

UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA
MACROAREA DI INGEGNERIA



CORSO DI STUDIO IN
INGEGNERIA INFORMATICA

PROGETTO DI
PERFORMANCE MODELING OF COMPUTER
SYSTEMS AND NETWORKS

TITOLO
SIMULAZIONE NEXT-EVENT PER SISTEMA DI
RIDE-HAILING E RIDE-SHARING

Studenti:

Greta Luna Ancora (0369455)
Gaia Meola (0369454)
Nicola Violante (0362469)

Anno Accademico 2024/2025

Abstract

Lo studio si concentra sulla modellazione e sulla simulazione del sistema di **ride-hailing** di Bolt, facendo ricorso ai principi fondamentali della teoria delle code. L'obiettivo principale è analizzare l'efficienza operativa dell'architettura del sistema attualmente adottata e valutare l'impatto dell'introduzione di un servizio di **ride-sharing**. Il lavoro si articola in due fasi principali:

1. **Analisi del sistema esistente di ride-hailing**, con particolare attenzione alla distribuzione delle richieste di servizio e all'impiego dei veicoli.
2. **Introduzione e valutazione di un servizio di ride-sharing**, concepito per ridurre l'impatto ambientale del sistema, in linea con le normative emergenti in materia di mobilità sostenibile.

Quest'ultima fase, infatti, si inserisce nel contesto normativo definito dal **Regolamento (UE) 2019/631 del Parlamento Europeo e del Consiglio** [1][2], che stabilisce i livelli di prestazione in materia di emissioni di CO_2 per le nuove autovetture e i veicoli commerciali leggeri. Tale regolamento mira a incentivare soluzioni di trasporto più sostenibili, promuovendo l'adozione di soluzioni condivise, come il ride-sharing, capace di ridurre il numero di veicoli circolanti e, di conseguenza, le emissioni per passeggero.

Indice

1	Introduzione	1
2	Oggetto di studio	2
3	Obiettivo dell'analisi	3
4	Modello concettuale	4
4.1	Stato del sistema	4
4.2	Descrizione degli eventi	5
5	Modello delle specifiche	6
5.1	Matrice di routing	6
5.2	Equazioni di traffico	7
5.3	Modellazione centri	8
5.4	Scelta delle distribuzioni	8
6	Modello computazionale	9
6.1	Pseudo-Random Number Generator	9
6.2	Stream	10
6.3	Simulation Clock	10
6.4	Eventi	11
6.5	SimpleSystem	11
6.6	SimpleMultiServerNode	11
6.7	getNextArrivalTimeSimpleCenter() e getServiceTimeSimple() .	14
7	Design degli esperimenti	16
7.1	Intervalli di confidenza	16
7.2	Analisi del transitorio	16
7.3	Simulazione ad orizzonte infinito	19
7.4	Autocorrelazione	24
8	Verifica	25
8.1	Centro automobili small M/M/33	26
8.2	Centro automobili medium M/M/6	27
8.3	Centro automobili large M/M/16	28
9	Validazione	30
10	Modello migliorativo	31

11 Modello concettuale	32
11.1 Stato del sistema	33
11.2 Descrizione degli eventi	33
12 Modello delle specifiche	34
12.1 Matrice di routing	34
12.2 Equazioni di traffico	36
12.3 Modellazione dei centri di servizio	36
12.4 Scelta delle distribuzioni	37
13 Modello computazionale	38
13.1 Eventi	38
13.2 RideSharingSystem	38
13.3 RideSharingMultiServerNode	39
13.4 getNextArrivalTimeRideSharing() e getServiceTimeRideSharing()	43
14 Design degli esperimenti	44
14.1 Intervalli di confidenza	44
14.2 Primo obiettivo	45
14.2.1 Analisi del transitorio	45
14.2.2 Simulazione ad orizzonte infinito	49
14.3 Secondo obiettivo	59
14.3.1 Analisi del transitorio	60
14.4 Studio Orizzonte Infinito	61
14.5 Autocorrelazione	64
15 Verifica	65
15.1 Controllo della coerenza delle statistiche raccolte	65
15.2 Controllo specifico sull'interazione tra le due componenti	65
16 Validazione	67
17 Conclusioni	68

1 Introduzione

Bolt è un'azienda estone attiva nel settore della mobilità urbana che, tra i servizi offerti, include il **ride-hailing** tramite una piattaforma digitale. La missione dichiarata dall'azienda è contribuire alla riduzione delle emissioni derivanti dal trasporto passeggeri, responsabile di una quota significativa delle emissioni globali. Grazie all'integrazione di più modalità di trasporto in un'unica piattaforma (automobili, biciclette, monopattini elettrici, ecc.) e al supporto alla connessione con il trasporto pubblico, Bolt si propone di costruire un modello di mobilità urbana **più efficiente, accessibile e sostenibile** rispetto a quello tradizionale basato sull'uso dell'auto privata.

2 Oggetto di studio

Prima di descrivere nel dettaglio l'oggetto di studio, è opportuno chiarire alcune ipotesi semplificative adottate per la sua costruzione: il sistema studiato rappresenta una versione astratta e controllata del servizio di ride-hailing offerto da Bolt. In particolare, si assume:

1. Un **intervallo temporale fisso**, durante il quale si concentra l'analisi delle prestazioni.
2. Il **numero di veicoli**, sia complessivamente sia per **ciascuna tipologia**, rimane invariato per l'intero intervallo temporale considerato.
3. L'**assenza di una modellazione spaziale esplicita**: le distanze tra la posizione dei passeggeri e dei veicoli sono approssimate attraverso probabilità di abbinamento, senza considerare coordinate geografiche reali.
4. Un **domanda delle richieste di corsa stabile** durante tutto l'intervallo simulato.

Il sistema oggetto di studio è composto da **tre centri di servizio**, ciascuno dedicato alla gestione delle richieste per una specifica tipologia di veicolo tra quelle offerte da Bolt. Le tipologie selezionate sono:

1. **small** che rappresenta i veicoli con una capacità da 1 a 3 posti.
2. **medium** che rappresenta i veicoli con una capacità da 4 posti.
3. **large** che rappresenta i veicoli con una capacità da 5 a 8 posti.

Queste tre tipologie rappresentano le classi di veicoli più significative per il funzionamento del sistema, pur non esaurendo l'intera gamma di opzioni presenti sulla piattaforma. Un utente che desidera richiedere una corsa accede alla piattaforma digitale, seleziona la tipologia di veicolo desiderata e invia la richiesta di corsa. Nella realtà, l'assegnazione di una richiesta ad un veicolo avviene tramite un algoritmo multi-parametrico di **matching**, che considera vari fattori come distanza, priorità, disponibilità del conducente etc. Nel nostro studio tale processo è **semplificato** adottando una politica di tipo **selection in order**, in cui ogni richiesta viene assegnata al primo veicolo disponibile. Il veicolo assegnato rimane occupato per tutta la durata della corsa, al termine della quale torna automaticamente disponibile e può accettare nuove richieste. Inoltre, un utente potrebbe decidere di cancellare la propria richiesta di corsa prima che questa sia assegnata ad un veicolo.

3 Obiettivo dell'analisi

Seguendo la politica di Bolt incentrata su una mobilità sostenibile, si è scelto di introdurre nel modello esistente un nuovo servizio di **ride-sharing** con l'obiettivo di contenere le emissioni e ottimizzare l'utilizzo dei veicoli. Lo studio, in particolare, si propone due obiettivi principali grazie all'introduzione di tale servizio:

1. Il primo obiettivo è **gestire lo stesso carico di lavoro**, individuando il **numero ottimale di veicoli** per tipologia da assegnare al servizio introdotto, in modo da ottenere una distribuzione del carico tra i centri di servizio quanto più omogenea possibile.
2. Il secondo obiettivo è la **riduzione del numero totale di veicoli** impiegati, senza compromettere le prestazioni osservate nel modello base.

4 Modello concettuale

Il servizio di Bolt è stato modellato come in Figura 1.

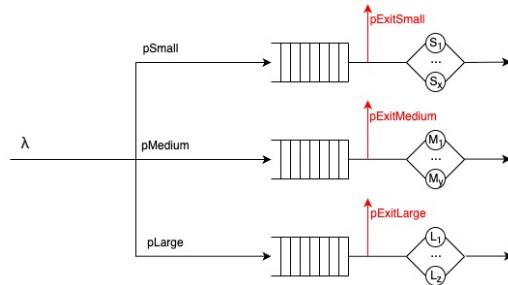


Figura 1: Modello concettuale del servizio di ride-hailing

Gli elementi principali di questo sistema sono:

1. **Utente**: colui che desidera richiedere una corsa.
2. **Auto**: la tipologia di veicolo che può essere richiesta da un utente.

In particolare si ha:

1. Centro di **servizio tradizionale** per veicoli **small**: gestisce le corse effettuate da veicoli con capacità da **1 a 3 passeggeri**.
2. Centro di **servizio tradizionale** per veicoli **medium**: gestisce le corse effettuate da veicoli con capacità da **4 passeggeri**.
3. Centro di **servizio tradizionale** per veicoli **large**: gestisce le corse effettuate da veicoli con capacità da **5 a 8 passeggeri**.

4.1 Stato del sistema

Nel sistema **tradizionale**, lo stato in un determinato istante è definito dal vettore

$$(n_{small}, n_{medium}, n_{large})$$

- $n_{small}, n_{medium}, n_{large}$: numero di richieste attualmente presenti nei centri di servizio, suddivise in base alla tipologia di veicolo richiesta (small, medium, large).

4.2 Descrizione degli eventi

Nel sistema **tradizionale**, gli eventi che modificano lo stato sono:

1. L'**arrivo di una richiesta** di corsa per una specifica **tipologia** di veicolo.
2. Il **termine di una corsa** che rende un veicolo nuovamente **disponibile**.
3. L'**uscita dal sistema** di un utente che ha annullato la propria **richiesta di servizio**.

5 Modello delle specifiche

Il **tempo medio di servizio** $E[S]$ e il **numero totale di veicoli** N sono stati ricavati dai dati presentati nello studio [3]. Successivamente, il totale N è stato suddiviso nelle tre partizioni N_{Small} , N_{Medium} e N_{Large} , in proporzione alla probabilità di ricevere una richiesta associata a ciascuna tipologia di veicolo (si veda §5.1). In questo modo, il modello rispecchia la distribuzione attesa della domanda, allocando più veicoli alle categorie con maggiore frequenza di richieste.

Poiché lo studio si concentra su una **fascia operativa a carico medio-alto**, il coefficiente di utilizzo ρ è stato fissato a **0.7**. A partire dai valori di $E[S]$, dal numero di veicoli disponibili per tipologia e dal valore di ρ , è possibile calcolare il **tasso di arrivo medio** λ tramite le seguenti formule:

$$\begin{aligned} \bullet \quad \lambda_{Small} &= \frac{\rho}{\frac{E(S_{Server})}{N_{Small}}} \\ \bullet \quad \lambda_{Small} &= (1 - p_{ExitSmall}) \cdot p_{Small} \cdot \lambda \\ \implies \lambda &= \frac{\rho \cdot N_{Small}}{E(S_{Server}) \cdot (1 - p_{ExitSmall}) \cdot p_{Small}} \end{aligned}$$

5.1 Matrice di routing

Il modello utilizza tre probabilità per ripartire le richieste tra le diverse tipologie di veicoli, indicate rispettivamente con \mathbf{P}_{Small} , \mathbf{P}_{Medium} e \mathbf{P}_{Large} , a cui sono stati assegnati i valori **0.6**, **0.1** e **0.3**. In particolare, esse rappresentano la probabilità di ricevere una richiesta da 1 a 3 posti (\mathbf{P}_{Small}), da 4 posti (\mathbf{P}_{Medium}) o da 5 a 8 posti (\mathbf{P}_{Large}). Questa ripartizione riflette l'ipotesi che la maggior parte delle richieste riguardi un numero ridotto di posti, mentre solo una quota minore coinvolga gruppi più numerosi. Inoltre, vengono introdotte delle probabilità di abbandono per ogni coda, denotate con $\mathbf{P}_{ExitSmall}$, $\mathbf{P}_{ExitMedium}$ e $\mathbf{P}_{ExitLarge}$, che rappresentano la possibilità che un utente in attesa decida di annullare la propria richiesta di corsa. A quest'ultime è stato assegnato il valore **0.05**, ipotizzando che sia poco probabile che un utente scelga di cancellare la propria richiesta.

Probabilità	Descrizione	Valore
P_{Small}	Probabilità di ricevere una richiesta da 1 a 3 posti	0.6
P_{Medium}	Probabilità di ricevere una richiesta da 4 posti	0.1
P_{Large}	Probabilità di ricevere una richiesta da 5 a 8 posti	0.3
$P_{ExitSmall}$	Probabilità che un utente in coda Small annulli la richiesta	0.05
$P_{ExitMedium}$	Probabilità che un utente in coda Medium annulli la richiesta	0.05
$P_{ExitLarge}$	Probabilità che un utente in coda Large annulli la richiesta	0.05

Tabella 1: Riepilogo delle probabilità utilizzate nel modello

	Esterno	Centro Small	Centro Medium	Centro Large
Esterno	0	p_{Small}	p_{Medium}	p_{Large}
Centro Small	1	0	0	0
Centro Medium	1	0	0	0
Centro Large	1	0	0	0

Tabella 2: Tabella di routing del modello base.

5.2 Equazioni di traffico

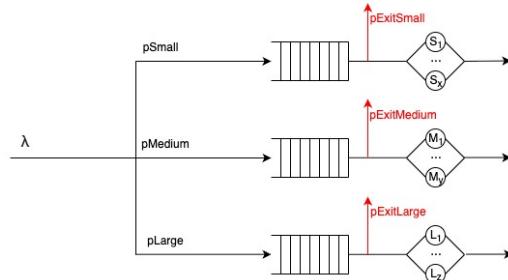


Figura 2: Modello concettuale del servizio di ride-hailing

Le equazioni di traffico del sistema sono:

$$\begin{cases} \lambda_{Small} = p_{Small}(1 - p_{ExitSmall})\lambda \\ \lambda_{Medium} = p_{Medium}(1 - p_{ExitMedium})\lambda \\ \lambda_{Large} = p_{Large}(1 - p_{ExitLarge})\lambda \end{cases}$$

5.3 Modellazione centri

I centri di servizio "classici" sono modellati come code **M/N/k**, ovvero caratterizzati da arrivi di tipo **Poisson**, servizi di tipo **normali**, un numero **finito** di server e da una **coda di attesa a capacità infinita**, alimentata da un pool di utenti. Nel sistema sono presenti tre centri di questo tipo, ciascuno dedicato a una specifica tipologia di veicolo: **small**, **medium** e **large**. Il numero di server in ciascun centro è determinato in funzione della categoria di veicolo che esso gestisce.

5.4 Scelta delle distribuzioni

In assenza di dati sufficienti per un'analisi statistica precisa, le scelte adottate per modellare i tempi di arrivo e di servizio nei diversi centri mirano a rappresentare in modo realistico il comportamento dei processi.

- **Distribuzione Gaussiana Troncata – Tempi di Servizio**

Per modellare i tempi di servizio è stata adottata una distribuzione **Gaussiana troncata**, considerata adatta a descrivere un processo con forte concentrazione dei valori attorno alla media. Questa distribuzione permette, infatti, di catturare l'andamento centrato dei tempi di servizio, pur lasciando spazio a variazioni legate a corse più brevi o più lunghe. La troncatura inferiore evita la generazione di tempi irrealisticamente bassi, mentre quella superiore pone un limite massimo coerente con le tempistiche reali del sistema.

- **Distribuzione Esponenziale – Tempi di Arrivo**

Per modellare gli istanti di arrivo delle richieste si è scelta la distribuzione **esponenziale**, ampiamente impiegata in ambito modellistico per rappresentare processi di arrivo casuali. La scelta è giustificata dalla proprietà di assenza di memoria tipica di questa distribuzione: la probabilità di un prossimo arrivo non dipende dal tempo trascorso dall'ultimo evento. Questo riflette adeguatamente l'elevata aleatorietà del flusso di richieste e l'indipendenza tra arrivi successivi.

6 Modello computazionale

Come linguaggio di programmazione per la simulazione è stato scelto **Java**, principalmente per i vantaggi che offre, tra cui la gestione automatica della memoria e l'approccio **object-oriented**, particolarmente adatto alla modellazione di sistemi complessi. Per la generazione dei grafici è stato, invece, utilizzato **Python**, grazie alla disponibilità della libreria `matplotlib`, che risulta semplice ed efficace per la creazione di rappresentazioni grafiche.

Per garantire una struttura chiara, modulare e facilmente manutenibile, il codice è stato organizzato in **sei packages principali**:

- **Centers**: contiene le classi che implementano le due tipologie di centri (base e migliorativo), incapsulando la logica di creazione e di processamento degli eventi del sistema.
- **Configuration**: include la classe di supporto per la gestione e l'impostazione dei parametri di configurazione.
- **Controller**: raccoglie le classi che implementano i due sistemi (base e migliorativo) e che integrano al loro interno la logica simulativa sia per l'**analisi del transitorio** che per lo studio ad **orizzonte infinito**.
- **Libs**: contiene il codice sviluppato da *Steve Park* e *Dave Geyer* che fornisce l'implementazione delle distribuzioni statistiche e dei generatori pseudo-casuali utilizzati nella simulazione.
- **Model**: comprende le classi di supporto alla simulazione **next-event** e alla raccolta delle statistiche.
- **Utils**: raccoglie le classi ausiliarie per l'elaborazione e l'analisi delle statistiche prodotte durante le diverse simulazioni.

6.1 Pseudo-Random Number Generator

Per la generazione dei numeri pseudo-casuali, il modello utilizza la libreria `rngs`, una versione multi-stream compatibile con la libreria `rng`. Questa libreria gestisce fino a 256 stream di numeri casuali, dei quali solo uno è attivo alla volta. Durante la simulazione, ogni volta che è necessario generare un numero casuale per una componente del modello, lo stream corrispondente viene attivato tramite `SelectStream` e il numero viene prodotto tramite `Random`. Tutti gli stream di numeri casuali vengono inizializzati una sola volta

tramite `PlantSeeds`, che assegna a ciascun stream uno stato iniziale differente. All'inizio di ogni replica, lo stato di ciascun stream viene recuperato tramite `GetSeed`.

6.2 Stream

Gli stream vengono assegnati come segue:

- Lo **stream 1** è dedicato alla funzione `getNextArrivalTimeSimpleCenter()`;
- Lo **stream 2** è dedicato alla funzione `getNextArrivalTimeRideSharing()`;
- Lo **stream 3** è dedicato alla funzione `getServiceTimeSimple()`;
- Lo **stream 4** è dedicato alla funzione `getServiceTimeRideSharing()`;
- Lo **stream 5** gestisce la probabilità di uscita dal servizio tradizionale (`P_EXIT`);
- Lo **stream 6** gestisce la probabilità di uscita dal servizio ride-sharing (`P_EXIT`);
- Lo **stream 7** gestisce la probabilità di match con risorsa occupata (`P_MATCH_BUSY`);
- Lo **stream 8** è dedicato alla funzione `getNumeroPosti()`.

La simulazione adotta l'approccio della **Next-Event Simulation**, che consiste nel far avanzare il **clock** della simulazione elaborando gli eventi in ordine cronologico. Si identifica l'evento più imminente, si esegue l'azione corrispondente e si aggiornano di conseguenza lo stato del sistema e il **clock** della simulazione. Nei paragrafi seguenti verranno analizzate alcune delle componenti più significative di questo meccanismo.

6.3 Simulation Clock

Una delle principali strutture dati del simulatore è `MsqTime`, che rappresenta il **clock** della simulazione:

```
public class MsqTime {  
    double current;  
    double next;  
}
```

Gli attributi della classe hanno il seguente significato:

- **current**: istante temporale corrente della simulazione;
- **next**: istante temporale in cui si verificherà il prossimo evento.

6.4 Eventi

Gli eventi del sistema sono modellati tramite la classe `MsqEvent`, definita come segue:

```
public class MsqEvent {  
    double t;  
    int x;  
}
```

Gli attributi della classe sono:

- **t**: istante temporale in cui l'evento si verifica;
- **x**: variabile di stato che specifica se l'evento è attivo oppure no.

6.5 SimpleSystem

La logica di simulazione è implementata all'interno della classe `SimpleSystem`, che mette a disposizione due modalità di esecuzione:

- `runFiniteSimulation()`, che utilizza la **tecnica delle repliche** per analizzare la fase transitoria del sistema;
- `runInfiniteSimulation()`, che implementa una simulazione a orizzonte infinito basata sulla **tecnica delle batch means**. Questa modalità costituisce l'approccio effettivamente adottato per il calcolo delle **statistiche prestazionali**, consentendo di ottenere stime affidabili in regime stazionario.

6.6 SimpleMultiServerNode

La classe `SimpleMultiServerNode` incapsula la logica operativa dei centri di servizio. Il metodo `processNextEvent()` si occupa della gestione dell'evento temporalmente più vicino nella coda degli eventi del nodo: esso aggiorna lo stato interno ed esegue le azioni associate all'evento.

```

{
    public int processNextEvent(double t) {
        int e = peekNextEventType();
        MsqEvent ev = event.get(e);
        clock.next = ev.t;

        clock.current = clock.next;

        if (e == ARRIVAL || e > numberOfServersInTheCenter) {
            if (e == ARRIVAL) {

                lastArrivalTimeInBatch = clock.current;

                numberJobInSystem++;

                MsqEvent arr = event.getFirst();
                arr.t = distrs.getNextArrivalTimeSimpleCenter(rng,
                    ↳ system, centerIndex, clock.current);
                arr.x = 1;

                rng.selectStream(5);
                double rnd = rng.random();

                if (rnd < P_EXIT) {
                    numberJobInSystem--;
                    return -1;
                }
            }

            int serverIndex = findOne();

            if (serverIndex != -1) {

                double serviceTimeSimple = distrs.getServiceTimeSimple
                    ↳ (rng);

                MsqEvent sEvent = event.get(serverIndex);
                sEvent.t = clock.current + serviceTimeSimple;

                serversCompletion[serverIndex].setLastCompletionTime
                    ↳ (sEvent.t);
            }
        }
    }
}

```

```

        sEvent.x = 1;
        sum[serverIndex].service += serviceTimeSimple;

        if (e > numberServersInTheCenter) {
            event.remove(e);
        }
        return serverIndex;
    }
} else {
    numberJobInSystem--;
    sum[e].served++;

    if (numberJobInSystem >= numberServersInTheCenter) {
        double serviceTimeSimple = distrs.getServiceTimeSimple
            ↪ (rng);

        MsqEvent sEvent = event.get(e);
        sEvent.t = clock.current + serviceTimeSimple;

        serversCompletion[e].setLastCompletionTime(sEvent.t)
            ↪ ;
        sum[e].service += serviceTimeSimple;
        return e;
    }

    } else {
        event.get(e).x = 0;
    }
}
return -1;
}
}

```

Nel caso di elaborazione di un arrivo, il sistema verifica la possibilità di assegnare la richiesta a uno dei server disponibili. A tal fine viene adottata una politica di **selection in order**, implementata dal metodo `findOne()`: la richiesta viene quindi instradata al primo server libero individuato nell'ordine di scansione.

```
{
private int findOne() {
    for (int i = 1; i < event.size(); i++) {
        if (event.get(i).x == 0) return i;
    }
}
```

```

        }
        return -1;
    }
}

```

6.7 getNextArrivalTimeSimpleCenter() e getServiceTimeSimple()

Per generare i tempi di arrivo e di servizio, il modello sfrutta la classe `Rngs` con un approccio multi-stream, garantendo l'indipendenza tra le sequenze di numeri casuali utilizzate dalle diverse componenti del sistema.

Si mostra come esempio il codice di `getNextArrivalTimeSimpleCenter()`:

```

{
    public double getNextArrivalTimeSimpleCenter(Rngs r, Sistema system
        , int centerIndex, double sarrival) {
        r.selectStream(1);
        double lambda = config.getDouble("simulation","lambdasimple")
            ;

        if(system instanceof SimpleSystem) {
            switch (centerIndex) {
                case 0 -> lambda *= config.getDouble("simulation",
                    "p_small");
                case 1 -> lambda *= config.getDouble("simulation",
                    "p_medium");
                case 2 -> lambda *= config.getDouble("simulation",
                    "p_large");
                default -> System.out.println("Centro inesistente!");
            }
        }

    }else{

        switch (centerIndex) {
            case 0 -> lambda *= config.getDouble("simulation",
                "p_small") * config.getDouble("simulation",
                "psimple");
            case 1 -> lambda *= config.getDouble("simulation",
                "p_medium") * config.getDouble("simulation",
                "psimple");
    }
}

```

```

        case 2 -> lambda *= config.getDouble("simulation",
            "p_large") * config.getDouble("simulation",
            "psimple") ;
        default -> System.out.println("Centro inesistente!");

    }
}

sarrival += exponential(1/lambda, r);
return sarrival;
}
}

```

Mentre `getServiceTimeSimple()` ha la seguente implementazione:

```

{
    public double getServiceTimeSimple(Rngs r) {
        r.selectStream(3);
        double esi = config.getDouble("simulation","esi");
        double alpha, beta;
        double a = 2;
        double b = 30;

        alpha = cdfNormal(esi, 4, a);
        beta = 1 - cdfNormal(esi, 4, b);

        double u = uniform(alpha, 1 - beta, r);
        return idfNormal(esi, 4, u);
    }
}

```

7 Design degli esperimenti

7.1 Intervalli di confidenza

All'interno della trattazione sono stati utilizzati gli **intervalli di confidenza**. Un intervallo di confidenza è un intervallo calcolato a partire dai dati campionari, che ha una certa probabilità di contenere il valore reale della media della popolazione. In altre parole, fornisce una stima dell'incertezza associata alla media campionaria, indicando un intervallo plausibile entro cui ci si può ragionevolmente aspettare che la media reale del sistema si trovi.

Per questo studio sono stati scelti intervalli con un **livello di confidenza** pari al **95%**. Gli intervalli di confidenza sono stati calcolati utilizzando la seguente formula:

$$\text{Intervallo di Confidenza} = t^* \left(\frac{s}{\sqrt{n-1}} \right)$$

dove

- **s** è la deviazione standard campionaria calcolata tramite l'algoritmo di **Welford**;
- **n** è la dimensione del campione;
- **t*** è il valore critico della distribuzione *t* di **Student** (nel codice, il calcolo è stato effettuato utilizzando la funzione `idfStudent()` fornita dalla classe `Rvms`).

7.2 Analisi del transitorio

Prima di procedere con l'esecuzione degli esperimenti, è fondamentale analizzare la fase transitoria del sistema per verificare quando esso tende a uno stato stazionario. A tal fine, si utilizza la tecnica delle **repliche**: la simulazione viene eseguita più volte a partire dallo stesso stato iniziale del sistema. Per ciascuna replica, l'intervallo temporale considerato è fissato a **2.880 minuti**, corrispondenti a due giorni di attività del sistema.

La variabile utilizzata come indicatore della convergenza transitoria è il tempo medio di risposta $E[T_s]$, come illustrato nelle figure sottostanti:

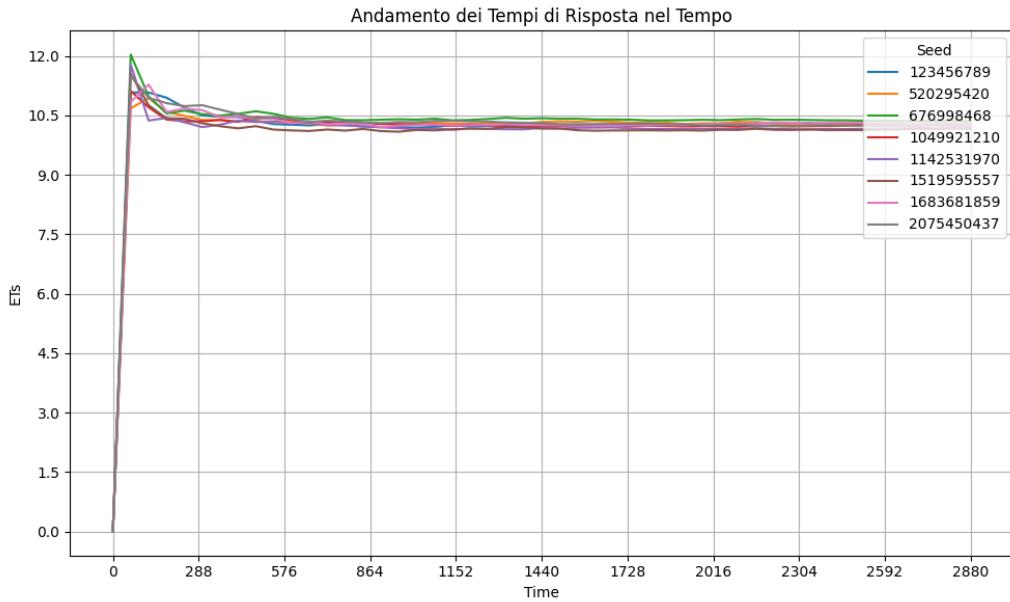


Figura 3: Centro small

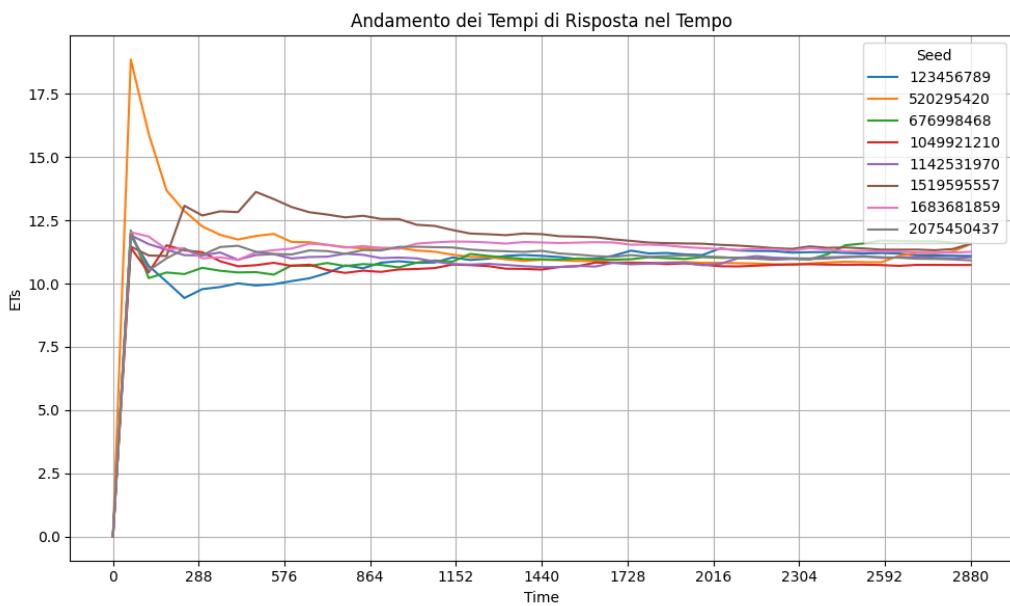


Figura 4: Centro medium

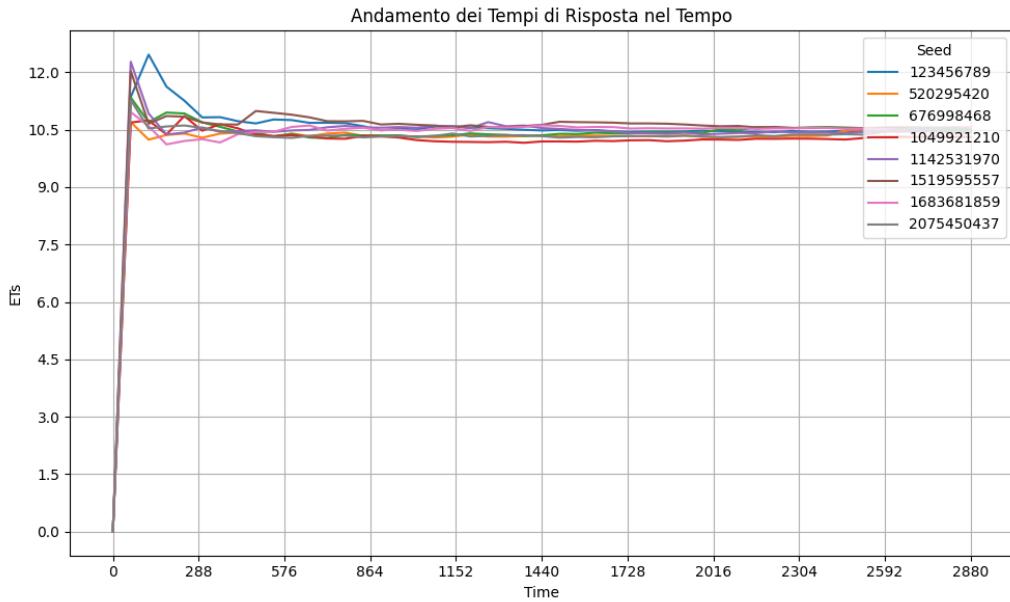


Figura 5: Centro large

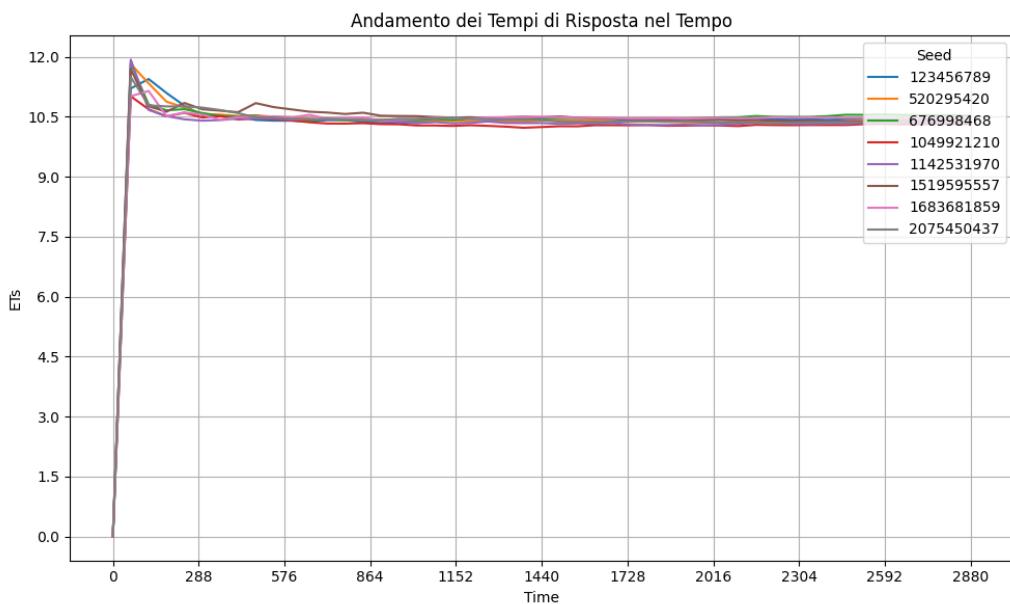


Figura 6: Risultati globali

Si osserva che sia il sistema sia i centri raggiungono il regime stazionario, con il sistema che in particolare converge dopo circa **864 minuti**.

7.3 Simulazione ad orizzonte infinito

Considerando che Bolt è un'applicazione già consolidata e ampiamente adottata dagli utenti, che lo studio si concentra su una singola fascia operativa e che la simulazione parte da uno stato iniziale vuoto, l'analisi ad orizzonte infinito risulta particolarmente appropriata, in quanto consente di ottenere stime delle prestazioni operative prive del bias legato alla fase transitoria.

Per stimare le statistiche di interesse, si è applicato il metodo delle **Batch Means**. I parametri scelti sono stati **b = 2048** (dimensione di ciascun batch) e **k = 512** (numero complessivo di batch). Tali valori non sono stati fissati a priori: inizialmente si era optato per $b = 256$ e $k = 64$, ma entrambi i parametri sono stati aumentati progressivamente fino a garantire che la **lag-1 autocorrelazione** tra le medie di batch risultasse inferiore a **0.2**, in accordo con quanto raccomandato in [4]. In questo modo si è raggiunto un compromesso soddisfacente perché da un lato le medie dei batch risultano sufficientemente indipendenti; dall'altro, il numero di batch rimane abbastanza ampio da permettere una stima affidabile della varianza.

Eseguendo la simulazione si osservano i seguenti risultati:

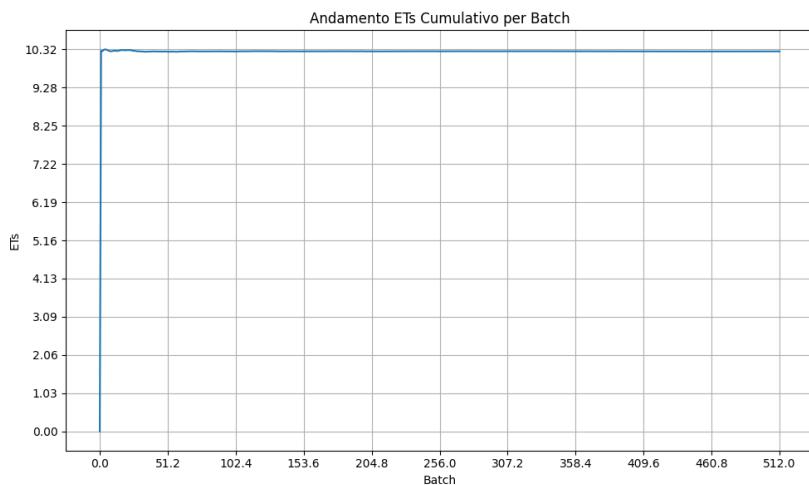


Figura 7: Centro small

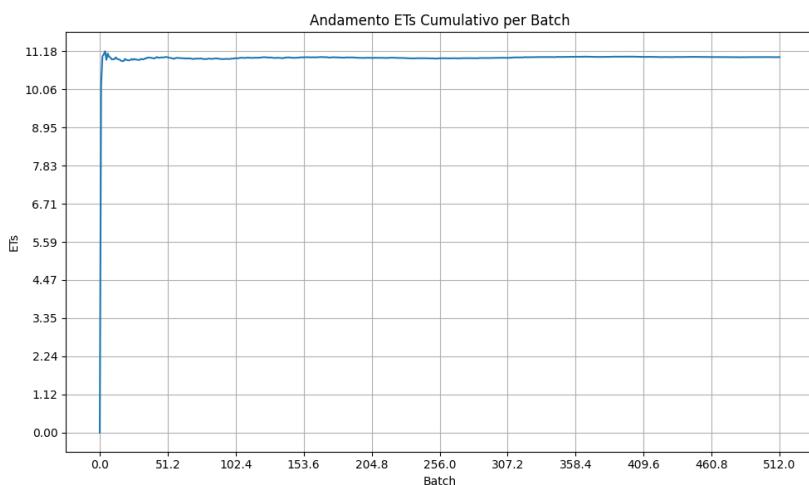


Figura 8: Centro medium

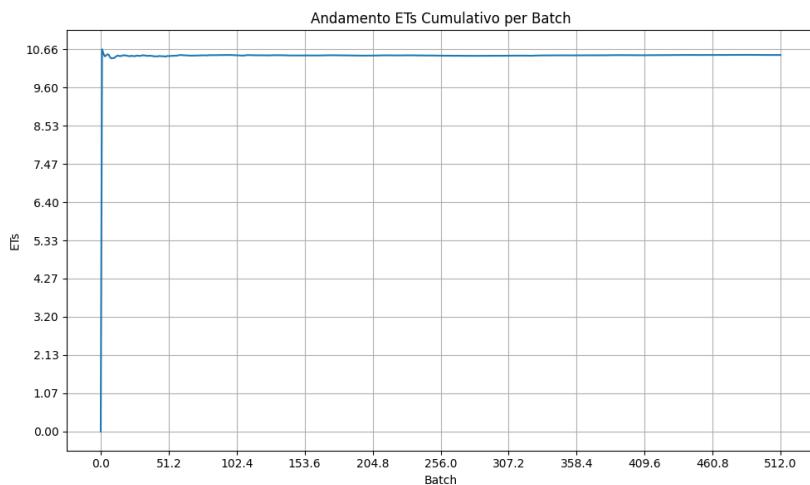


Figura 9: Centro large

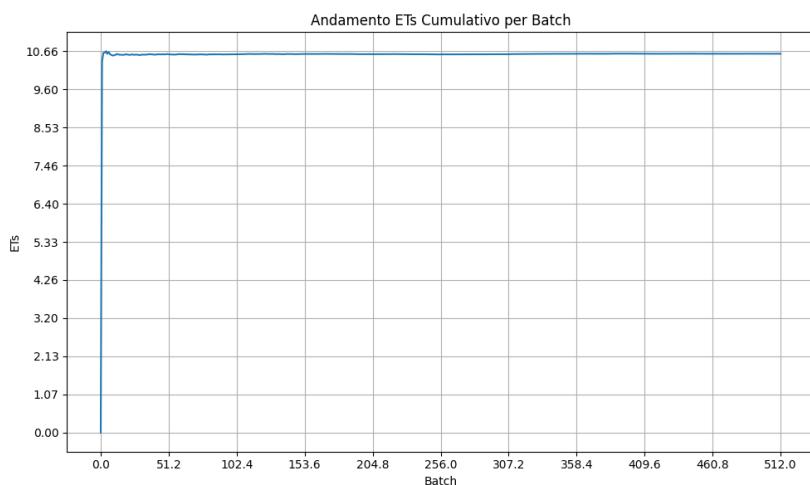


Figura 10: Risultati globali

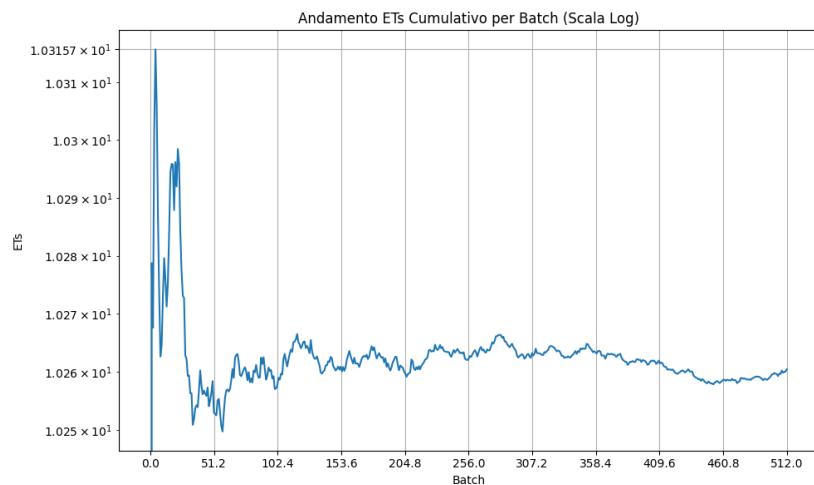


Figura 11: Centro small in scala logaritmica

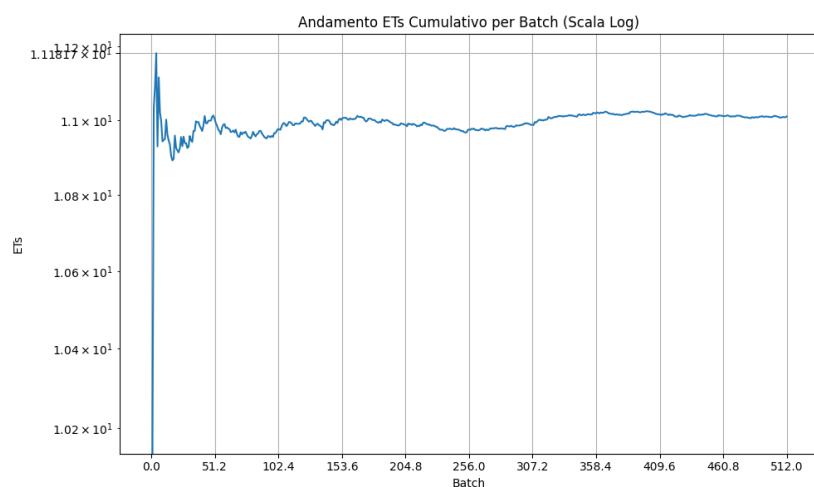


Figura 12: Centro medium in scala logaritmica

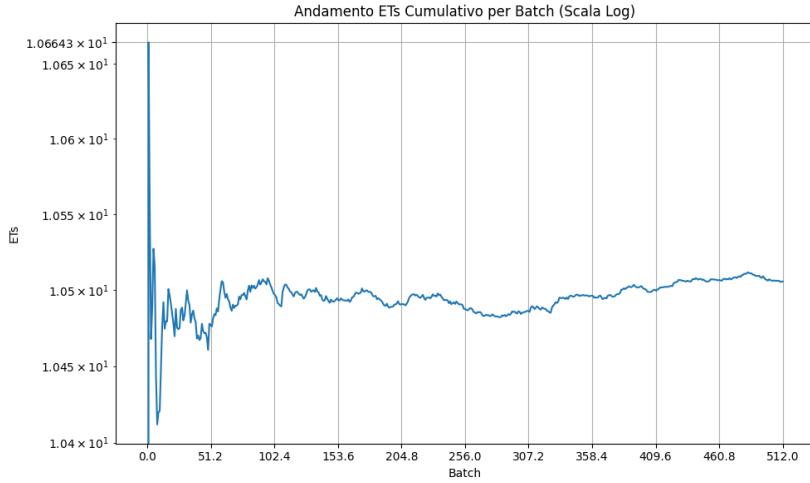


Figura 13: Centro large in scala logaritmica

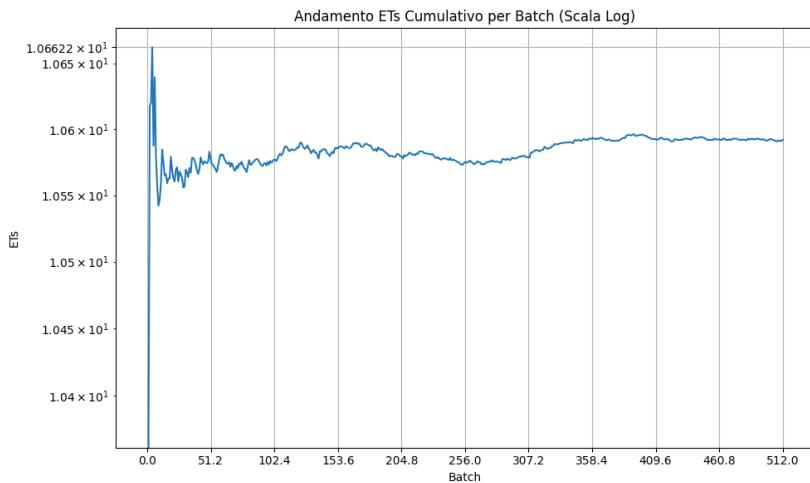


Figura 14: Risultati globali in scala logaritmica

Centro	$E[N_S]$	$E[T_S]$	$E[N_Q]$	$E[T_Q]$	ρ	$E[S_i]$
small	23.6930 ± 0.0584	10.2604 ± 0.0099	0.0813 ± 0.0076	0.0348 ± 0.0032	0.7155 ± 0.0017	10.2256 ± 0.0090
medium	4.2429 ± 0.0368	11.0098 ± 0.0498	0.3093 ± 0.0164	0.7873 ± 0.0388	0.6556 ± 0.0039	10.2225 ± 0.0218
large	12.1358 ± 0.0518	10.5056 ± 0.0217	0.3333 ± 0.0181	0.2854 ± 0.0148	0.7377 ± 0.0024	10.2202 ± 0.0129

Tabella 3: Statistiche riassuntive simulazione ad orizzonte infinito del modello base

7.4 Autocorrelazione

Di seguito sono riportati i valori dell'autocorrelazione delle statistiche forniti dal simulatore, che utilizza la classe di libreria `Acs.java`:

```
*****  
AUTOCORRELATION VALUES FOR Center0 [B:2048|K:512]  
*****  
E[Ts]: -0,0116  
E[Tq]: 0,0751  
E[Si]: -0,0108  
E[Ns]: 0,0300  
E[Nq]: 0,0772  
ρ: 0,0317  
λ: 0,0337  
*****
```

Figura 15: Valori di autocorrelazione del centro small

```
*****  
AUTOCORRELATION VALUES FOR Center1 [B:2048|K:512]  
*****  
E[Ts]: -0,0621  
E[Tq]: -0,0420  
E[Si]: -0,0279  
E[Ns]: -0,0517  
E[Nq]: -0,0394  
ρ: -0,0506  
λ: -0,0233  
*****
```

Figura 16: Valori di autocorrelazione del centro medium

```
*****  
AUTOCORRELATION VALUES FOR Center2 [B:2048|K:512]  
*****  
E[Ts]: -0,0225  
E[Tq]: -0,0057  
E[Si]: 0,0216  
E[Ns]: 0,0491  
E[Nq]: 0,0006  
ρ: 0,0775  
λ: 0,1078  
*****
```

Figura 17: Valori di autocorrelazione del centro large

8 Verifica

La verifica mira a dimostrare la correttezza dell’implementazione confrontando i risultati di simulazione con soluzioni analitiche entro un intervallo di confidenza del 95% (vedi §9.1). Per rendere possibile il confronto, è stata cambiata la distribuzione dei tempi di servizio in una **esponenziale**. La verifica si concentra sul caso stazionario, utilizzando una simulazione ad **orizzonte infinito**. Per ogni centro vengono riportati i calcoli analitici e allegato lo screenshot della verifica automatica eseguita dal simulatore.

8.1 Centro automobili small M/M/33

$$\lambda = 4.05263161 \text{ richieste/min}$$

$$\lambda_{Small} = \lambda \cdot p_{Small} \cdot (1 - p_{ExitSmall}) = 2.31000002 \text{ richieste/min}$$

$$E(S_i) = \frac{1}{\mu_i} = 10 \text{ min}$$

$$E(S) = \frac{E(S_i)}{33} = 0.30 \text{ min}$$

$$\mu_i = \frac{1}{E(S_i)} = 0.1 \text{ job/min}$$

$$\rho = \frac{\lambda_1}{33 \mu_i} = 0.70000001$$

$$m = 33$$

$$p_0 = \left[\sum_{i=0}^{m-1} \frac{(m \rho)^i}{i!} + \frac{(m \rho)^m}{m! (1-\rho)} \right]^{-1} = 9.23785633 \times 10^{-11}$$

$$P_Q = \frac{(m \rho)^m p_0}{m! (1-\rho)} = 0.03539644$$

$$E(T_q) = \frac{P_Q \cdot E(S)}{1-\rho} = 0.03575398 \text{ min}$$

$$E(N_q) = E(T_q) \lambda_1 = 0.0825917$$

$$E(T_s) = E(T_q) + E(S_i) = 10.03575398 \text{ min}$$

$$E(N_s) = E(T_s) \lambda_1 = 23.18259187$$

```
Center0
E[Ts]: mean 10,0549, diff 0,0191 is within ±0,0249
E[Tq]: mean 0,0364, diff 0,0006 is within ±0,0042
E[Si]: mean 10,0185, diff 0,0185 is within ±0,0235
E[Ns]: mean 23,2178, diff 0,0352 is within ±0,0763
E[Nq]: mean 0,0849, diff 0,0023 is within ±0,0100
ρ: mean 0,7010, diff 0,0010 is within ±0,0022
λ: mean 2,3092, diff 0,0008 is within ±0,0051
```

Figura 18: Valori di verifica automatica per il centro small

8.2 Centro automobili medium M/M/6

$$\lambda = 4.05263161 \text{ richieste/min}$$

$$\lambda_{Medium} = \lambda \cdot p_{Medium} \cdot (1 - p_{ExitMedium}) = 0.385 \text{ richieste/min}$$

$$E(S_i) = \frac{1}{\mu_i} = 10 \text{ min}$$

$$E(S) = \frac{E(S_i)}{6} = 1.6 \text{ min}$$

$$\mu_i = \frac{1}{E(S_i)} = 0.1 \text{ job/min}$$

$$\rho = \frac{\lambda_2}{6 \mu_i} = 0.6416$$

$$m = 6$$

$$p_0 = \left[\sum_{i=0}^{m-1} \frac{(m \rho)^i}{i!} + \frac{(m \rho)^m}{m! (1-\rho)} \right]^{-1} = 0.01976355$$

$$P_Q = \frac{(m \rho)^m p_0}{m! (1-\rho)} = 0.24946504$$

$$E(T_q) = \frac{P_Q \cdot E(S)}{1 - \rho} = 1.16030201 \text{ min}$$

$$E(N_q) = E(T_q) \lambda_2 = 0.44671627$$

$$E(T_s) = E(T_q) + E(S_i) = 11.16030201 \text{ min}$$

$$E(N_s) = E(T_s) \lambda_2 = 4.29671627$$

```
Center1
E[Ts]: mean 11,0491, diff 0,1112 is within ±0,1183
E[Tq]: mean 1,0977, diff 0,0626 is within ±0,0801
E[Si]: mean 9,9514, diff 0,0486 is within ±0,0555
E[Ns]: mean 4,2615, diff 0,0352 is within ±0,0582
E[Nq]: mean 0,4318, diff 0,0149 is within ±0,0331
ρ: mean 0,6383, diff 0,0034 is within ±0,0051
λ: mean 0,3848, diff 0,0002 is within ±0,0021
```

Figura 19: Valori di verifica automatica per il centro medium

8.3 Centro automobili large M/M/16

$$\lambda = 4.05263161 \text{ richieste/min}$$

$$\lambda_{Large} = \lambda \cdot p_{Large} \cdot (1 - p_{ExitLarge}) = 1.15500001 \text{ richieste/min}$$

$$E(S_i) = \frac{1}{\mu_i} = 10 \text{ min}$$

$$E(S) = \frac{E(S_i)}{16} = 0.625 \text{ min}$$

$$\mu_i = \frac{1}{E(S_i)} = 0.1 \text{ job/min}$$

$$\rho = \frac{\lambda_3}{16 \mu_i} = 0.72187501$$

$$m = 16$$

$$p_0 = \left[\sum_{i=0}^{m-1} \frac{(m \rho)^i}{i!} + \frac{(m \rho)^m}{m! (1-\rho)} \right]^{-1} = 9.25463973 \times 10^{-6}$$

$$P_Q = \frac{(m \rho)^m p_0}{m! (1-\rho)} = 0.15951876$$

$$E(T_q) = \frac{P_Q \cdot E(S)}{1 - \rho} = 0.35846913 \text{ min}$$

$$E(N_q) = E(T_q) \lambda_3 = 0.41403185$$

$$E(T_s) = E(T_q) + E(S_i) = 10.35846913 \text{ min}$$

$$E(N_s) = E(T_s) \lambda_3 = 11.96403195$$

```
Center2
E[Ts]: mean 10,3514, diff 0,0071 is within ±0,0488
E[Tq]: mean 0,3583, diff 0,0001 is within ±0,0260
E[Si]: mean 9,9930, diff 0,0070 is within ±0,0331
E[Ns]: mean 11,9575, diff 0,0066 is within ±0,0720
E[Nq]: mean 0,4186, diff 0,0046 is within ±0,0312
ρ: mean 0,7212, diff 0,0007 is within ±0,0031
λ: mean 1,1548, diff 0,0002 is within ±0,0035
```

Figura 20: Valori di verifica automatica per il centro large

Indice	Valore analitico	Valore empirico	Intervallo di confidenza
$E[N_S]$	23.1826	23.2178	± 0.0763
$E[T_S]$	10.0358	10.0549	± 0.0249
$E[N_Q]$	0.0826	0.0849	± 0.0100
$E[T_Q]$	0.0358	0.0364	± 0.0042
ρ	0.7	0.7010	± 0.0022

Tabella 4: Medie analitiche, empiriche e intervalli di confidenza del centro small

Indice	Valore analitico	Valore empirico	Intervallo di confidenza
$E[N_S]$	4.2967	4.2611	± 0.0582
$E[T_S]$	11.1603	11.0491	± 0.1183
$E[N_Q]$	0.4467	0.4318	± 0.0331
$E[T_Q]$	1.1603	1.0977	± 0.0801
ρ	0.6417	0.6383	± 0.0051

Tabella 5: Medie analitiche, empiriche e intervalli di confidenza del centro medium

Indice	Valore analitico	Valore empirico	Intervallo di confidenza
$E[N_S]$	11.9640	11.9875	± 0.0720
$E[T_S]$	10.3585	10.3514	± 0.0488
$E[N_Q]$	0.4140	0.4186	± 0.0312
$E[T_Q]$	0.3585	0.3583	± 0.0260
ρ	0.7218	0.7212	± 0.0031

Tabella 6: Medie analitiche, empiriche e intervalli di confidenza del centro large

9 Validazione

La fase di validazione ha l’obiettivo di verificare che il modello computazionale sia coerente e rappresentativo del sistema reale che si intende simulare. In genere, questa fase prevede il confronto dei risultati del modello con dati empirici o con misurazioni del sistema reale, al fine di accettarne la correttezza e la capacità predittiva.

Nel nostro caso, tuttavia, non è stato possibile effettuare una validazione, in quanto non sono disponibili dati osservati del sistema in esame. Tale mancanza di informazioni ci impedisce di confrontare direttamente le prestazioni simulate con il sistema reale.

10 Modello migliorativo

Nella seconda fase dello studio, si ipotizza l'introduzione di un servizio di **ride-sharing** all'interno dell'applicazione Bolt, con l'obiettivo di consentire a più passeggeri di condividere lo stesso veicolo. In questo scenario, una richiesta può essere abbinata a un veicolo condiviso solo se entrambe le seguenti condizioni risultano soddisfatte:

1. Il veicolo deve avere un numero **sufficiente** di **posti liberi** per ospitare tutti i passeggeri coinvolti.
2. L'itinerario della nuova richiesta deve essere **compatibile** con quelle già in corso.

Nelle piattaforme che offrono già il servizio di ride-sharing, l'assegnazione di una richiesta a un veicolo avviene tramite complessi algoritmi di matching, che valutano contemporaneamente molteplici fattori. Nel nostro studio, questo processo viene **semplificato** adottando un algoritmo che verifica la **prima condizione** sopra indicata. La **seconda condizione**, invece, viene modellata tramite una probabilità di matching, che rappresenta la possibilità che la richiesta condivida lo stesso percorso con altri utenti in corsa. La richiesta viene quindi assegnata al primo veicolo compatibile e disponibile secondo queste regole. Nel nostro modello, un veicolo adibito al ride-sharing può accettare più richieste finché ha posti liberi disponibili. Inoltre, come nel servizio tradizionale, un utente potrebbe decidere di cancellare la propria richiesta di corsa prima che questa venga assegnata ad un veicolo. Infine, nel caso in cui non sia stato possibile effettuare il matching, la richiesta di corsa verrà dirottata verso il servizio di trasporto tradizionale.

11 Modello concettuale

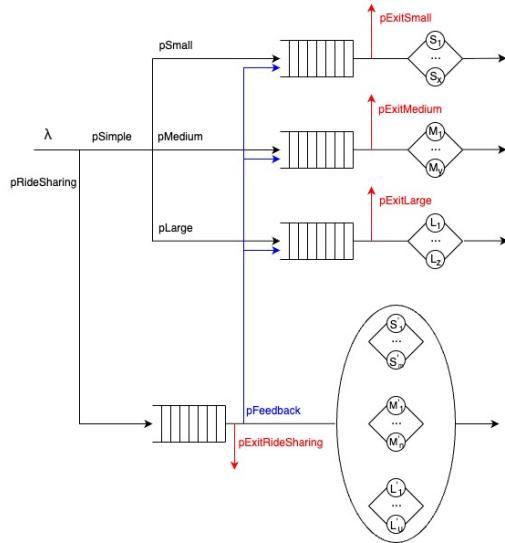


Figura 21: Modello concettuale del servizio di ride-hailing con aggiunta del servizio di ride-sharing

Gli elementi principali di questo sistema sono:

1. **Utente**: colui che desidera richiedere una corsa.
2. **Auto**: la tipologia di veicolo che può essere richiesta da un utente.

In particolare si ha:

1. Centro di **servizio tradizionale** per veicoli **small**: gestisce le corse effettuate da veicoli con capacità da **1 a 3 passeggeri**.
2. Centro di **servizio tradizionale** per veicoli **medium**: gestisce le corse effettuate da veicoli con capacità da **4 passeggeri**.
3. Centro di **servizio tradizionale** per veicoli **large**: gestisce le corse effettuate da veicoli con capacità da **5 a 8 passeggeri**.
4. Centro di **servizio ride-sharing**: gestisce le corse effettuate da veicoli adibiti al **ride-sharing**.

11.1 Stato del sistema

Nel sistema **migliorativo**, lo stato in un determinato istante è definito dal vettore

$$(n_{small}, n_{medium}, n_{large}, n_{ride}, s_1, \dots, s_m, L_{richieste})$$

- $n_{small}, n_{medium}, n_{large}, n_{ride}$: il **numero di richieste nei centri di servizio**.
- s_1, \dots, s_m : la **capacità rimanente**, espressa in termine di **posti disponibili**, dei veicoli **attualmente impegnati in servizio di ride-sharing**.
- $L_{richieste}$: l'**elenco delle richieste in attesa di matching** nel centro ride-sharing.

11.2 Descrizione degli eventi

Nel **sistema migliorativo**, gli eventi che modificano lo stato sono:

1. L'**arrivo di una richiesta** di corsa per una specifica **tipologia** di veicolo per il servizio **tradizionale**.
2. Il **termine di una corsa** che rende un veicolo nuovamente **disponibile** per il servizio **tradizionale**.
3. L'**uscita dal sistema** di un utente che ha annullato la propria **richiesta di servizio**.
4. L'**arrivo di una richiesta** di corsa che prevede l'utilizzo del servizio di **ride-sharing**.
5. Il **matching** tra una richiesta e un veicolo con posti disponibili per il servizio di **ride-sharing**.
6. Il **termine di una corsa** che prevede l'utilizzo del servizio di **ride-sharing**.

12 Modello delle specifiche

Il tasso di arrivo medio λ rimane analogo a quello illustrato nel modello base (si veda §5). Per quanto riguarda invece $E[S]$, si introduce un leggero incremento proporzionale al numero di richieste servite, in modo da simulare il tempo necessario per le soste durante la presa o la discesa degli utenti ed eventuali deviazioni dal percorso.

12.1 Matrice di routing

Il modello migliorativo è caratterizzato da due probabilità, P_{Simple} e $P_{\text{RideSharing}}$, che rappresentano rispettivamente la probabilità di ricevere una richiesta destinata al servizio tradizionale e quella di ricevere una richiesta destinata al nuovo servizio di ride-sharing. Considerando che il servizio di ride-sharing sia appena stato introdotto nell'applicazione, è ragionevole ipotizzare che, almeno nel periodo iniziale, soltanto una quota ridotta di utenti scelga di sperimentarlo. Per questo motivo, le probabilità vengono fissate a **0.7** (P_{Simple}) e **0.3** ($P_{\text{RideSharing}}$). Oltre alle probabilità di abbandono associate ai centri dei servizi tradizionali ($P_{\text{ExitSmall}}$, $P_{\text{ExitMedium}}$, $P_{\text{ExitLarge}}$), il modello migliorativo introduce anche la probabilità di abbandono dalla coda del servizio di ride-sharing, indicata con $P_{\text{ExitRideSharing}}$, fissata anch'essa a **0.05**. In aggiunta, viene introdotta una nuova probabilità, denotata con P_{Feedback}^1 , che indica la possibilità che una richiesta di ride-sharing non possa essere soddisfatta. In tal caso, la richiesta non viene scartata, ma reindirizzata al servizio tradizionale, assicurando così che l'utente non perda completamente l'opportunità di ottenere un veicolo.

¹Si noti che, sebbene nel modello concettuale P_{Feedback} sia rappresentata come una probabilità, nel codice implementativo la gestione è in realtà deterministica: si verifica il numero di posti relativi alla richiesta di corsa reindirizzata e si individua la tipologia di veicolo con capacità minima in grado di soddisfare tale richiesta.

Probabilità	Descrizione	Valore
P_{Simple}	Probabilità di ricevere una richiesta destinata al servizio tradizionale	0.7
$P_{RideSharing}$	Probabilità di ricevere una richiesta destinata al servizio di ride-sharing	0.3
P_{Small}	Probabilità di ricevere una richiesta da 1 a 3 posti per il servizio tradizionale	0.6
P_{Medium}	Probabilità di ricevere una richiesta da 4 posti per il servizio tradizionale	0.1
P_{Large}	Probabilità di ricevere una richiesta da 5 a 8 posti per il servizio tradizionale	0.3
$P_{ExitSmall}$	Probabilità che un utente in coda Small annulli la richiesta	0.05
$P_{ExitMedium}$	Probabilità che un utente in coda Medium annulli la richiesta	0.05
$P_{ExitLarge}$	Probabilità che un utente in coda Large annulli la richiesta	0.05
$P_{ExitRideSharing}$	Probabilità che un utente in coda RideSharing annulli la richiesta	0.05

Tabella 7: Riepilogo delle probabilità utilizzate nel modello migliorativo, inclusa la probabilità di matching per le richieste ride-sharing.

La tabella di routing risultante è la seguente:

	Esterno	Centro small	Centro medium	Centro large	Centro ride-sharing
Esterno	0	$p_{Simple} p_{Small}$	$p_{Simple} p_{Medium}$	$p_{Simple} p_{Large}$	$p_{RideSharing}$
Centro small	1	0	0	0	0
Centro medium	1	0	0	0	0
Centro large	1	0	0	0	0
Centro ride-sharing	$1 - p_{Feedback}$	$p_{Feedback} p_{Small}$	$p_{Feedback} p_{Medium}$	$p_{Feedback} p_{Large}$	0

12.2 Equazioni di traffico

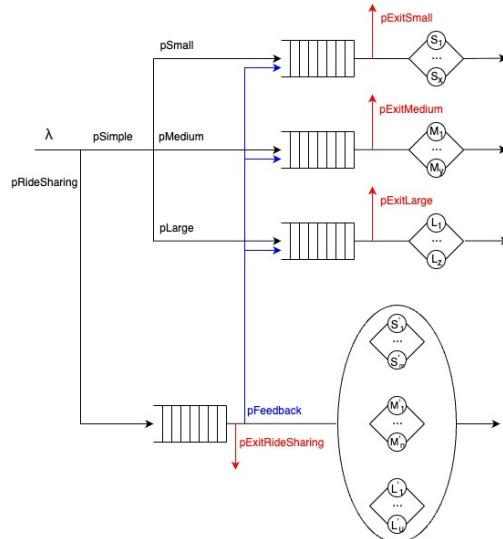


Figura 22: Modello concettuale del servizio di ride-hailing con aggiunta del servizio di ride-sharing

Le equazioni di traffico del sistema sono:

$$\begin{cases} \lambda_{Small} = p_{Small} p_{Simple} (1 - p_{ExitSmall}) \lambda + p_{RideSharing} p_{Feedback} (1 - p_{ExitSmall}) p_{Small} \lambda \\ \lambda_{Medium} = p_{Medium} p_{Simple} (1 - p_{ExitMedium}) \lambda + p_{RideSharing} p_{Feedback} (1 - p_{ExitMedium}) p_{Medium} \lambda \\ \lambda_{Large} = p_{Large} p_{Simple} (1 - p_{ExitLarge}) \lambda + p_{RideSharing} p_{Feedback} (1 - p_{ExitLarge}) p_{Large} \lambda \\ \lambda_{RideSharing} = p_{RideSharing} (1 - p_{ExitRideSharing}) (1 - p_{Feedback}) \lambda \end{cases}$$

12.3 Modellazione dei centri di servizio

- I centri di servizio "classici" sono modellati come code **M/N/k**, ovvero caratterizzati da arrivi di tipo **Poisson**, servizi di tipo **normali**, un numero **finito** di server e da una **coda di attesa a capacità infinita**, alimentata da un pool di utenti. Nel sistema sono presenti tre centri di questo tipo, ciascuno dedicato a una specifica tipologia di veicolo: **small**, **medium** e **large**. Il numero di server in ciascun centro è determinato in funzione della categoria di veicolo che esso gestisce.

- Il centro di servizio "ride-sharing" è modellato come coda $M/N/k^2$, ovvero caratterizzato da arrivi di tipo **Poisson**, servizi di tipo **normali**, un numero **finito** di server e da una **coda di attesa a capacità infinita**, alimentata da un pool di utenti. A differenza dei sistemi $M/N/k$ classici, i server di questo centro non sono tutti identici: per ciascuna tipologia di veicolo (**small**, **medium**, **large**) è previsto un numero specifico di server all'interno dello stesso centro.

12.4 Scelta delle distribuzioni

In assenza di dati sufficienti per un'analisi statistica precisa, le scelte adottate per modellare i tempi di arrivo e di servizio nei diversi centri mirano a rappresentare in modo realistico il comportamento dei processi.

- **Distribuzione Gaussiana Troncata – Tempi di Servizio**

Per modellare i tempi di servizio è stata adottata una distribuzione **Gaussiana troncata**, considerata adatta a descrivere un processo con forte concentrazione dei valori attorno alla media. Questa distribuzione permette, infatti, di catturare l'andamento centrato dei tempi di servizio, pur lasciando spazio a variazioni legate a corse più brevi o più lunghe. La troncatura inferiore evita la generazione di tempi irrealisticamente bassi, mentre quella superiore pone un limite massimo coerente con le tempistiche reali del sistema.

- **Distribuzione Esponenziale – Tempi di Arrivo**

Per modellare gli istanti di arrivo delle richieste si è scelta la distribuzione **esponenziale**, ampiamente impiegata in ambito modellistico per rappresentare processi di arrivo casuali. La scelta è giustificata dalla proprietà di assenza di memoria tipica di questa distribuzione: la probabilità di un prossimo arrivo non dipende dal tempo trascorso dall'ultimo evento. Questo riflette adeguatamente l'elevata aleatorietà del flusso di richieste e l'indipendenza tra arrivi successivi.

²Si precisa che la notazione $M/N/k$ è utilizzata *per convenzione*: il modello considera esplicitamente le differenze tra i server, quindi il comportamento simulato riflette la realtà del centro e non è limitato dalla semplificazione della notazione.

13 Modello computazionale

Il modello computazionale mantiene la struttura del modello di base, integrando estensioni e adattamenti per rappresentare fedelmente le specificità del nuovo sistema in esame.

13.1 Eventi

La classe `MsqEvent` viene estesa nel modello ride-sharing come segue:

```
public class MsqEvent {  
    double t;  
    int x;  
    int postiRichiesti;  
    int capacitaRimanente;  
    int numRichiesteServite;  
    int capacita;  
    double svc;  
}
```

Gli attributi principali della classe sono:

- **t**: istante temporale in cui l'evento si verifica;
- **x**: variabile di stato che indica se l'evento è attivo o meno;
- **postiRichiesti**: numero di posti richiesti dall'utente;
- **numRichiesteServite**: numero di richieste aggregate su un singolo server;
- **capacita**: capacità totale del server;
- **capacitaRimanente**: posti ancora disponibili sul server;
- **svc**: tempo di servizio cumulato erogato dal server fino a quel momento.

13.2 RideSharingSystem

La logica di simulazione del modello ride-sharing mantiene la stessa struttura di base implementata da `SimpleSystem`, con le opportune estensioni per rappresentare le specificità del nuovo servizio.

13.3 RideSharingMultiServerNode

La classe `RideSharingMultiServerNode` incapsula la logica operativa del centro ride-sharing. Nel modello sono presenti due varianti di questa classe:

- `RideSharingMultiServerNodeSimple`, che replica il comportamento del servizio tradizionale;
- `RideSharingMultiServerNode`, che implementa la logica del servizio ride-sharing.

Per motivi di chiarezza e sintesi, nella presente documentazione viene riportata esclusivamente la versione dedicata al servizio ride-sharing, poiché la logica del servizio tradizionale è già stata descritta nel modello base.

Il metodo `processNextEvent()` nella versione ride-sharing è quindi modificato come illustrato di seguito:

```
{  
    public int processNextEvent(double t) {  
        int e = peekNextEventType();  
        clock.current = t;  
  
        if (e == ARRIVAL) {  
            numberJobInSystem++;  
  
            MsqEvent arr = new MsqEvent();  
            arr.t = distrs.getNextArrivalTimeRideSharing(rng, clock.  
                ↪ current);  
            arr.x = 1;  
            arr.postiRichiesti = getNumPosti();  
            event.set(ARRIVAL, arr);  
  
            rng.selectStream(6);  
            double p = rng.random();  
            if (p < P_EXIT) {  
                numberJobInSystem--;  
                return -1;  
            }  
  
            pendingArrivals.add(arr);  
  
            if (Double.isInfinite(nextMatchTime)) {  
                nextMatchTime = clock.current + TIME_WINDOW;  
            }  
        }  
    }  
}
```

```

    }

} else {
    MsqEvent sEvent = event.get(e);
    int numRichiesteServite = sEvent.numRichiesteServite;
    numberJobInSystem -= numRichiesteServite;
    sum[e].served += numRichiesteServite;
    sum[e].service += event.get(e).svc;

    sEvent.x = 0;
    sEvent.capacitaRimanente = sEvent.capacita;
    sEvent.numRichiesteServite = 0;
    sEvent.postiRichiesti = 0;

    return e;
}

if (clock.current >= nextMatchTime) {
    while (true) {
        int matched = findOne();
        if (matched == 0) {
            if (!pendingArrivals.isEmpty()) {
                numberJobInSystem--;
                MsqEvent toFb = pendingArrivals.getFirst();
                generateFeedback(toFb);
                pendingArrivals.removeFirst();
                continue;
            }
            break;
        }
        nextMatchTime = Double.POSITIVE_INFINITY;
    }
    return -1;
}
}

```

Nel caso di elaborazione di arrivi nel servizio ride-sharing, il sistema valuta la possibilità di assegnare una richiesta a un server disponibile. L'algoritmo di matching è stato esteso rispetto al modello tradizionale per tener conto di due aspetti specifici del ride-sharing:

- **Capacità residua:** il server può accogliere più richieste contemporaneamente.

neamente fino a esaurire la propria capacità;

- **Match con server già attivo:** una richiesta può essere assegnata a un server già in servizio con una certa probabilità P_{MATCH_BUSY} ³

Per realizzare questa logica, si utilizza una politica di **selection in order**, implementata dal metodo `findOne()`. In sintesi, l'algoritmo procede come segue:

1. Si seleziona il primo evento di arrivo in attesa (`firstReq`).
2. Si scansionano prima i server attivi ($x = 1$) per verificare se possono accogliere la richiesta senza superare la propria capacità residua e rispettando la probabilità di match con server occupato (P_{MATCH_BUSY}).
3. Se un server attivo idoneo viene trovato, la richiesta viene assegnata a quel server.
4. In caso contrario, si cercano server inattivi ($x = 0$) in grado di ospitare la richiesta; se viene individuato un server libero, la richiesta viene assegnata e si tenta di aggregare ulteriori richieste in coda fino al completamento della capacità disponibile.

```
{  
public int findOne() {  
    if (pendingArrivals.isEmpty()) return 0;  
  
    MsqEvent firstReq = pendingArrivals.getFirst();  
  
    int bestActive = -1;  
    double bestCapActive = -1;  
  
    rng.selectStream(7);  
    for (int i = 1; i <= RIDESERVERS; i++) {  
        if (event.get(i).x == 1  
            && event.get(i).capacitaRimanente >= firstReq.  
                → postiRichiesti  
            && rng.random() < P_MATCH_BUSY
```

³La probabilità $P_{MatchBusy}$ è fissata a **0.6** e modella la **seconda condizione** (si veda §10). Essa rappresenta la possibilità che una richiesta di ride-sharing possa condividere il medesimo percorso con altri utenti in corsa, rispettando comunque il vincolo sulla capacità residua.

```

        && event.get(i).capacitaRimanente > bestCapActive)
        ↪ {
    bestCapActive = event.get(i).capacitaRimanente;
    bestActive = i;
}
}

if (bestActive != -1) {
    assignToServer(bestActive, firstReq);
    pendingArrivals.removeFirst();
    return 1;
}

int bestIdle = -1;
double bestCapIdle = -1;
rng.selectStream(8);
for (int i = 1; i <= RIDESERVERS; i++) {
    if (event.get(i).x == 0
        && event.get(i).capacitaRimanente >= firstReq.
            ↪ postiRichiesti
        && event.get(i).capacitaRimanente > bestCapIdle) {
        bestCapIdle = event.get(i).capacitaRimanente;
        bestIdle = i;
    }
}
if (bestIdle == -1) return 0;

event.get(bestIdle).x = 1;
int totalMatched = 0;
Iterator<MsqEvent> it = pendingArrivals.iterator();
while (it.hasNext()) {
    MsqEvent req = it.next();
    if (req.postiRichiesti <= event.get(bestIdle).
        ↪ capacitaRimanente) {
        assignToServer(bestIdle, req);
        it.remove();
        totalMatched++;
        if (event.get(bestIdle).capacitaRimanente == 0) break;
    }
}
return totalMatched;
}
}

```

13.4 getNextArrivalTimeRideSharing() e getServiceTimeRideSharing()

Per generare i tempi di arrivo e di servizio, il modello sfrutta la classe `Rngs` con un approccio multi-stream, garantendo l'indipendenza tra le sequenze di numeri casuali utilizzate dalle diverse componenti del sistema.

Si mostra come esempio il codice di `getNextArrivalTimeRideSharing()`:

```
{  
    public double getNextArrivalTimeRideSharing(Rngs r, double sarrival  
    ↪ ) {  
        r.selectStream(2);  
        double lambda = config.getDouble("simulation","lambdaride") *  
            ↪ config.getDouble("simulation","pride");  
        sarrival += exponential(1/lambda, r);  
        return sarrival;  
    }  
}
```

Mentre `getServiceTimeRideSharing()` ha la seguente implementazione:

```
{  
    public double getServiceTimeRideSharing(Rngs r) {  
        r.selectStream(4);  
        double esi = config.getDouble("simulation","esi");  
        double alpha, beta;  
        double a = 2;  
        double b = 30;  
  
        alpha = cdfNormal(esi, 4, a);  
        beta = 1- cdfNormal(esi, 4, b);  
  
        double u = uniform(alpha, 1- beta, r);  
        return idfNormal(esi, 4, u);  
    }  
}
```

14 Design degli esperimenti

14.1 Intervalli di confidenza

All'interno della trattazione sono stati utilizzati gli intervalli di confidenza. Un intervallo di confidenza è un intervallo calcolato a partire dai dati campionari, che ha una certa probabilità di contenere il valore reale della media della popolazione. In altre parole, fornisce una stima dell'incertezza associata alla media campionaria, indicando un intervallo plausibile entro cui ci si può ragionevolmente aspettare che la media reale del sistema si trovi.

Per questo studio sono stati scelti intervalli con un **livello di confidenza** pari al **95%**. Gli intervalli di confidenza sono stati calcolati utilizzando la seguente formula:

$$\text{Intervallo di Confidenza} = t^* \left(\frac{s}{\sqrt{n-1}} \right)$$

dove

- **s** è la deviazione standard campionaria calcolata tramite l'algoritmo di **Welford**;
- **n** è la dimensione del campione;
- **t*** è il valore critico della distribuzione *t* di **Student** (nel codice, il calcolo è stato effettuato utilizzando la funzione `idfStudent()` fornita dalla classe `Rvms`).

14.2 Primo obiettivo

Il primo obiettivo è gestire lo stesso carico di lavoro con l'introduzione del servizio di ride-sharing, individuando il numero ottimale di veicoli da assegnare a ciascuna tipologia, in modo da ottenere una distribuzione del carico tra i centri di servizio quanto più omogenea possibile. Ogni nuova configurazione è stata impostata cercando di mitigare il collo di bottiglia individuato nella configurazione precedente.

14.2.1 Analisi del transitorio

Prima di procedere con l'esecuzione degli esperimenti, è fondamentale analizzare la fase transitoria del sistema per verificare quando esso tende a uno stato stazionario. A tal fine, si utilizza la tecnica delle **repliche**: la simulazione viene eseguita più volte a partire dallo stesso stato iniziale del sistema. Per ciascuna replica, l'intervallo temporale considerato è fissato a **2.880 minuti**, corrispondenti a due giorni di attività del sistema.

Analogamente al modello base, il tempo medio di risposta $E[T_s]$ è stato scelto come indicatore della convergenza transitoria. D'ora in avanti la descrizione delle configurazioni dei veicoli seguirà la seguente notazione, costituita da due triplettre:

- la **prima tripletta** rappresenta il numero di veicoli assegnati ai centri tradizionali (small, medium, large);
- la **seconda tripletta** indica invece i veicoli destinati al servizio di ride-sharing, sempre suddivisi nelle tre categorie (small, medium, large).

Nella prima configurazione, considerando che solo una quota limitata di utenti sceglie il servizio di ride-sharing, sono stati assegnati a questo servizio **14 veicoli**, distribuiti secondo la proporzione [(24-5-12), (9-1-4)], in modo da rispecchiare la distribuzione attesa della domanda. I principali risultati ottenuti per questa configurazione sono riportati di seguito:

14.2.1.1 Configurazione [(24-5-12), (9-1-4)]

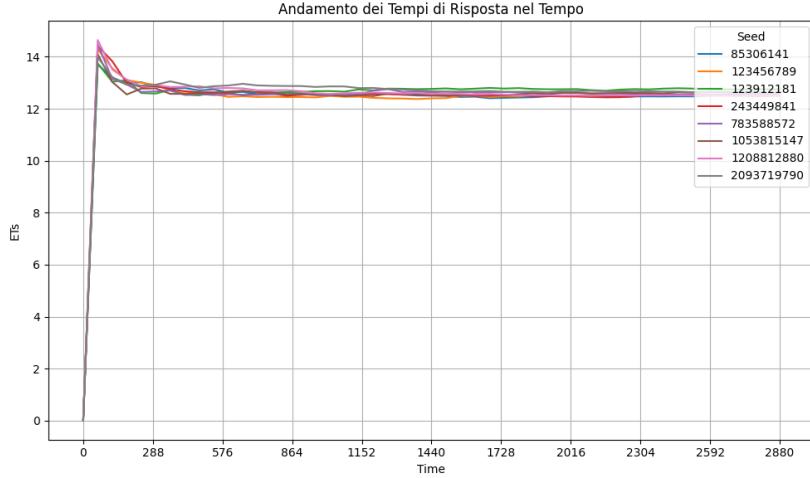
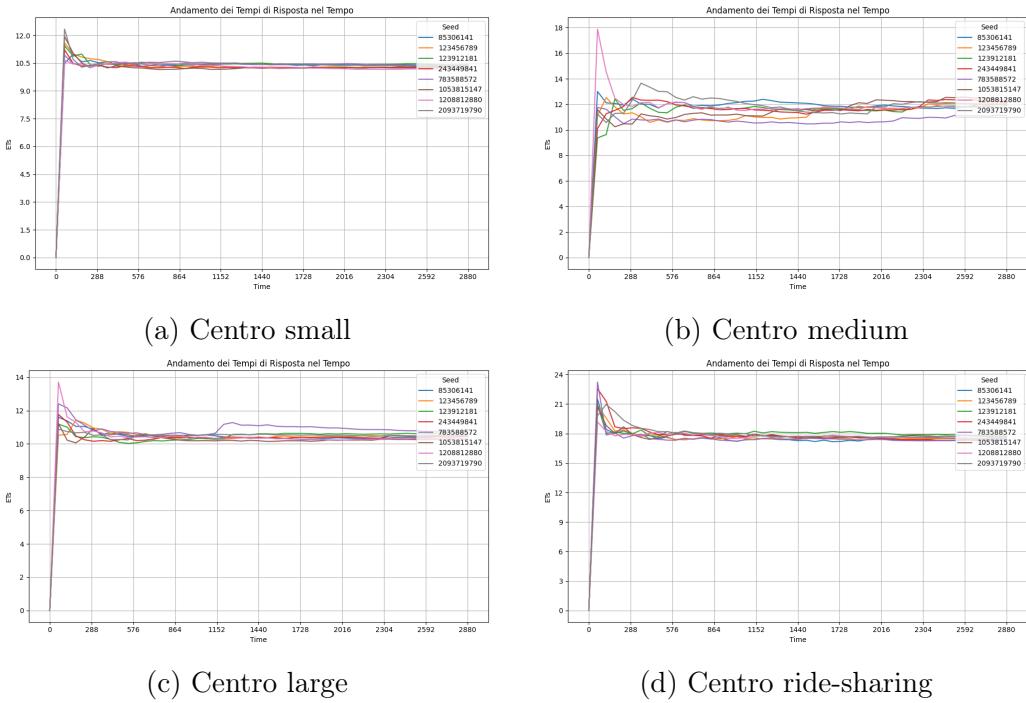


Figura 23: Risultati globali



Si osserva che sia il sistema sia i centri raggiungono il regime stazionario, con il sistema che in particolare converge dopo circa **288 minuti**.

14.2.1.2 Configurazione [(25-5-13), (8-1-3)]

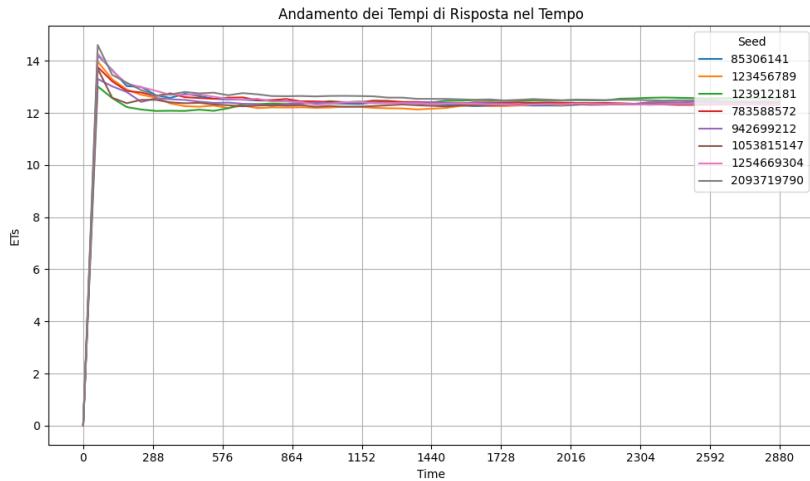
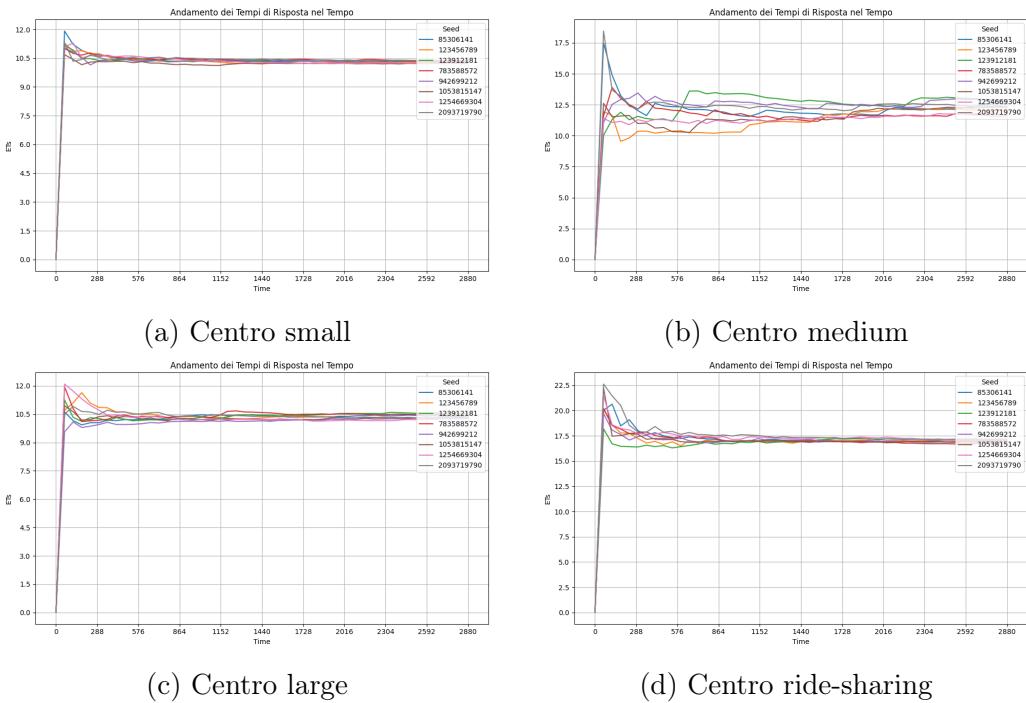


Figura 25: Risultati globali



Si osserva che sia il sistema sia i centri raggiungono il regime stazionario, con il sistema che in particolare converge dopo circa **576 minuti**.

14.2.1.3 Configurazione [(27-6-13), (6-0-3)]

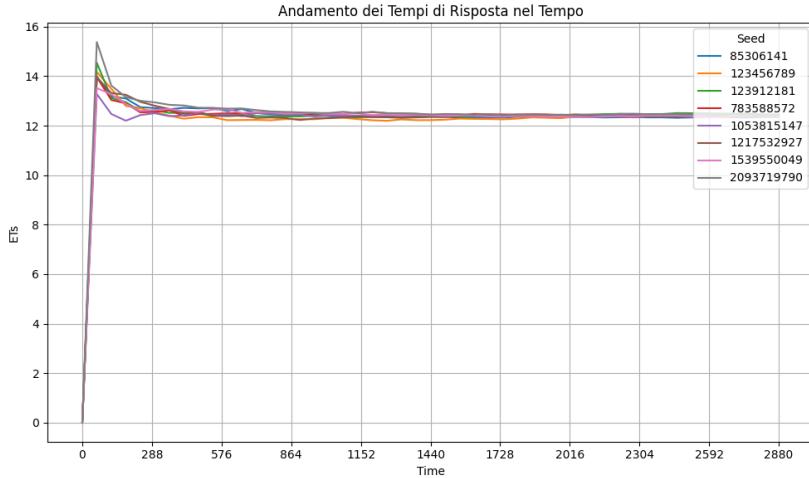
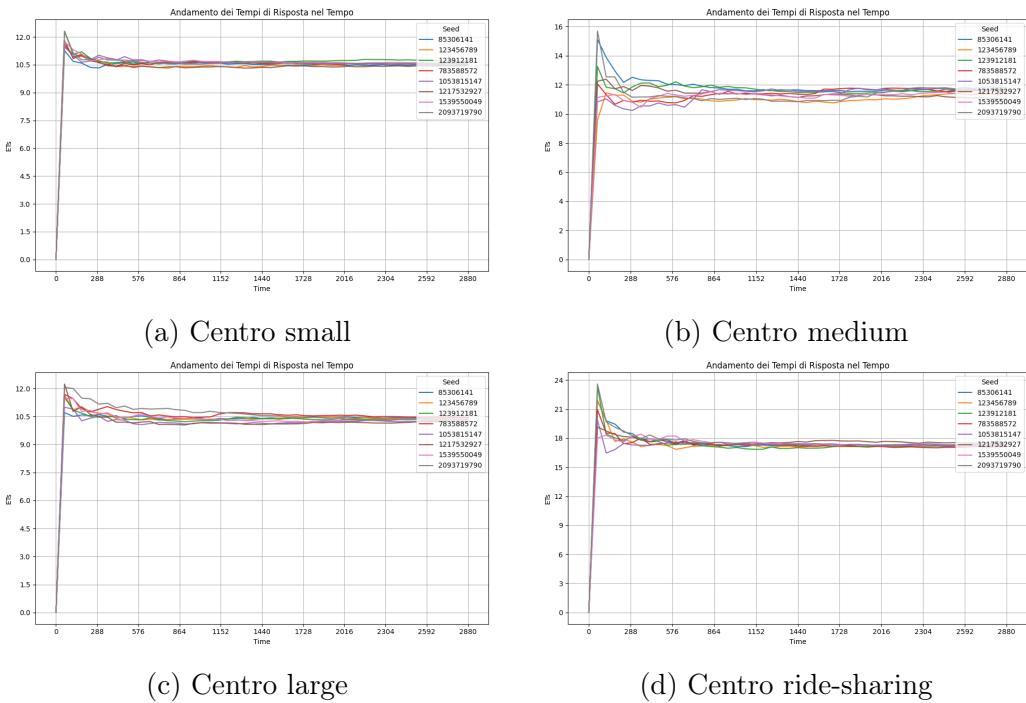


Figura 27: Risultati globali



Si osserva che sia il sistema sia i centri raggiungono il regime stazionario, con il sistema che in particolare converge dopo circa **288 minuti**.

14.2.2 Simulazione ad orizzonte infinito

Considerando che Bolt è un'applicazione già consolidata e ampiamente adottata dagli utenti, che lo studio si concentra su una singola fascia operativa e che la simulazione parte da uno stato iniziale vuoto, l'analisi ad orizzonte infinito risulta particolarmente appropriata, in quanto consente di ottenere stime delle prestazioni operative prive del bias legato alla fase transitoria.

Per stimare le statistiche di interesse, si è applicato il metodo delle **Batch Means**. I parametri scelti sono stati **b = 2048** (dimensione di ciascun batch) e **k = 512** (numero complessivo di batch). Tali valori non sono stati fissati a priori: inizialmente si era optato per $b = 256$ e $k = 64$, ma entrambi i parametri sono stati aumentati progressivamente fino a garantire che la **lag-1 autocorrelazione** tra le medie di batch risultasse inferiore a **0.2**, in accordo con quanto raccomandato in [4]. In questo modo si è raggiunto un compromesso soddisfacente perché da un lato le medie di batch risultano sufficientemente indipendenti; dall'altro, il numero di batch rimane abbastanza ampio da permettere una stima affidabile della varianza.

Eseguendo la simulazione, per le medesime configurazioni valutate nell'analisi del transitorio, si ottengono i risultati:

14.2.2.1 Configurazione [(24-5-12), (9-1-4)]

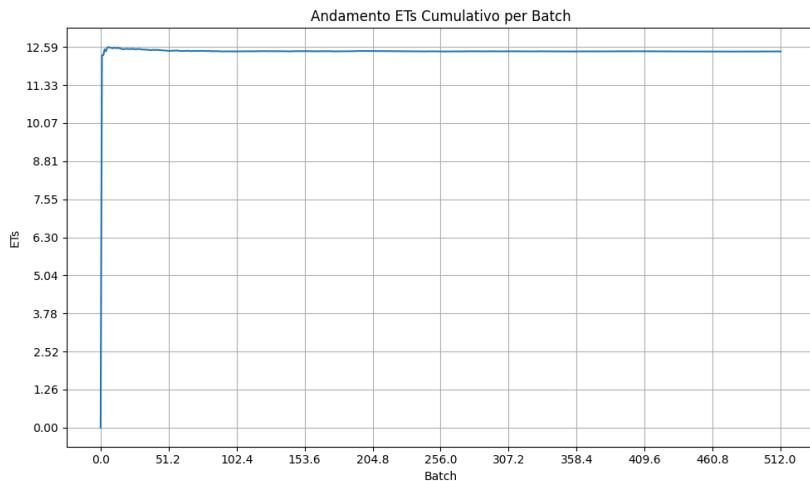
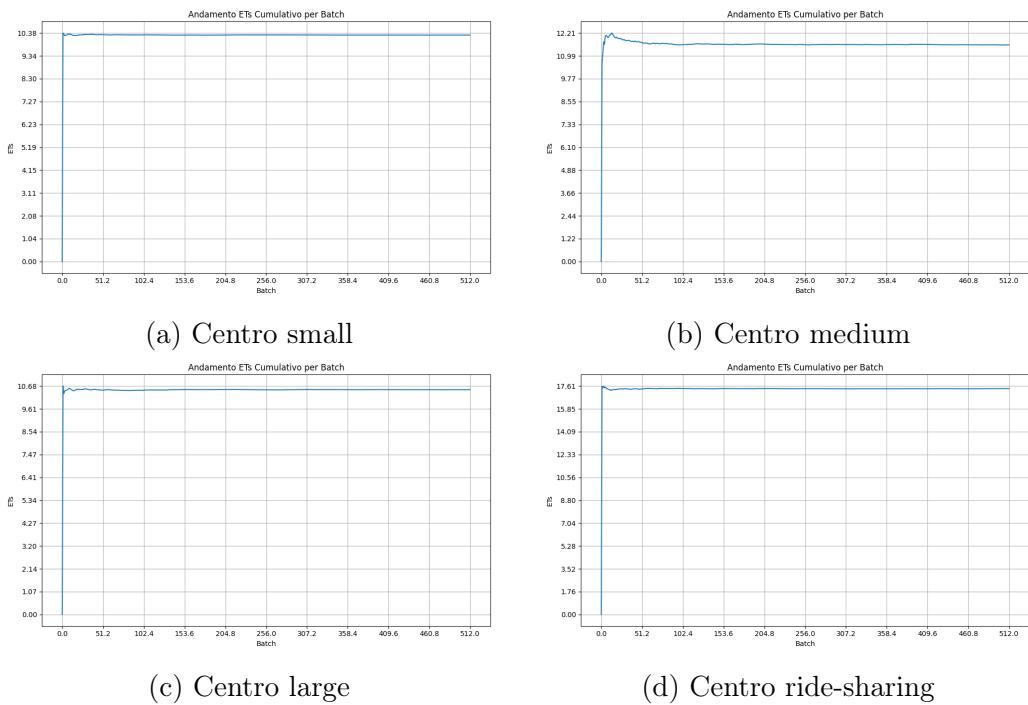


Figura 29: Risultati globali



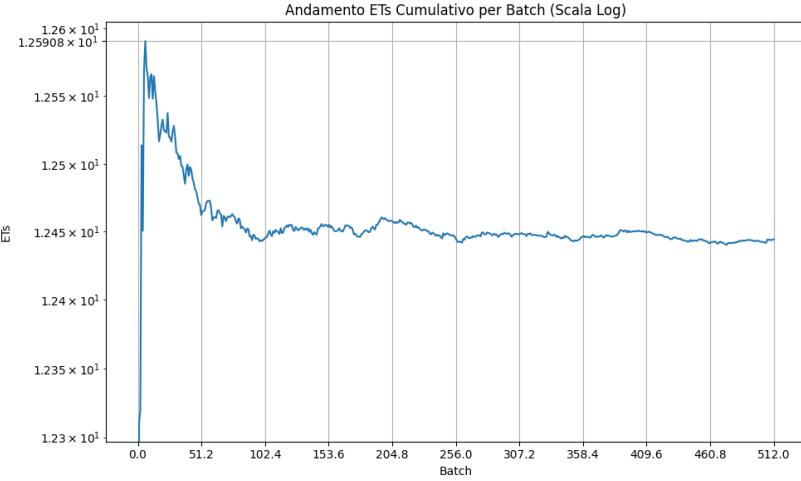
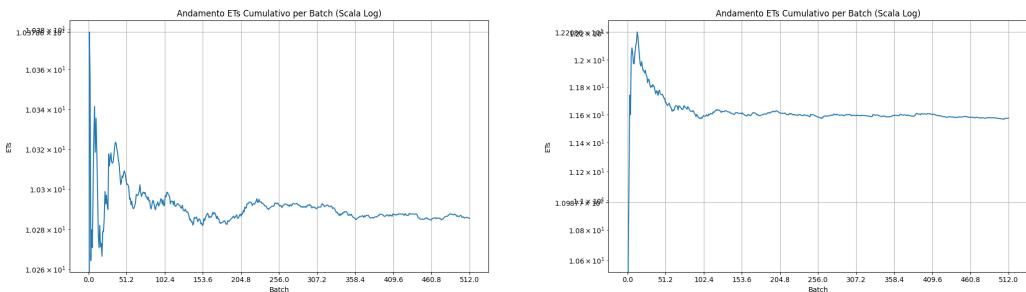
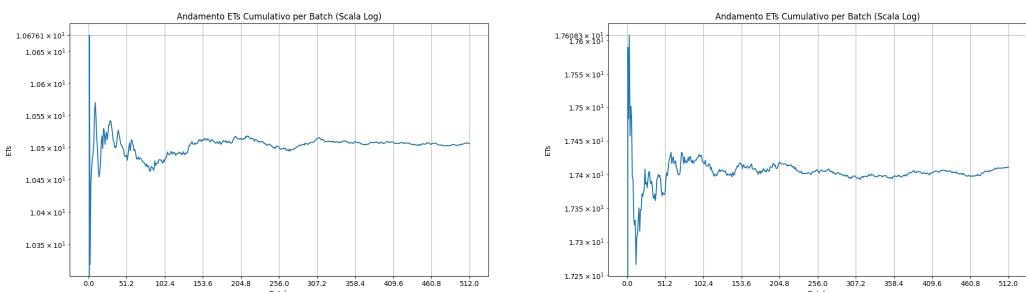


Figura 31: Risultati globali scala log.



(a) Centro small scala log.

(b) Centro medium scala log.



(c) Centro large scala log.

(d) Centro ride-sharing scala log.

Centro	$E[N_S]$	$E[T_S]$	$E[N_Q]$	$E[T_Q]$	ρ	$E[S_i]$	λ
small	16.6780 ± 0.0582	10.2854 ± 0.0128	0.0989 ± 0.0095	0.0600 ± 0.0056	0.6908 ± 0.0022	10.2254 ± 0.01165	1.6214 ± 0.0048
medium	3.5547 ± 0.0417	11.5754 ± 0.0747	0.3002 ± 0.0195	0.9524 ± 0.0563	0.6509 ± 0.0052	10.6230 ± 0.0311	0.3063 ± 0.0021
large	8.4705 ± 0.0448	10.5063 ± 0.0264	0.2295 ± 0.0133	0.2811 ± 0.0156	0.6868 ± 0.0030	10.2253 ± 0.0175	0.8060 ± 0.0033
ride-sharing	19.5314 ± 0.0657	17.4111 ± 0.0351	3.1704 ± 0.0125	2.8263 ± 0.0083	0.4177 ± 0.0138	14.5848 ± 0.0330	1.1222 ± 0.0041

Tabella 8: Statistiche riassuntive simulazione ad orizzonte infinito del modello migliorativo configurazione [(24-5-12), (9-1-4)]

14.2.2.2 Configurazione [(25-5-13), (8-1-3)]

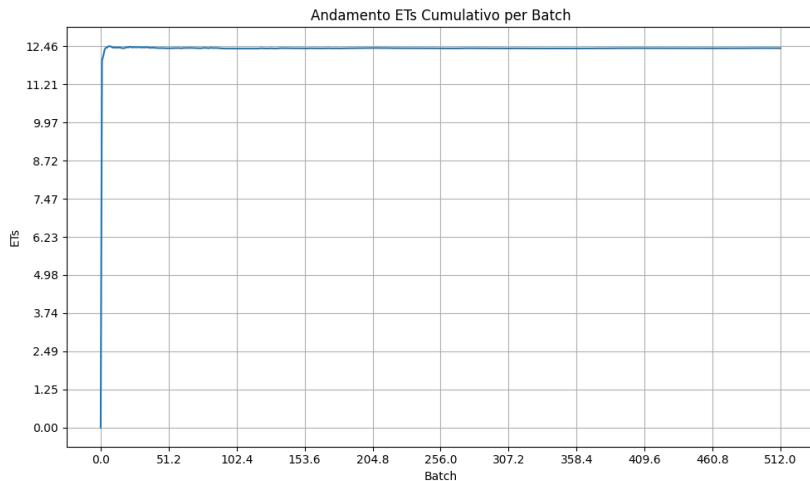
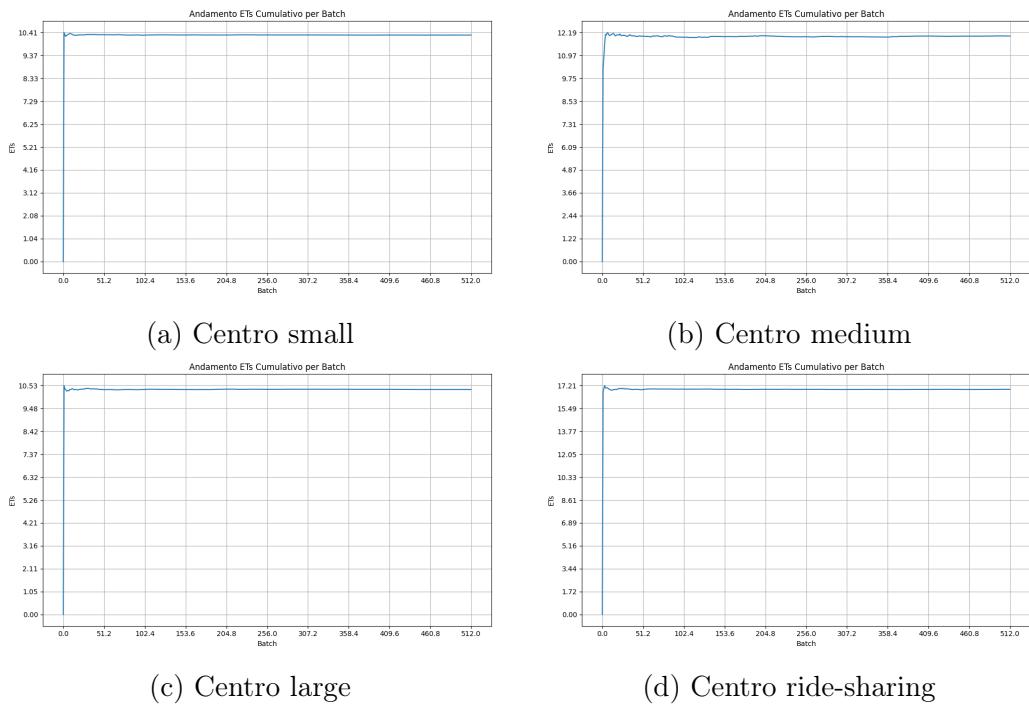


Figura 33: Risultati globali



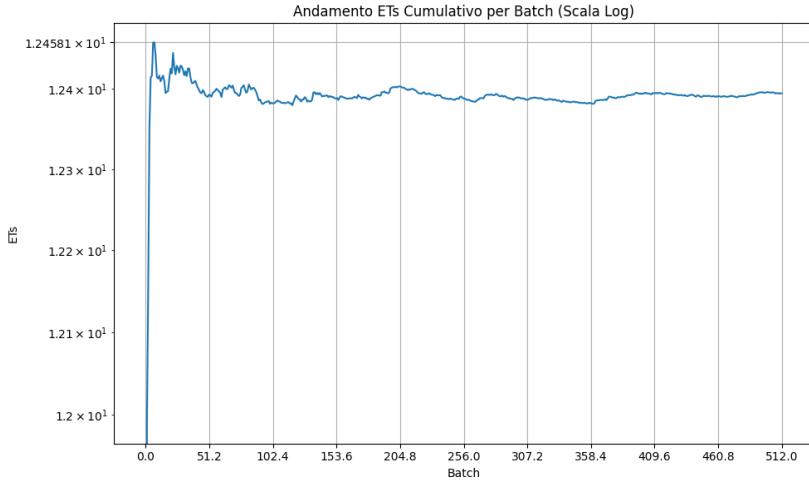
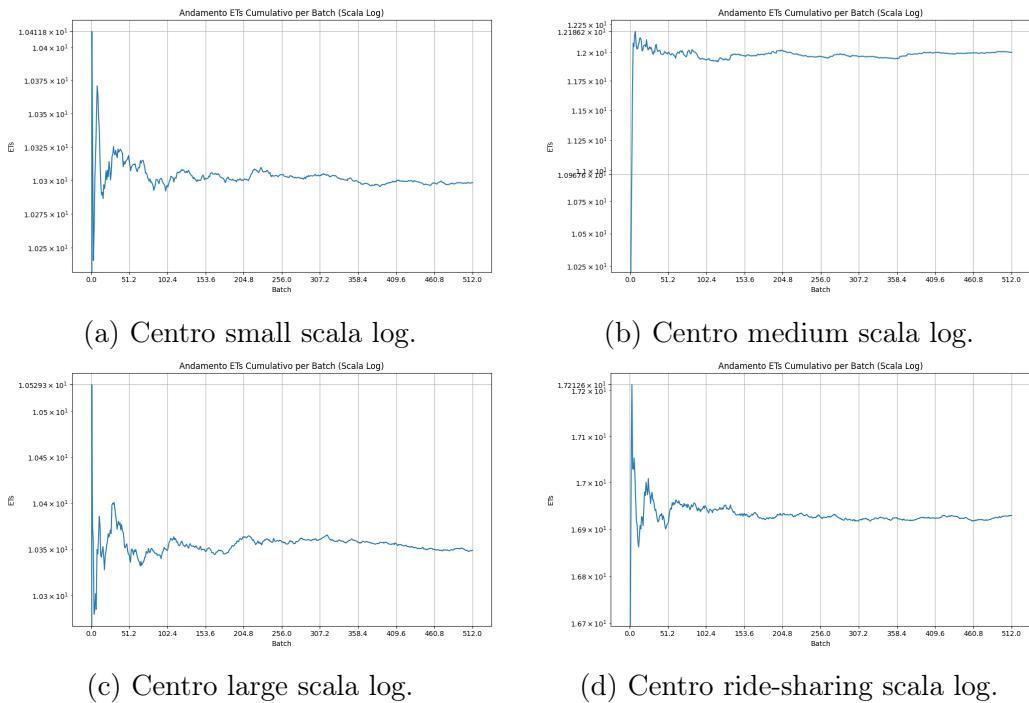


Figura 35: Risultati globali scala log.



Centro	$E[N_S]$	$E[T_S]$	$E[N_Q]$	$E[T_Q]$	ρ	$E[S_i]$	λ
small	16.7830 ± 0.0590	10.2985 ± 0.0138	0.0607 ± 0.0068	0.0366 ± 0.0040	0.6689 ± 0.0022	10.2619 ± 0.0127	1.6295 ± 0.0049
medium	3.8779 ± 0.0447	12.0008 ± 0.0788	0.3913 ± 0.0231	1.1814 ± 0.0638	0.6973 ± 0.0051	10.8194 ± 0.0316	0.3222 ± 0.0021
large	8.3423 ± 0.0400	10.3486 ± 0.0205	0.1024 ± 0.0073	0.1251 ± 0.0087	0.6338 ± 0.0028	10.2235 ± 0.0170	0.8060 ± 0.0033
ride-sharing	18.5727 ± 0.0619	16.9296 ± 0.0348	3.1699 ± 0.0126	2.8895 ± 0.0089	0.5024 ± 0.0160	14.0400 ± 0.0321	1.0975 ± 0.0040

Tabella 9: Statistiche riassuntive simulazione ad orizzonte infinito del modello migliorativo configurazione [(25-5-13), (8-1-3)]

14.2.2.3 Configurazione [(27-6-13), (6-0-3)]

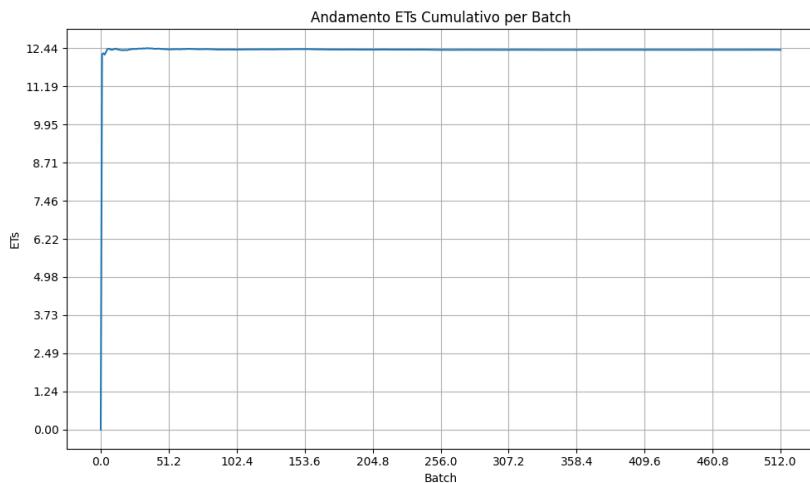
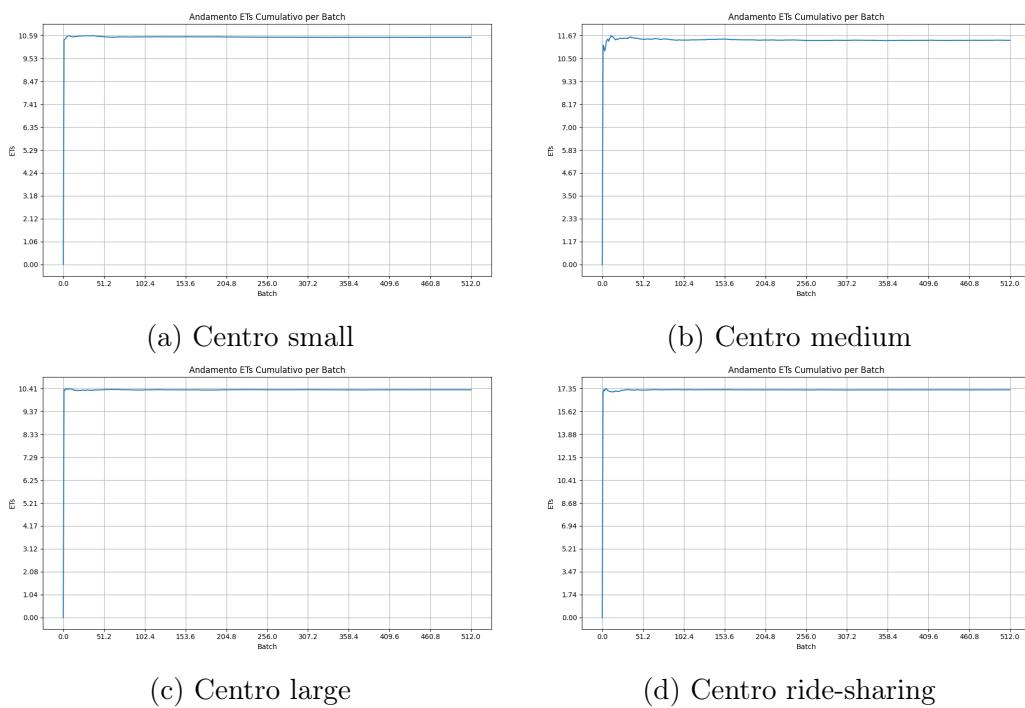


Figura 37: Risultati globali



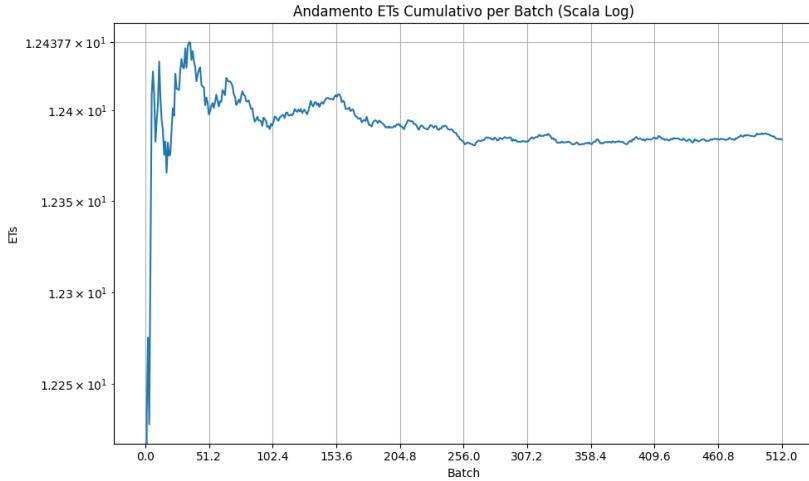
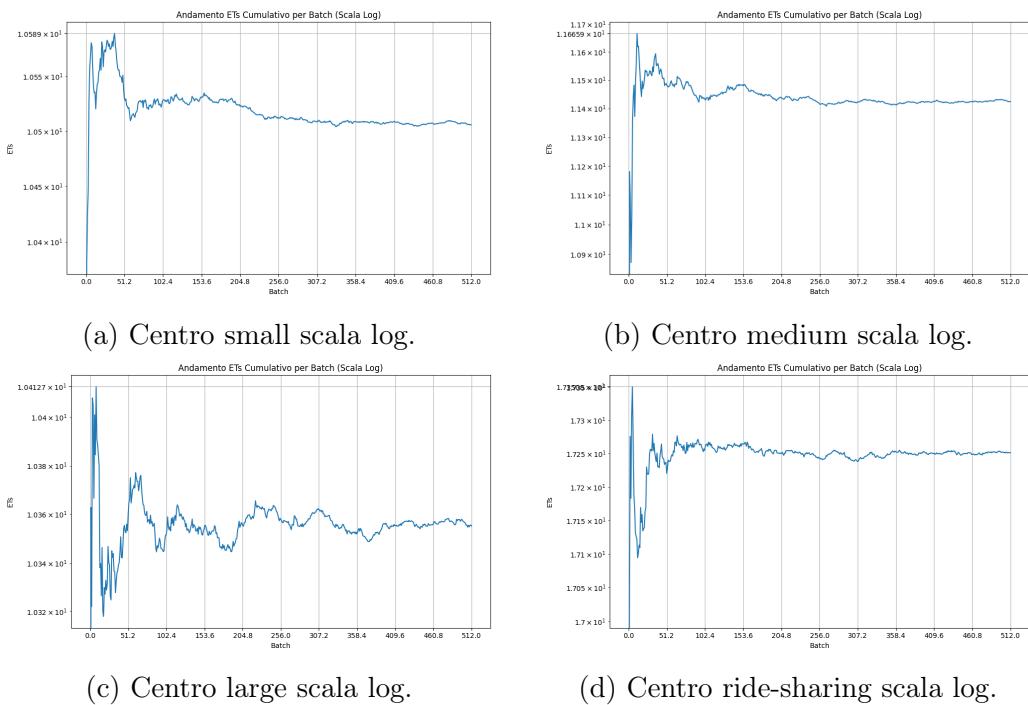


Figura 39: Risultati globali scala log.



Centro	$E[N_S]$	$E[T_S]$	$E[N_Q]$	$E[T_Q]$	ρ	$E[S_i]$	λ
small	17.6455 ± 0.0640	10.5059 ± 0.0174	0.0342 ± 0.0038	0.0201 ± 0.0022	0.6523 ± 0.0023	10.4858 ± 0.0168	1.6794 ± 0.0049
medium	3.8197 ± 0.0355	11.4241 ± 0.0497	0.1501 ± 0.0104	0.4365 ± 0.0281	0.6116 ± 0.0047	10.9876 ± 0.0324	0.3338 ± 0.0022
large	8.3488 ± 0.0413	10.3552 ± 0.0206	0.1037 ± 0.0072	0.1268 ± 0.0085	0.6342 ± 0.0029	10.2284 ± 0.0170	0.8061 ± 0.0034
ride-sharing	17.8600 ± 0.0524	17.2511 ± 0.0378	3.1693 ± 0.0123	3.0612 ± 0.0099	0.6335 ± 0.0188	14.1900 ± 0.0341	1.0357 ± 0.0033

Tabella 10: Statistiche riassuntive simulazione ad orizzonte infinito del modello migliorativo configurazione [(27-6-13), (6-0-3)]

14.3 Secondo obiettivo

Il secondo obiettivo dello studio consiste nel valutare l'effettiva possibilità di **ridurre il numero di veicoli** necessari per gestire il carico di lavoro iniziale grazie all'introduzione del servizio di ride-sharing. Dall'analisi dei risultati è emerso che la configurazione minima in grado di riportare l'utilizzo dei vari centri ai livelli precedenti all'introduzione del ride-sharing è la seguente:

- **(23-5-11)**: veicoli allocati per il servizio tradizionale per tipologia (small-medium-large);
- **(5-0-4)**: veicoli allocati per il servizio di ride-sharing per tipologia (small-medium-large).

Risulta quindi possibile ridurre il numero di veicoli small da **33** a **28**, il numero di veicoli medium da **6** a **5**, e il numero di veicoli large da **16** a **15**, ottenendo una riduzione complessiva del numero di veicoli pari al **12.72%**.

14.3.1 Analisi del transitorio

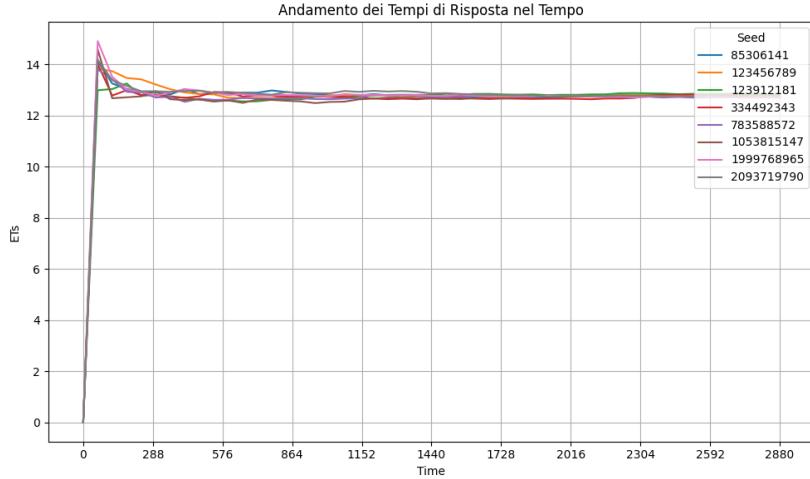
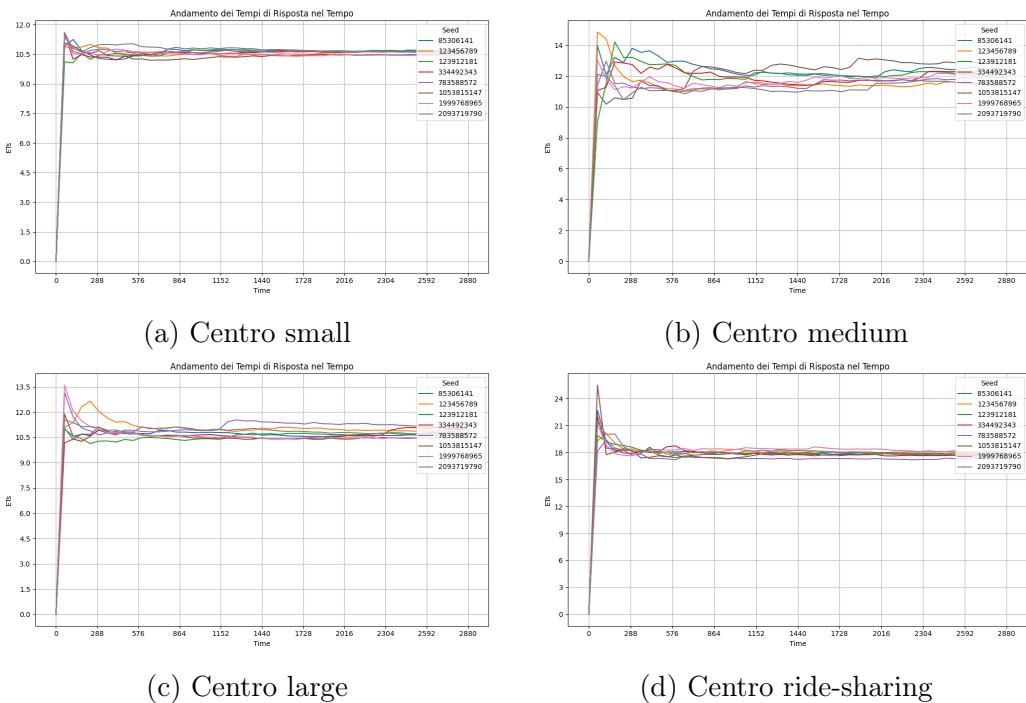


Figura 41: Risultati globali



Si osserva che sia il sistema sia i centri raggiungono il regime stazionario, con il sistema che in particolare converge dopo circa **576 minuti**.

14.4 Studio Orizzonte Infinito

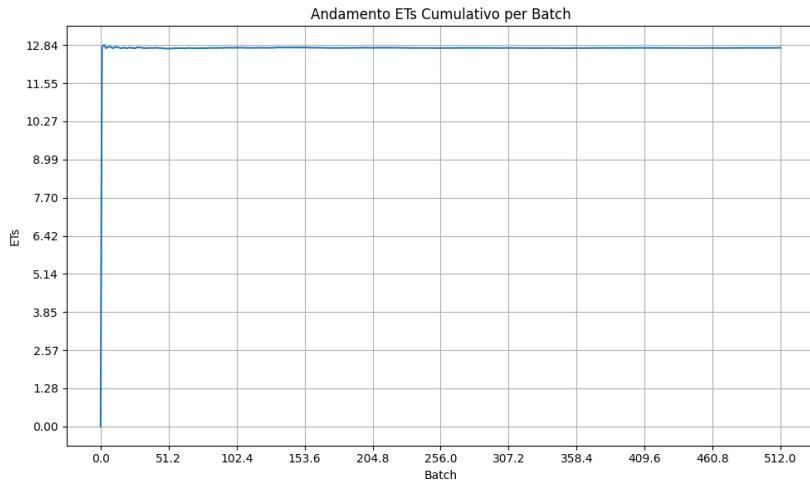
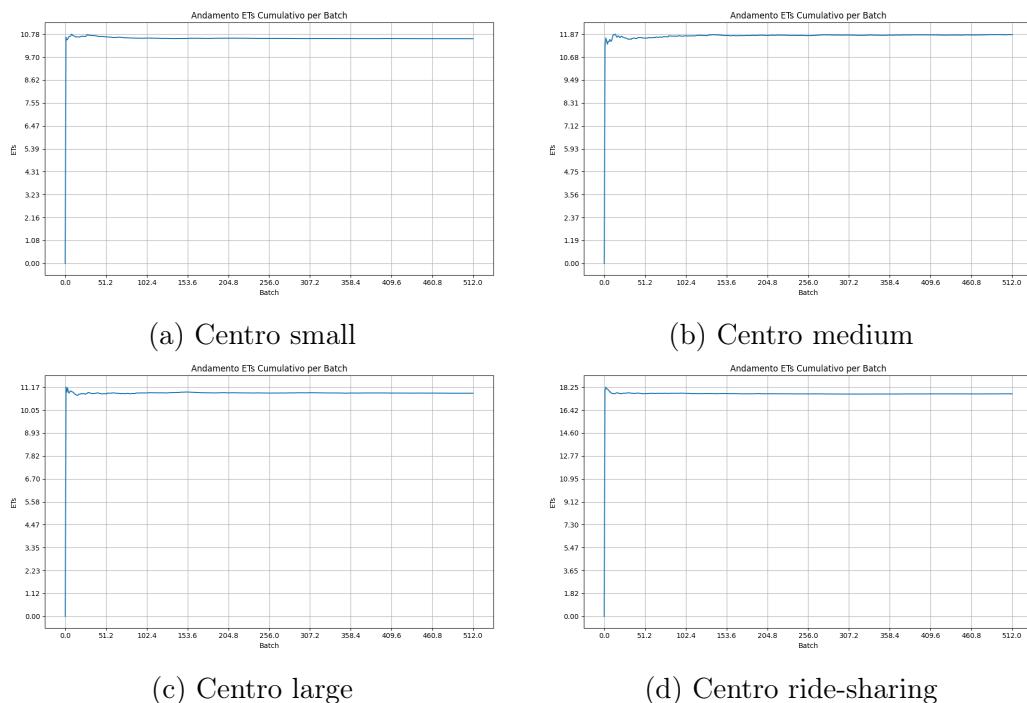


Figura 43: Risultati globali



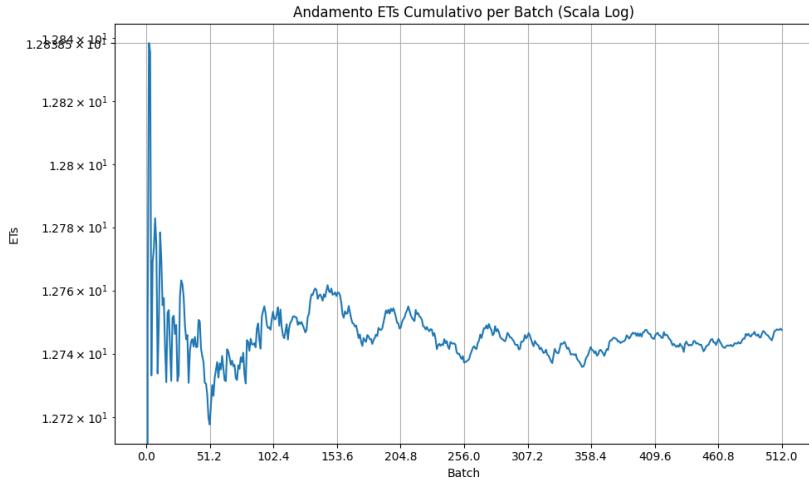
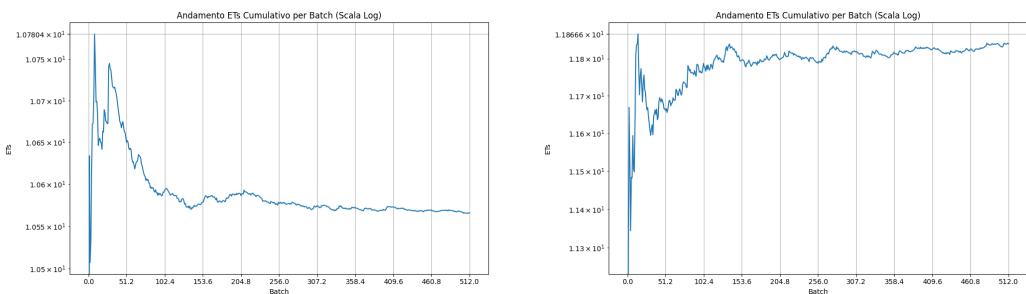
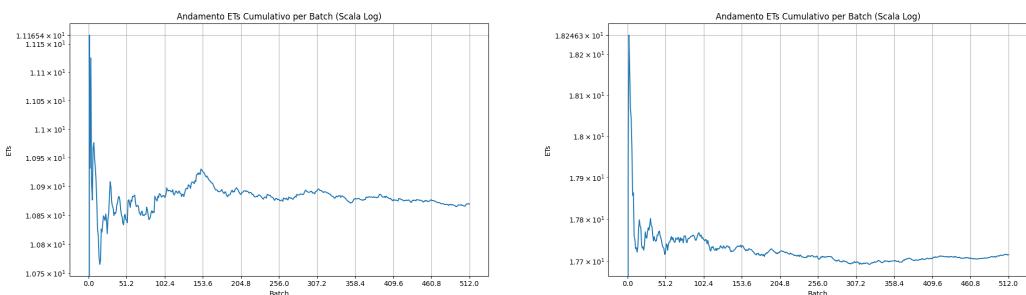


Figura 45: Risultati globali scala log.



(a) Centro small scala log.

(b) Centro medium scala log.



(c) Centro large scala log.

(d) Centro ride-sharing scala log.

Centro	$E[N_S]$	$E[T_S]$	$E[N_Q]$	$E[T_Q]$	ρ	$E[S_i]$	λ
small	17.5685 ± 0.0740	10.5657 ± 0.0224	0.2658 ± 0.0220	0.1577 ± 0.0125	0.7523 ± 0.0026	10.4081 ± 0.0155	1.6623 ± 0.0048
medium	3.7619 ± 0.0458	11.8403 ± 0.0810	0.3632 ± 0.0232	1.1116 ± 0.0646	0.6797 ± 0.0053	10.7287 ± 0.0322	0.3167 ± 0.0022
large	8.7670 ± 0.0593	10.8689 ± 0.0411	0.5232 ± 0.0292	0.6389 ± 0.0336	0.7494 ± 0.0032	10.2300 ± 0.0164	0.8058 ± 0.0032
ride-sharing	18.9584 ± 0.0541	17.7150 ± 0.0377	3.1700 ± 0.0123	2.9618 ± 0.0092	0.6146 ± 0.0174	14.7533 ± 0.0346	1.0706 ± 0.0034

Tabella 11: Statistiche riassuntive simulazione ad orizzonte infinito del modello migliorativo configurazione [(23-5-11),(5-0-4)]

14.5 Autocorrelazione

Di seguito sono riportati i valori dell'autocorrelazione delle statistiche della **configurazione [(27-6-13), (6-0-3)]** forniti dal simulatore, che utilizza la classe di libreria `Acs.java`:

```
*****  
AUTOCORRELATION VALUES FOR Center0 [B:2048|K:512]  
*****  
E[Ts]: -0,0619  
E[Tq]: -0,0571  
E[Si]: -0,0677  
E[Ns]: 0,0102  
E[Nq]: -0,0570  
ρ: 0,0090  
λ: 0,0107  
*****
```

Figura 47: Valori di autocorrelazione del centro small

```
*****  
AUTOCORRELATION VALUES FOR Center1 [B:2048|K:512]  
*****  
E[Ts]: -0,0428  
E[Tq]: -0,0349  
E[Si]: -0,0054  
E[Ns]: 0,0066  
E[Nq]: -0,0364  
ρ: 0,0269  
λ: 0,0435  
*****
```

Figura 48: Valori di autocorrelazione del centro medium

```
*****  
AUTOCORRELATION VALUES FOR Center2 [B:2048|K:512]  
*****  
E[Ts]: -0,0828  
E[Tq]: -0,0612  
E[Si]: -0,0631  
E[Ns]: -0,1121  
E[Nq]: -0,0672  
ρ: -0,0917  
λ: -0,0513  
*****
```

Figura 49: Valori di autocorrelazione del centro large

```
*****  
AUTOCORRELATION VALUES FOR Center3 [B:2048|K:512]  
*****  
E[Ts]: -0,1770  
E[Tq]: -0,1212  
E[Si]: -0,1435  
E[Ns]: -0,0209  
E[Nq]: -0,0193  
ρ: -0,0453  
λ: -0,0027  
*****
```

Figura 50: Valori di autocorrelazione del centro ride-sharing

15 Verifica

La verifica del modello migliorativo può essere condotta distinguendo le due principali componenti del sistema:

1. **Centri di servizio classici:** disabilitando la generazione del feedback per le richieste non soddisfatte dall'algoritmo di matching del ride-sharing, il comportamento dei centri semplici rimane identico a quello del modello base. Infatti, senza il feedback, i flussi di arrivo rimangono separati: uno diretto ai centri semplici e l'altro al centro ride-sharing. Pertanto, non sono necessari ulteriori controlli rispetto a quelli già eseguiti nella verifica del modello semplice, a cui si rimanda per i dettagli.
2. **Centro Ride Sharing:** per questa componente, la struttura dei server, con capacità differenziate e la possibilità di gestire contemporaneamente più richieste, impedisce l'applicazione dei tradizionali approcci analitici di verifica. Di conseguenza, l'unica strada percorribile consiste nell'effettuare controlli di consistenza interna sui risultati della simulazione, finalizzati a confermare che la logica implementata rispetti le ipotesi del modello. I principali controlli includono:
 - Controllo della coerenza delle statistiche raccolte;
 - Controllo specifico sull'interazione tra le due componenti.

15.1 Controllo della coerenza delle statistiche raccolte

Sui risultati degli esperimenti riportati nel paragrafo precedente sono stati effettuati i seguenti controlli di consistenza, tenendo conto degli intervalli di confidenza:

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

Legge di Little: $E(N_S) = \lambda E(T_S); \quad E(N_Q) = \lambda E(T_Q)$

15.2 Controllo specifico sull'interazione tra le due componenti

Si è verificato che le richieste non soddisfatte dal centro ride-sharing vengano effettivamente reindirizzate ai centri di servizio tradizionali e che questo trasferimento comporti un aumento proporzionale del flusso nei centri di servizio

classici. Tale controllo garantisce la corretta intrerazione tra i due servizi offerti dal sistema. Nella tabella sottostante vengono riportati i risultati della **configurazione [(27-6-13), (6-0-3)]**.

Tabella 12: Utilizzazione dei centri

(a) Senza feedback		(b) Con feedback	
Centro	ρ	Centro	ρ
small	0.6140 ± 0.0018	small	0.6523 ± 0.0023
medium	0.4583 ± 0.0036	medium	0.6116 ± 0.0047
large	0.6328 ± 0.0027	large	0.6342 ± 0.0029

Si osservi che l'utilizzazione del centro large è assimilabile al sistema con feedback poiché le richieste di ride-sharing possono essere effettuate per un numero di posti massimo di quattro persone.

16 Validazione

La fase di validazione ha l'obiettivo di verificare che il modello computazionale sia coerente e rappresentativo del sistema reale che si intende simulare. In genere, questa fase prevede il confronto dei risultati del modello con dati empirici o con misurazioni del sistema reale, al fine di accettarne la correttezza e la capacità predittiva.

Nel nostro caso, tuttavia, non è stato possibile effettuare una validazione tradizionale, in quanto non sono disponibili dati osservati dettagliati sul servizio di ride-hailing dell'oggetto di studio. La mancanza di informazioni reali ci impedisce di confrontare direttamente le prestazioni simulate con il sistema reale.

17 Conclusioni

Le simulazioni evidenziano come il **modello migliorativo** sia in grado di gestire lo stesso volume di richieste del **modello tradizionale**, riducendo al contempo l'utilizzazione dei centri di servizio e senza impatti significativi sui tempi di risposta percepiti dagli utenti. L'analisi del secondo obiettivo mostra inoltre che lo stesso carico di lavoro può essere soddisfatto con un numero inferiore di veicoli. L'ultima configurazione analizzata conferma infatti la possibilità di contenere il parco veicoli complessivo senza compromettere la qualità del servizio. Questo risultato si traduce in un vantaggio concreto sia in termini di **sostenibilità ambientale**, sia come supporto a politiche di **mobilità condivisa**, perfettamente coerenti con la **strategia green** di **Bolt**.

Riferimenti bibliografici

- [1] <https://eur-lex.europa.eu/legal-content/IT/TXT/HTML/?uri=CELEX:32019R0631>
- [2] <https://www.consilium.europa.eu/it/policies/fit-for-55>
- [3] <https://etrr.springeropen.com/articles/10.1186/s12544-025-00708-x>
- [4] Jerry Banks, John S Carson II, Barry L Nelson, and David M Nicol
Discrete-event system simulation fourth edition