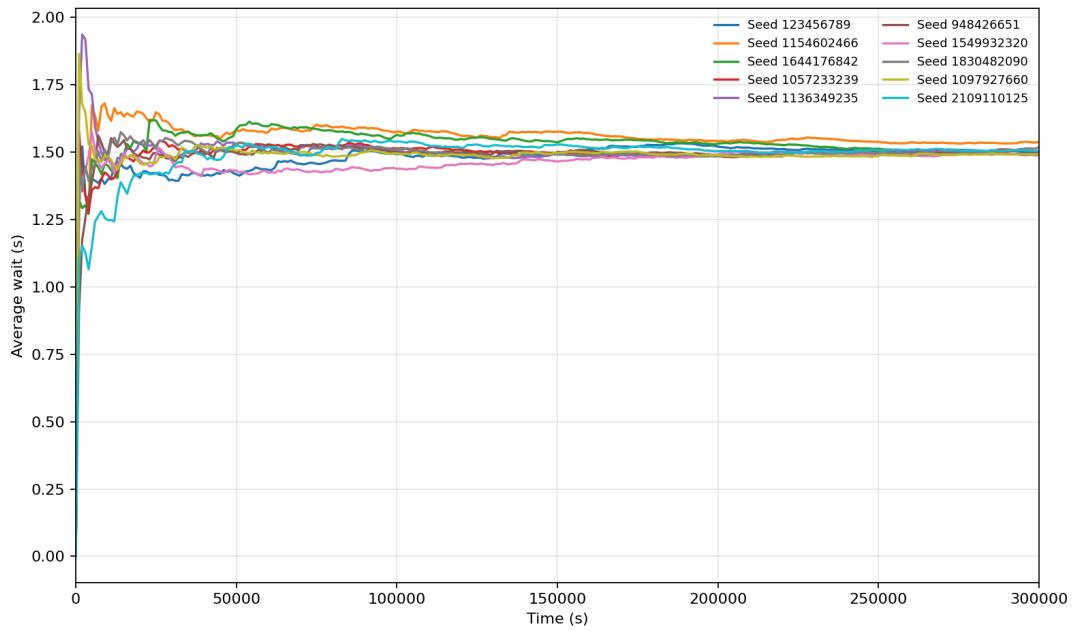


Figura 5: Tempo di risposta medio per il nodo Edge per i pacchetti di classe C.

Questo componente specifico mostra un transitorio più pronunciato rispetto alla media generale, con un picco iniziale che si smorza progressivamente. La piena stabilità è raggiunta dopo circa 200.000 secondi.

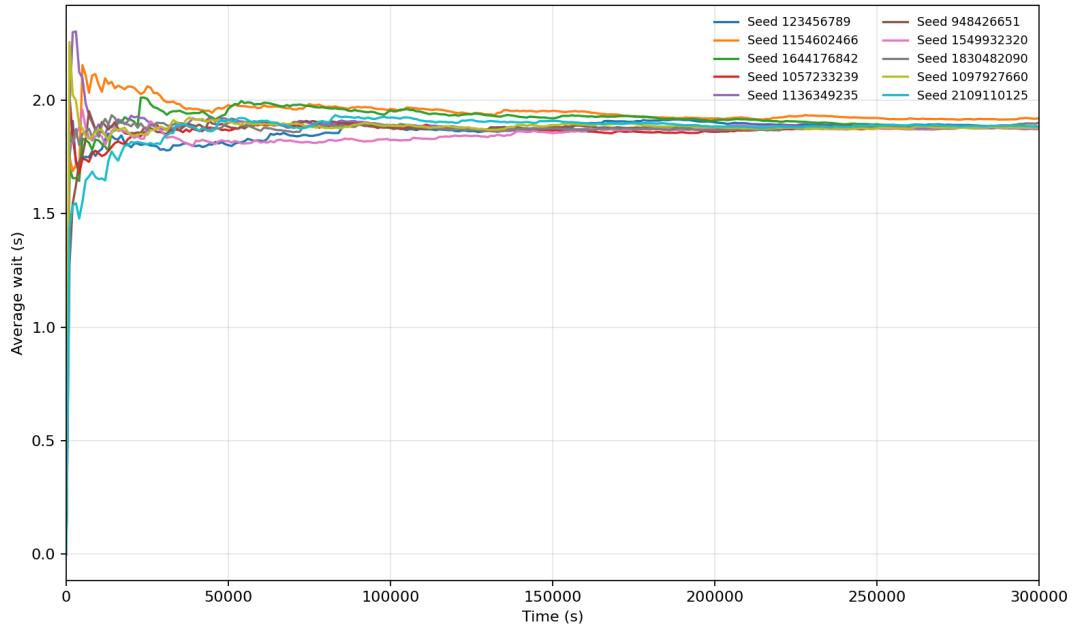


Figura 6: Tempo di attesa medio per il nodo Edge per i pacchetti di classe E.

Il comportamento è simile a quello del nodo C, sebbene le oscillazioni nel transitorio siano di minore entità. Il regime stazionario è consolidato a partire da circa 150.000-200.000 secondi.

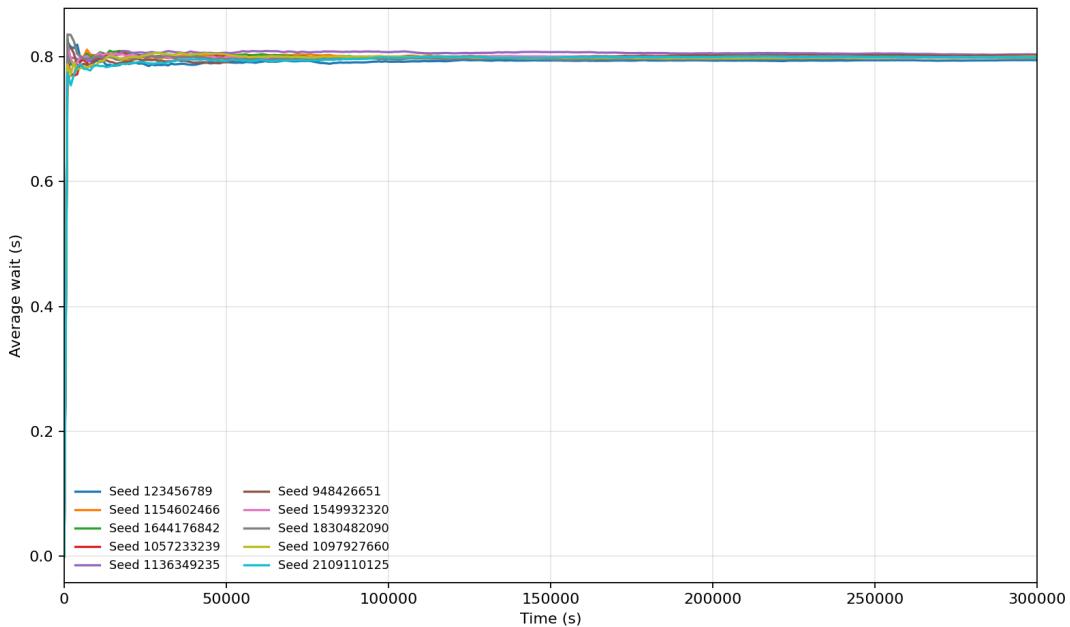
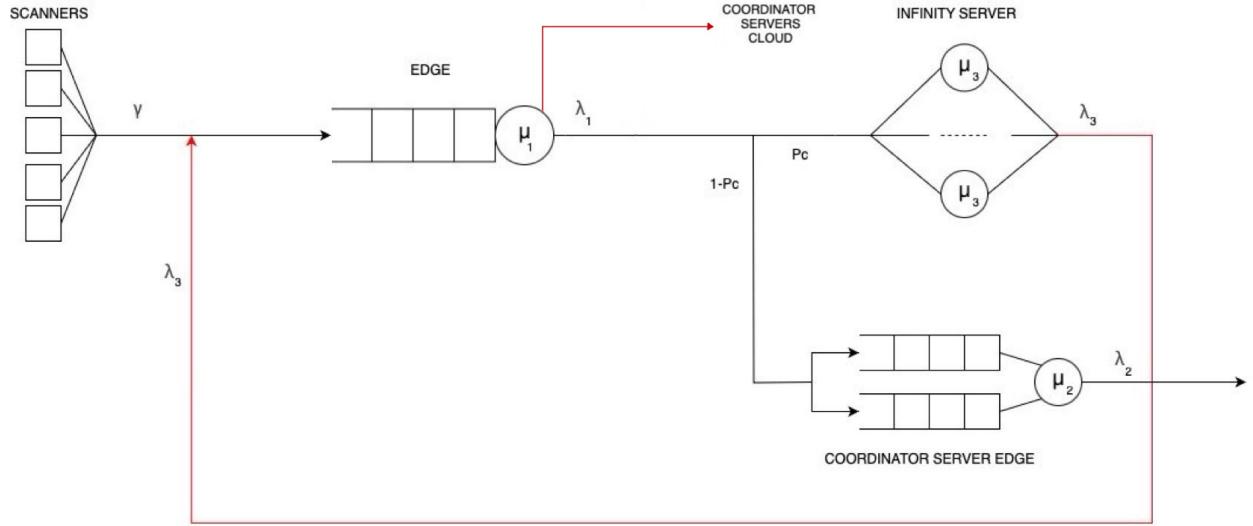


Figura 7: Tempo di attesa medio presso il nodo Cloud

Il componente cloud è caratterizzato da un transitorio molto breve. La convergenza a un valore medio stabile avviene entro i primi 50.000 secondi, mostrando successivamente un andamento piatto e privo di variazioni significative.

In conclusione, l'analisi del transitorio dimostra che tutte le componenti del sistema - Coordinator Server, nodi Edge (nelle loro diverse classi) e nodo Cloud - convergono efficacemente verso un regime stazionario. Sebbene i tempi di assestamento varino tra le diverse entità (dal Cloud, più rapido, ai nodi Edge di classe C, più lenti), tutte le repliche raggiungono una condizione di stabilità ben prima del termine del periodo di osservazione, validando così la correttezza della configurazione simulativa e permettendo un'analisi significativa delle performance di sistema nello stato stazionario.

11 Analisi del collo di bottiglia



Nei calcoli che seguono, indicheremo con γ il tasso di arrivo esterno (jobs/s), con λ_i il tasso di uscita dal centro i -esimo

$$\lambda_1 = \frac{\gamma}{1 - P_c}$$

Questo deriva dal Bilancio dei flussi: il flusso in ingresso all'Edge include sia i job nuovi (γ) che quelli che ritornano con probabilità P_c :

$$\lambda_1 = \gamma + \lambda_3 = \gamma + \lambda_1 \cdot P_c$$

Allora:

$$\lambda_1 \cdot (1 - P_c) = \gamma$$

Quindi:

$$\lambda_1 = \frac{\gamma}{1 - P_c}$$

Invece,

$$\lambda_3 = P_c \cdot \lambda_1$$

Questi sono i job che vengono reindirizzati dall'Edge al Cloud.

Infine,

$$\lambda_2 = (1 - P_c)\lambda_1.$$

Ora definiamo il numero medio di visite che ciascun job compie in ogni centro:

$$v_i = \frac{\lambda_i}{\gamma}$$

Misura quante volte un job , in media, passa dal centro i -esimo rispetto al flusso esterno.
Visite all'Edge:

$$v_1 = \frac{\lambda_1}{\gamma} = \frac{1}{1 - P_c}$$

Visite al Cloud:

$$v_3 = \frac{\lambda_3}{\gamma} = \frac{P_c}{1 - P_c}$$

Visite al Coordinator:

$$v_2 = \frac{\lambda_2}{\gamma} = 1$$

Valori numerici per $p_c = 0.4$:

$$\begin{aligned} v_1 &= \frac{1}{1 - 0.4} = \frac{1}{0.6} = \frac{10}{6} = \frac{5}{3} \approx 1.6667 \\ v_3 &= \frac{0.4}{0.6} = \frac{4}{6} = \frac{2}{3} \approx 0.6667 \\ v_2 &= 1 \end{aligned}$$

Tempi medi di servizio:

Poiché all'Edge Node i job che arrivano dall'esterno richiedono un tempo di servizio pari a 0.5 s, mentre i job che effettuano il feedback richiedono un tempo di servizio pari a 0.1 s, consideriamo come S_1 la media dei tempi di servizio all'Edge Node, ottenuta dalla simulazione, con $P_c = 0.4$, ossia:

$$S_1 = 0.385816 \text{ s}$$

Per il Coordinator Server Edge:

$$S_2 = 0.45 \text{ s}$$

Per il Cloud :

$$S_3 = 0.8 \text{ s}$$

Il demand D_i di ciascun centro è il prodotto tra il numero medio di visite v_i e il tempo medio di servizio S_i :

$$D_i = v_i \cdot S_i$$

In particolare:

$$\begin{aligned} D_1 &= v_1 \cdot S_1 = \frac{1}{1 - P_c} \cdot 0.385816 \\ D_3 &= v_3 \cdot S_3 = \frac{P_c}{1 - P_c} \cdot 0.8 \\ D_2 &= v_2 \cdot S_2 = 1 \cdot 0.45 = 0.45 \end{aligned}$$

Valori numerici per $P_c = 0.4$:

$$\begin{aligned} D_1 &= 0.6430 \text{ s} \\ D_3 &= \frac{8}{15} \approx 0.5333 \text{ s} \\ D_2 &= 0.45 \end{aligned}$$

Identificazione del **collo di bottiglia**:

Confrontiamo i centri a *capacità finita* ($E[T_{s,\text{Edge-E}}]$ Coordinator):

$$D_1 = 0.6430 \text{ s} \quad \text{vs} \quad D_2 = 0.45 \text{ s.}$$

Poiché $D_1 > D_2$, il collo di bottiglia è : Edge Node

Il Cloud è *Infinite Server* (non accoda), quindi non può essere collo di bottiglia anche se D_3 è elevato.

Throughput massimo Per un centro a servente finito vale:

$$X_i = \frac{1}{D_i}.$$

Il throughput massimo del sistema è uguale a:

$$X_{\max} = \frac{1}{\max_{i \in \{1,2\}} \{D_i\}}$$

Calcoliamo i reciproci usando le forme frazionarie:

Edge:

$$\frac{1}{D_1} = \frac{1}{0.6430} = 1.5552 \text{ job/s}$$

Coordinator:

$$\frac{1}{D_2} = \frac{1}{0.45} \approx 2.2222 \text{ job/s}$$

Confronto: $1.5552 < 2.2222$, quindi

$$X_{\max} = 1.5552 \text{ job/s}$$

determinato dall'Edge (collo di bottiglia).

12 Intervallo di confidenza

L'intervallo di confidenza è stato determinato utilizzando la formula:

$$IC = t^* \cdot \frac{s}{\sqrt{n-1}}$$

dove:

- s rappresenta la deviazione standard del campione;
- $n - 1$ indica i gradi di libertà, ossia la dimensione del campione meno uno (repliche per l'orizzonte finito, medie di batch per l'orizzonte infinito);
- t^* è il valore critico ottenuto dalla distribuzione di Student con $n - 1$ gradi di libertà.

Il valore t^* è stato calcolato mediante la funzione inversa della distribuzione di Student:

$$t^* = \text{idfStudent}(n - 1, 1 - \alpha/2)$$

dove il livello di significatività è stato fissato a $\alpha = 0.05$, corrispondente a un intervallo di confidenza del 95%.

13 Simulazione Orizzonte Finito

Nel progetto, lo *studio ad orizzonte finito* è stato realizzato con l'obiettivo di osservare il comportamento del sistema su un intervallo temporale realistico di 24 ore, mantenendo il controllo sulle condizioni di carico e consentendo l'analisi delle fasi transitorie. L'orizzonte massimo simulato è fissato a **STOP = 86400** secondi, corrispondente a un'intera giornata.

L'implementazione di questo approccio adotta un *tasso di arrivo medio fisso* (**forced_lambda**) per l'intera durata della simulazione. Questa scelta consente di studiare il comportamento del sistema sotto condizioni di carico medio ottenuto durante una giornata lavorativa.

La simulazione è implementata con un *motore next event-driven* che avanza il tempo in modo discreto, saltando da un evento al successivo. Questo approccio garantisce efficienza computazionale evitando di processare istanti temporali in cui non occorre alcun evento significativo. Il cuore della simulazione è rappresentato dal ciclo principale che gestisce sequenzialmente gli eventi di arrivo e completamento:

```
while (stats.t.arrival < stop) or (jobs_in_system > 0):
    # Determina il prossimo evento
    stats.t.next = Min(stats.t.arrival,
                        stats.t.completion_edge,
                        stats.t.completion_cloud,
                        stats.t.completion_coord)

    # Avanza il tempo all'evento successivo
    dt = stats.t.next - stats.t.current
    # Aggiorna statistiche accumulate
    stats.t.current = stats.t.next

    # Gestione dell'evento
    if stats.t.current == stats.t.arrival:
        handle_arrival()
    elif stats.t.current == stats.t.completion_edge:
        handle_edge_completion()
    elif stats.t.current == stats.t.completion_cloud:
        handle_cloud_completion()
    elif stats.t.current == stats.t.completion_coord:
        handle_coord_completion()
```

La gestione degli eventi segue una logica precisa:

- **Arrivi:** I job arrivano secondo un processo di Poisson con tasso λ costante. Ogni arrivo incrementa il contatore dei job nel sistema e, se il server Edge è libero, avvia immediatamente il servizio.
- **Completamenti Edge:** Al termine del servizio Edge, il job viene instradato verso il Cloud (con probabilità P_C) o verso il Coordinator (con probabilità $1 - P_C$), con ulteriore routing tra le classi P1-P4 nel caso del Coordinator.
- **Completamenti Cloud:** I job completati al Cloud ritornano all'Edge come job di classe C per l'elaborazione finale.
- **Completamenti Coordinator:** I job processati dal Coordinator lasciano definitivamente il sistema.

La *riproducibilità* è garantita dall'inizializzazione controllata dei generatori di numeri casuali. Il seed globale `cs.SEED` viene impostato una volta all'inizio del modulo, mentre stream separati gestiscono le diverse componenti stocastiche (arrivi, servizi, routing). Prima di ogni replica, lo stato temporaneo degli arrivi viene resettato con `reset_arrival_temp()` per garantire condizioni iniziali identiche.

La struttura operativa prevede l'esecuzione di multiple repliche indipendenti, ciascuna con un seed diverso derivato deterministicamente dal seed base. Per ogni replica, la simulazione procede fino al tempo `STOP`, smaltendo eventuali job rimasti in coda al termine dell'orizzonte temporale:

```
# Inizializzazione globale degli stream
plantSeeds(cs.SEED)

for rep in range(cs.REPLICATIONS):
```

```

seed = getSeed() # Seed per la replica corrente
reset_arrival_temp()
stats = SimulationStats()
stats.reset(cs.START)

# Configurazione iniziale
stats.t.arrival = GetArrival(stats.t.current, forced_lambda)

# Esegui la simulazione
results, stats = finite_simulation(cs.STOP, forced_lambda=lam)
results.update({"seed": seed, "lambda": lam})
write_file(results, "finite_statistics.csv")

```

L'impiego di *repliche multiple* (definito da `cs.REPLICATIONS = 100`) è essenziale per garantire robustezza statistica ai risultati. Ogni replica produce stime indipendenti delle metriche di performance, permettendo il calcolo di intervalli di confidenza e la riduzione della variabilità casuale insita nelle simulazioni stocastiche.

Le metriche raccolte includono tempi di risposta, tempi di attesa in coda, utilizzo dei server, popolazione media nei centri e throughput, sia aggregate che disaggregate per classe di job (E e C). Questa ricca set di dati consente un'analisi approfondita del comportamento del sistema sotto diverse condizioni di carico.

In definitiva, la combinazione di un motore event-driven efficiente, tassi di arrivo costanti, gestione controllata della casualità e repliche multiple rende lo studio ad orizzonte finito uno strumento preciso e affidabile per l'analisi delle performance del sistema in scenari realistici e stabili.

Qui di seguito i grafici ottenuti dalla simulazione ad orizzonte finito, ottenuti con $P_c = 0.4$, $\lambda_{\text{globale}} = 1.38j/s$ e con 100 repliche con seed indipendente:

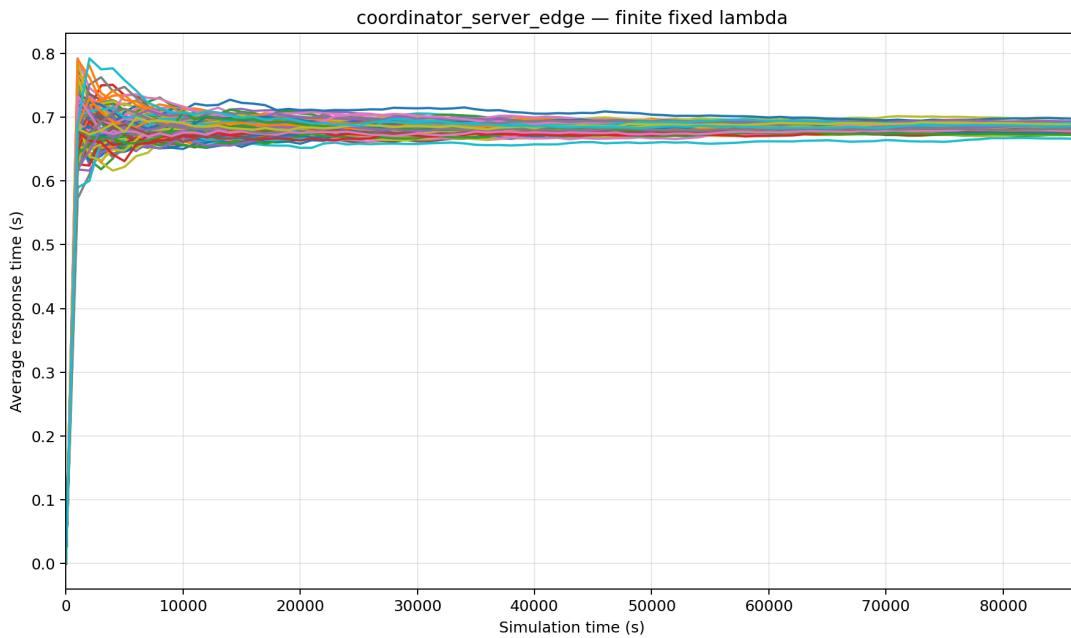


Figura 8: Tempo di risposta medio presso il Coordinator Server - Orizzonte Finito

Il Coordinator Server mostra un transitorio iniziale molto breve, convergendo rapidamente allo stato stazionario entro i primi 25.000 secondi. L'andamento si mantiene stabile per tutta la durata delle 24 ore.

Metrica	Valore	Intervallo di Confidenza (95%)
$E(T_S)$	0.683407	± 0.001046
$E(T_Q)$	0.245970	± 0.000821
$E(S)$	0.437437	± 0.000338
ρ	0.362415	± 0.000379
$E(N_S)$	0.566205	± 0.001032
$E(N_Q)$	0.203789	± 0.000734

Tabella 11: Metriche del Coordinator Server - Orizzonte Finito

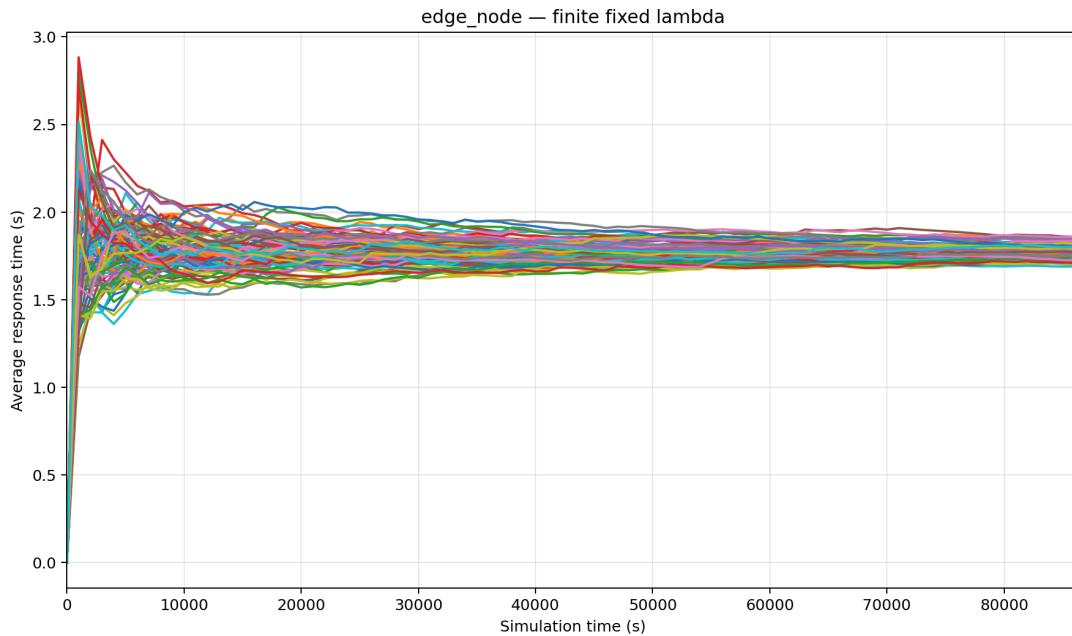


Figura 9: Tempo di risposta medio complessivo presso il nodo Edge - Orizzonte Finito

Il nodo Edge complessivo presenta un transitorio più marcato, con oscillazioni che si smorzano progressivamente fino al raggiungimento dello stato stazionario attorno ai 50.000 secondi. La stabilità successiva dimostra una buona capacità del sistema di gestire il carico di lavoro.

Metrica	Valore	Intervallo di Confidenza (95%)
$E(T_S)$	1.774659	± 0.007308
$E(T_Q)$	1.388843	± 0.007211
$E(S)$	0.385816	± 0.000182
ρ	0.745843	± 0.000609
$E(N_S)$	3.430833	± 0.015577
$E(N_Q)$	2.684990	± 0.015070

Tabella 12: Metriche complessive del nodo Edge - Orizzonte Finito

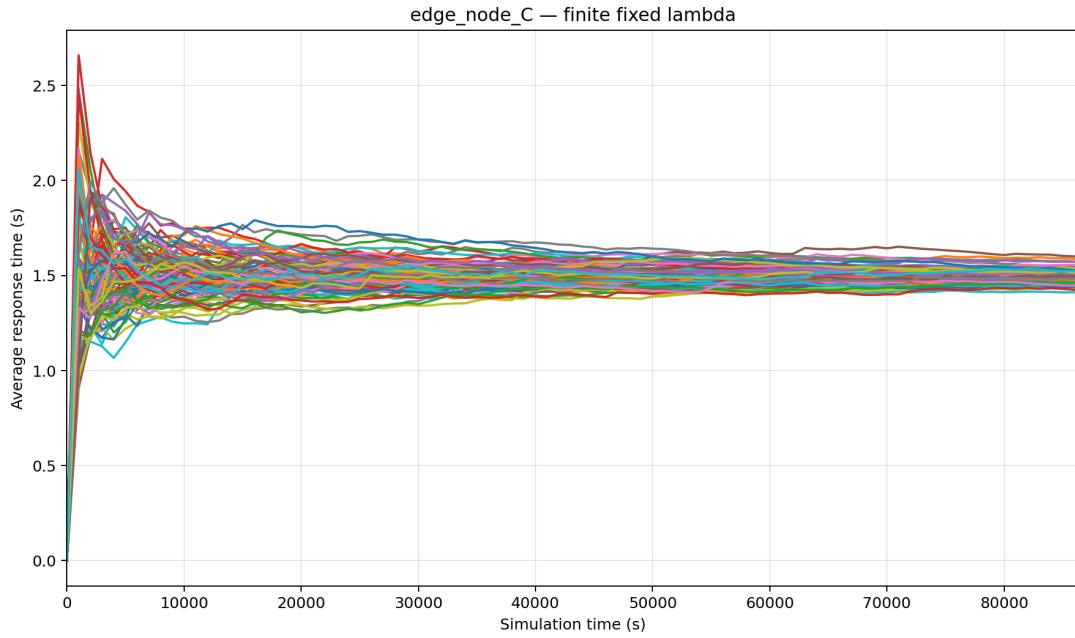


Figura 10: Tempo di risposta medio per pacchetti di classe C - Orizzonte Finito

I pacchetti di classe C mostrano un comportamento caratterizzato da un transitorio iniziale con elevata variabilità, che si stabilizza gradualmente dopo circa 40.000 secondi. La maggiore durata del transitorio riflette la natura intermittente di questo tipo di traffico.

Metrica	Valore	Intervallo di Confidenza (95%)
$E(T_S)$	1.500010	± 0.007480
$E(T_Q)$	1.399990	± 0.007476
ρ	0.055244	± 0.000072
$E(N_S)$	0.828544	± 0.004541
$E(N_Q)$	0.773300	± 0.004509

Tabella 13: Metriche per pacchetti di classe C - Orizzonte Finito

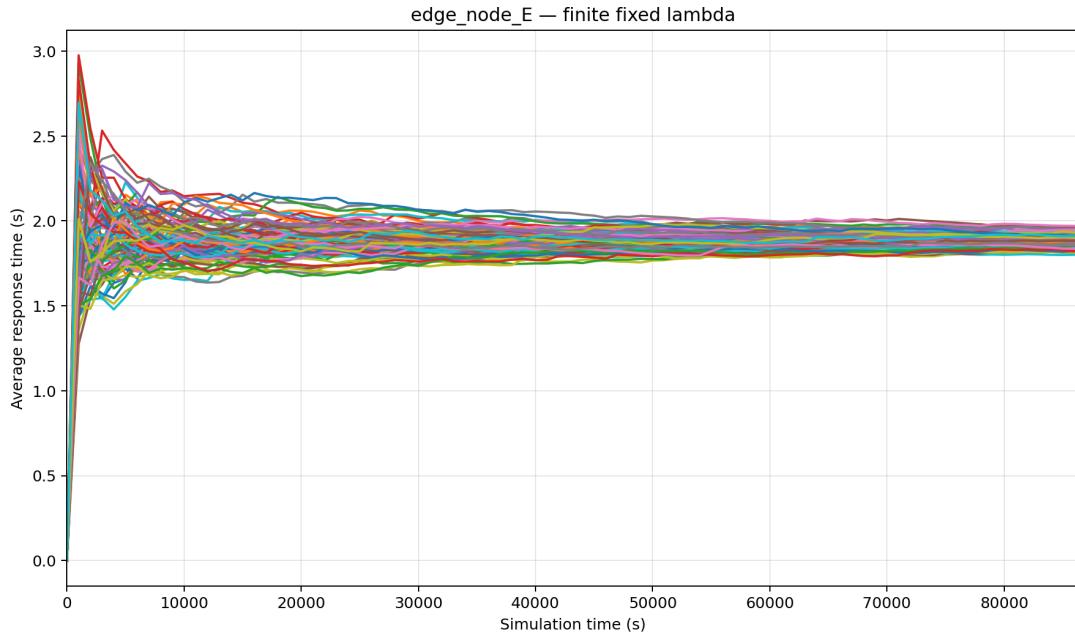


Figura 11: Tempo di risposta medio per pacchetti di classe E - Orizzonte Finito

I pacchetti di classe E dimostrano un transitorio più regolare rispetto alla classe C, raggiungendo lo stato stazionario entro 35.000 secondi. L'andamento stabile successivo indica una gestione efficiente del traffico principale del sistema.

Metrica	Valore	Intervallo di Confidenza (95%)
$E(T_S)$	1.884516	± 0.007289
$E(T_Q)$	1.384382	± 0.007149
ρ	0.690599	± 0.000577
$E(N_S)$	2.602289	± 0.011135
$E(N_Q)$	1.911690	± 0.010654

Tabella 14: Metriche per pacchetti di classe E - Orizzonte Finito

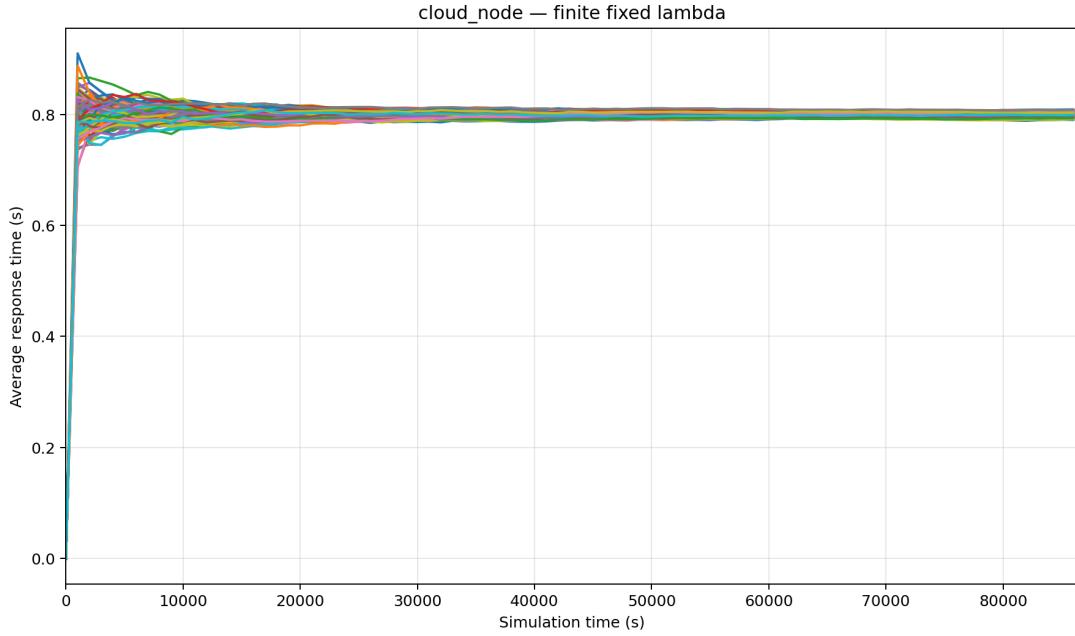


Figura 12: Tempo di risposta medio presso il nodo Cloud - Orizzonte Finito

Il nodo Cloud mostra il transitorio più breve tra tutte le componenti, convergendo allo stato stazionario in meno di 10.000 secondi. La rapida stabilizzazione e l'andamento piatto successivo evidenziano l'elevata efficienza del centro cloud.

Metrica	Valore	Intervallo di Confidenza (95%)
$E(T_S)$	0.799755	± 0.000702
$E(T_Q)$	0.000000	± 0.000000
ρ	0.441728	± 0.000600
$E(N_S)$	0.441728	± 0.000600
$E(N_Q)$	0.000000	± 0.000000

Tabella 15: Metriche del nodo Cloud - Orizzonte Finito

L'analisi complessiva del transitorio dimostra una convergenza differenziata ma efficace verso lo stato stazionario da parte di tutte le componenti del sistema. Il Coordinator Server e il nodo Cloud evidenziano le performance più efficienti, raggiungendo la stabilizzazione rispettivamente entro 25.000 secondi e in meno di 10.000 secondi, confermando la loro capacità di gestire rapidamente il carico di lavoro.

Il nodo Edge complessivo richiede un periodo transitorio più esteso (circa 50.000 secondi), mostrando oscillazioni iniziali più marcate che si smorzano progressivamente. L'analisi disaggregata rivela un comportamento differenziato tra le classi di traffico: i pacchetti di classe E, rappresentanti il traffico principale, convergono stabilmente entro 35.000 secondi, mentre quelli di classe C, caratterizzati da natura più intermittente, richiedono circa 40.000 secondi per raggiungere la piena stabilità.

Tutti i componenti raggiungono e mantengono condizioni di stabilità ben prima del termine delle 24 ore simulate, validando la robustezza del sistema e l'affidabilità delle metriche di performance raccolte per l'analisi dello stato stazionario.

14 Simulazione ad Orizzonte Infinito

Lo studio a **orizzonte infinito** è stato introdotto nel simulatore con lo scopo di analizzare il comportamento del sistema in **regime stazionario**.

Il principio metodologico seguito è quello della tecnica dei *batch-means*, in cui il processo simulato non viene mai interrotto, ma viene suddiviso in lotti (batch) successivi; ogni lotto raccoglie un insieme di osservazioni statisticamente omogenee, che vengono poi utilizzate per stimare le metriche di interesse.

Nel codice, la configurazione di questo meccanismo è centralizzata in `constants.py`, dove vengono definiti il numero totale di batch da simulare (`K`) e la dimensione di ciascun batch (`B`), misurata in **numero di job arrivati** piuttosto che in tempo simulato:

Listing 1: Parametri per simulazione infinita

```
# Parametri per simulazione infinita
K = 128 # numero di batch
B = 4080 # dimensione di ciascun batch
```

Per la scelta del valore di `B` abbiamo eseguito il programma `acs.py` già fornito, aumentando tale valore man mano finché l'autocorrelazione tra i vari batch non è scesa sotto a 0.2, come proposto nel libro di testo “Discrete Event System Simulation” [ref13].

Questa scelta — fissare il batch per numero di arrivi anziché per durata temporale — garantisce che ogni lotto contenga una quantità simile di informazioni statistiche, anche se la durata in tempo reale di un batch può variare in base al carico di lavoro e alla configurazione corrente. La simulazione non ha quindi un orizzonte temporale di stop: il parametro `STOP_INFINITE` è impostato a infinito, e l'avanzamento dipende unicamente dal raggiungimento della dimensione `B` in termini di job processati.

L'implementazione della simulazione a orizzonte infinito si trova nella funzione `infinite_simulation` di `simulator.py`. Questa funzione esegue un ciclo principale che continua fino a quando il numero di batch raccolti non raggiunge `K`. All'interno di ogni batch, si procede a simulare eventi tramite la funzione `execute` finché non si raggiunge la soglia di `B` arrivi:

Listing 2: Logica principale della simulazione infinita

```
while len(batch_stats.edge_wait_times) < cs.K:
    while stats.job_arrived < cs.B:
        execute(stats, cs.STOP_INFINITE, forced_lambda)

    stop_time = stats.t.current - start_time
    start_time = stats.t.current

    stats.calculate_area_queue()
    results = return_stats(stats, stop_time, seed)
    results_list.append(results)
    append_stats(batch_stats, results, stats)

    stats.reset_infinite()
```

Al termine di ogni batch, vengono calcolate le aree delle code (`calculate_area_queue`), estratte le statistiche (`return_stats`) e memorizzati i risultati in una struttura dedicata (`append_stats`). Il reset tra batch non reimposta il tempo globale, né gli arrivi, né lo stato delle code o dei job in corso: azzera unicamente i contatori e le aree utilizzate per calcolare le statistiche del batch. In questo modo il sistema evolve senza soluzione di continuità, mentre le misure statistiche vengono raccolte in modo indipendente per ciascun lotto.

Il reset è definito come segue:

Listing 3: Reset dello stato tra batch consecutivi

```

def reset_infinite(self):
    self.job_arrived = 0

    self.index_edge = 0
    self.index_cloud = 0
    self.index_coord = 0

    self.count_E = 0
    self.count_C = 0

    self.count_E_P1 = 0
    self.count_E_P2 = 0
    self.count_E_P3 = 0
    self.count_E_P4 = 0

    self.index_E = 0
    self.index_C = 0

    self.area_edge = Track()
    self.area_cloud = Track()
    self.area_coord = Track()
    self.area_E = Track()
    self.area_C = Track()

    self.index_edge_E = 0
    self.index_edge_C = 0

```

Si noti che l'orologio degli arrivi (`arrival_temp`) viene azzerato solo all'inizio della simulazione tramite `reset_arrival_temp`, mentre non viene mai reimpostato tra un batch e l'altro: in questo modo il processo di arrivo rimane continuo e i lotti successivi rappresentano finestre statistiche dello stesso flusso, senza discontinuità artificiali.

Il motore vero e proprio della simulazione, `execute`, rimane identico rispetto all'orizzonte finito: si tratta di un **next-event engine** che sceglie il prossimo evento tra arrivo e completamento dei vari nodi, avanza il tempo simulato al minimo tra questi, e aggiorna le aree secondo la formula integrale di Little. Inoltre, nel codice è presente anche una raccolta opzionale di statistiche transitorie (dump ogni 1000 secondi di simulazione), utilizzata per analizzare l'evoluzione temporale delle metriche ma non essenziale ai fini della stima a regime.

Un estratto della selezione dell'evento imminente è riportato di seguito:

Listing 4: Scelta del prossimo evento e aggiornamento aree

```

stats.t.next = Min(stats.t.arrival,
                    stats.t.completion_edge,
                    stats.t.completion_cloud,
                    stats.t.completion_coord)

dt = stats.t.next - stats.t.current
if dt < 0:
    dt = 0.0

if stats.number_edge > 0:
    stats.area_edge.node += dt * stats.number_edge
if stats.number_cloud > 0:
    stats.area_cloud.node += dt * stats.number_cloud
if stats.number_coord > 0:
    stats.area_coord.node += dt * stats.number_coord

```

```

if stats.number_E > 0:
    stats.area_E.node += dt * stats.number_E
if stats.number_C > 0:
    stats.area_C.node += dt * stats.number_C
...
stats.t.current = stats.t.next

```

La generazione degli arrivi avviene tramite la funzione `GetArrival`.

Ogni chiamata somma al tempo dell'ultimo arrivo un intervallo casuale esponenziale con parametro λ , che può essere fissato (`forced_lambda`) oppure dipendere dal tempo corrente (`GetLambda`). In questo modo il sistema viene alimentato da un flusso potenzialmente infinito di job: nella simulazione a orizzonte infinito, infatti, non esiste un tempo massimo di stop e gli arrivi continuano indefinitamente, finché non viene raggiunta la dimensione B del batch.

Listing 5: Generazione degli arrivi

```

def GetArrival(current_time, forced_lambda=None):

    global arrival_temp
    selectStream(0)
    lam = forced_lambda if forced_lambda is not None else GetLambda(current_time)
    arrival_temp += Exponential(1 / lam)
    return arrival_temp

```

In sintesi, questa architettura consente di ottenere misure stabili del comportamento di lungo periodo senza duplicare la logica della simulazione. La scelta di un reset “soft” garantisce che i batch rappresentino un processo continuo, mentre la segmentazione per numero di arrivi assicura coerenza statistica tra i lotti. La tecnica dei batch-means è così applicata in maniera pulita ed efficiente, adattandosi perfettamente alla struttura event-driven del simulatore.

I risultati della simulazione con $P_c = 0.4$ e $\lambda_{\text{globale}} = 1.38j/s$ sono i seguenti:

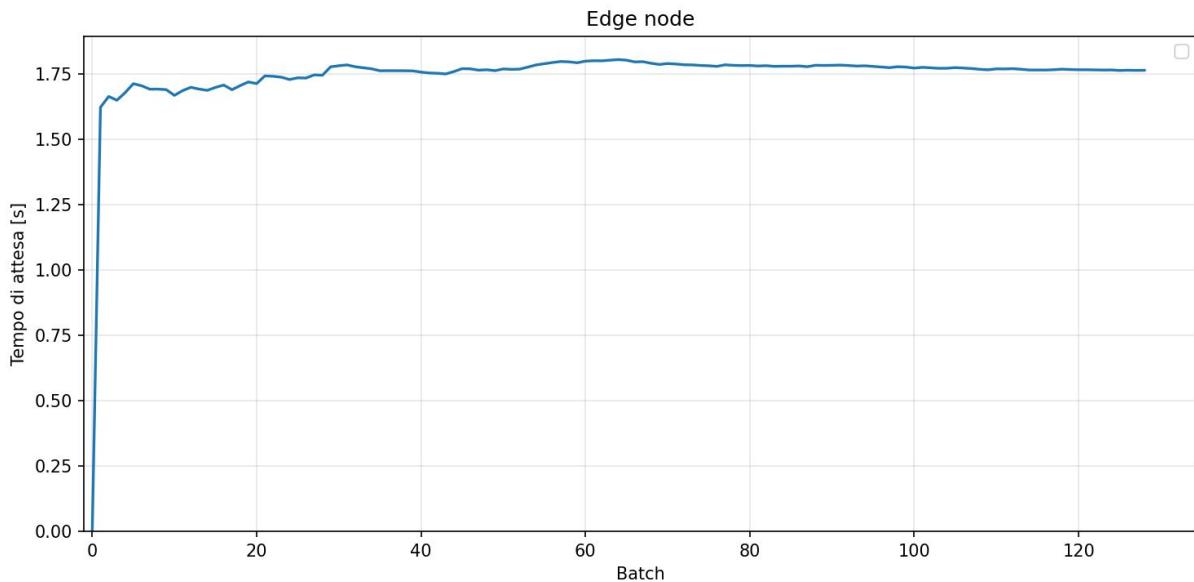


Figura 13: Tempo di risposta medio presso il nodo Edge

Il grafico mostra l'evoluzione del tempo medio di attesa al nodo Edge, stimato con tecnica batch-means. Nella parte iniziale (primi batch) è evidente il transitorio. Dopo circa 20 batch, la curva si stabilizza attorno a 1.75 secondi, indicando che il sistema ha raggiunto il regime

stazionario. Le leggere fluttuazioni successive sono dovute alla variabilità intrinseca del processo simulato.

Metrica (Edge)	Valore medio	IC 95%
$E(T_S)$	1.764	± 0.038
$E(T_Q)$	1.378	± 0.038
ρ	0.745	± 0.003
$E(N_S)$	3.407	± 0.079
$E(N_Q)$	2.662	± 0.077

Tabella 16: Metriche principali del nodo Edge nella simulazione a orizzonte infinito (media \pm intervallo di confidenza al 95%).

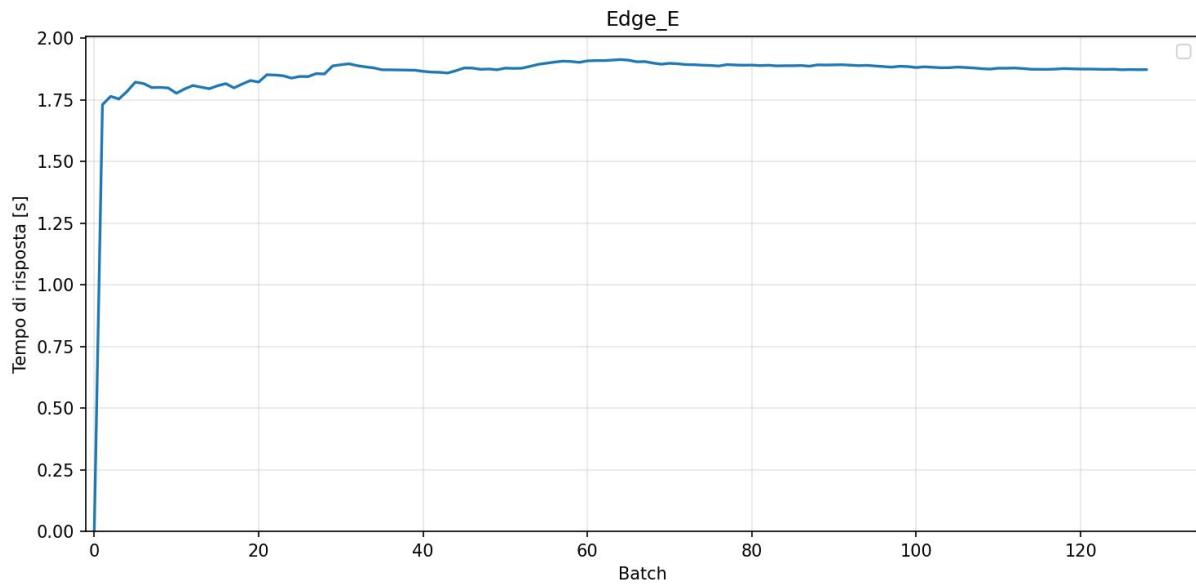


Figura 14: Tempo di risposta medio presso il nodo Edge per i pacchetti di Classe E

Il grafico relativo al nodo Edge per la classe E mostra il tempo medio di risposta in funzione dei batch. Dopo una breve fase di crescita iniziale, la curva si stabilizza attorno a 1.87 secondi, in linea con il valore medio stimato dalla simulazione. Questo conferma che il metodo batch-means ha raggiunto il regime stazionario e che le misure ottenute sono affidabili.

Metrica (Edge-E)	Valore medio	IC 95%
$E(T_S)$	1.874	± 0.038
$E(T_Q)$	1.373	± 0.037
ρ	0.690	± 0.003
$E(N_S)$	2.586	± 0.057
$E(N_Q)$	1.896	± 0.054

Tabella 17: Metriche principali per i job di classe E al nodo Edge (media \pm intervallo di confidenza al 95%).

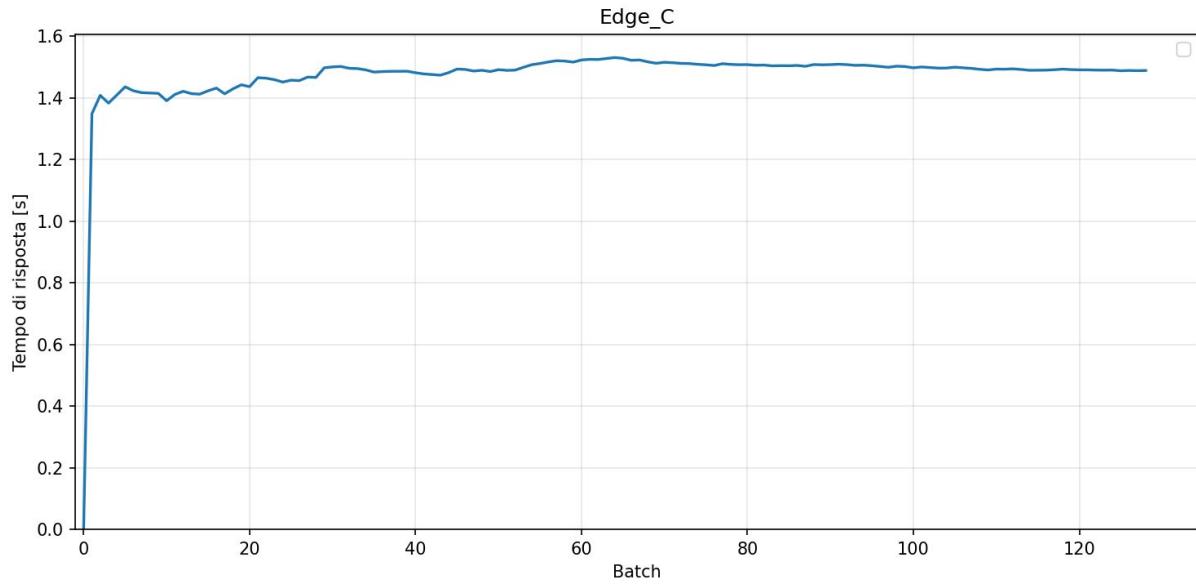


Figura 15: Tempo di risposta medio presso il nodo Edge per i pacchetti di Classe C

Il grafico del tempo medio di risposta al nodo Edge per la classe C mostra un iniziale aumento dovuto al transitorio, seguito da una stabilizzazione attorno a 1.49 secondi. Questo valore è coerente con le stime ottenute dall'analisi statistica, confermando che il sistema raggiunge il regime stazionario dopo circa 20 batch. Le variazioni residue sono imputabili alla natura stocastica del processo simulato.

Metrica (Edge-C)	Valore medio	IC 95%
$E(T_S)$	1.489	± 0.038
$E(T_Q)$	1.390	± 0.038
ρ	0.055	± 0.0003
$E(N_S)$	0.821	± 0.023
$E(N_Q)$	0.766	± 0.023

Tabella 18: Metriche principali per i job di classe C al nodo Edge (media \pm intervallo di confidenza al 95%).

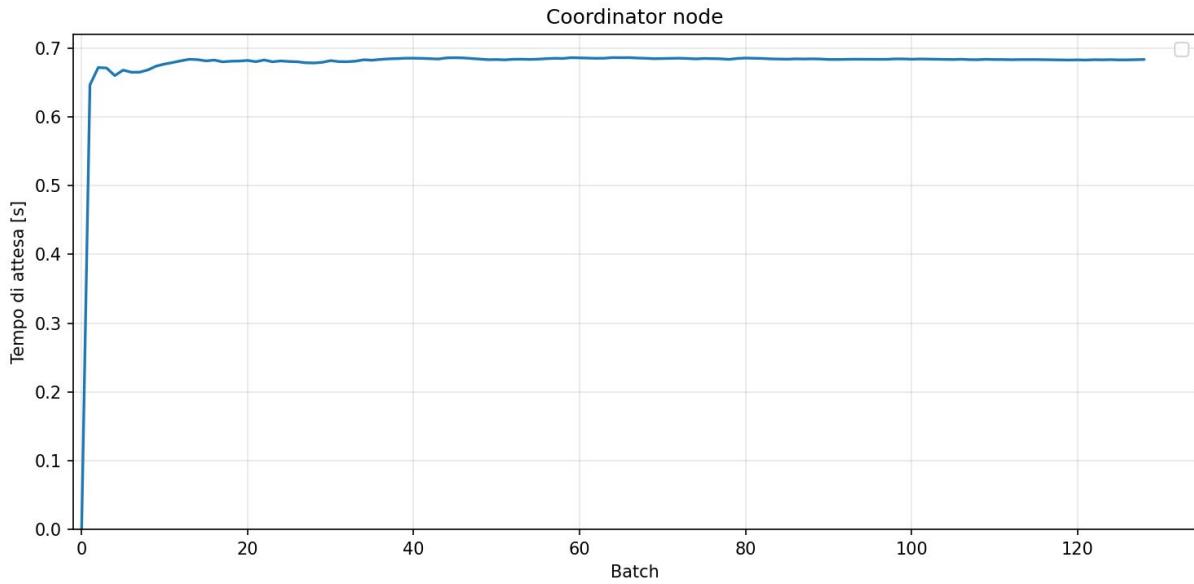


Figura 16: Tempo di risposta medio presso il nodo Coordinator Server Edge

Il grafico mostra che, per il nodo Coordinator, il tempo di attesa si stabilizza rapidamente e rimane costante a regime, segno di un comportamento stazionario e prevedibile in condizioni di orizzonte infinito.

Metrica (Coord)	Valore Medio	IC (95%)
$E(T_S)$	0.683424	± 0.005742
$E(T_Q)$	0.246348	± 0.004873
ρ	0.362171	± 0.001885
$E(N_S)$	0.566551	± 0.006067
$E(N_Q)$	0.204380	± 0.004521

Tabella 19: Metriche per il nodo Coordinator – Orizzonte Infinito

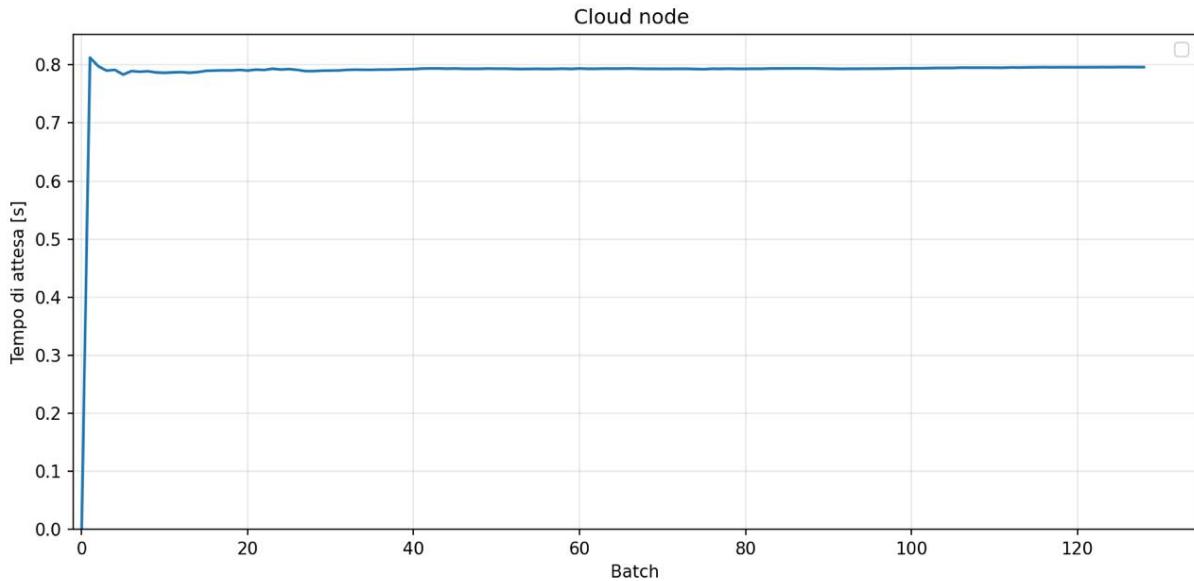


Figura 17: Tempo di risposta medio presso il nodo Cloud

Il grafico conferma che, per il nodo cloud, il sistema raggiunge rapidamente l'equilibrio e rimane stabile, garantendo un tempo di attesa prevedibile per i job in orizzonte infinito.

Metrica (Cloud)	Valore Medio	IC (95%)
$E(T_S)$	0.796145	± 0.003467
ρ	0.438196	± 0.002648
$E(N_S)$	0.438196	± 0.002648

Tabella 20: Metriche per il nodo Cloud – Orizzonte Infinito

La simulazione a orizzonte infinito mostra che il sistema raggiunge rapidamente il regime stazionario: dopo pochi batch i tempi medi di risposta e di attesa si stabilizzano e restano costanti, confermando la stabilità del modello.

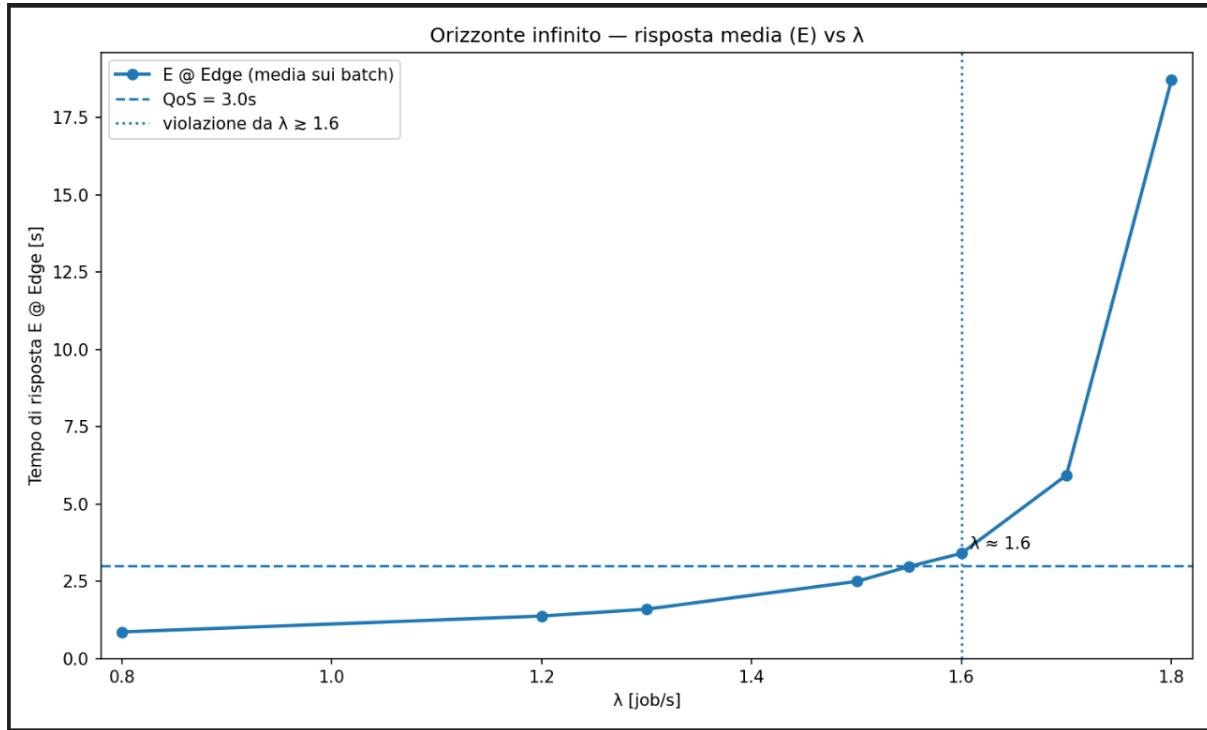
Il nodo Edge è quello più carico, con tempi medi di risposta attorno a 1.76 s e un livello di utilizzo vicino alla saturazione. Le classi E soffrono di più, con tempi di risposta più alti, mentre le classi C restano poco impattate.

I nodi Coordinator e Cloud operano invece con tempi di risposta molto più bassi e utilizzazioni moderate, senza code significative.

Di seguito i valori dell'autocorrelazione per le metriche d'interesse:

- edge_avg_wait: n=128, mean=1.7639, sd=0.2194, rho(1)=0.039, rho(5)=-0.027
- edge_E_avg_response: n=128, mean=1.8735, sd=0.2204, rho(1)=0.050, rho(5)=-0.019
- edge_C_avg_response: n=128, mean=1.4894, sd=0.2203, rho(1)=0.015, rho(5)=-0.047

dove $\rho(1)$ e $\rho(5)$ è rispettivamente l'autocorrelazione per batch adiacenti e per batch separati da 5 passi. Entrambe le metriche sono abbondantemente sotto 0.2.



Andando ad eseguire questa simulazione variando il tasso degli arrivi λ , vediamo come per i pacchetti di tipo 'E' il QoS venga violato quando λ supera $1.55j/s$

15 Scalabilità

Per estendere il modello base e renderlo più realistico rispetto a scenari di carico variabile, è stata implementata una simulazione con scalabilità dinamica dei server. L'obiettivo era verificare come il sistema reagisse a variazioni della probabilità p_c e del tasso di arrivo λ . Per fare ciò sono state introdotte delle fasce orarie con lambda variabili calcolati utilizzando i dati dell'aeroporto di Ciampino , come definito in precedenza dal paragrafo 4.4.

15.1 Implementazione

Per gestire scenari con carico variabile nel tempo è stato sviluppato un simulatore unificato Edge + Coordinator con scalabilità dinamica, contenuto nel file `edge_cord_merged_scalability_simulator.py`.

La funzione principale è `edge_coord_scalability_simulation(stop, forced_lambda=None, slot_index=None)`, che gestisce con un approccio event-driven la simulazione di 24 ore, con arrivi modellati da un processo di Poisson non omogeneo in base alle fasce di carico (LAMBDA_SLOTS).

Gli aspetti chiave dell'implementazione sono:

1. **Pool di server e strutture dati dedicate Edge/Coordinator** passano da 1 servente a m serventi: array paralleli per stato e tempi di fine servizio

```
edge_server_busy = [False] * cs.EDGE_SERVERS
edge_completion_times = [cs.INFINITY] * cs.EDGE_SERVERS
# analogo per il Coordinator
```

- 2. Stima d'utilizzazione e decisioni di scaling (80% / 30%)** A ogni iterazione si valuta la necessità di aggiungere o rimuovere server in Edge/Coordinator, sulla base di una stima d'utilizzazione coerente con il mix corrente:

```
edge_utilization = current_lambda * (0.5 + 0.1 * p_c)
coord_utilization = (current_lambda * p_c) * 0.8

# Scale-up: se oltre 80% (per server)
if cs.EDGE_SERVERS < cs.EDGE_SERVERS_MAX and edge_utilization / cs.EDGE_SERVERS >
    0.8:
    cs.increment_edge()

# Scale-down: se sotto 30% (mantenendo almeno 1 server)
if cs.EDGE_SERVERS > 1 and edge_utilization / cs.EDGE_SERVERS < 0.3:
    cs.decrement_edge()
# logica analoga per il Coordinator
```

Le soglie 0.8 / 0.3 sono i valori di guardia; la valutazione avviene dentro il ciclo così che i server aggiunti possano essere subito riempiti.

- 3. Dispatch multi-server: locale + “kick” globale** Quando si libera un server si tenta prima un'assegnazione locale:

```
ssigned = edge_assign_if_possible(i) # riempie il server i
if not assigned:
    kick_assign_all() # passata globale su tutti i server liberi
```

edge_assign_if_possible(i): se il server *i* è libero e la coda non è vuota estrai il primo job e lo avvia sul server *i*.

```
def edge_assign_if_possible(sidx):
    if not edge_server_busy[sidx] and stats.queue_edge:
        job = stats.queue_edge.pop(0) # "E" o "C"
        service = GetServiceEdgeE() if job == "E" else GetServiceEdgeC()
        edge_completion_times[sidx] = stats.t.current + service
        edge_server_busy[sidx] = True
        # update aree/contatori per E o C ...
        return True
    return False
```

kick_assign_all() quando l'assegnazione locale non basta (p.es. più server liberi, completamenti simultanei, scale-up), esegue un riesame coordinato: tenta di riempire tutti i server Edge liberi usando la stessa coda unica, poi fa lo stesso per il Coordinator (code high/low).

Vantaggi principali:

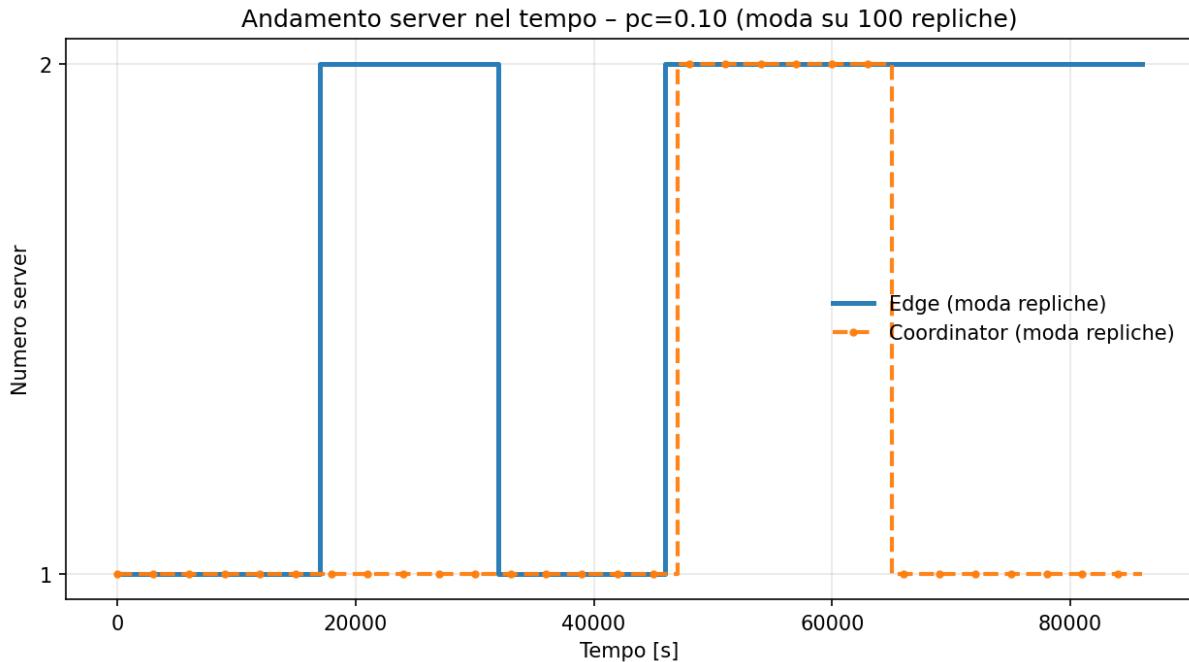
- **Reattività ai burst** (ritorni Cloud): distribuisce immediatamente molti job su più server.
- i server appena aggiunti vengono sfruttati subito.
- **Minore idleness**: evita che ci siano server liberi mentre la coda è non vuota.

```
# Edge: riempi tutti i server liberi dalla coda unica
for i in range(cs.EDGE_SERVERS):
    if not stats.queue_edge:
        break
    edge_assign_if_possible(i)
```

```
# Coordinator: riempi rispettando priorit (P3/P4 > P1/P2)
for i in range(cs.COORD_EDGE_SERVERS):
    if not (stats.queue_coord_high or stats.queue_coord_low):
        break
    coord_assign_if_possible(i)
```

15.2 Risultati

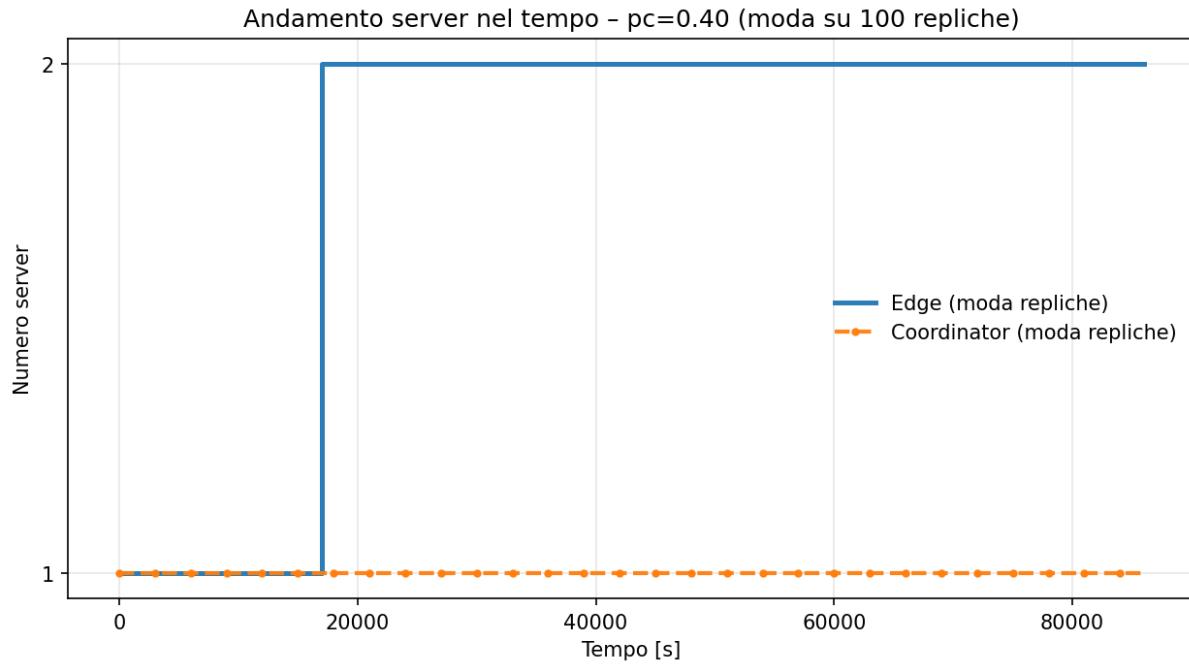
- $p_c = 0.1$



Quando $p_c = 0.1$ il feedback del cloud verso il nodo edge è quasi completamente assente, l'**edge node** scala comunque a due server nella prima fascia oraria di picco (dalle 4 alle 8 di mattina, cioè da 14400 s a 28799 s, con $\lambda = 1.66$), per poi tornare ad un server alla fascia successiva. Ritorna poi a due server nella fascia più alta (dalle 12 alle 15:59, cioè da 43200 s a 57599 s, con $\lambda = 1.9$) per poi rimanere fissa a 2 fino alla fine della giornata, nonostante nella fascia successiva il tasso di arrivi diminuisca. Questo perché l'utilizzazione rimane alta dovendo smaltire il backlog dovuto alla fascia precedente.

Per quanto riguarda il **coordinator server edge**, essendo il 90% dei pacchetti riconosciuti dall'edge la sua utilizzazione supera il valore di soglia dell'80% solo in corrispondenza della fascia con il più alto tasso di arrivi all'edge (da 43200 s a 57599 s, con $\lambda = 1.9$)

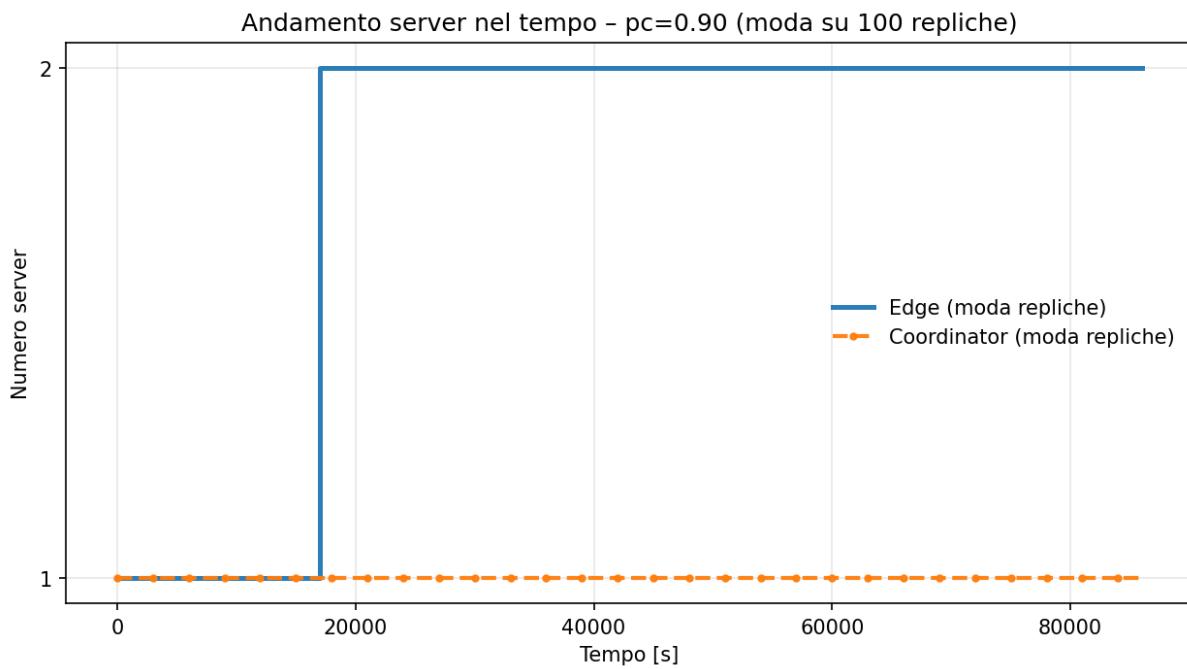
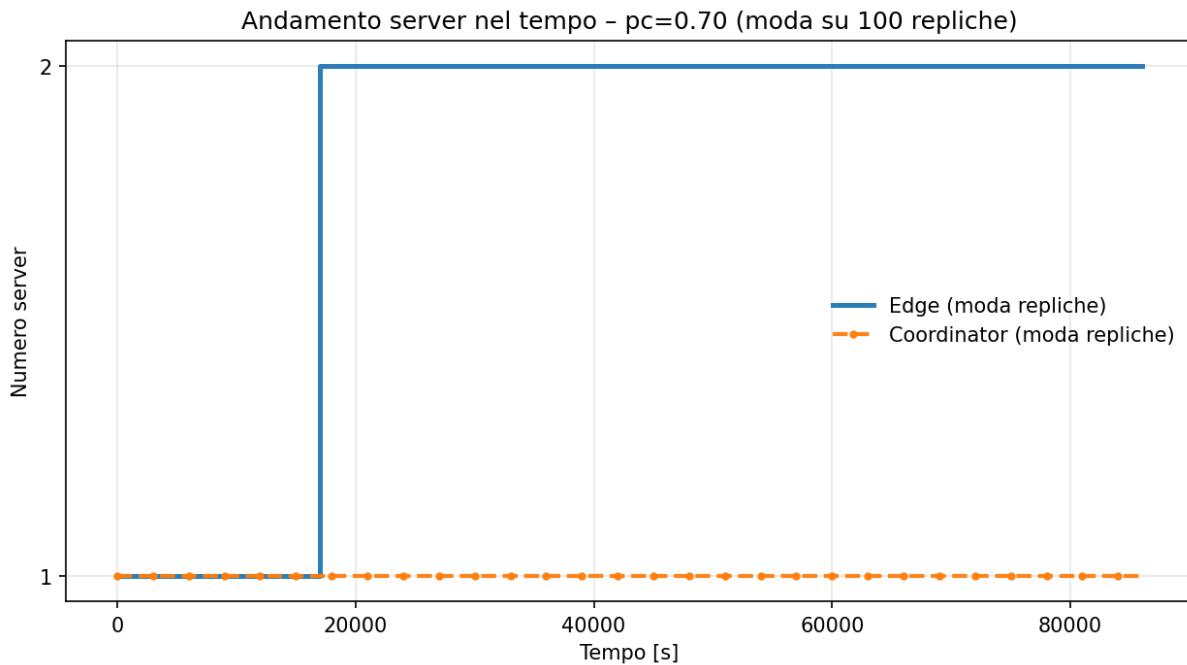
- $p_c = 0.4$



Inizialmente il nodo **edge** resta con un solo server, ma poco dopo le 14.400 s (fascia HIGH, 04:00–07:59, $\lambda = 1.66$) scala a 2 server. Da quel momento non ritorna più a 1, rimanendo stabilmente a 2 fino a fine giornata. Questo comportamento è riconducibile al **backlog accumulato**: durante le fasce di traffico più intenso (in particolare quella VERY HIGH, 12:00–15:59, $\lambda = 1.90$) si genera coda che impedisce all'utilizzazione di scendere sotto la soglia di scale-down, mantenendo il pool di Edge sovradimensionato anche quando il carico diminuisce.

Per quanto riguarda il **coordinator server edge** rimane sempre con un solo server. La quota di job che raggiunge il Coordinator con $p_c = 0.4$ non è sufficiente a saturarne la capacità, e quindi non si innesca mai alcun incremento.

- $p_c = 0.7 \& 0.9$



Aumentando ancora la percentuale di pacchetti indirizzati al cloud che quindi torneranno nell'**edge** la situazione rimane analoga a prima, per cui dalla prima fascia con un più alto tasso degli arrivi l'**edge** non riesce a smaltire il backlog delle fasce precedenti e quindi rimane fisso a 2.

Avendo ancora meno pacchetti riconosciuti il **coordinator server edge** non supera mai il valore di soglia per attivare lo scale-up.

16 Valutazione dei Risultati

Gli esperimenti a **orizzonte finito** e **orizzonte infinito** hanno permesso di analizzare uno scenario in cui il modello veniva sollecitato con un tasso di arrivo medio realistico. Entrambi gli approcci hanno evidenziato con successo il raggiungimento degli obiettivi iniziali. In particolare, la simulazione su 24 ore ha mostrato che:

- il tempo di risposta medio si stabilizza rapidamente dopo la fase di transitorio;
- il sistema garantisce un livello di *QoS* soddisfacente sia per job di classe E (traffico principale) sia per job di classe C (traffico intermittente), con tempi di assestamento compatibili con scenari realistici di carico;
- la ripetizione delle simulazioni su più repliche indipendenti ha validato la robustezza statistica delle metriche, confermando la capacità del modello di descrivere correttamente il regime stazionario.

Diverso è il caso dello scenario di **scalabilità dinamica con fasce orarie**, in cui il modello è stato testato in un contesto più realistico, caratterizzato non più da un λ medio costante, ma da un λ variabile nell'arco della giornata. Pur mantenendo la stessa media globale, questa variabilità ha introdotto picchi di carico significativi.

In questo contesto emerge pertanto che, ad eccezione della prima fase di picco, la configurazione di sistema necessita di un'infrastruttura dual-core per la maggior parte del tempo.

17 Modello migliorativo

Dall'analisi dei risultati simulativi precedenti si osserva che, con un tasso medio di arrivo fisso, tutti gli obiettivi sono rispettati. Tuttavia, in un contesto realistico—come nello studio di scalabilità—caratterizzato da tassi di arrivo variabili secondo un profilo orario giornaliero con picchi di carico pronunciati, emerge la necessità di un approccio migliorativo. In particolare:

- il nodo *Edge* richiede stabilmente una configurazione *dual-core* per far fronte alla domanda operativa;
- Il *Coordinator* mostra una capacità di scalabilità limitata, attivandosi solo marginalmente e per un breve intervallo quando la probabilità P_c è fissata a 0.1.

Alla luce di queste evidenze, fissiamo la configurazione di riferimento con due core/serventi all'*Edge* ($m_{\text{Edge}} = 2$) e un solo servente al *Coordinator* ($m_{\text{Coord}} = 1$).

Per valutare il comportamento del sistema in condizioni realistiche, è stato condotto uno studio a orizzonte finito della durata di

$$T_{\text{sim}} = 48 \text{ ore} = 172\,800 \text{ s.}$$

Questa scelta consente di osservare la dinamica del modello migliorativo in presenza di picchi di carico ripetuti e in uno scenario operativo non vuoto .

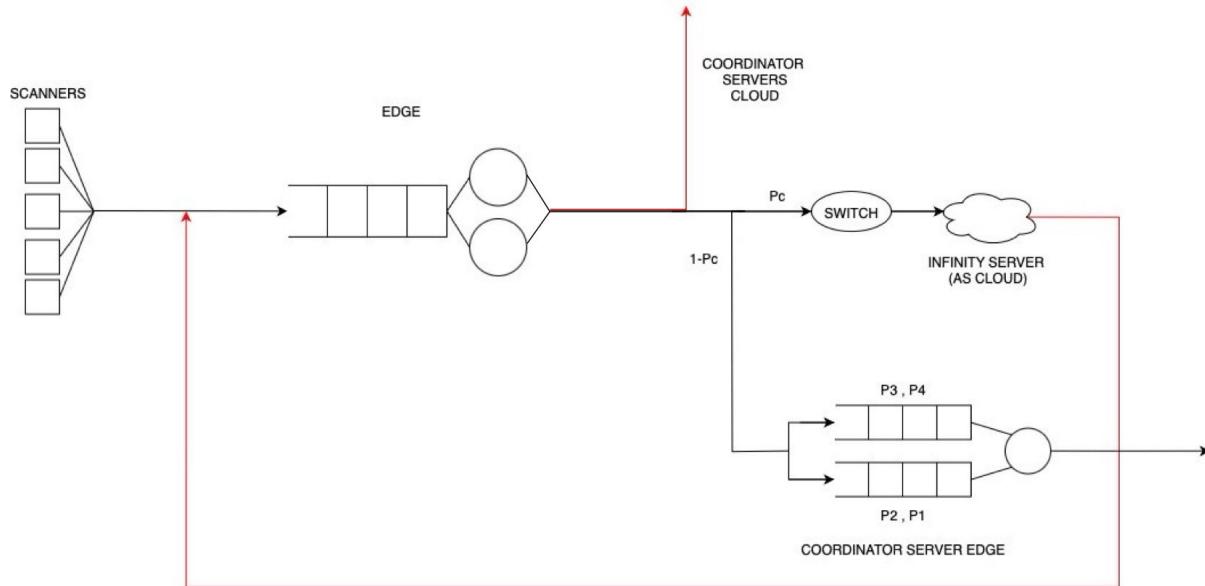
In assenza di un valore di P_c universalmente valido o consolidato in letteratura per il caso d'uso considerato, si è adottata un'analisi di sensibilità rispetto a differenti valori di P_c . I valori specifici di P_c impiegati nello studio, insieme alle motivazioni e ai risultati di dettaglio, **saranno presentati e discusssi nelle sezioni successive**, con l'obiettivo di valutare la robustezza del sistema e il rispetto dei vincoli di *Quality of Service* (*QoS*) definiti per il nodo *Edge*.

18 Obiettivi Modello migliorativo

Gli obiettivi sono gli stessi già proposti e raggiunti nel modello precedente , ma studiati in un contesto con un tasso di arrivi variabile , con picchi di carico durante il periodo simualto.

19 Modello Concettuale

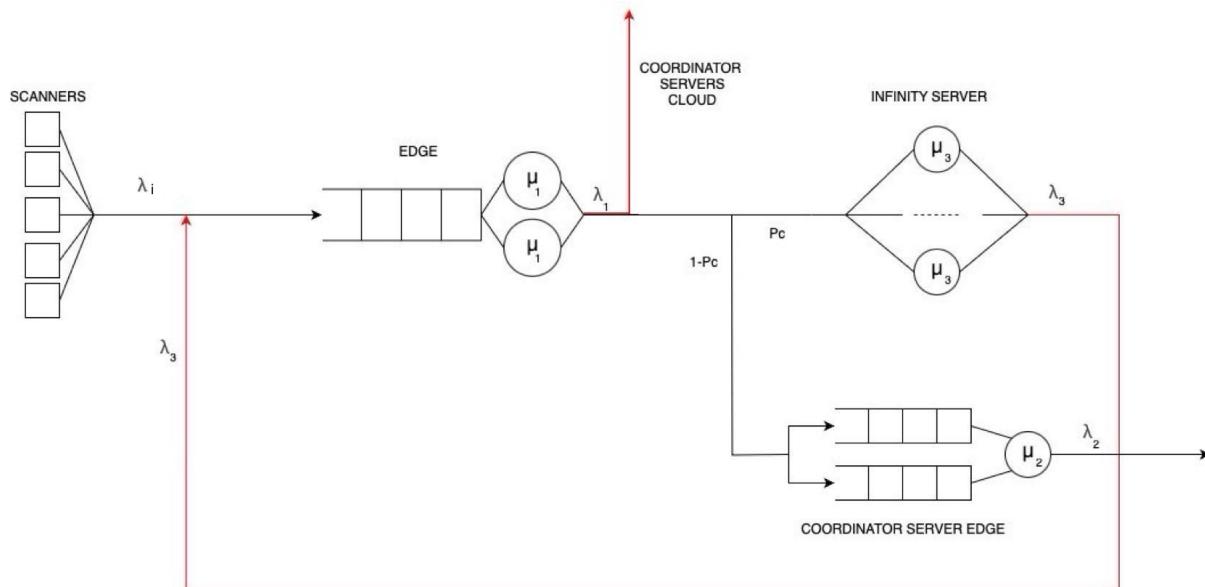
In questa sezione si ha lo scopo di chiarire la struttura del modello nella fase concettuale:



L'unica differenza risiede nella composizione del nodo edge che stavolta possiede due core.

20 Modello delle Specifiche

In questa fase dello studio , il modello viene osservato con un tasso medio di arrivo esponenziale variabile in base all' arco della giornata. Questo perchè osservando un diverso λ_i in base al momento della giornata possiamo studiare la variabilità del carico con i relativi picchi di lavoro. Al livello strutturale nel modello l'unica differenza riesede sulla computazione del nodo edge , le equazioni di traffico e di routing rimangono sostanzialmente invariate, che sono qui di seguito :



	Coordinator	Server	Edge	Edge	Cloud	Esterno
Coordinator	0		0	0	1	
Edge		$1 - p_c$	0	p_c	0	
Cloud	0		1	0	0	
Esterno	0		1	0	0	

Tabella 21: Matrice di routing

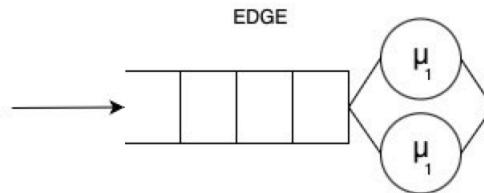
Come definito in precedenza, viene modellato in questo studio con λ_i , poichè lo studio è stato affrontato con un tasso medio di arrivo variabile nel l'arco della giornata. Quindi in base alla fascia della gironata durante il periodo simulato riscontremo un diverso λ_i . Qui di seguito l'equazioni di traffico:

$$\begin{cases} \lambda_1 = \lambda_i + \lambda_3 \\ \lambda_2 = \lambda_1(1 - p_c) \\ \lambda_3 = \lambda_1 p_c \end{cases}$$

Si evidenzia che tutti i centri mantengono le stesse distribuzioni di servizio medio del caso precedentemente studiato, compreso il nodo edge.

20.1 Modellazione dei centri

Qui di seguito viene affrontata la descrizione delle scelte di modellazione effettuate:



Il nodo Edge è stato modellato come un sistema **dual core**, con un'unica coda FIFO infinita, così da garantire l'elaborazione di tutti i pacchetti in arrivo e preservare la sicurezza aeroportuale. Entrambi i core presentano la stessa distribuzione di servizio e i medesimi tempi medi:

$$E[S] = 0.5 \text{ s} \quad \text{per i job di classe E}$$

$$E[S] = 0.1 \text{ s} \quad \text{per i job di classe C.}$$

La scelta di modellare i tempi di servizio tramite una distribuzione esponenziale è motivata da diversi fattori:

- riflette la natura regolare e ripetitiva delle operazioni svolte all'Edge;
- consente di rappresentare in modo semplice la variabilità dei tempi di servizio attorno a un valore medio;
- gode della proprietà di memoria nulla (*memoryless*), che rende il modello coerente con l'ipotesi di indipendenza tra i job e permette un'analisi più agevole tramite la teoria delle code.

Infine, il tempo tra due arrivi successivi è modellato, per via del teorema di Burke, con una distribuzione esponenziale di parametro:

$$\frac{1}{\lambda_i + \lambda_3}$$

dove λ_i rappresenta il tasso complessivo di arrivo in base alla fascia oraia e λ_3 eventuali pacchetti aggiuntivi di feedback.

21 Modello Computazionale

L'unica differenza dal modello precedenza risiede nell'implementazione del nodo Edge con *dual core*. La sua implementazione è descritta qua di seguito: Rispetto alla versione base (un M/M/1 con un solo t.completion_edge in simulation/simulator.py), l'Edge è ora modellato come M/M/2: due core paralleli con coda FCFS condivisa. Si mantiene stato per-core (occupato/libero) e due timer di completamento; all'arrivo si tenta l'assegnazione a un core libero, altrimenti il job entra in coda. Al completamento si libera il core specifico e si innesca il drain della coda finché c'è capacità. Nel ciclo next-event, l'istante di completamento dell'Edge è il minimo tra i due timer per-core.

```
# Stato Edge dual-core
EDGE_CORES = 2
edge_busy = [False] * EDGE_CORES
edge_comp = [float('inf')] * EDGE_CORES
queue_E = collections.deque() # coda FCFS condivisa

def edge_assign_if_possible(job, now):
    for k in range(EDGE_CORES):
        if not edge_busy[k]:
            edge_busy[k] = True
            st = sample_service_time_E(job)
            edge_comp[k] = now + st
            return True
    return False

def kick_assign_all(now):
    while queue_E and any(not b for b in edge_busy):
        job = queue_E.popleft()
        if not edge_assign_if_possible(job, now):
            queue_E.appendleft(job); break

def handle_arrival(job, now):
    if not edge_assign_if_possible(job, now):
        queue_E.append(job)

def handle_completion_edge(now):
    k = min(range(EDGE_CORES), key=lambda i: edge_comp[i])
    edge_busy[k] = False
    edge_comp[k] = float('inf')
    kick_assign_all(now)

def next_event_time(t):
    edge_min = min(edge_comp)
    return min(t.arrival, edge_min, t.completion_cloud, t.completion_coord)
```

- **edge_assign_if_possible(job, now)**: tenta di assegnare subito il job a un core libero: marca il core come occupato, campiona il tempo di servizio e imposta il relativo timer di completamento. Restituisce True se assegnato, altrimenti False.
- **kick_assign_all(now)**: “drena” la coda condivisa finché esiste almeno un core libero: prende job dalla coda e richiama edge_assign_if_possible; si ferma quando i core sono pieni o la coda è vuota.
- **handle_arrival(job, now)**: gestisce un arrivo all'Edge: prova l'assegnazione immediata; se nessun core è libero, il job viene accodato in FCFS nella coda condivisa.

- **handle_completion_edge(now)**: identifica quale core ha il completamento più vicino, libera quel core, azzera il suo timer e invoca kick_assign_all per riempire la capacità liberata.
- **next_event_time(t)**: calcola il prossimo istante evento del ciclo next-event come minimo tra: prossimo arrivo, minimo dei due timer di completamento dell'Edge, completamento al Cloud e completamento al Coordinator.

Il resto del modello è rimasto invariato rispetto alla versione precedente.

22 Verifica

In questa fase abbiamo verificato la correttezza del modello computazionale implementato tramite la simulazione ad orizzonte infinito. Il seed base utilizzato è 123456789 con 100 repliche, con l'intervallo di confidenza del 95 %.

Non abbiamo fatto la verifica del Cloud e del Coordinator Server Edge in quanto risulterebbe uguale al modello base.

Nodo Edge (dual-core)

Dati di base:

$$P_c = 0.0, \quad m = 2, \quad E(S_i) = 0.5 \text{ s} \quad \Rightarrow \quad \mu_i = \frac{1}{E(S_i)} = 2 \text{ s}^{-1}$$

Formule utilizzate:

$$\begin{aligned} E(S) &= \frac{1}{m\mu_i} \\ \rho &= \frac{\lambda_i}{m\mu_i} \\ P(0) &= \left[\sum_{i=0}^{m-1} \frac{(m\rho)^i}{i!} + \frac{(m\rho)^m}{m! (1-\rho)} \right]^{-1} \\ P_Q &= \frac{(m\rho)^m}{m! (1-\rho)} \cdot P(0) \\ E(T_Q) &= \frac{P_Q \cdot E(S)}{1-\rho} \\ E(T_S) &= E(T_Q) + E(S_i) \\ E(N_Q) &= \lambda_i \cdot E(T_Q) \\ E(N_S) &= \lambda_i \cdot E(T_S) \end{aligned}$$

Risultati analitici:

Fascia oraria	λ_i [job/s]	ρ	$E(T_Q)$ [s]	$E(T_S)$ [s]	$E(N_Q)$ [job]	$E(N_S)$ [job]
00:00–03:59	0.83	0.2075	0.0225	0.5225	0.0187	0.4337
04:00–07:59	1.66	0.4150	0.1040	0.6040	0.1726	1.0027
08:00–11:59	1.16	0.2900	0.0427	0.5427	0.0495	0.6295
12:00–15:59	1.90	0.4750	0.1851	0.6851	0.3517	1.3017
16:00–19:59	1.49	0.3725	0.0806	0.5806	0.1201	0.8650
20:00–23:59	1.24	0.3100	0.0532	0.5532	0.0660	0.6859

Tabella 22: Risultati analitici per ciascuna fascia oraria

Ai fini di questa verifica, data l'impossibilità di validare i risultati in modo analitico a causa del complesso meccanismo di feedback, abbiamo simulato il sistema assumendo $P_c = 0$.

Risultati di simulazione:

Fascia oraria	λ_i [job/s]	$E(T_S)$ [s]	CI	$E(N_S)$ [job]	CI
00:00–03:59	0.83	0.5228	± 0.00043	0.4342	± 0.00048
04:00–07:59	1.66	0.6040	± 0.00053	1.0025	± 0.00113
08:00–11:59	1.16	0.5468	± 0.00044	0.6352	± 0.00073
12:00–15:59	1.90	0.6453	± 0.00066	1.2261	± 0.00144
16:00–19:59	1.49	0.5802	± 0.00049	0.8643	± 0.00092
20:00–23:59	1.24	0.5527	± 0.00043	0.6856	± 0.00071

Tabella 23: Risultati di simulazione per ciascuna fascia oraria, con intervalli di confidenza al 95%

Controlli di consistenza

Dai risultati ottenuti è possibile verificare che sono rispettati i seguenti controlli di consistenza:

- $E(T_S) = E(T_Q) + mE(S)$
- $0 \leq \rho \leq 1$
- $E(N_S) = E(N_Q) + m\rho$

23 Validazione

In questa fase verifichiamo se il modello implementato è coerente con il sistema reale di riferimento e se riproduce correttamente i comportamenti attesi. Rispetto al modello base, ci aspettiamo di ritrovare una corrispondenza con le considerazioni già fatte per quest'ultimo.

Edge Node – Dual-Core Con $P_c = 0$ non abbiamo il feedback, quindi ci aspettiamo dei tempi di risposta inferiori. Impostando $P_c = 0.4$ introduciamo il feedback dal Cloud verso l'Edge. Ci si aspetta quindi un aumento del tasso di arrivi e, di conseguenza, un peggioramento della saturazione e dei tempi medi, sia dei job di classe E che quelli di classe C . La simulazione conferma queste previsioni, riproducendo il comportamento atteso e confermando la correttezza del modello.

Tuttavia, grazie alla presenza dei due server in parallelo (dual-core) con lo stesso tempo di servizio del caso base, l'incremento dei tempi medi sarà più contenuto rispetto al singolo core. Di conseguenza, aumentando ulteriormente il valore di P_c , ci aspettiamo ancora un peggioramento della saturazione e dei tempi medi sia dei job di classe E che quelli di classe C , ma l'impatto sarà mitigato dal parallelismo introdotto, come confermato dai risultati della simulazione.

Cloud L’ipotesi di partenza è che, essendo il Cloud un sistema $M/M/\infty$ (cioè con un numero illimitato di server), non si formano code e la capacità del sistema sia sempre disponibile. Di conseguenza, ci si aspetta che un aumento del carico non influisca sui tempi di risposta. Con $P_c = 0.4$, la simulazione ha confermato questa previsione: i tempi di risposta sono rimasti praticamente invariati. Questo comportamento è coerente con quanto ci si attende da un sistema $M/M/\infty$. Aumentando ulteriormente P_c , ci si aspetta ancora che i tempi di risposta del Cloud restino stabili, come confermato dalla simulazione. Tuttavia, questo incremento del feedback provoca un aumento del carico sull’Edge, che subirà un peggioramento dei tempi di risposta complessivi.

Coordinator Per $P_c = 0.4$, ci si aspettano dei miglioramenti nelle prestazioni rispetto al caso senza feedback. All’aumentare di P_c , la frazione di traffico diretta al Coordinator diminuisce, poiché solo $1 - P_c$ dei job raggiunge questo centro. Questo porta a una riduzione dell’utilizzazione e dei tempi di risposta, con un conseguente miglioramento delle prestazioni, esattamente come previsto teoricamente. La simulazione conferma questa relazione inversa tra P_c e le prestazioni. Analogamente, per $P_c = 0$, il flusso diretto al Coordinator aumenta, causando un peggioramento dei tempi di risposta, scenario che viene anch’esso confermato dalla simulazione.

24 Design degli esperimenti

La fase di progettazione degli esperimenti si articola nei seguenti passaggi:

- **Analisi del collo di bottiglia:** questa fase ha l’obiettivo di individuare il centro con la domanda più elevata, utilizzando strumenti di analisi operazionale.
- **Simulazione a orizzonte finito:** in questa fase il sistema viene simulato per un periodo di 48 ore impiegando il metodo delle repliche. Nel nostro caso, sono state effettuate 100 repliche.

Si è scelto un orizzonte finito più lungo rispetto agli esperimenti precedenti, per osservare il comportamento del modello in una condizione non vuota. L’orizzonte infinito non è stato adottato perché avrebbe prodotto indicatori simili a quelli dell’orizzonte finito ma su un arco temporale più esteso, finendo per nascondere il transitorio iniziale e la successiva stabilizzazione della gestione del carico quando il modello non parte da condizioni vuote.

24.1 Calcolo del seed indipendente tra repliche

Per garantire l’**indipendenza dei seed tra le repliche** abbiamo adottato la stessa tecnica già utilizzata in precedenza e descritta nel paragrafo 9 .

24.2 Analisi del transitorio

Prima di analizzare nel dettaglio le fasi sopra menzionate, è opportuno condurre uno studio del transitorio per verificare la convergenza del sistema allo stato stazionario. A tal fine, sono state effettuate 10 simulazioni tramite il metodo delle repliche, utilizzando seed differenti e indipendenti tra loro, fissando $P_c = 0.4$ e adottando il tempo di risposta medio come metrica di riferimento per valutare la convergenza del sistema. Inoltre i grafici fanno riferimento ad un periodo di 4 giorni esatti, ovvero a (345600s.).

Di seguito presentiamo i risultati della simulazione con $P_c = 0.4$, in cui il tasso di arrivo λ varia per fascia oraria come descritto nel paragrafo 4.4. Il grafico riportano il tempo di risposta medio dei pacchetti di **classe E** nel nodo Edge migliorato.

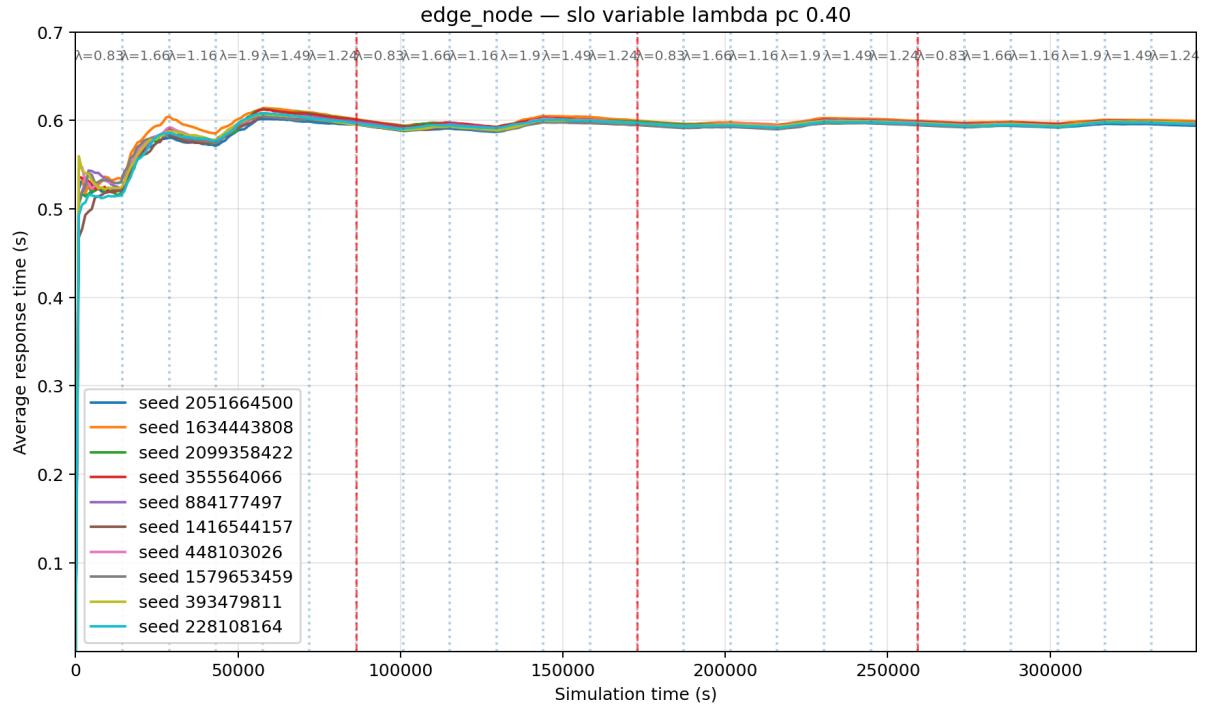
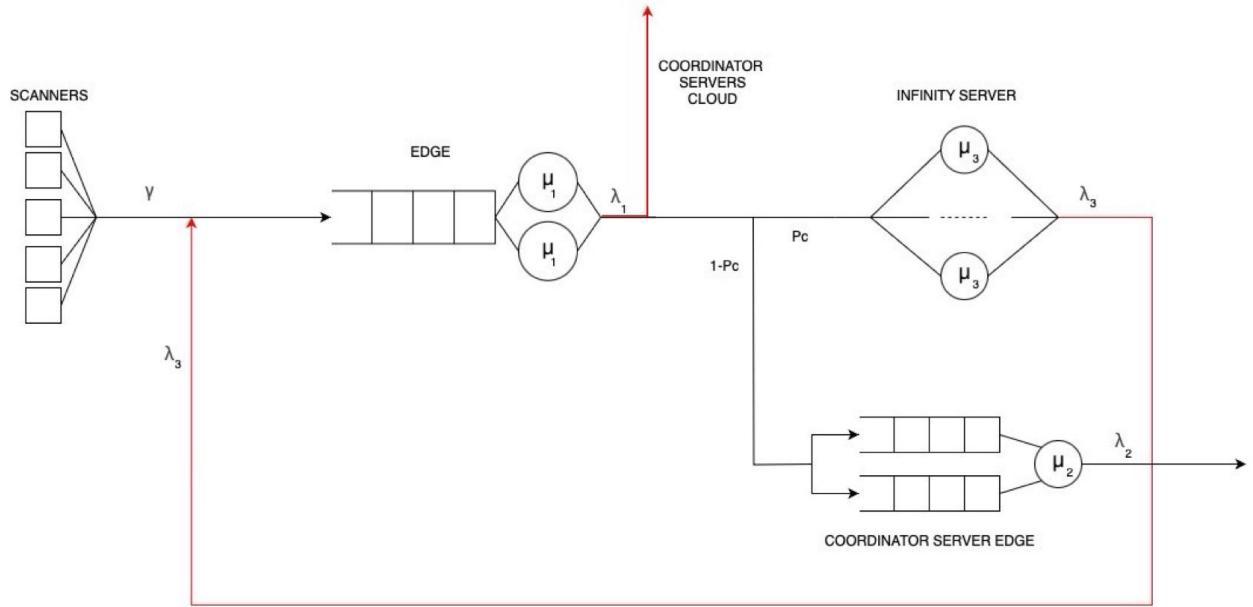


Figura 18: `edge_node_slo_variable_lambda_pc_0.40.png` — Tempo di risposta medio all’Edge per dieci repliche con seed differenti; linee tratteggiate rosse: confini di giornata (24 h), linee puntinate azzurre: confini delle fasce orarie di λ .

Il grafico in Fig. 18 mostra l’evoluzione del tempo di risposta medio dei job di classe E all’Edge con $P_c = 0.4$ e profilo di arrivo giornaliero a fasce (paragrafo 4.4). Le dieci traiettorie, corrispondenti a seed indipendenti, presentano un breve transitorio iniziale in cui il tempo di risposta cresce rapidamente da valori prossimi a 0.52–0.54 s fino a circa 0.59 s. Entro le prime 2–6 ore (circa 10^4 – 2×10^4 secondi) le curve si allineano e la variabilità tra repliche diviene trascurabile.

Una volta esaurito il transitorio, il sistema entra in un *regime stazionario periodico* indotto dalla variazione ciclica di λ : il tempo di risposta oscilla debolmente attorno a ~ 0.57 –0.60 s, con leggere increspature sincronizzate con i confini di fascia (linee puntinate). Il ripetersi della stessa dinamica tra i confini di giornata (linee rosse a 24 h, 48 h, 72 h) e la sovrapposizione delle repliche confermano sia la stabilità del modello sia l’irrilevanza dell’effetto del seed una volta raggiunto il regime.

24.3 Analisi del collo di bottiglia



Nei calcoli che seguono, indicheremo con γ il tasso di arrivo esterno (jobs/s), con λ_i il tasso di uscita dal centro i -esimo

$$\lambda_1 = \frac{\gamma}{1 - P_c}$$

Questo deriva dal Bilancio dei flussi: il flusso in ingresso all'Edge include sia i job nuovi (γ) che quelli che ritornano con probabilità P_c :

$$\lambda_1 = \gamma + \lambda_3 = \gamma + \lambda_1 \cdot P_c$$

Allora:

$$\lambda_1 \cdot (1 - P_c) = \gamma$$

Quindi:

$$\lambda_1 = \frac{\gamma}{1 - P_c}$$

Invece,

$$\lambda_3 = P_c \cdot \lambda_1$$

Questi sono i job che vengono reindirizzati dall'Edge al Cloud.

Infine,

$$\lambda_2 = (1 - P_c)\lambda_1.$$

Ora definiamo il numero medio di visite che ciascun job compie in ogni centro:

$$v_i = \frac{\lambda_i}{\gamma}$$

Misura quante volte un job , in media, passa dal centro i -esimo rispetto al flusso esterno.

Visite all'Edge:

$$v_1 = \frac{\lambda_1}{\gamma} = \frac{1}{1 - P_c}$$

Visite al Cloud:

$$v_3 = \frac{\lambda_3}{\gamma} = \frac{P_c}{1 - P_c}$$

Visite al Coordinator:

$$v_2 = \frac{\lambda_2}{\gamma} = 1$$

Valori numerici per $p_c = 0.4$.

$$\begin{aligned} v_1 &= \frac{1}{1 - 0.4} = \frac{1}{0.6} = \frac{10}{6} = \frac{5}{3} \approx 1.6667 \\ v_3 &= \frac{0.4}{0.6} = \frac{4}{6} = \frac{2}{3} \approx 0.6667 \\ v_2 &= 1 \end{aligned}$$

Tempi medi di servizio:

Poiché all'Edge Node i job che arrivano dall'esterno richiedono un tempo di servizio pari a 0.5 s, mentre i job che effettuano il feedback richiedono un tempo di servizio pari a 0.1 s, precedentemente abbiamo considerato come S_1 la media dei tempi di servizio all'Edge Node, ottenuta dalla simulazione con $P_c = 0.4$:

$$S_1 = 0.385816 \text{ s}$$

L'Edge Node è ora dual-core, quindi è composto da due server identici in parallelo. Indichiamo con $S_{1,i}$ il tempo medio di servizio *per singolo server* (valore ottenuto dalla simulazione):

$$S_{1,i} = 0.385816 \text{ s}$$

Il tempo medio di servizio effettivo per job sull'intero Edge Node (con due server in parallelo) diventa:

$$S_1 = \frac{S_{1,i}}{2} = \frac{0.385816}{2} = 0.192908 \text{ s}$$

In forma generale, per un Edge Node con m server identici in parallelo e tempo medio per server $S_{1,i}$, si ha:

$$S_1 = \frac{S_{1,i}}{m}$$

Per il Coordinator Server Edge:

$$S_2 = 0.45 \text{ s}$$

Per il Cloud :

$$S_3 = 0.8 \text{ s}$$

Il demand D_i di ciascun centro è il prodotto tra il numero medio di visite v_i e il tempo medio di servizio S_i :

$$D_i = v_i \cdot S_i$$

In particolare:

$$D_1 = v_1 \cdot S_1 = \frac{1}{1 - P_c} \cdot 0.192908$$

$$D_3 = v_3 \cdot S_3 = \frac{P_c}{1 - P_c} \cdot 0.8$$

$$D_2 = v_2 \cdot S_2 = 1 \cdot 0.45 = 0.45$$

Valori numerici per $P_c = 0.4$:

$$D_1 = \frac{1}{0.6} \cdot 0.192908 = 1.6667 \cdot 0.192908 \approx 0.3215 \text{ s}$$

$$D_3 = \frac{0.4}{0.6} \cdot 0.8 = 0.6667 \cdot 0.8 = 0.5333 \text{ s}$$

$$D_2 = 0.45$$

Identificazione del collo di bottiglia:

Confrontiamo i centri a capacità finita ($E[T_{s,\text{Edge-E}}]$ Coordinator):

$$D_1 = 0.3215 \text{ s} \quad \text{vs} \quad D_2 = 0.45 \text{ s.}$$

Poiché $D_1 < D_2$, il **collo di bottiglia diventa il Coordinator Server** (e non più l'Edge Node).

Il Cloud resta un *Infinite Server*, quindi non può essere collo di bottiglia anche se D_3 è elevato.

Throughput massimo Per un centro a servente finito vale:

$$X_i = \frac{1}{D_i}.$$

Il throughput massimo del sistema è uguale a:

$$X_{\max} = \frac{1}{\max_{i \in \{1,2\}} \{D_i\}}$$

Calcoliamo i reciproci usando le forme numeriche:

$$\frac{1}{D_1} = \frac{1}{0.3215} \approx 3.111 \text{ job/s}$$

$$\frac{1}{D_2} = \frac{1}{0.45} \approx 2.222 \text{ job/s}$$

Poiché $3.111 > 2.222$, il throughput massimo del sistema è determinato dal **Coordinator (collo di bottiglia)**:

$$X_{\max} = 2.222 \text{ job/s.}$$

24.4 Simulazione ad Orizzonte Finito

Nel progetto, lo *studio ad orizzonte finito* è stato realizzato con l'obiettivo di osservare il comportamento del sistema su un intervallo temporale realistico di 48 ore, mantenendo il controllo sulle condizioni di carico e consentendo l'analisi delle fasi transitorie. L'orizzonte massimo simulato è fissato a **STOP = 172800** secondi, corrispondente a due intere giornate. La scelta nasce dall'esigenza di valutare il modello non solo in assenza di carico iniziale, ma anche in presenza di backlog, quando il sistema non inizia vuoto.

24.4.1 Implementazione

La simulazione rimane event-driven, gli unici cambiamenti riguardano l'orizzonte finito che diventa di due giorni (172800 s) con arrivi non omogenei: il tasso degli arrivi segue fasce orarie (definite in constants.py in LAMBDA_SLOTS), il profilo di 24h viene poi replicato sul secondo giorno tramite $t \bmod 86400$.

La logica degli eventi rimane invariata, cambia la modellazione del nodo **Edge** che diventa **dual-core**.

Edge dual-core Rispetto alla versione base (un M/M/1 con un solo t.completion_edge in simulation/simulator.py), l'Edge è ora modellato come M/M/2: due core paralleli con coda FCFS condivisa. Si mantiene stato per-core (occupato/libero) e due timer di completamento; all'arrivo si tenta l'assegnazione a un core libero, altrimenti il job entra in coda. Al completamento si libera il core specifico e si innesca il drain della coda finché c'è capacità. Nel ciclo next-event, l'istante di completamento dell'Edge è il minimo tra i due timer per-core.

```
# Stato Edge dual-core
EDGE_CORES = 2
edge_busy = [False] * EDGE_CORES
edge_comp = [float('inf')] * EDGE_CORES
queue_E = collections.deque() # coda FCFS condivisa

def edge_assign_if_possible(job, now):
    for k in range(EDGE_CORES):
        if not edge_busy[k]:
            edge_busy[k] = True
            st = sample_service_time_E(job)
            edge_comp[k] = now + st
            return True
    return False

def kick_assign_all(now):
    while queue_E and any(not b for b in edge_busy):
        job = queue_E.popleft()
        if not edge_assign_if_possible(job, now):
            queue_E.appendleft(job); break

def handle_arrival(job, now):
    if not edge_assign_if_possible(job, now):
        queue_E.append(job)

def handle_completion_edge(now):
    k = min(range(EDGE_CORES), key=lambda i: edge_comp[i])
    edge_busy[k] = False
    edge_comp[k] = float('inf')
    kick_assign_all(now)

def next_event_time(t):
    edge_min = min(edge_comp)
    return min(t.arrival, edge_min, t.completion_cloud, t.completion_coord)
```

- **edge_assign_if_possible(job, now):** tenta di assegnare subito il job a un core libero: marca il core come occupato, campiona il tempo di servizio e imposta il relativo timer di completamento. Restituisce True se assegnato, altrimenti False.
- **kick_assign_all(now):** “drena” la coda condivisa finché esiste almeno un core libero: prende job dalla coda e richiama edge_assign_if_possible; si ferma quando i core sono pieni o la coda è vuota.
- **handle_arrival(job, now):** gestisce un arrivo all'Edge: prova l'assegnazione immediata; se nessun core è libero, il job viene accodato in FCFS nella coda condivisa.
- **handle_completion_edge(now):** identifica quale core ha il completamento più vicino, libera quel core, azzera il suo timer e invoca kick_assign_all per riempire la capacità liberata.

- **next_event_time(t)**: calcola il prossimo istante evento del ciclo next-event come minimo tra: prossimo arrivo, minimo dei due timer di completamento dell'Edge, completamento al Cloud e completamento al Coordinator.

24.4.2 Risultati

L'analisi dei risultati ottenuti conferma l'impatto congiunto delle variazioni di carico (λ) e della probabilità P_c di inoltro verso il nodo *Cloud* sulle prestazioni dell'Edge. In tutti i casi, il tempo medio di risposta mostra un andamento crescente con l'aumentare del tasso di arrivi, mentre un incremento di P_c aumenta il carico effettivo sull'Edge, alzando il tempo medio di attesa e di risposta per le richieste. Segue la presentazione dei risultati distinti per i diversi valori di P_c considerati.

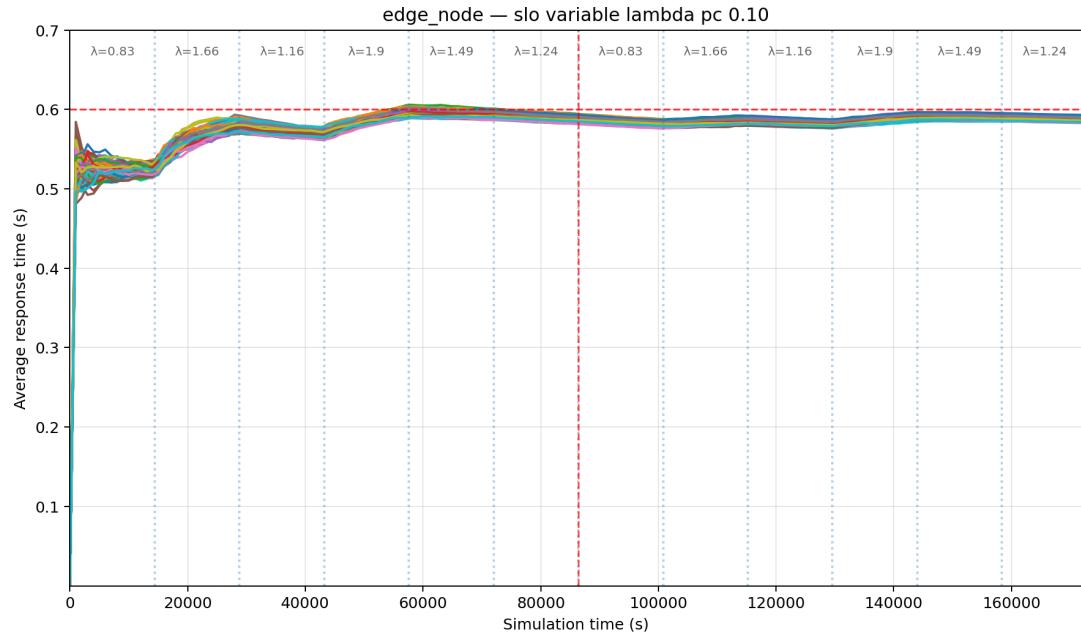


Figura 19: Andamento del tempo medio di risposta all'Edge per $P_c = 0.10$ al variare di λ .

Caso $P_c = 0.10$ In questo scenario, l'Edge è minornemente sollecitato poiché solo il 10% dei job viene inoltrato al Cloud. Il tempo medio di risposta per i pacchetti di classe *E* cresce rapidamente con λ toccando lievemente la soglia dei (0.6s) già per $\lambda \geq 1.9$.

λ	$E[T_{s,\text{Edge-E}}]$	$E[T_{s,\text{Edge-C}}]$
0.83	0.523233 ± 0.000749	0.123228 ± 0.000695
1.16	0.547882 ± 0.000770	0.147545 ± 0.000788
1.24	0.554004 ± 0.000747	0.154486 ± 0.000798
1.49	0.582345 ± 0.000848	0.182661 ± 0.000926
1.66	0.600984 ± 0.000936	0.206877 ± 0.001063
1.90	0.610064 ± 0.001158	0.250765 ± 0.001200

Tabella 24: Tempi medi di risposta all'Edge (media \pm CI95) per $P_c = 0.10$ al variare di λ .

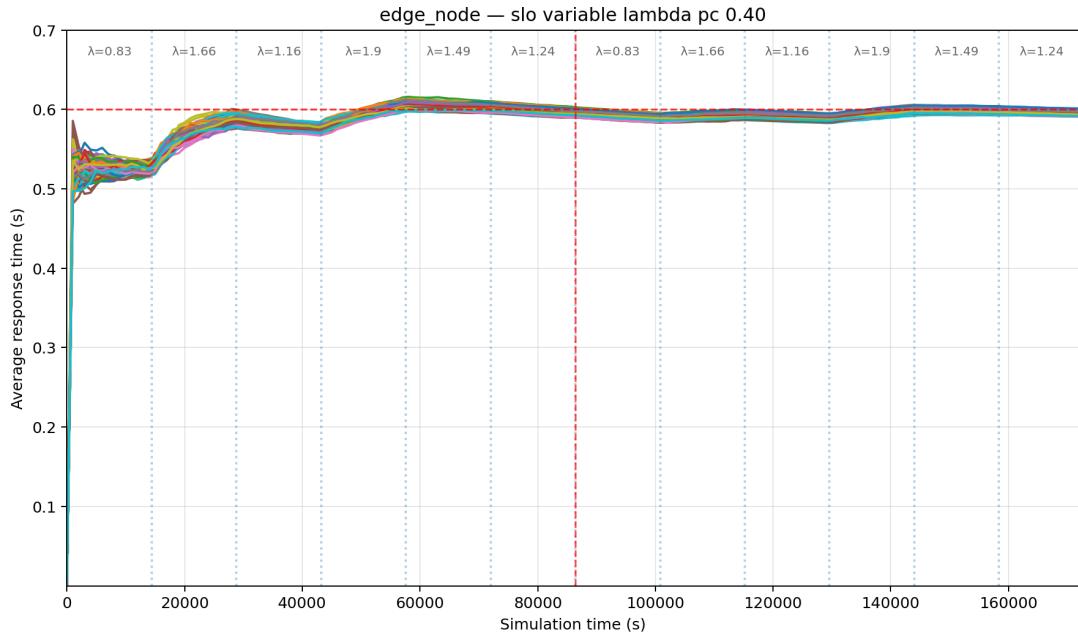
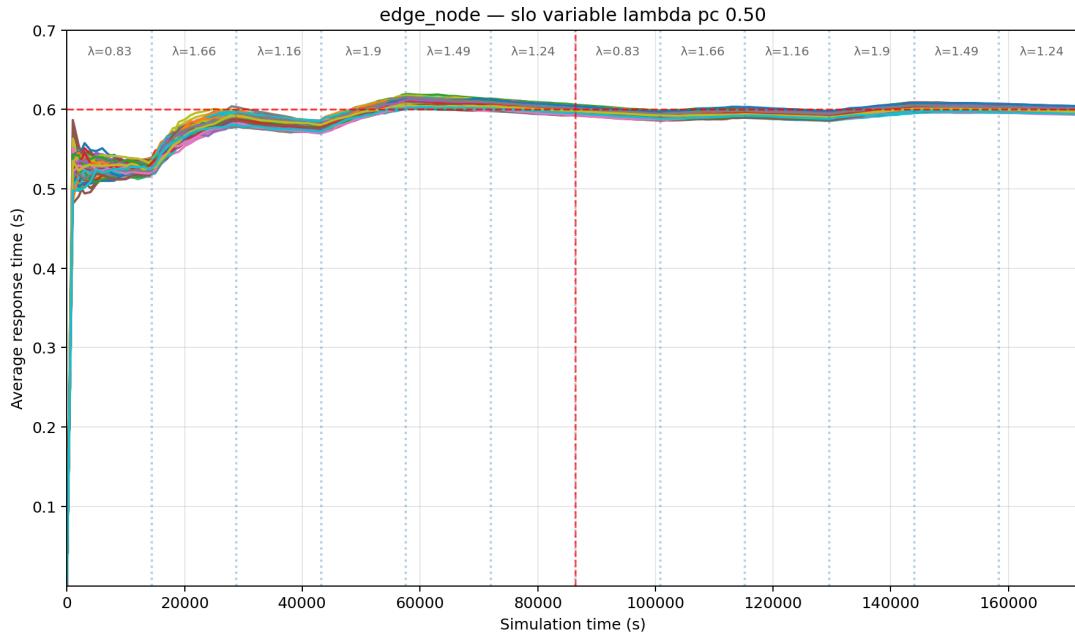


Figura 20: Tempo medio di risposta all’Edge per $P_c = 0.40$.

Caso $P_c = 0.40$ Con una quota maggiore di job inoltrati (40%) al nodo Cloud, l'Edge soffre di un aumento del carico dovuto all'aumento dei pacchetti di feedback provenienti dal Cloud, aumentando ulteriormente i tempi medi di risposta per i pacchetti di *classe E* rispetto al caso precedente.

λ	$E[T_{s,\text{Edge-E}}]$	$E[T_{s,\text{Edge-C}}]$
0.83	0.525 ± 0.0008	0.125 ± 0.0004
1.16	0.551 ± 0.0008	0.151 ± 0.0005
1.24	0.558 ± 0.0008	0.159 ± 0.0005
1.49	0.589 ± 0.0009	0.190 ± 0.0007
1.66	0.607 ± 0.0010	0.218 ± 0.0008
1.90	0.616 ± 0.0012	0.268 ± 0.0011

Tabella 25: Tempi medi di risposta all'Edge (media \pm CI_{95%}) per $P_c = 0.40$.

Figura 21: Tempo medio di risposta all'Edge per $P_c = 0.50$.

Caso $P_c = 0.50$ Quando metà dei job viene diventano pacchetti di feedback, i tempi medi risultano ulteriormente aumentati, mostrando come per il $\lambda = 1.9$ si tocchi il picco a 0.6s. in maniera più marcata rispetto al precedente caso.

λ	$E[T_{s,\text{Edge-E}}]$	$E[T_{s,\text{Edge-C}}]$
0.83	0.525 ± 0.0008	0.125 ± 0.0004
1.16	0.552 ± 0.0008	0.152 ± 0.0004
1.24	0.559 ± 0.0008	0.161 ± 0.0005
1.49	0.592 ± 0.0009	0.193 ± 0.0006
1.66	0.602 ± 0.0010	0.222 ± 0.0008
1.90	0.612 ± 0.0013	0.274 ± 0.0011

Tabella 26: Tempi medi di risposta all'Edge (media \pm CI95) per $P_c = 0.50$.

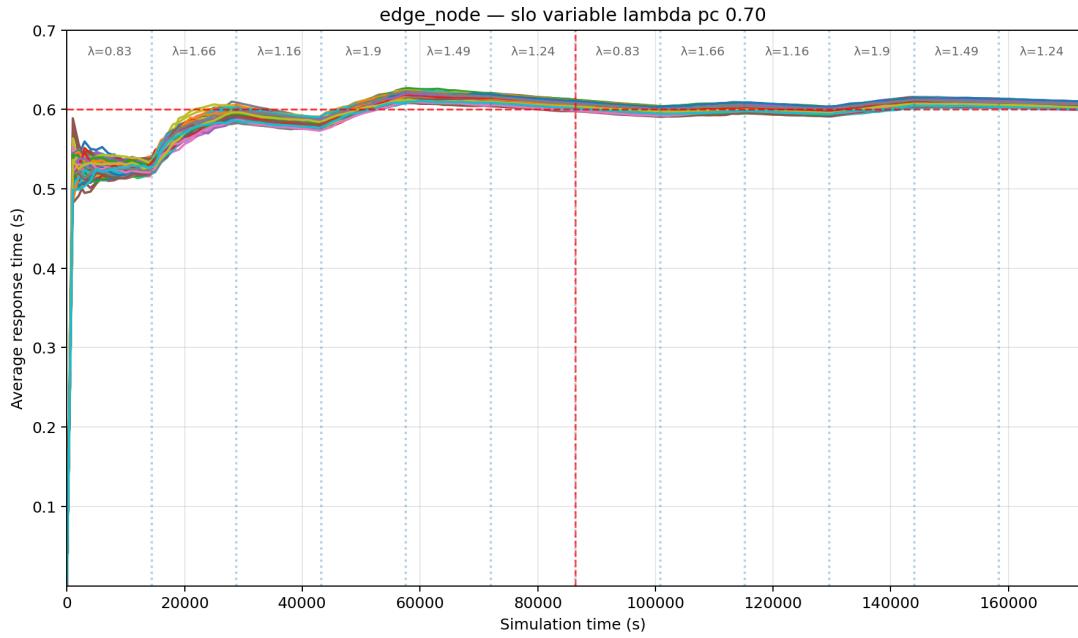
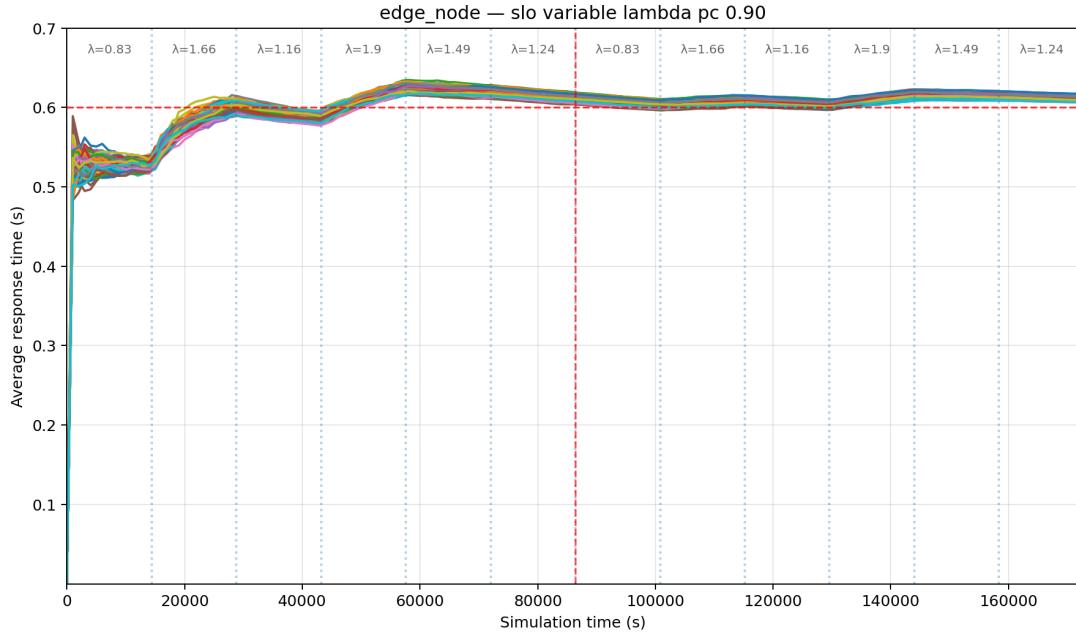


Figura 22: Tempo medio di risposta all'Edge per $P_c = 0.70$.

Caso $P_c = 0.70$ Con il 70% dei job inoltrati, il nodo Edge subisce un sovraccarico importante, superando in maniera marcata il picco a 0.6s che prima gli altri casi hanno leggermente sfiorato.

λ	$E[T_{s,\text{Edge-E}}]$	$E[T_{s,\text{Edge-C}}]$
0.83	0.526 ± 0.0008	0.126 ± 0.0003
1.16	0.555 ± 0.0008	0.155 ± 0.0004
1.24	0.562 ± 0.0008	0.164 ± 0.0005
1.49	0.597 ± 0.0009	0.199 ± 0.0006
1.66	0.618 ± 0.0010	0.230 ± 0.0008
1.90	0.624 ± 0.0014	0.288 ± 0.0011

Tabella 27: Tempi medi di risposta all'Edge (media \pm CI_{95%}) per $P_c = 0.70$.

Figura 23: Tempo medio di risposta all'Edge per $P_c = 0.90$.

Caso $P_c = 0.90$ Nel caso estremo di $P_c = 0.90$, la gran parte del carico viene delegata al Cloud. Il ciò evidenzia il caso peggiore per il nodo Edge, che però mantiene i tempi sotto al Qos di 3 secondi definito negli obiettivi, ma che registra un picco più alto dei casi precedenti, ma sempre conforme a quanto aspettato e imposto dai nostri obiettivi

λ	$E[T_{s,\text{Edge-E}}]$	$E[T_{s,\text{Edge-C}}]$
0.83	0.527 ± 0.0008	0.128 ± 0.0003
1.16	0.557 ± 0.0008	0.158 ± 0.0004
1.24	0.565 ± 0.0008	0.167 ± 0.0004
1.49	0.602 ± 0.0009	0.205 ± 0.0007
1.66	0.625 ± 0.0011	0.239 ± 0.0008
1.90	0.648 ± 0.0014	0.303 ± 0.0011

Tabella 28: Tempi medi di risposta all'Edge (media \pm CI_{95%}) per $P_c = 0.90$.

In sintesi, i risultati mostrano come l'aumento di P_c alleggerisca in modo significativo il carico sul nodo Edge, consentendo di mantenere i tempi di risposta entro lo SLO anche in condizioni di λ elevato. Viceversa, bassi valori di P_c portano rapidamente al superamento della soglia di servizio per carichi intensi.

25 Conclusioni finali

Dall'analisi emerge con chiarezza che il comportamento del nodo Edge dipende fortemente sia dal tasso di arrivi λ sia dalla probabilità P_c di inoltro al Cloud.

- Per bassi valori di P_c (es. 0.10), l'Edge mantiene tempi medi di risposta contenuti, con un impatto marginale anche al crescere di λ , superando solo in condizioni di carico molto elevate la soglia SLO.
- Per valori intermedi ($P_c = 0.40\text{--}0.50$), i pacchetti di feedback aumentano la pressione sull'Edge e determinano un incremento progressivo dei tempi medi, che arrivano a toccare la soglia di 0.6 s già a $\lambda = 1.9$.

- Con valori alti ($P_c = 0.70\text{--}0.90$), l'Edge è gravato da un sovraccarico crescente, con tempi medi sensibilmente più elevati per i pacchetti di classe E e C. Tuttavia, i valori restano sempre entro i limiti QoS fissati a 3 s.

In sintesi, l'aumento di P_c comporta un aggravio del carico per l'Edge, con un peggioramento dei tempi medi di risposta, mentre l'effetto di λ si manifesta in modo uniforme come crescita monotona. L'analisi conferma la necessità di un bilanciamento ottimale dei parametri per garantire il rispetto degli SLO in scenari realistici di traffico.

25.1 Applicabilità operativa

Il modello migliorativo è quindi indicato negli slot ad alto carico e, più in generale, in contesti con variazioni diurne marcate: replicando il profilo giornaliero su 48 h si pianifica quando tenere attivi due core e quando no, mantenendo invariati routing e meccanismi di misura. In aeroporti o ambienti analoghi, il beneficio è massimo sui punti latency-critical all'Edge, dove le misure precedenti avevano evidenziato oscillazioni e code in crescita durante i picchi: con $m_{Edge} = 2$ tali fenomeni si attenuano, mentre il Coordinator rimane poco sollecitato nelle condizioni di riferimento (scala marginalmente solo per p_c molto basso), e il Cloud continua a non costituire collo di bottiglia.

26 Bibliografia

Riferimenti bibliografici

- [1] Park, S., & Geyer, D. (2018). *Performance Engineering: Learning Through Applications Using JMT*. Springer.
- [2] Aeroporti di Roma. (2024). *Dati di traffico*. Disponibile su https://www.adr.it/bsn-dati-di-traffico?p_p_id=it_adr_trafficdata_web_portlet_TrafficDataWebPortlet&p_p_lifecycle=0&p_p_state=normal&p_p_mode=view&_it_adr_trafficdata_web_portlet_TrafficDataWebPortlet_tabs1=CIA.
- [3] Aeroporti di Roma. (2024). *E-Gates*. Disponibile su <https://www.adr.it/e-gates1>, accesso il 30 agosto 2025.
- [4] Galit, W., & Ward, P. (2013). *Predictive modeling of inbound demand in airports*. Journal of Airport Operations. Disponibile su <https://www.sciencedirect.com/science/article/pii/S0377221719300438>.
- [5] Tokunaga, H. (2017). *Queue-Based Modeling of the Aircraft Arrival Process at Tokyo International Airport*. MDPI. Disponibile su <https://www.mdpi.com/2226-4310/6/10/103>.
- [6] Columbia University. (2013). *Lecture Notes on Poisson Processes*. Disponibile su https://www.columbia.edu/~ww2040/4615S15/LectureNotes_Galit_Ward_2013.pdf.
- [7] Probability Course. (2020). *The Ultimate Guide to the Poisson Process*. Disponibile su https://www.probabilitycourse.com/chapter11/11_1_2_basic_concepts_of_the_poisson_process.php.
- [8] Working with Crowds. (2018). *Analysis of Airport Security Screening Checkpoints Using Queuing Theory*. Disponibile su <https://www.workingwithcrowds.com/wp-content/uploads/2018/02/Analysis-of-Airport-Security-Screening-Checkpoints-using-Queuing.pdf>.
- [9] Smiths Detection. (2021). *Low False Alarm Rates Reduce Touch Points in Airport Security*. Disponibile su <https://www.smithsdetection.com/insights/low-false-alarm-rates-reduce-touch-points/>.
- [10] Wikipedia. (2022). *Exponential Distribution*. Disponibile su https://en.wikipedia.org/wiki/Exponential_distribution.
- [11] MIT OpenCourseWare. (2011). *Discrete Stochastic Processes: Chapter 2*. Disponibile su https://ocw.mit.edu/courses/6-262-discrete-stochastic-processes-spring-2011/3a19ce0e02d0008877351bfa24f3716a/MIT6_262S11_chap02.pdf.
- [12] Number Analytics. (2020). *The Ultimate Guide to the Arrival Process in Queueing Theory*. Disponibile su <https://www.numberanalytics.com/blog/ultimate-guide-arrival-process-queueing-theory>.
- [13] Jerry Banks, John S Carson II, Brooks Automation, Barry L Nelson, and David M Nicol. *Discrete-event system simulation fourth edition*.