Gestione di una Stazione Ferroviaria

Progetto di Modellistica, simulazione e valutazione delle prestazioni

Danilo Dell'Orco 0300229
Michele Salvatori 0306362
AA 2020/2021

Indice

1 Introduzione

A seguito della pandemia globale Covid-19, diverse attività hanno dovuto adeguare le proprie strutture e protocolli per rispettare le direttive sanitarie indicate dal governo.

Il nostro studio prende in esame una stazione ferroviaria. Per permettere l'accesso in stazione, è necessario controllare la temperatura dei passeggeri, verificando che questa non superi i 37°. A tale scopo, sono stati introdotti all'ingresso della stazione dei sistemi di rilevamento automatico della temperatura.

Gli utenti che passano il controllo, possono essere di tre tipi: abbonati, con biglietto online, o senza biglietto.

- Gli utenti abbonati avranno accesso diretto ai tornelli automatici in cui potranno validare tramite NFC il loro abbonamento.
- Gli utenti senza biglietto dovranno acquistare un biglietto presso uno degli appositi sportelli, e poi accedere ai gate in cui un operatore verifica il biglietto tramite codice QR.
- Gli utenti con biglietto acquistato online accederanno direttamente al Gate per la verifica del codice QR, senza passare per la biglietteria.

Per viaggiare con i mezzi pubblici è obbligatorio essere in possesso di un Green Pass valido; per questo ogni stazione, effettua dei controlli a campione prima dell'accesso effettivo al treno, verificando la certificazione vaccinale dei passeggeri presenti sui binari. Tipicamente il controllo sul green pass viene effettuato a bordo dai controllori della compagnia ferroviaria, ma molte stazioni effettuano un controllo preliminare anche lungo i binari.

2 Obiettivo

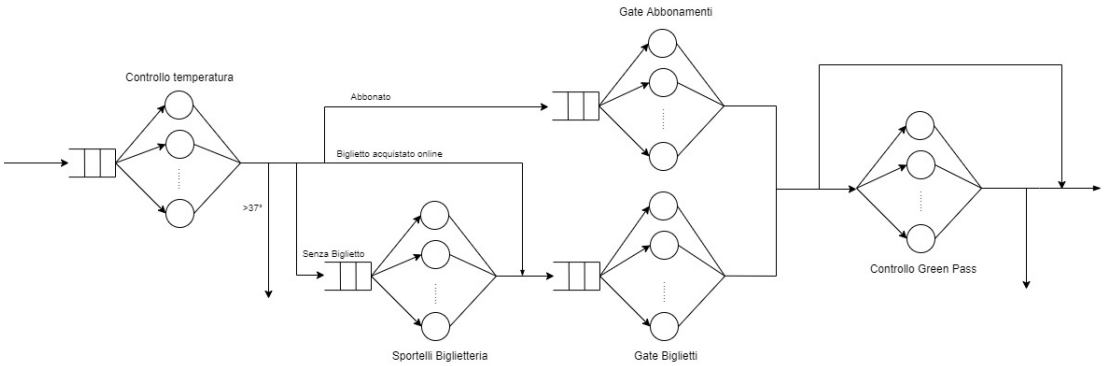
L'obiettivo dello studio è quello di minimizzare i costi per la gestione dei servizi ferroviari e anti-covid. Quindi in particolare si vuole individuare il numero ottimale di serventi ed operatori per ogni sottosistema della stazione nelle diverse fasce orarie.

In particolare si vogliono rispettare due **QoS**:

- Mantenere il tempo di risposta complessivo del sistema inferiore ai 2 minuti, senza considerare i tempi di percorrenza tra i vari sottosistemi della stazione.
- Verificare il Green Pass per almeno il 70% dei passeggeri che escono dai gate ed accedono ai binari.

A tale scopo si effettua uno studio sia dello stato stazionario che del transiente, provando diverse configurazioni di serventi e cercando di individuare quale di queste rispetta i due vincoli mantenendo il minor costo totale.

3 Modello Concettuale



La stazione è stata modellata secondo la rete riportata in figura. Si distinguono 5 *sottosistemi* o *blocchi* differenti:

- Blocco 0: Controllo Temperatura
- Blocco 1: Acquisto Biglietti
- Blocco 2: Gate Abbonamenti
- Blocco 3: Gate Biglietti
- Blocco 4: Controllo Green Pass

I primi 4 blocchi sono modellati come una **M/M/k**; le code sono infinite in quanto una stazione ferroviaria non limita il numero dei clienti che può servire. Quando c'è un arrivo in un blocco, si verifica sempre se c'è un servente libero; se sono presenti più sportelli liberi si sceglie sempre il primo tra questi, altrimenti si entra in coda.

Il controllo del green pass è modellato come un **M/M/k/k** (loss system), in quanto se i controllori sono tutti occupati semplicemente i passeggeri accederanno direttamente al treno. In tal senso l'obiettivo dello studio è quindi quello di dimensionare la rete in modo che il numero di passeggeri che non effettuano il controllo sia al più del 30%.

Ai fini dello studio i passeggeri con green pass non valido non sono rilevanti in quanto l'analisi non comprende la fase di accesso al treno ma soltanto l'insieme di controlli prima di accedere effettivamente ai binari. Sia il Green Pass valido che non valido quindi vengono considerati come completamenti.

Una stazione ferroviaria è aperta tipicamente dalle 05:00 a 00:00, ed in questo periodo il numero di passeggeri non è costante, ma ci sono dei periodi di rush. Per questo sono state identificate 3 differenti fasce orarie:

- 05:00 – 08:00: traffico medio
- 08:00 – 19:00: traffico intenso
- 19:00 – 00:00: traffico basso

Le **variabili di stato** considerate sono

- Numero di Serventi Disponibili nei vari blocchi in ogni fascia oraria
- Stato di uno Servente (libero/occupato/online)

Gli **utenti** in ingresso al sistema sono di tre tipi:

- Senza Biglietto
- Con Biglietto acquistato online
- Abbonati

Gli **eventi** che possiamo avere sono:

- Arrivo di un utente
- Servizio di un utente presso l'apposito servente
- Cambio di Fascia oraria

Il cambio di fascia oraria può causare l'apertura o la chiusura dei gate/serventi attivi nella specifica fascia oraria per mantenere il sistema stabile e far fronte alla variazione del flusso in ingresso.

Ogni servente attivo implica avere un maggiore costo di manutenzione/gestione, per cui l'obiettivo è quello di mantenere il numero ottimo di serventi per ogni fascia oraria, in modo da minimizzare le spese rispettando comunque i due QoS.

4 Modello di Specifiche

4.1 Dati di Input

Per la scelta dei parametri di input del sistema sono stati considerati i dati della stazione ferroviaria di Standford(UK), forniti dal report dell' anno finanziario 2020/2021 prodotto da *The Office of Rail Regulation (ORR)*.

In particolare sono stati presi come riferimento i dati riguardanti il numero di passeggeri durante l'anno, il numero di abbonati, ed il numero di persone che acquistano un biglietto online.

I tempi di interarrivo ed i tempi di servizio sono modellati secondo una distribuzione **Esponenziale** che, come discusso in letteratura, rappresenta una buona scelta per il sistema in esame.[1][2]

- Periodo di osservazione: 19 ore
- Numero di Passeggeri al Giorno: 43800[3]
    - 0,640351 arrivi medi al secondo

- Gli arrivi sono suddivisi in 3 differenti fasce orarie
    - **05:00** – **08:00**: 10% dei passeggeri giornalieri, 4380 passeggeri totali
        - 0,405556 arrivi/sec
    - **08:00** – **19:00**: 75% dei passeggeri giornalieri, 39600 passeggeri totali
        - 0,829545 arrivi/sec
    - **19:00** – **24:00**: 15% dei passeggeri giornalieri, 18000 passeggeri totali
        - 0,365 arrivi/sec
- Tempo verifica Green Pass 30 secondi
    - 0.033333 verifiche al secondo, incluso il tempo per il controllo dei documenti
- Tempo di verifica Temperatura 15 secondi
    - 0,066667 scansioni al secondo
- Tempo per l'acquisto di un biglietto 90 secondi
    - 0.011111 acquisti al secondo
- Tempo per la verifica manuale di un biglietto 25 secondi
    - 0.04 verifiche biglietto al secondo, incluso il tempo per il controllo dei documenti
- Tempo per la verifica automatica di un abbonamento 10 secondi
    - 0.1 verifiche abbonamento al secondo
- Percentuale con temperatura '>37°': 0.2%
    - Probabilità di uscire dal sistema dopo il controllo $p_T = 0.002$
    - Probabilità di entrare in stazione dopo il controllo $p_S = 0.998$
- Percentuale di utenti abbonati: 25,7 [4]
    - Probabilità di utente abbonato $p_A = 0.257$
    - Probabilità di utente con biglietto $p_B = 0.743$
- Percentuale di Biglietti venduti online: 35% [5]
    - Probabilità di utente con biglietto online $p_O = 0.35$
    - Probabilità di utente che deve acquistare il biglietto $p_{AB} = 0.65$

4.2 Costi di Gestione

Si è assunto per ogni tipo di servente un **costo di gestione mensile**. Il costo viene calcolato su ogni servente per il tempo totale in cui questo è attivo, anche se poco utilizzato o non utilizzato.

- Costo mensile scanner temperatura: $C_0 = 300€$
- Costo mensile biglietteria automatica: $C_1 = 200€$
- Costo mensile gate abbonamento: $C_2 = 50€$
- Costo mensile dipendente gate biglietto: $C_3 = 1300€$[6]
- Costo mensile dipendente verifica green pass: $C_4 = 800€$
    - Si assume che alla verifica del green pass non vi siano controllori specifici, ma anche dipendenti part-time.

4.3 Probabilità di Routing

A livello di modellazione non generiamo in modo aleatorio il tipo di utente al suo arrivo, ma distribuiamo il flusso totale in ingresso verso i vari blocchi tramite le **probabilità di routing**, definite utilizzando i dati percentuali ricavati in precedenza.

- Probabilità di routing in uscita dal controllo temperatura
    - Uscita dal sistema: $p_{0,exit} = p_T = 0.002$
    - Acquisto Biglietti: $p_{0,1} = p_S * p_B * p_{AB} = 0{,}481984$
    - Controllo Abbonamento: $p_{0,2} = p_S * p_A = 0.256486$
    - Controllo Biglietto: $p_{0,3} = p_S * p_B * p_O = 0.25953$
- Le altre probabilità di routing sono tutte pari a 1, in quanto in uscita dai blocchi 1-4 la destinazione è deterministica.
    - $p_{1,3} = 1$
    - $p_{2,4} = 1$
    - $p_{3,4} = 1$

5 Modello Computazionale

Per la simulazione della rete è stato utilizzato l'approccio di tipo **Next-Event Simulation**, in cui l'avanzamento del tempo si effettua tramite processamento dell'evento successivo. Il modello è sviluppato in linguaggio `c`, utilizzando la libreria *http://www.math.wm.edu/~leemis/simtext.code.c*. Il codice della simulazione è implementato in `simulate.c`, In `utils.c` sono presenti diverse funzioni ausiliarie utilizzate per l'implementazione, mentre in `conf.h` è possibile modificare i parametri di input e di configurazione del sistema.

5.1 Strutture Dati

Descriviamo le principali strutture dati utilizzate nel nostro software.

- Struttura `block` che tiene le informazioni sul singolo blocco/nodo del sistema
    - I job in coda nel blocco sono implementati tramite una *Linked List* di `job`, in cui ogni Job punta al job successivo.
    - La linked list mantiene i job attualmente nei serventi, più quelli in attesa di essere serviti nel blocco
- Struttura `server` che rappresenta un servente
- Struttura `clock` che mantiene il tempo di simulazione. Sono presenti 3 campi:
    - `current` mantiene il clock attuale di simulazione.
    - `arrival` mantiene il clock del prossimo arrivo da processare.
    - `next` mantiene il clock del prossimo evento da processare.
- Struttura `network_status` che mantiene lo stato complessivo della rete e dei serventi, quindi quali di questi sono online, e quali di questi sono idle/busy/online
- Struttura `sorted_completions` che mantiene tutti gli eventi di completamento in una lista ordinata. In questo modo per ottenere il prossimo completamento è sufficiente accedere a `sorted_completions[0]`.
    - Le operazioni in media sono `O(log(n))` dove `n` è il numero di completamenti generati. La simulazione è molto più efficiente non dovendo effettuare una scansione lineare di tutti i server.
    - `insertSorted()` inserisce un elemento ordinato, `O(n)` nel caso peggiore.
    - `deleteElement()` cancella un elemento tramite ricerca binaria, `O(n)` nel caso peggiore.

5.2 Gestione degli eventi

Gli eventi di completamento ed arrivo vengono gestiti in modo differente, per cui è sempre necessario verificare se il prossimo evento è un arrivo o un completamento.

1. Si ottiene il `next_completion` tramite `sorted_completions[0]`.
2. Si avanza il clock settando `clock.next` al minimo tra `clock.arrival` e `next_completion` e `clock.current` a `clock.next`.
3. Se `clock.current` è uguale a `clock.arrival` devo gestire un *arrivo*, altrimenti un *completamento*.

5.2.1 Arrivo

Si genera il primo tempo di arrivo tramite `getArrival()` che utilizza la funzione `Exponential()` di `rvgs.c`. Ogni arrivo dall'esterno viene gestito dal blocco per il controllo temperatura. Si verifica se c'è un servente libero in questo blocco tramite `findFreeServer()`.

Se c'è un servente libero nel blocco temperatura, allora:

1. Genero un tempo di servizio esponenziale per il completamento tramite `getService()`.
2. Registro il `job` nel servente, inserendolo nella linked list dei job tramite `enqueue()`.
3. Inserisco il completamento in `sorted_completions` tramite `insertSorted()`.
4. Setto lo stato del server come `BUSY`.

Se invece non c'è nessun servente libero inserisco soltanto il job nella coda del blocco.

5.2.2 Completamento

Quando viene processato un completamento si rimuove il job dal servente tramite `dequeue()` e si rimuove il completamento dalla struttura ausiliaria `sorted_completions` tramite `deleteElement()`.

Il completamento di un servente equivale ad un arrivo sul blocco successivo.

1. Tramite `findDestination(block from)` si trova in base al nodo corrente `from` quale è il nodo destinatario secondo le probabilità di routing.
    - Si genera un valore casuale tra 0 e 100 tramite `Uniform()`.
    - Si confronta questo valore con le diverse probabilità, individuando quindi il blocco di destinazione.
2. Si gestisce la consegna verso il blocco destinatario del job appena servito come descritto in 5.2.1.

5.2.3 Cambio di Fascia oraria

Al cambio di fascia oraria varia il flusso di ingresso e viene quindi riconfigurata la rete per far fronte al nuovo traffico. A livello implementativo si utilizza una funzione `update_network()` che confronta la configurazione attuale con quella della nuova fascia, e attiva o disattiva per ogni blocco il numero di server necessari.

Se un server è in servizio quando viene scelto per la chiusura si aspetta che termini l'esecuzione attuale per andare a disattivarlo in un secondo momento. In questo caso quindi si flagga quel server come `need_resched`, e al prossimo completamento si procede con la sua terminazione. Questo permette di non sprecare inutilmente il lavoro; infatti non è conveniente né realistico spegnere immediatamente uno sportello già in servizio. Il tempo online in eccesso viene considerato nel calcolo dei costi.

5.3 Simulazione ad Orizzonte Infinito

Nella simulazione ad orizzonte infinito il sistema viene simulato per un tempo di simulazione "infinito", quindi di molto superiore al tempo reale. In questo modo si producono le statistiche a stato stazionario del sistema.

Per ridurre il bias dello stato iniziale si effettua una run di simulazione più lunga rispetto alla dimensione della fascia oraria. In ogni simulazione si assume che il sistema sia statico, quindi per ogni fase il tasso `arrival_rate` resta costante.

Per ricavare la media campionaria del tempo di risposta si utilizza il metodo delle **Batch Means**, suddividendo la run di simulazione in `k` batches di dimensione `b`. Da ogni batch si ricavano le statistiche tramite `calculate_statistics_inf()`. Si resettano quindi le statistiche tramite `reset_statistics()`, ma senza azzerare lo stato del sistema, ovvero mantenendo tutti i job rimanenti dal precedente batch. In questo modo si genera un campione di `k` batches indipendenti, sul quale è possibile valutare la media campionaria.

Le dimensioni di `b` e `k` utilizzate influenzano la qualità del campione; un valore di `b` grande permette di avere un campione con bassa autocorrelazione, mentre un maggior numero `k` di batch permette di avere un valore migliore in termini di intervallo di confidenza.

Sono stati quindi utilizzati `k` = 128 batch, e seguendo le linee guida proposte da *Banks, Carson, Nelson, and Nicol* [7] si è cercato di individuare il valore di `b` per cui cui l'autocorrelazione del campione prodotto è inferiore a 0.2 per lag j = 1.

E' stata definita la funzione `find_batch_b()` per generare un campione al variare di diversi valori di b. Su ogni campione è stata valutata quindi l'autocorrelazione tramite il programma `acs.c`. Si è visto che per b=1024 si ha un'autocorrelazione di 0.033 per j=1, quindi si è scelto tale valore come dimensione del batch.

- In totale, per ogni simulazione vengono quindi processati `b*k=131072 job`.

Gli obiettivi della simulazione ad orizzonte infinito ai fini del nostro studio sono principalmente 3:

- Analisi dei tempi di risposta a steady state
- Analisi della probabilità $\mathbf{P}_{\text{bypass}}$
    - Valutata come il rapporto tra il numero di passeggeri che trovano i serventi tutti occupati (bypassano il controllo) ed il numero totale di arrivi nel blocco 4
- Ricerca della configurazione ottima che permette per ogni fascia oraria di rispettare i due QoS e allo stesso tempo minimizzare i costi.

5.4 Simulazione ad Orizzonte Finito

Nella simulazione ad orizzonte finito viene effettuata una simulazione del sistema lungo le 19 ore lavorative; il sistema è assunto quindi dinamico, e viene considerata la variazione del flusso nelle diverse fasce orarie e la riorganizzazione del numero di serventi attivi. Il sistema è `IDLE` sia all'inizio che alla fine della simulazione. Un'analisi di questo tipo permette di ottenere le statistiche di sistema transienti.

Per effettuare un'analisi statistica dei valori ottenuti ad orizzonte finito, il procedimento di misurazione è stato replicato 128 volte, ottenendo quindi un ensemble di dimensione pari a 128. Ogni replica viene utilizzata per misurare le stesse statistiche, e fornisce quindi un punto del nostro campione.

Per ottenere la media campionaria e l'intervallo di confidenza al 95% si è utilizzato il programma `estimate.c`, che internamente utilizza la distribuzione *Student*.

Come da *linee guida*[8], il seed viene impostato tramite `PlantSeeds()` fuori dal ciclo di replicazione, mentre per le prove successive alla prima viene usato come stato iniziale di ogni stream rng lo stato finale degli stessi stream per la replica precedente. Facendo ciò si evitano possibili sovrapposizioni degli eventi generati dalle singole repliche.

La condizione di terminazione della simulazione è il raggiungimento dell'orario di chiusura (00:00), e quindi quando il clock di simulazione raggiunge il valore di 68400 secondi (=19 ore).

Per analizzare le statistiche ottenute nel continuo, per ogni ripetizione, effettuiamo ogni 5 minuti una misurazione del tempo di risposta, che viene salvato in file `csv`.

- Per ogni istante di tempo (05:00, 05:05, 05:10…) si avranno quindi 128 misurazioni del tempo di risposta
- Calcolando la media di queste stime si ottiene quindi il valore del tempo di risposta considerato ai fini dell'analisi per quello specifico orario.

Nella nostra analisi, la simulazione ad orizzonte finito ha come obiettivi principali:

- Analizzare il comportamento del sistema al cambio fascia, verificando che le configurazioni individuate ad orizzonte infinito siano valide effettivamente anche nel sistema reale.
- Analizzare i costi reali sull'arco dell'intera giornata.
- Testare diverse configurazioni per valutare più nel dettaglio il comportamento del sistema nel transiente ed in particolare al cambio di fascia oraria.

6 Verifica

Vediamo che il modello computazionale è effettivamente conforme al modello di specifica, quindi che l'implementazione del modello computazionale sia effettivamente corretta.

Per fare questo abbiamo verificato 4 condizioni:

A. Dati N job in ingresso le percentuali di job in ingresso su ogni blocco sono conformi al tasso di arrivo e alle probabilità di routing specificate.

B. Il tempo di risposta (`wait`) per un server è sempre uguale alla somma del tempo di attesa (`delay`) e del tempo di servizio (`service`).

C. Il numero di job in ingresso è conforme a quello indicato nel modello delle specifiche

D. Il numero di arrivi sul blocco è sempre uguale al numero di job in coda più il numero di job completati

Per valutare queste 4 condizioni sono state eseguite diverse run ad orizzonte sia finito che infinito. Riportiamo qui i risultati di una run nell'arco delle 19 ore lavorative.

```
Result for block TEMPERATURE_CTRL
Arrivals ................. = 44070
Completions............... = 43981
Dropped................... = 89
Average wait ............. = 15.196649
Average delay ............. = 0.206358
Average service time ...... = 14.990291
```

A. 15.196649 = 0.206358 + 14.990291

B. 44070 = 43981 + 89

```
Result for block TICKET_BUY
Arrivals ................. = 21238
Completions............... = 21238
Average wait ............. = 91.714337
Average delay ............. = 2.191239
Average service time ...... = 89.523098
```

A. 91.714337 = 2.191239 + 89.523098

B. 21238 arrivi = 21238 completamenti

```
Result for block SEASON_GATE
Arrivals ................. = 11297
Completions............... = 11297
Average wait ............. = 10.018958
Average delay ............. = 0.251060
Average service time ...... = 9.767898
```

A. 10.018958 = 0.251060 + 9.767898

B. 11297 arrivi = 11297 completamenti

```
Result for block TICKET_GATE
Arrivals ................. = 32684
Completions............... = 32684
Average wait ............. = 25.048388
Average delay ............ = 0.258535
Average service time ..... = 24.789853
```

A. 25.048388 = 0.258535 + 24.789853

B. 32684 arrivi = 32684 completamenti

```
Result for block GREEN_PASS
Arrivals ................. = 43981
Completions............... = 38825
Average wait ............. = 29.697514
Average wait (2).......... = 26.216002
Number bypassed .......... = 5156
Average delay ............ = 0.000000
```

`wait (2)` rappresenta l'attesa media di un qualsiasi job che entra nel blocco green pass, quindi sia se effettua il controllo, sia se lo bypassa.

`Wait` rappresenta l'attesa media dei soli job che effettuano il controllo green pass.

Possiamo quindi verificare che la media tra il tempo di risposta per i soli job che entrano in servizio (29.697514) ed i job che bypassano il controllo (tempo di risposta 0) corrisponde al valore `wait (2)`

A. $\frac{(43981 - 5156) * 29.697514 + 5156 * 0\ ]}{43981} = 26.216002$

B. 43981 = 38825 + 5156

Per la *condizione C*, vediamo che abbiamo 44070 arrivi nel blocco temperatura, che potrebbero sembrare non allineati con il valore del modello di specifica. In realtà questo valore è un outlier in quanto gli arrivi sono generati secondo un processo stocastico. Infatti calcolando la media del numero di arrivi su 128 ripetizioni della simulazione abbiamo un numero di job in ingresso pari a 43797

```
Mean Arrivals for #128 repetitions: 43797
```

Per la *condizione D*, vediamo che i 44070 job in ingresso in questa run si distribuiscono in maniera conforme alle probabilità di routing specificate.

`DROPPED` *(p=0.2%)*: 44070 * 0.002 = 88.14 ≈ 89 job scartati.

`TICKET_BUY` *(p=48.1984%)*: 44070 * 0.481984 = 21241.03488 ≈ 21249 job in ingresso all'acquisto dei biglietti.

`SEASON_GATE` *(p=25.6486%)*: 44070 * 0.256486 = 11303.33802 ≈ 11297 job in ingresso al controllo temperatura.

`TICKET_GATE` *(p=25.953%)*: 44070 * 0.25953 = 11437.4871. Sommando a questo valore i completamenti del ticket buy abbiamo 11437.4871 + 21241.03488 = 32678.52198 ≈ 32684 job in ingresso al gate abbonamenti.

`GREEN_PASS` *(p=100%)*: Tutti i job completati dal controllo temperatura (43981) vediamo che arrivano effettivamente in ingresso al controllo green pass.

7 Validazione

Nella fase di validazione vogliamo verificare che il modello computazionale è coerente con il sistema reale che abbiamo analizzato. Non avendo un dataset con dati reali a disposizione, si opera una validazione rispetto al modello analitico, quindi verificando che i risultati ottenuti rispettino le leggi teoriche.

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

La validazione del sistema è stata effettuata ad orizzonte finito durante l'intera giornata. È stata testata la seguente configurazione, scelta casualmente:

- Fascia 05:00 – 08:00: `#TEMP=9 #BUY=22 #SEAS=3 #TICK=11 #GREEN=10`
- Fascia 08:00 – 19:00: `#TEMP=14 #BUY=42 #SEAS=4 #TICK=20 #GREEN=20`
- Fascia 19:00 – 00:00: `#TEMP=9 #BUY=20 #SEAS=3 #TICK=12 #GREEN=10`

Nella fase di validazione si confrontano i risultati della simulazione rispetto al modello analitico. In particolare le leggi controllate sono:

- $E(TQ_k) = P_Q * \frac{\rho}{\lambda(1-\rho)}$ *per* $k \in [0,3]$
  - $P_Q = \frac{(m\rho)^m}{m!(1-\rho)} * P(0)$
  - $P(0) = \frac{1}{\sum_0^{m-1} \frac{(m\rho)^i}{i!} + \frac{(m\rho)^m}{m!(1-\rho)}}$
- $E(T_{s,k}) = E(TQ_k) + E(S_i)$ *per* $k \in [0,3]$
- $P_{bypass} = \Pi_m = \frac{\frac{1}{m} * (\frac{\lambda}{\mu})^m}{\sum_0^m (\frac{\lambda}{\mu})^j * \frac{1}{j!}}$
- $E(TQ_4) = 0$
  - Il tempo di coda per un M/M/k/k è sempre 0, in quanto non ha coda.
- $E(T_{s,4}) = P_{bypass} * 0 + (1 - P_{bypass}) * E(S_4)$
  - Nel nostro modello i job che bypassano il controllo sono comunque completati, e quindi nella media pesata hanno tempo di servizio pari a 0.
- $E(T_{s,tot}) = \sum_0^4 v_k * E(T_{s,k})$
  - Il tempo di risposta della rete si ottiene tramite la somma di tutti i tempi di risposta dei singoli blocchi, pesandola per il numero di visite del blocco

05:00 – 08:00		
Statistica	**Risultato Analitico**	**Risultato Sperimentale (α=0.05)**
$\mathbf{E(TQ_0)}$	1,073706144	1.035781 +/- 0.095251
$\mathbf{E(TQ_1)}$	4,798709172	5.039429 +/- 1.528223
$\mathbf{E(TQ_2)}$	0,510800481	0.492448 +/- 0.047513
$\mathbf{E(TQ_3)}$	1,27843378	1.429423 +/- 0.181819
$\mathbf{P_{bypass}}$	0,307710371	0.305489 +/- 0.004120
$\mathbf{E(T_{s,tot})}$	104,6740257	104.807374 +/- 1.085204
08:00 – 19:00		
$\mathbf{E(TQ_0)}$	5,567558552	5.320638 +/- 0.652626
$\mathbf{E(TQ_1)}$	3,6597019	4.005438 +/- 1.459421
$\mathbf{E(TQ_2)}$	1,10457884	1.066030 +/- 0.161535
$\mathbf{E(TQ_3)}$	1,043555698	1.030215 +/- 0.159809
$\mathbf{P_{bypass}}$	0,276229453	0.273317 +/- 0.005148
$\mathbf{E(T_{s,tot})}$	109,5396437	109.336112 +/- 1.306189
19:00 – 00:00		
$\mathbf{E(TQ_0)}$	0,540070613	0.536954 +/- 0.052692
$\mathbf{E(TQ_1)}$	5,139317634	5.511025 +/- 1.160988
$\mathbf{E(TQ_2)}$	0,374300989	0.361841 +/- 0.043293
$\mathbf{E(TQ_3)}$	0,240801791	0.251061 +/- 0.039264
$\mathbf{P_{bypass}}$	0,256430139	0.254103 +/- 0.003774
$\mathbf{E(T_s)}$	105,035357	105.744819 +/- 0.889858

Possiamo quindi vedere come il modello simulativo rispetti le leggi del modello analitico. Anche non avendo dati reali a disposizione questo ci conferma che il nostro modello rappresenta bene un modello reale, e possiamo quindi proseguire con la fase di analisi dei risultati.

8 Progettazione ed esecuzione degli Esperimenti

L'approccio seguito nella simulazione è quello di testare diverse configurazioni di server per le diverse fasce orarie. Una configurazione è mantenuta tramite una struttura `config` che contiene i serventi che vogliamo mantenere per ogni blocco in ogni fascia oraria.

Sono state quindi definite diverse configurazioni, utilizzando combinazioni differenti di serventi per ogni fase, che sono state utilizzate in diverse simulazioni sia ad orizzonte finito che infinito.

I risultati di ogni simulazione vengono scritti su un file csv differente, analizzabile poi tramite `estimate.c` e/o `uvs.c` per ottenere i valori di *media*, *deviazione standard* ed *intervallo di confidenza*.

9 Analisi dei risultati

9.1 Analisi ad Orizzonte Infinito

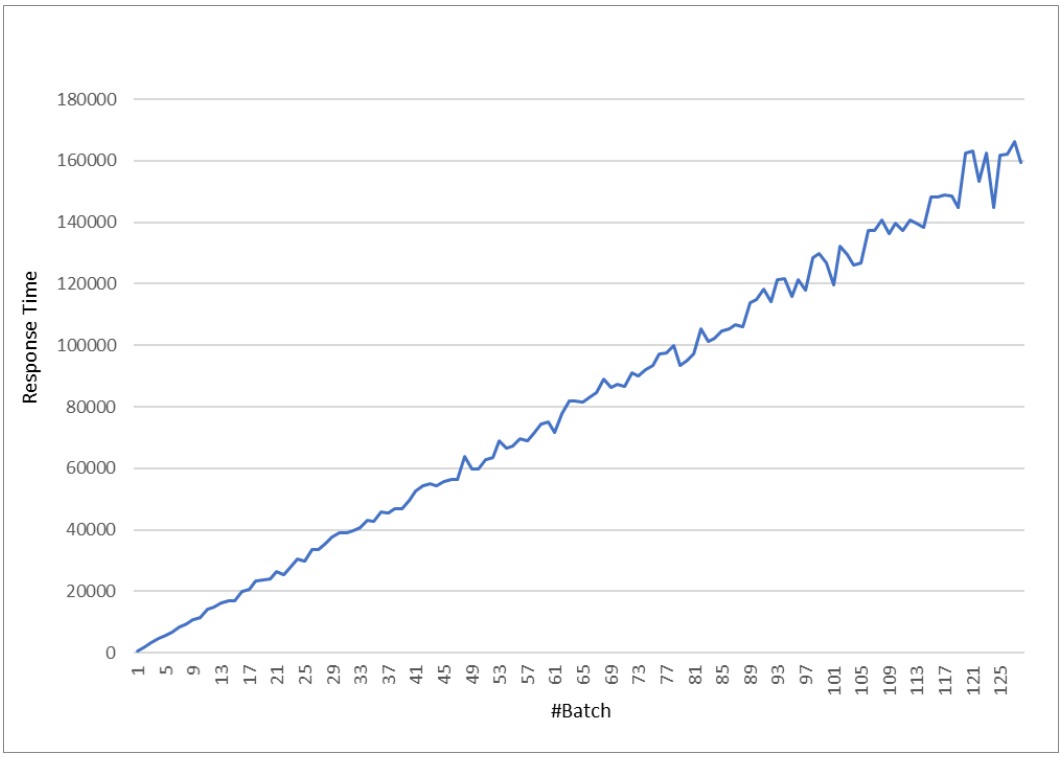
L'approccio seguito è quello di simulare ad orizzonte infinito per cercare la configurazione ottima e per verificare che i tempi di risposta del modello siano conformi a quelli analitici.

Per ogni configurazione testata si valuta quindi se rispetta il QoS del tempo di risposta e se il numero di passeggeri cui viene controllato il green pass è almeno del 70% ($P_{bypass}<0.30$).

Si simulano quindi in questo caso le singole fasce, e si cerca quale tra le diverse configurazioni testate permette di minimizzare il costo ottenuto rispettando entrambi i vincoli.

9.1.1 Fascia 05:00 – 08:00

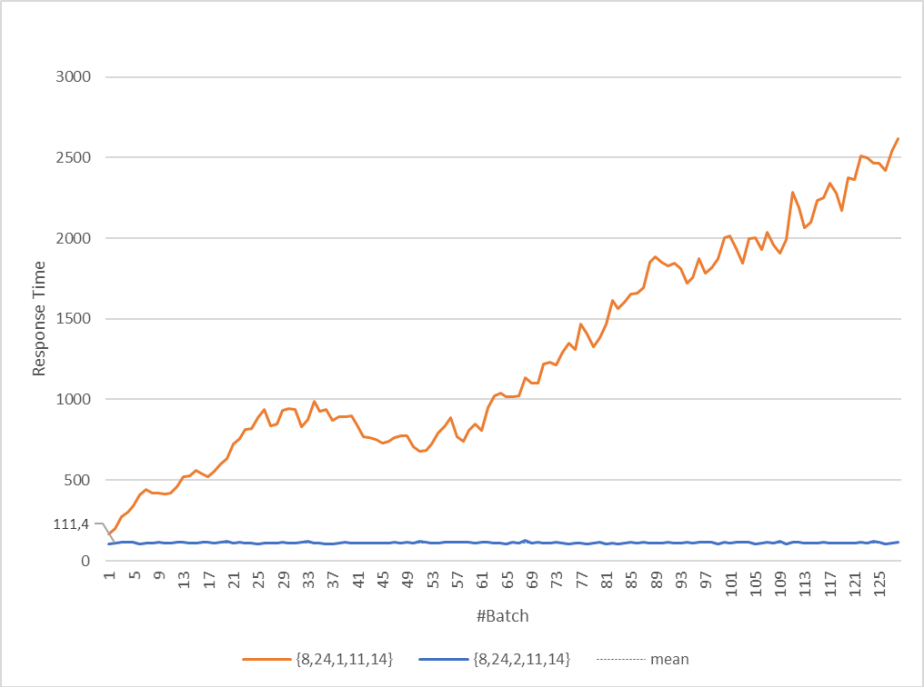
Configurazione {3,20,1,5,15}



Vediamo che con questa configurazione non raggiungiamo mai lo stato stazionario; il sistema non è quindi stabile ed aumenta all'infinito il tempo di risposta. Questa configurazione infatti ha utilizzazione pari a 1 per il controllo temperatura, e quindi tutti i job in ingresso non vengono smaltiti, causando accodamenti già nel primo blocco del sistema.

```
Mean Utilization for block TEMPERATURE_CTRL......1.000000
Mean Utilization for block TICKET_BUY............0.445988
Mean Utilization for block SEASON_GATE...........0.515863
Mean Utilization for block TICKET_GATE...........0.753230
Mean Utilization for block GREEN_PASS............0.399698
Loss Percentage for block GREEN_PASS.............0.000794
------------------------------------------------------------
TOTAL SLOT 0 CONFIGURATION COST..................3694.26€
```

Configurazione {8,24,1,11,14} vs Configurazione {8,24,2,11,14}



Testiamo innanzitutto la configurazione {8,24,1,11,14} (in arancione), ed osserviamo dai risultati che il sistema non è stabile; infatti il tempo di risposta tende a crescere e supera ampiamente il QoS di 120 secondi. Analizzando le utilizzazioni dei vari blocchi vediamo che per la verifica degli abbonamenti si ha un'utilizzazione che tende a 1, e quindi non viene smaltito il traffico in quel blocco.

```
Mean Utilization for block TEMPERATURE_CTRL......0.756365
Mean Utilization for block TICKET_BUY............0.740677
Mean Utilization for block SEASON_GATE...........0.998506
Mean Utilization for block TICKET_GATE...........0.685108
Mean Utilization for block GREEN_PASS............0.753161
Loss Percentage for block GREEN_PASS.............0.115769
-------------------------------------------------------------
TOTAL SLOT 0 CONFIGURATION COST..................5153.62€
```

Miglioriamo quindi la configurazione aggiungendo un server nel gate abbonamenti; vediamo che l'utilizzazione passa da 0.998506 a 0.515732. Il costo cresce solo sensibilmente, ed il tempo di risposta medio è di 111.44 +/- 0.70.

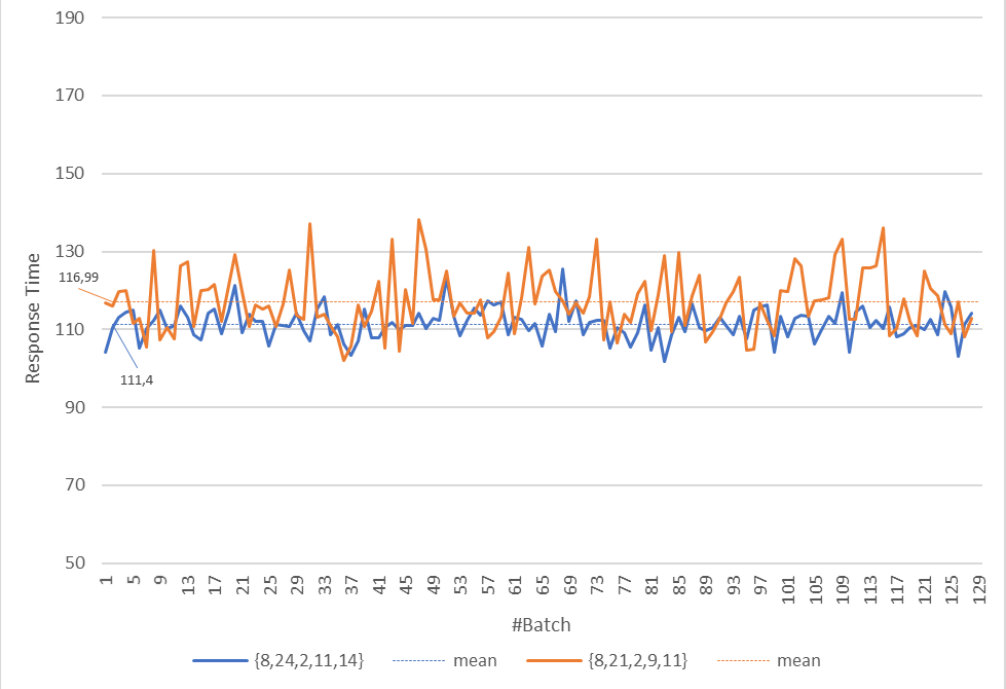
```
Mean Utilization for block TEMPERATURE_CTRL......0.753277
Mean Utilization for block TICKET_BUY............0.738213
Mean Utilization for block SEASON_GATE...........0.515732
Mean Utilization for block TICKET_GATE...........0.681536
Mean Utilization for block GREEN_PASS............0.756859
Loss Percentage for block GREEN_PASS.............0.119791
-------------------------------------------------------------
TOTAL SLOT 0 CONFIGURATION COST..................5161.18€
```

Configurazione {8,24,2,11,14} vs Configurazione {8,21,2,9,11}

La configurazione {8,24,2,11,14} rispetta entrambi i QoS, ma le utilizzazioni sono ampiamente inferiori a 1, per cui possiamo rimuovere qualche server in modo da ottimizzare i costi.

- $E(T_{s,tot})$: 111.44 +/- 0.70 < 120 sec
- P_{bypass}: 0,119791 < 0,30

Passiamo quindi alla configurazione {8,21,2,9,11} e confrontiamo i risultati ottenuti.



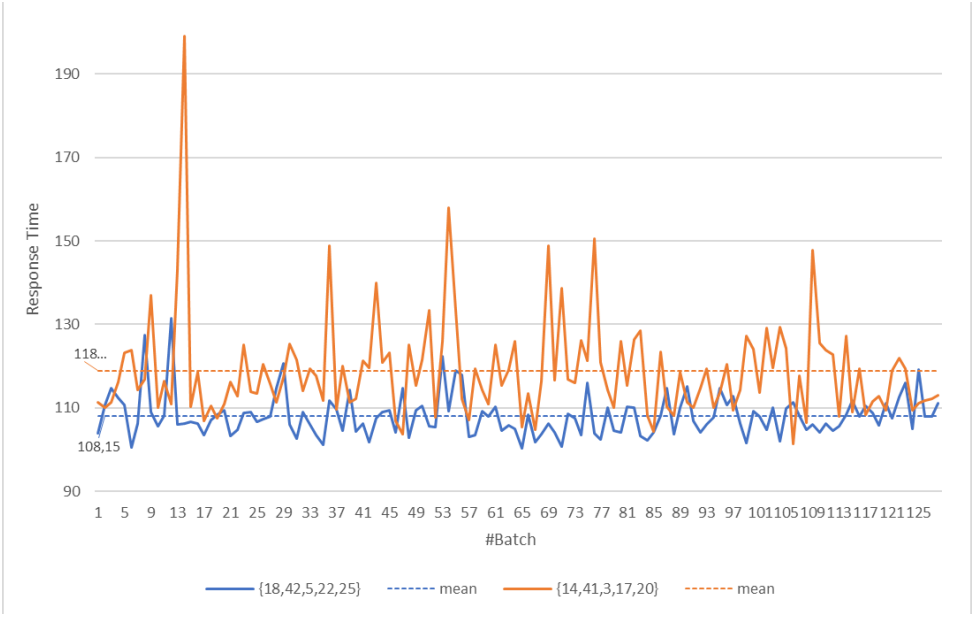
Vediamo che i tempi di risposta salgono, fino a 116.99 +/- 1.35, che rispetta comunque il QoS. Analizzando le altre statistiche osserviamo che il costo scende notevolmente da 5161.18€ a 4286.19€, e che rispettiamo anche il vincolo sui passeggeri che bypassano il controllo green pass (`0.248802 < 0.30`).

```
Mean Utilization for block TEMPERATURE_CTRL......0.756277
Mean Utilization for block TICKET_BUY............0.847541
Mean Utilization for block SEASON_GATE...........0.518329
Mean Utilization for block TICKET_GATE...........0.831935
Mean Utilization for block GREEN_PASS............0.820030
Loss Percentage for block GREEN_PASS.............0.248802
------------------------------------------------------------
TOTAL SLOT 0 CONFIGURATION COST..................4286.19€
```

Provando a rimuovere ulteriori server da questa configurazione si viola il QoS relativo al tempo di risposta, per cui **{8,21,2,9,11}** è la **configurazione ottima** per la prima fascia oraria.

9.1.2 Fascia 08:00 – 19:00

Configurazione {18,42,5,22,25} vs Configurazione {14,41,3,17,20}



Proviamo la configurazione {18,42,5,22,25} (in blu) e vediamo che abbiamo tempi di risposta bassi (108.15 +/- 0.89) e anche una bassa P_{bypass} (0.139192).

```
Mean Utilization for block TEMPERATURE_CTRL......0.692108
Mean Utilization for block TICKET_BUY............0.868167
Mean Utilization for block SEASON_GATE...........0.423539
Mean Utilization for block TICKET_GATE...........0.696386
Mean Utilization for block GREEN_PASS............0.850676
Loss Percentage for block GREEN_PASS.............0.139192
------------------------------------------------------------
TOTAL SLOT 0 CONFIGURATION COST..................4816.44€
```

Analizzando le utilizzazioni vediamo che tutti i blocchi possono essere ottimizzati rimuovendo dei serventi, per cui passiamo alla configurazione {14,41,3,17,20}.

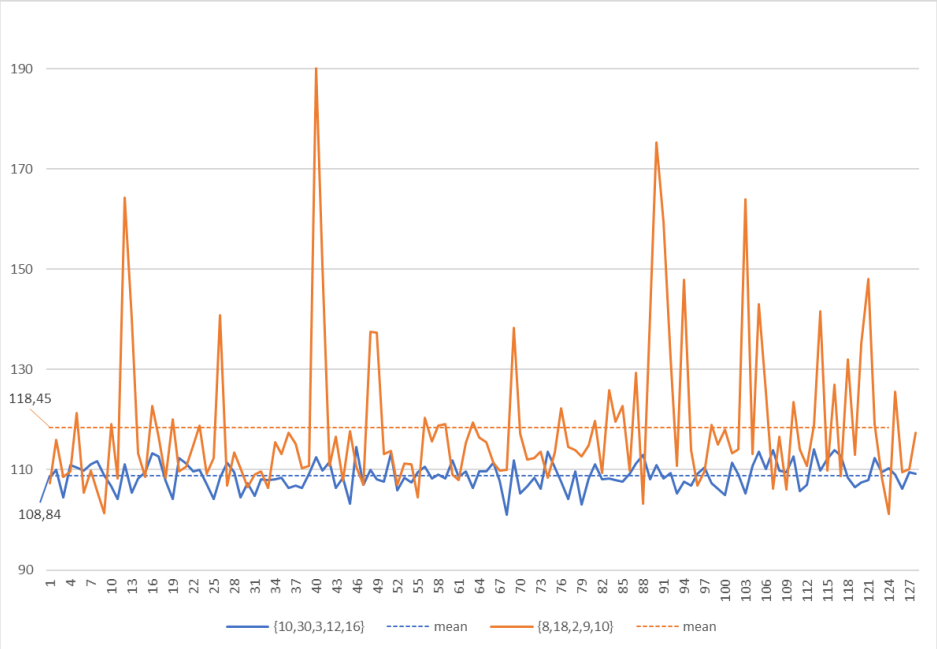
```
Mean Utilization for block TEMPERATURE_CTRL......0.888364
Mean Utilization for block TICKET_BUY............0.890921
Mean Utilization for block SEASON_GATE...........0.695427
Mean Utilization for block TICKET_GATE...........0.903949
Mean Utilization for block GREEN_PASS............0.850676
Loss Percentage for block GREEN_PASS.............0.274171
------------------------------------------------------------
TOTAL SLOT 0 CONFIGURATION COST..................3899.12€
```

I costi totali vengono notevolmente ridotti (da 4816.44€ a 3899.12€), pur mantenendo le statistiche nei limiti del QoS: il tempo di risposta medio è di 118.94 +/- 2.18 (<120) mentre P_{bypass} vale 0.274171 (<0.30).

È facile vedere che rimuovendo un qualsiasi servente non si rispetta uno dei due QoS, per cui **{14,41,3,17,20}** è la **configurazione ottima** per la seconda fascia oraria.

9.1.3 Fascia 19:00 – 00:00

Configurazione {10,30,3,12,16} vs Configurazione {8,18,2,9,10}



Proviamo la configurazione {18,42,5,22,25} e vediamo che abbiamo tempi di risposta bassi (108.15 +/- 0.89) e anche una bassa P_{bypass} (0.139192).

```
Mean Utilization for block TEMPERATURE_CTRL......0.545523
Mean Utilization for block TICKET_BUY............0.531467
Mean Utilization for block SEASON_GATE...........0.310815
Mean Utilization for block TICKET_GATE...........0.560211
Mean Utilization for block GREEN_PASS............0.651201
Loss Percentage for block GREEN_PASS.............0.038549
------------------------------------------------------------
TOTAL SLOT 0 CONFIGURATION COST..................6554.25€
```

Analizzando le utilizzazioni vediamo che tutti i blocchi possono essere ottimizzati rimuovendo dei serventi, per cui passiamo alla configurazione {8,18,2,9,10}.

```
Mean Utilization for block TEMPERATURE_CTRL......0.680598
Mean Utilization for block TICKET_BUY............0.879193
Mean Utilization for block SEASON_GATE...........0.465129
Mean Utilization for block TICKET_GATE...........0.750972
Mean Utilization for block GREEN_PASS............0.808505
Loss Percentage for block GREEN_PASS.............0.254454
------------------------------------------------------------
TOTAL SLOT 0 CONFIGURATION COST..................4505.36€
```

I costi totali vengono notevolmente ridotti (da 6554.25€ a 4505.36€€), pur mantenendo le statistiche nei limiti del QoS: il tempo di risposta medio è di 118.45 +/- 1.73 (<120) mentre P_{bypass} vale 0.254454 (<0.30). Rimuovendo un qualsiasi servente non si rispettano i QoS, per cui **{8,18,2,9,10}** è la **configurazione ottima** per l'ultima fascia oraria.

9.2 Analisi ad Orizzonte Finito

Con l'analisi ad orizzonte infinito delle singole fasce abbiamo individuato la configurazione ottima, che permette di minimizzare il costo rispettando i QoS durante l'intera giornata lavorativa.

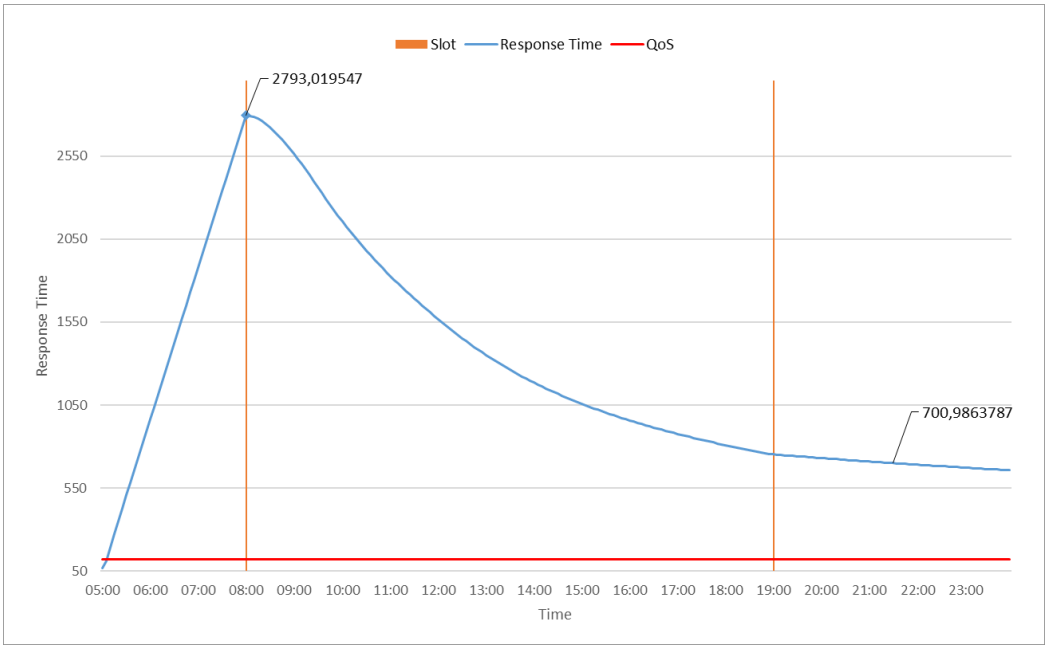
`{8,21,2,9,11},{14,41,3,17,20},{8,18,2,9,10}`

Con l'analisi ad orizzonte finito andiamo a verificare che lo scenario ottimo minimizzi effettivamente i costi, confrontandolo con uno scenario non-ottimo durante tutta la giornata.

Andiamo inoltre ad analizzare diversi scenari specifici, per valutare il comportamento del sistema nel transiente, in particolare al cambio di fascia oraria in cui varia il tasso di ingresso e vengono di conseguenza accesi o spenti un certo numero di server.

9.2.1 Scenario 1 - {3,20,1,5,15}, {18,42,5,22,25}, {10,30,3,12,16}

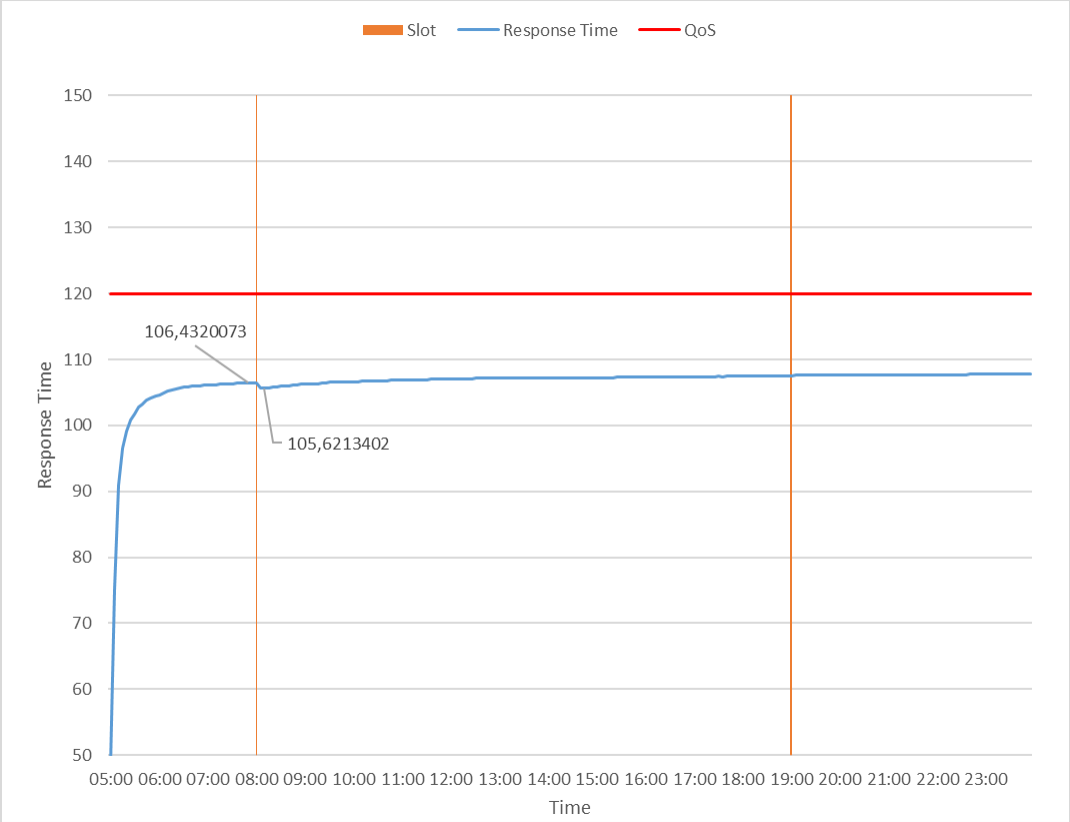
In questa configurazione, abbiamo una prima fascia oraria sottodimensionata, in cui il numero di serventi non mantiene il sistema stabile; nella seconda e nella terza fascia invece vengono allocate più risorse del necessario, quindi permettono di smaltire il traffico accumulato in precedenza.



Come possiamo vedere dal grafico, nella prima fascia aumenta il tempo di risposta in quanto il sistema è instabile, fino ad un massimo di 2793 sec. Al cambio di fascia oraria viene incrementata la configurazione da {3, 20, 1, 5, 15} a {18, 42, 5, 22, 25} per cui vengono attivati molti server. Questo porta il sistema a smaltire velocemente i job rimasti in coda dalla prima fascia, per poi avere una decrescita del tempo di risposta medio. Al termine della giornata lavorativa il tempo medio di risposta resta comunque di molto superiore al limite di 120 secondi dell'obiettivo.

9.2.2 Scenario 2 - {10,30,3,20,15}, {18,42,5,22,25}, {10,30,3,12,16}

In questa configurazione abbiamo un numero di serventi sovradimensionato rispetto alla configurazione ottima. Ci aspettiamo quindi che il tempo di risposta medio sia sempre inferiore al QoS.



Come possiamo osservare dal grafico, al cambio di fascia oraria (08:00) si ha un leggero decremento del tempo di risposta medio. Questo avviene perché aprendo immediatamente 34 server vengono smaltiti istantaneamente 34 job dalle code dei rispettivi blocchi.

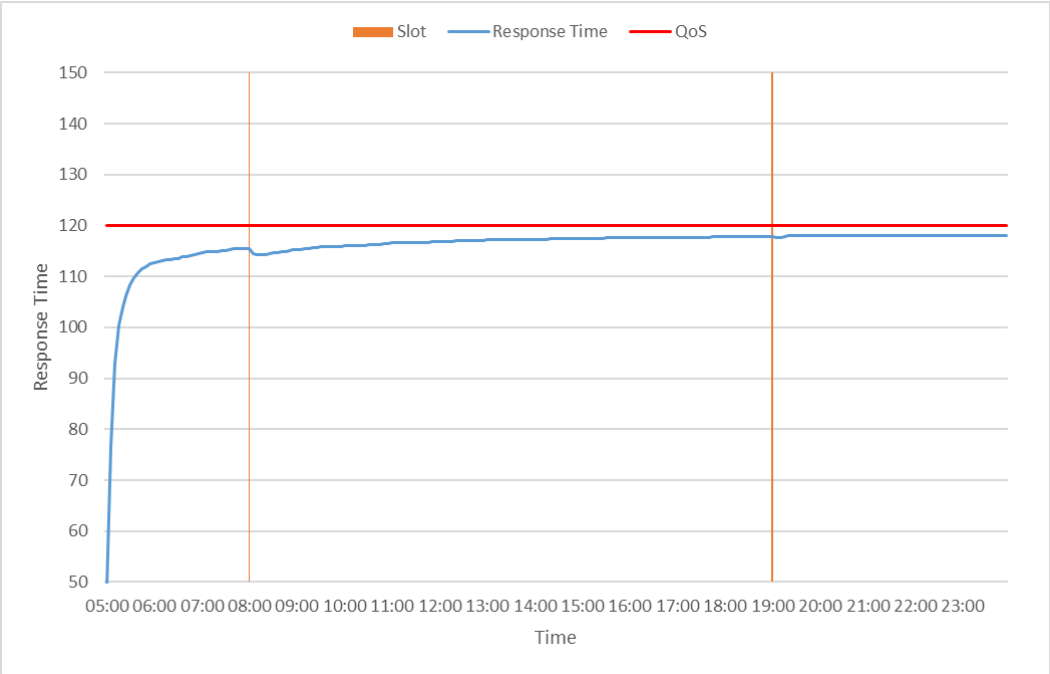
Successivamente il tempo di risposta medio cresce fino a raggiungere la stazionarietà, con media pari a 107.457562 +/- 0.161081 per la fascia centrale e 107.755702 +/- 0.139401 per l'ultima fascia.

Restiamo quindi sempre ampiamente sotto il limite di 120 secondi imposto dal QoS, ma il costo totale della configurazione nell'arco delle 19 ore è di 1786.75€, sicuramente migliorabile.

I risultati mostrano inoltre che nelle tre fasce orarie le probabilità di bypassare il controllo green pass sono rispettivamente 0.081941, 0.130731 e 0.116898, quindi molto al di sotto del QoS di 0.30.

9.2.3 Scenario Ottimo - {8,21,2,9,11}, {14,41,3,17,20}, {8,18,2,9,10}

Analizziamo il comportamento nel transiente della configurazione ottima, verificando che effettivamente il costo sulle 19 ore sia minimo, e che venga rispettato il QoS di 120 secondi.



Nelle tre fasce orarie i tempi di risposta medi sono rispettivamente 115.528435 +/- 0.814483, 117.856171 +/- 0.364914, 117.976557 +/- 0.327956 che confermano quindi i risultati ottenuti tramite analisi ad orizzonte infinito.

Vediamo nella prima fascia come si parta da un tempo di risposta molto basso per poi stabilizzarsi intorno al valore medio circa alle 07:00. Questo comportamento è ragionevole in quanto il sistema è `IDLE` inizialmente e quindi per i primi arrivi si avranno tempi di risposta molto bassi.

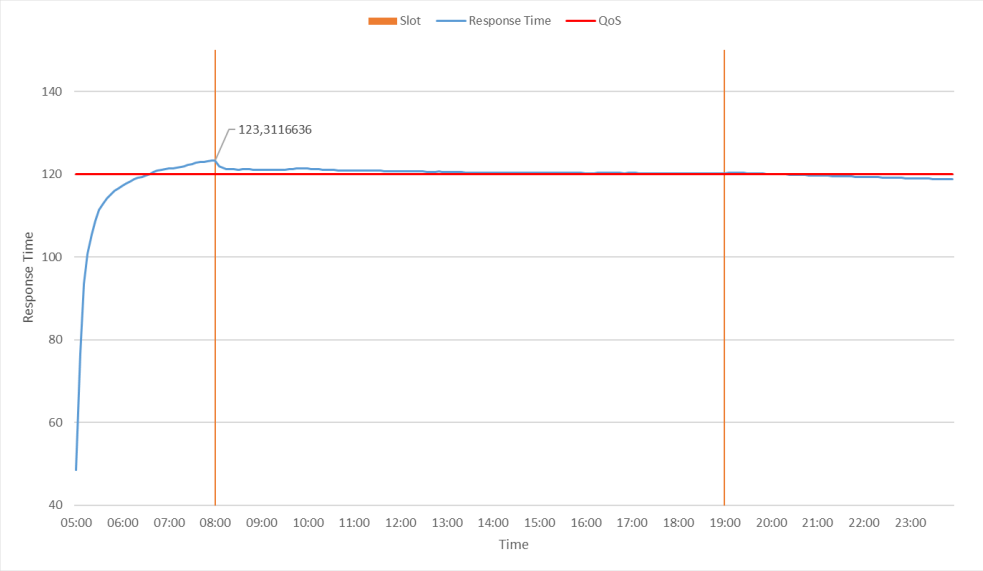
Il costo totale è di 1347.26€, quindi molto inferiore a 1786.75€ visto nella configurazione di *over-provisioning*. Come vediamo dal grafico inoltre in tutte le fasce orarie siamo sotto i 120 secondi per il tempo di risposta, confermando quindi l'ottimalità della configurazione.

Valutando il valore di P_{bypass} nelle 3 fasce orarie vediamo che viene sempre rispettato il QoS di 0.3, in particolare:

- Fascia 05:00 – 08:00: 0.247849 +/- 0.002984
- Fascia 08:00 – 19:00: 0.272511 +/- 0.006007
- Fascia 19:00 – 24:00: 0.271137 +/- 0.008372

9.2.4 Scenario 3 - {9,19,3,11,15}, {14,40,3,17,20}, {10,30,3,12,16}

Analizziamo lo scenario di una configurazione in cui rimuoviamo 2 server dal `TICKET_BUY` della prima fascia rispetto alla configurazione ottimale, ma aumentando i serventi disponibili in tutti gli altri blocchi.



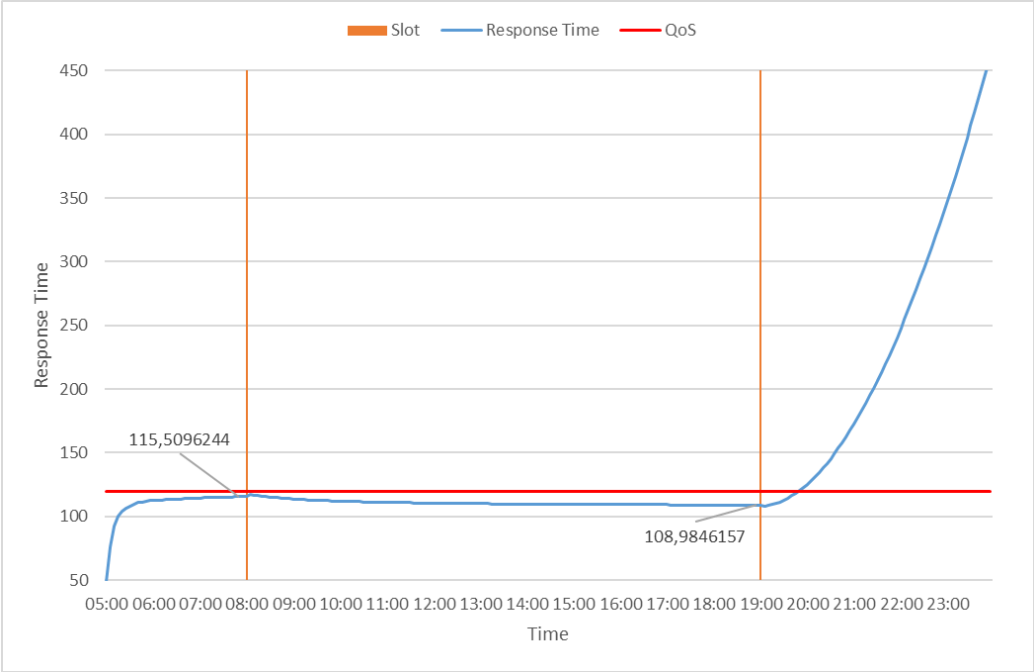
Come vediamo dai risultati nel grafico, per il tempo di risposta superiamo leggermente il QoS di 120 sec nella prima e seconda fascia, con un picco intorno alle 08:00 di 123,3116 sec. vediamo che riduciamo sensibilmente i costi rispetto allo scenario 4 ottimale, ma restiamo purtroppo sopra i tempi attesi. Infatti i tempi medi nella fascia centrale sono di 120.240624 +/- 0.483843, mentre nell'ultima fascia riusciamo a tornare sotto il QoS con tempi medi di 118.709267 +/- 0.418142.

Da questa analisi osserviamo quindi che pur incrementando il numero di server rispetto allo scenario ottimo, i 2 server rimossi dal blocco 1 portano l'utilizzazione a 0,927781 nella fascia 0 e 0,901424 nella fascia 1. Questo ci indica che il blocco per l'acquisto dei biglietti è il *collo di bottiglia* della rete, in quanto causa un degrado importante delle prestazioni all'aumentare della sua utilizzazione.

Questo risultato è ragionevole, in quanto il blocco 1 ha tempo di servizio mediamente molto più alto degli altri serventi, per cui se abbiamo un'alta utilizzazione si generano molti accodamenti che impiegheranno più tempo ad essere smaltiti, anche andando a fare *over-provisioning* negli altri blocchi.

9.2.5 Scenario 4 - {8,21,2,9,11}, {18,42,5,22,25}, {4,14,1,6,8}

In questo scenario utilizziamo una configurazione ottima per la prima fascia, over-provisioning per la fascia centrale ed invece under-provisioning per l'ultima fascia.



Come possiamo vedere dai risultati abbiamo per la prima fascia un comportamento atteso, analogo a quanto già visto nella configurazione ottima. Nella seconda fascia invece allochiamo un numero di server di molto superiore a quanto necessario, per cui notiamo un decremento del tempo di risposta medio, fino ad un minimo di 108,4737 secondi prima di cambiare fascia oraria alle 19:00.

Nelle tre fasce orarie i tempi medi di risposta sono rispettivamente:

- 115.509723 +/- 1.032371 sec
- 109.882444 +/- 0.196634 sec
- 469.626995 +/- 4.225227 sec

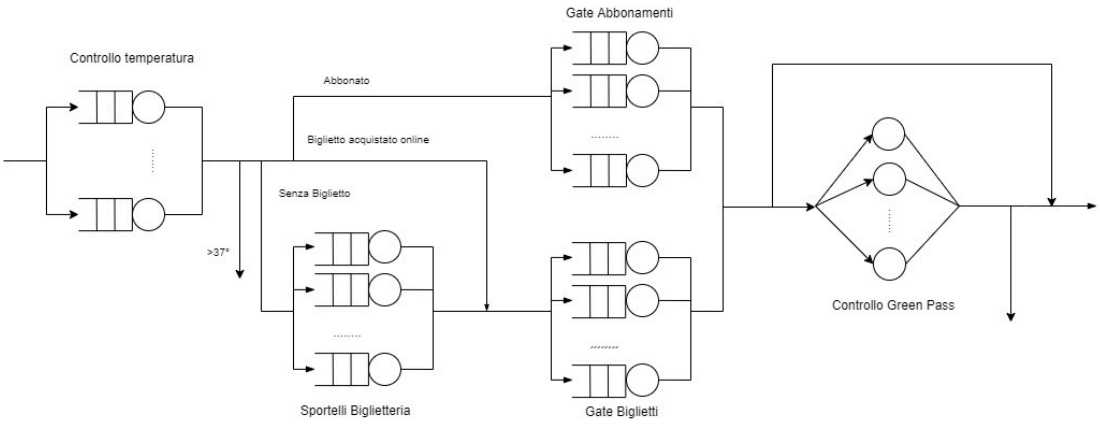
Quando cambiamo invece fascia oraria alle 19:00 vengono spenti troppi serventi nei vari blocchi, e questo causa una rapida crescita del tempo medio di risposta, che arriva a toccare i 450 secondi al termine delle 19 ore di osservazione, e quindi a violare il QoS.

10 Algoritmo Migliorativo

Come algoritmo migliorativo si propone una soluzione per bilanciare e suddividere in modo equo il traffico tra i vari serventi. Infatti la soluzione proposta in precedenza causa degli accodamenti massivi e poco organizzati presso la singola coda di ogni blocco. Questa rappresenta una soluzione sicuramente efficace a livello di modellazione, ma al contempo anche poco realistica considerando il sistema in esame.

Infatti molte stazioni ferroviarie tipicamente suddividono le code su ogni singolo sportello, soprattutto per questioni di ordine pubblico, ed in tempi più recenti per evitare assembramenti.

Come i risultati teorici dimostrano[9] con una soluzione di questo tipo non si avranno migliorie sui tempi di risposta, i quali tenderanno comunque ad aumentare a parità di configurazione rispetto all'algoritmo originale. Infatti le modifiche introdotte non sono volte a migliorare i tempi di risposta, ma agiscono in termini di bilanciamento delle code dei vari serventi; per questo è stato aggiunto un meccanismo di ridistribuzione dei job presenti in coda che avviene ad ogni cambio di fascia oraria quando si attivano o disattivano dei serventi.



Rispetto al modello concettuale originale, si aggiunge una nuova variabile di stato, ovvero il numero di job in coda sul singolo servente.

I blocchi 1-4 del sistema non sono più quindi modellati come una M/M/k ma come **k code M/M/1** su cui si distribuisce equamente il traffico in ingresso. L'obiettivo di questa analisi è dunque quello di trovare la nuova configurazione ottima che permetta di rispettare i due QoS già descritti in precedenza.

10.1 Modello Computazionale

Il modello computazionale dell'algoritmo migliorativo non differisce di molto rispetto a quello utilizzato per implementare il sistema originale.

La principale differenza riguarda la gestione delle code che viene realizzata con l'utilizzo di *liste doppiamente collegate* che tengono traccia dei job in coda in ogni singolo server del sistema.

10.1.1 Selezione del Servente destinazione

Ad ogni arrivo in un nuovo blocco, la scelta del server destinatario viene effettuata tramite una ricerca del server con meno job in coda tra tutti quelli attivi al momento. Attraverso la distribuzione `Equilikely()` di `rvgs.c` viene selezionato un server randomico dal quale iniziare la ricerca del server di destinazione.

In questo modo si riesce a raggiungere un bilanciamento del carico adeguato sui diversi server attivi. Infatti, se la ricerca avvenisse partendo sempre dal primo server attivo (o da un qualsiasi altro server), non si avrebbe una distribuzione equa dei job sui possibili server destinatari candidati, in quanto a parità di numero di job in coda la ricerca tenderebbe ad escludere gli ultimi server nella lista.

10.1.2 Load Balancing

Il bilanciamento del carico viene realizzato sfruttando le proprietà della lista doppiamente linkata di job in attesa per singolo server.

- Viene calcolato il numero di job che ogni server dovrà avere in coda una volta terminato il bilanciamento come $\frac{\#jobInCoda}{\#totalServers}$
- Iterando su ciascun server già attivo prima del bilanciamento, viene preso l'ultimo job in coda dalla sua lista doppiamente linkata, e viene distribuito sulla coda di uno dei server appena attivati dopo il cambio di fascia oraria.
- L'iterazione termina quando tutti i server avranno raggiunto il numero di job in coda calcolato precedentemente, garantendo dunque, alla fine dell'algoritmo, un bilanciamento equo delle code dei vari serventi.

10.2 Verifica

Per la verifica del sistema migliorato andiamo ad analizzare le statistiche sui singoli server, in quanto utilizziamo delle code singole. Analizziamo in particolare le statistiche con una configurazione di *under-provisioning*, dopo il cambio di fascia delle ore 19:00 (simulazione di 50400 secondi).

Per fare questo abbiamo verificato 3 condizioni:

A. Il tempo di risposta (`wait`) per un server è sempre uguale alla somma del tempo di attesa (`delay`) e del tempo di servizio (`service`).

B. Il numero di arrivi sul blocco è sempre uguale al numero di job in coda più il numero di job completati

C. Il numero di job in coda per i server di uno stesso blocco sono gli stessi. Questo perché misurando le statistiche dopo il oraria cambio di fascia è già avvenuto il bilanciamento del carico.

TEMPERATURE_CTRL server 0	TEMPERATURE_CTRL server 1	TEMPERATURE_CTRL server 2
Arrivals ..... = 5051	Arrivals ..... = 4829	Arrivals ..... = 4866
Completions..... = 3492	Completions..... = 3272	Completions..... = 3307
Job in Queue ..... = 1558	Job in Queue ..... = 1556	Job in Queue ..... = 1558
Server status..... = BUSY ONLINE	Server status..... = BUSY ONLINE	Server status..... = BUSY ONLINE
Average wait ..... = 6601.744842	Average wait ..... = 6905.388910	Average wait ..... = 6852.777123
Average delay .... = 6586.768759	Average delay .... = 6889.952976	Average delay .... = 6837.419576
Average service... = 14.976083	Average service... = 15.435934	Average service... = 15.357548

Vediamo che sono verificate tutte le condizioni. Per la condizione A, considerando ad esempio il server 0: 6601.744842 = 6586.768759 + 14.976083. Per la condizione B vediamo sul server 1 che 4829 = 3272 + 1556 (+ 1 job in servizio in quanto il server è `BUSY`). Per la condizione C vediamo che tutti hanno circa lo stesso numero di job in coda (1558,1556,1558).

TICKET_BUY server 6	TICKET_BUY server 7	TICKET_BUY server 8
Arrivals ..... = 595	Arrivals ..... = 622	Arrivals ..... = 601
Completions..... = 520	Completions..... = 547	Completions..... = 525
Job in Queue ..... = 74	Job in Queue ..... = 74	Job in Queue ..... = 75
Server status..... = BUSY ONLINE	Server status..... = BUSY ONLINE	Server status..... = BUSY ONLINE
Average wait ..... = 2693.972320	Average wait ..... = 2577.792454	Average wait ..... = 2666.168353
Average delay .... = 2603.815659	Average delay .... = 2488.588644	Average delay .... = 2576.700269
Average service... = 90.156661	Average service... = 89.203810	Average service... = 89.468084

Vediamo che anche per il secondo blocco sono verificate tutte le condizioni. Per la condizione A sul server 8: 2666.168353 = 2576.700269 + 89.468084. Per la condizione B vediamo sul server 6 che 595 = 520 + 74 (+ 1 job in servizio in quanto il server è `BUSY`). Per la condizione C vediamo che tutti hanno circa lo stesso numero di job in coda (74,74,75).

10.3 Validazione

Per validare il sistema verifichiamo che sono rispettate le seguenti condizioni:

- Leggi di Little
  - $E(N_S) = \lambda E(T_S)$
  - $E(N_Q) = \lambda E(T_Q)$
- Le utilizzazioni dei serventi sono uguali, e quindi che il meccanismo di load balancing introdotto funzioni come in un sistema reale.

TEMPERATURE_CTRL server 0 Arrivals ..... = 5051 Completions..... = 3492 Job in Queue ..... = 1558 Server status..... = BUSY ONLINE Average wait ..... = 6601.744842 Average delay .... = 6586.768759 Average service... = 14.976083 Average # queue .. = 660.111392 Average # node ... = 661.611173 Utilization ..... = 0.999781	TEMPERATURE_CTRL server 1 Arrivals ..... = 4829 Completions..... = 3272 Job in Queue ..... = 1556 Server status..... = BUSY ONLINE Average wait ..... = 6905.388910 Average delay .... = 6889.952976 Average service... = 15.435934 Average # queue .. = 660.147383 Average # node ... = 661.547281 Utilization ..... = 0.999899	TEMPERATURE_CTRL server 2 Arrivals ..... = 4866 Completions..... = 3307 Job in Queue ..... = 1558 Server status..... = BUSY ONLINE Average wait ..... = 6852.777123 Average delay .... = 6837.419576 Average service... = 15.357548 Average # queue .. = 660.136478 Average # node ... = 661.618900 Utilization ..... = 0.999992

Valgono le leggi di Little, ad esempio sul server 2, abbiamo λ=4866/50400=0.0965476, quindi:

- $E(N_Q)$=660.136478=6837.419576 * 0.0965476.
- $E(N_S)$=661.133500=6852.777123* 0.0965476.

Le utilizzazioni sono effettivamente bilanciate (0.999781, 0.999899, 0.999992), ma essendo il sistema saturo andiamo a valutare anche un ulteriore scenario più significativo

TICKET_GATE server 3 Arrivals ..... = 1756 Completions..... = 1756 Job in Queue ..... = 0 Server status..... = IDLE ONLINE ... Utilization ..... = 0.857013	TICKET_GATE server 4 Arrivals ..... = 1807 Completions..... = 1807 Job in Queue ..... = 0 Server status..... = IDLE ONLINE ... Utilization ..... = 0.859608	TICKET_GATE server 5 Arrivals ..... = 1730 Completions..... = 1728 Job in Queue ..... = 1 Server status..... = BUSY ONLINE ... Utilization ..... = 0.861908

Considerando il blocco `TICKET_GATE`, vediamo che tutti i server hanno effettivamente la stessa utilizzazione di circa 0.86.

Abbiamo quindi confermato che il load balancing funziona sia per sistemi stabili che instabili, per cui la miglioria introdotta presenta un comportamento coerente con un quello di un sistema reale.

10.4 Analisi

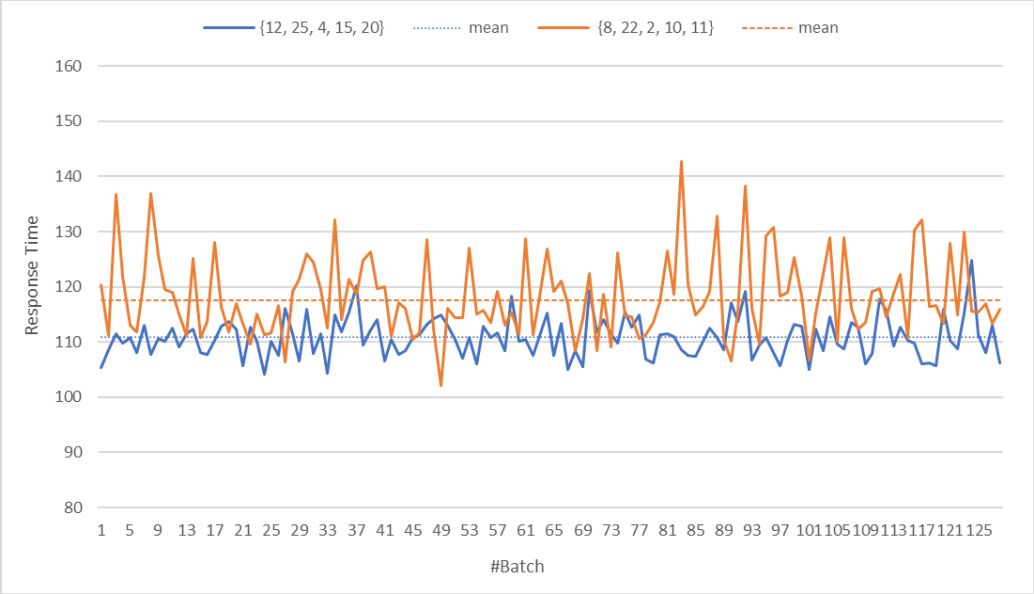
10.4.1 Analisi ad Orizzonte Infinito

Come effettuato in precedenza, tramite la simulazione ad orizzonte infinito cerchiamo la configurazione ottima verificando che i tempi di risposta del modello siano conformi a quelli analitici.

Per ogni configurazione testata si valuta quindi se rispetta il QoS del tempo di risposta e se il numero di passeggeri cui viene controllato il green pass è almeno del 70% (P_{bypass}<0.30). Oltre a tali vincoli si cerca di ottimizzare anche il costo della configurazione ottima.

Fascia 05:00 - 08:00

Configurazione {12, 25, 4, 15, 20} vs Configurazione {8, 22, 2, 10, 11}



Simulando il sistema con la configurazione {12, 25, 4, 15, 20} (in blu) notiamo dai risultati che vengono rispettati entrambi i QoS stabiliti:

- $E(T_{S,TOT})$: 110.08 +/- 0.63
- P_{bypass}: 0.009403 < 0.30

Come possiamo vedere, la probabilità che ad un passeggero non venga controllato il Green Pass è di molto sotto la soglia fissata dal QoS (0.30). Ciò indica che il numero di serventi dedicato al controllo del Green Pass è sovradimensionato e si possono dunque ridimensionare.

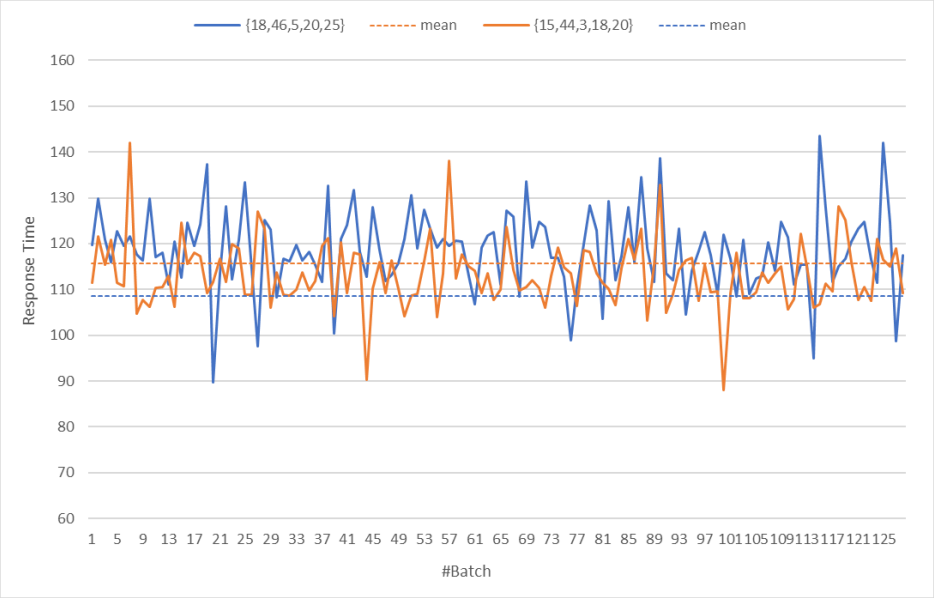
Inoltre, osservando le utilizzazioni riportate qui in basso, notiamo come anche i serventi degli altri blocchi siano sovradimensionati, portando dunque ad un costo spropositato.

```
Mean Utilization for block TEMPERATURE_CTRL......0.505720
Mean Utilization for block TICKET_BUY............0.697064
Mean Utilization for block SEASON_GATE...........0.518329
Mean Utilization for block TICKET_GATE...........0.258470
Mean Utilization for block GREEN_PASS............0.500461
Loss Percentage for block GREEN_PASS.............0.009403
------------------------------------------------------------
TOTAL SLOT 0 CONFIGURATION COST..................6962.26€
```

Possiamo dunque passare alla configurazione minima {8, 22, 2, 10, 11}, con la quale il sistema è in grado di rispettare i QoS e portare il costo giornaliero per questa fascia lavorativa a 4508.39€, con un tempo di risposta medio pari a 119.01 +/- 1.70 ed una P_{Bypass}=0.233678 (<30).

Fascia 08:00 - 19:00

Configurazione {18, 46, 5, 20, 25} vs Configurazione {15, 44, 3, 18, 20}



Un approccio simile all'analisi precedente ci permette di trovare la configurazione ottima applicabile alle 11 ore lavorative della fascia centrale.

Con la configurazione riportata in blu nel grafico {18, 46, 5, 20, 25}, otteniamo un tempo di risposta medio del sistema pari a 108.61 +/- 0.8. Tuttavia, a causa del grande numero di serventi istanziati, otteniamo un costo eccessivo, pari a 4675.28€.

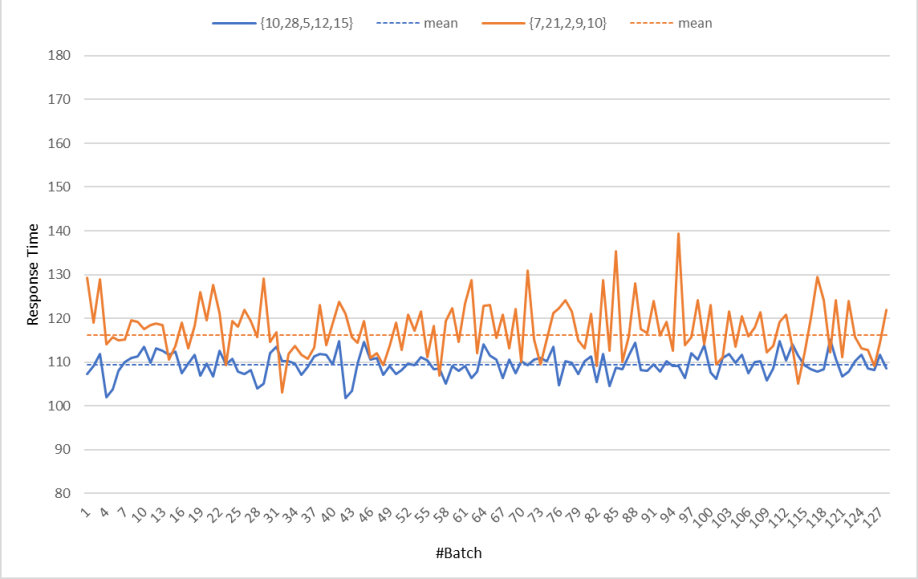
```
Mean Utilization for block TEMPERATURE_CTRL......0.684880
Mean Utilization for block TICKET_BUY............0.780093
Mean Utilization for block SEASON_GATE...........0.417564
Mean Utilization for block TICKET_GATE...........0.762185
Mean Utilization for block GREEN_PASS............0.865366
Loss Percentage for block GREEN_PASS.............0.116704
------------------------------------------------------------
TOTAL SLOT 1 CONFIGURATION COST..................4675.28€
```

Tramite l'analisi delle utilizzazioni riusciamo ad arrivare ad una configurazione con il numero minimo di server possibile, in modo tale da rispettare i QoS dettati dal sistema. Con la configurazione {15, 44, 3, 18, 20} otteniamo un tempo di risposta medio pari a 118.12 +/- 1.71, una P_{Bypass}=0.232280 ed una riduzione del costo a 3936.63€.

```
Mean Utilization for block TEMPERATURE_CTRL......0.878861
Mean Utilization for block TICKET_BUY............0.831733
Mean Utilization for block SEASON_GATE...........0.705111
Mean Utilization for block TICKET_GATE...........0.896536
Mean Utilization for block GREEN_PASS............0.936785
Loss Percentage for block GREEN_PASS.............0.232280
------------------------------------------------------------
TOTAL SLOT 1 CONFIGURATION COST..................3936.63
```

Fascia 19:00 - 00:00

Configurazione {10, 28, 5, 12, 15} vs Configurazione {7, 21, 2, 9, 10}



Nella restante fascia lavorativa, utilizzando lo stesso approccio applicato per le precedenti, andiamo a sovradimensionare la configurazione, {10, 28, 5, 12, 15}, ottenendo un tempo di risposta medio di 109.47 +/- 0.46 ed una P_{Bypass} molto bassa pari a 0.051528, ottenendo le seguenti utilizzazioni medie dei blocchi:

```
Mean Utilization for block TEMPERATURE_CTRL......0.547141
Mean Utilization for block TICKET_BUY............0.568179
Mean Utilization for block SEASON_GATE...........0.183530
Mean Utilization for block TICKET_GATE...........0.563377
Mean Utilization for block GREEN_PASS............0.685990
Loss Percentage for block GREEN_PASS.............0.051528
------------------------------------------------------------
TOTAL SLOT 2 CONFIGURATION COST..................6383.66€
```

Per diminuire i costi bisogna ridurre il numero di server assegnati ad ogni blocco, aumentando dunque le loro utilizzazioni. In particolare l'utilizzazione relativa ai server del blocco `SEASON_GATE` è estremamente bassa, 0.183530. Considerato infatti il flusso di accesso a tale blocco e la sua velocità nel servire i job, possiamo ridurlo a 2 server. Similmente è possibile aumentare le utilizzazioni degli altri blocchi fino ad arrivare alla seguente configurazione, {7, 21, 2, 9, 10}, che ci permette di raggiungere un tempo medio di risposta di 119.37 +/- 1.18 ed una P_{Bypass}= 0.235890, rispettando dunque i QoS stabiliti.

```
Mean Utilization for block TEMPERATURE_CTRL......0.771080
Mean Utilization for block TICKET_BUY............0.795777
Mean Utilization for block SEASON_GATE...........0.462275
Mean Utilization for block TICKET_GATE...........0.747286
Mean Utilization for block GREEN_PASS............0.827161
Loss Percentage for block GREEN_PASS.............0.235890
------------------------------------------------------------
TOTAL SLOT 2 CONFIGURATION COST..................4549.66€
```

10.4.2 Analisi ad Orizzonte Finito

Tramite l'analisi ad orizzonte infinito appena descritta, è stata individuata una configurazione server per ogni fascia:

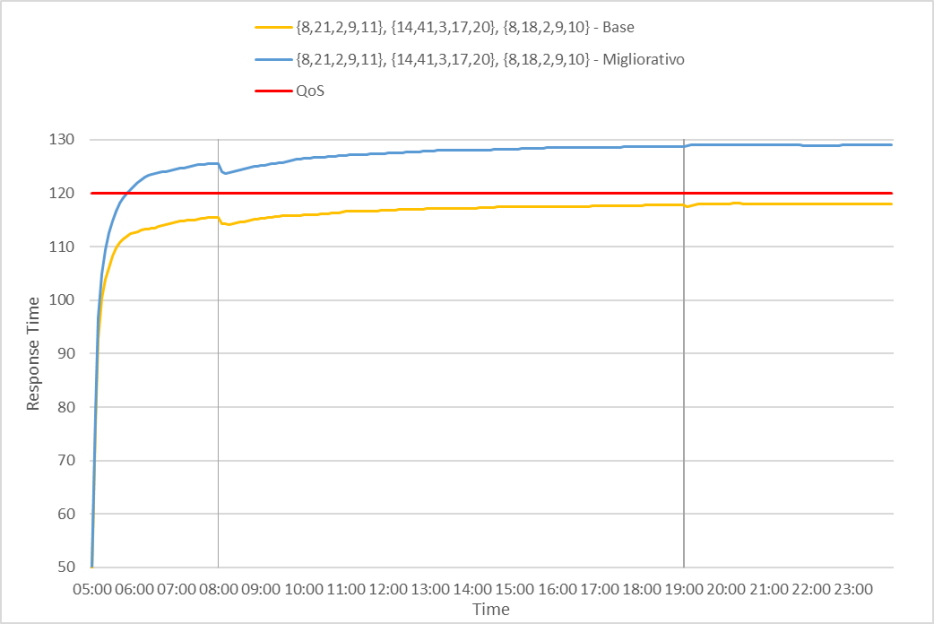
```
{8,22,2,10,11}, {15,44,3,18,20}, {7,21,2,9,10}
```

Andiamo quindi a considerare due scenari di simulazione *finite-horizon*: nel primo scenario utilizziamo la configurazione ottima vista nel sistema originale, mentre nel secondo scenario analizziamo l'ottimo individuato per il sistema migliorato. Inoltre andiamo anche a valutare le utilizzazioni dei singoli server, confrontandole a parità di configurazione con quelle per il sistema base.

Confronto configurazioni ottime

Nel grafico riportato di seguito è stata effettuata un'analisi ad orizzonte finito per verificare che la configurazione ottima trovata per il modello di sistema originale, se utilizzata nel modello migliorativo appena presentato, porta ad un incremento dei tempi di risposta del sistema, come ci si aspetta dai risultati teorici.

La configurazione analizzata è quindi quella ottima per il sistema base, ovvero {8,21,2,9,11}, {14,41,3,17,20}, {8,18,2,9,10}

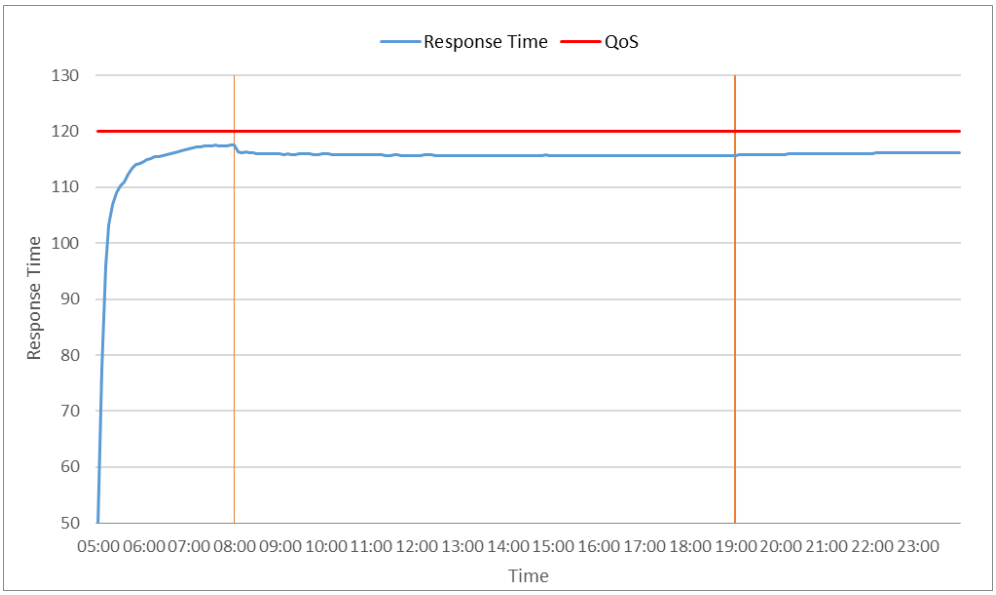


Possiamo vedere come i tempi registrati dall'algoritmo migliorativo (in blu) siano nettamente superiori rispetto all'algoritmo originale che implementa una singola coda di attesa per ogni blocco di servizio, superando inoltre il tempo minimo di 120s dettato dal QoS.

Questo ci suggerisce quindi (come previsto) che è necessario analizzare una nuova configurazione ottima per il sistema migliorato.

Configurazione Ottima - {8,22,2,10,11}, {15,44,3,18,20}, {7,21,2,9,10}

Attraverso un'analisi ad orizzonte finito andiamo quindi a verificare che effettivamente la configurazione ottima riesca a minimizzare i costi ed i tempi di risposta, rispettando i QoS, valutando inoltre i costi effettivi della soluzione proposta.



I tempi di risposta medi ottenuti nelle singole fasce lavorative sono rispettivamente pari a 117.567176 +/- 0.718870, 115.590476 +/- 0.198434 e 116.205546 +/- 0.180245. Tali risultati confermano il lavoro svolto nell'analisi ad orizzonte infinito che ha prodotto la configurazione ottima.

Analizzando l'andamento della curva possiamo notare come al cambio della prima fascia oraria ci sia un iniziale abbassamento del tempo di risposta medio dovuto all'aggiunta di nuovi server i quali, dopo aver effettuato un bilanciamento delle code, prendono in servizio job rimasti in attesa nelle code dei server attivi già nella prima fascia lavorativa.

Il QoS relativo alla percentuale di passeggeri che vengono sottoposti al controllo Green Pass viene rispettato in quanto otteniamo valori inferiori al 30%:

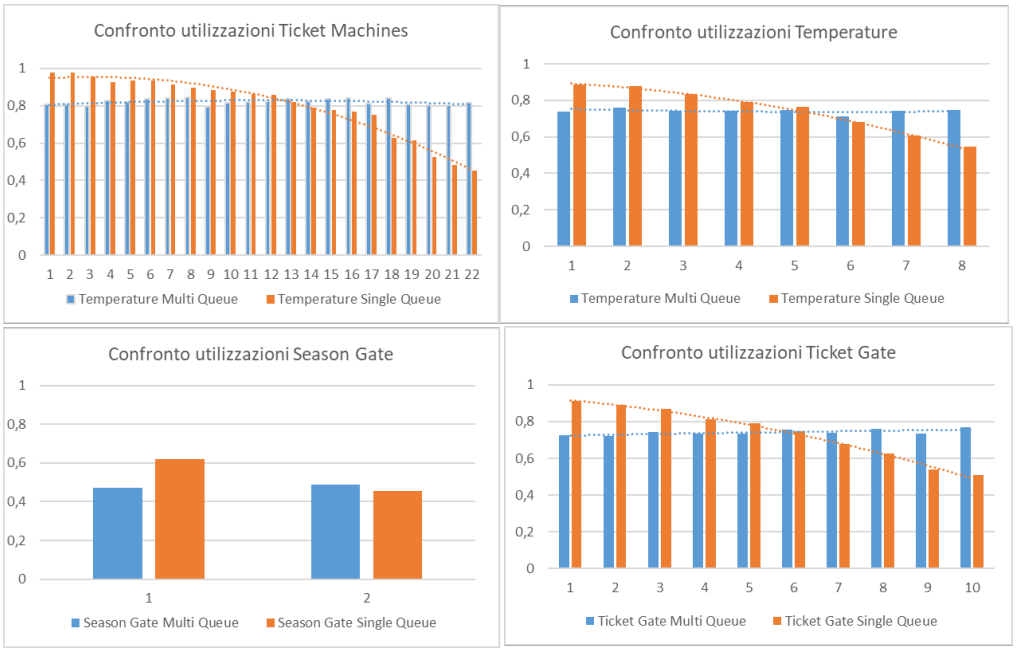
- Fascia 05:00 - 08:00 = 0.229954 +/- 0.001967
- Fascia 08:00 - 19:00 = 0.259645 +/- 0.004582
- Fascia 19:00 - 00:00 = 0.260081 +/- 0.003742

Il costo totale della soluzione ottima per il sistema migliorato è pari a 1527.39€. Il costo è come previsto maggiore rispetto a 1347.26€ del caso base.

10.4.3 Confronto utilizzazioni

Bilanciando il numero di job in coda al cambio di ogni fascia oraria, le utilizzazioni dei singoli serventi risultano bilanciate durante l'intera fascia lavorativa.

Tramite un'analisi ad orizzonte finito su una singola fascia, sono state messe a confronto le utilizzazioni del sistema originale con quelle ottenute dall'algoritmo migliorativo.



È facile vedere dai grafici come le utilizzazioni dei serventi nel sistema a coda unica (in arancione) non siano equamente distribuite, e si nota un andamento per il quale i primi serventi del blocco hanno sempre una maggiore utilizzazione a causa della politica di selezione adottata.

- Questo porta ad una gestione poco fair del sistema, a discapito dei primi serventi dei vari blocchi.

Al contrario nel sistema migliorato, come già discusso nella validazione, si hanno dei valori bilanciati su tutti i serventi allocati in un blocco in ogni istante di tempo.

11 Conclusioni

Per il sistema originale la configurazione ottima è {8,21,2,9,11}, {14,41,3,17,20}, {8,18,2,9,10} che presenta un costo totale di 1347.26€ e tempi di risposta medi nelle singole fasce rispettivamente di 115.528435 +/- 0.814483, 117.856171 +/- 0.364914, 117.976557 +/- 0.327956. Anche i QoS sul numero di passeggeri cui viene controllato il green pass sono verificati in ogni fase.

- Fascia 05:00 – 08:00: 0.247849 +/- 0.002984
- Fascia 08:00 – 19:00: 0.272511 +/- 0.006007
- Fascia 19:00 – 24:00: 0.271137 +/- 0.008372

Per il sistema migliorato la configurazione ottima è {8,22,2,10,11}, {15,44,3,18,20}, {7,21,2,9,10} che presenta un costo di 1527.39€. Anche in questo caso rispettiamo tutti i QoS sulla P_{Bypass}.

- Fascia 05:00 - 08:00 = 0.229954 +/- 0.001967
- Fascia 08:00 - 19:00 = 0.259645 +/- 0.004582
- Fascia 19:00 - 00:00 = 0.260081 +/- 0.003742
- 117.567176 +/- 0.718870, 115.590476 +/- 0.198434 e 116.205546 +/- 0.180245.

La soluzione migliorativa presenta un costo maggiore per rispettare i QoS considerati. Questo è ragionevole in quanto si va a gestire un caso più vicino al sistema reale, ed il modello proposto, come già discusso, ha tempi di risposta intrinsecamente maggiori.

Il motivo del degrado delle prestazioni è dovuto al fatto che una volta selezionata la coda il job non può essere migrato su un altro servente, anche se quest'ultimo diventa `IDLE`; il bilanciamento delle code avviene soltanto al cambio della fascia oraria, quando viene modificata la configurazione dei serventi. Al contrario nel sistema base, avendo un'unica coda, tutti i job vengono serviti il prima possibile. Per ottimizzare ulteriormente il sistema migliorato avremmo potuto effettuare un load balancing in maniera continuativa ad ogni arrivo/completamento, in modo da mitigare il problema descritto. Tuttavia tale soluzione non rappresenta il caso reale per due motivi:

- Lo spostamento di un passeggero da una coda ad un'altra introduce un certo delay dovuto alla distanza da percorrere.
- È poco pratico pensare di poter spostare ad ogni nuovo arrivo un numero considerevole di passeggeri tra le diverse code.

Per quanto riguarda invece i tempi di risposta, un modello che avrebbe portato sicuramente dei miglioramenti è quello in cui si utilizza per ogni blocco un unico servente **M/M/1** con *capacità aggregata*, pari a **m*u**.

Tuttavia tale modello non risulta realistico rispetto allo scenario che si cerca di modellare; ad esempio non è possibile avere un unico gate dal tempo di servizio infinitesimo, in quanto comunque ci sono dei tempi minimi di processamento per l'attraversamento fisico. Così come non è possibile avere una singola macchina per la

stampa di biglietti che fornisca un ticket in pochissimi secondi, dato che bisogna considerare oltre all'efficienza meccanica della macchina, anche i tempi impiegati dal passeggero per inserire i dati, selezionare il biglietto e pagare.

Un'altra possibile modifica considerata è quella di modelli basati su priority. Anche in questo caso la miglioria non risulta coerente con il caso in esame, in quanto difficilmente sono previsti agevolazioni riservate ad utenti di classi differenti nei vari controlli per l'accesso ai binari.

Considerati tutti questi aspetti è stato scelto di proporre un modello più realistico che implementi un blocco composto da k code M/M/1, pur presentando costi maggiori per mantenere lo stesso livello di prestazioni.

12 Bibliografia

[1] A. M. Nair, S. K.S and P. V. Ushakumari, "Application of Queuing Theory to a Railway ticket window," 2021 International Conference on Innovative Practices in Technology and Management (ICIPTM), 2021, pp. 154-158, doi: 10.1109/ICIPTM52218.2021.9388368.
(https://ieeexplore.ieee.org/document/9388368)

[2] Özgür Yalçınkaya, G. Mirac Bayhan, Modelling and optimization of average travel time for a metro line by simulation and response surface methodology, European Journal of Operational Research, Volume 196, Issue 1, 2009, pp. 221. (https://www.sciencedirect.com/science/article/pii/S0377221708002853)

[3] https://dataportal.orr.gov.uk/media/2024/station-usage-2020-21-statistical-release.pdf

[4] https://dataportal.orr.gov.uk/media/2041/passenger-rail-usage-2021-22-q2.pdf

[5] https://www.statista.com/statistics/675663/rail-tickets-booked-online-united-kingdom-uk/

[6] https://iltuosalario.it/carriera/italia-professioni-e-stipendi/italia-controllori-e-bigliettai-di-trasporti-pubblici#:~:text=Un%20Controllori%20e%20bigliettai%20di%20trasporti%20pubblici%20percepisce%20generalmente%20tra,settimana%20lavorativa%20di%2040%20ore.

[7] Hill, R. Discrete-Event Simulation: A First Course. J Simulation 1, 371 (2004)

[8] Hill, R. Discrete-Event Simulation: A First Course. J Simulation 1, 377 (2004)

[9] Hill, R. Discrete-Event Simulation: A First Course. J Simulation 1, 263 (2004)