Progetto di Performance Modeling of Computer Systems and Networks

Anno accademico 2021-2022

Adrian Petru Baba (0320578)
Sara Da Canal (0316044)
Matteo Federico (0321569)

Descrizione del sistema

Il sistema considerato è un pronto soccorso con tre diversi reparti per diversi tipi di cure: traumatologia, primo intervento e problemi di minore entità. In più, casi molto gravi vengono trattati separatamente. All'arrivo delle persone al pronto soccorso, passano l'accettazione, dove gli viene assegnato un codice in base alla gravità e vengono indirizzate verso il giusto reparto. I possibili codici sono quattro, rosso per i casi più gravi, e poi a scendere giallo, verde e bianco. I casi in codice rosso sono quelli trattati separatamente, i casi gialli e verdi vengono divisi tra i reparti in base al problema, mentre i casi in codice bianco vengono soltanto indirizzati verso i problemi di minore entità. I casi rossi e gialli hanno una probabilità di morte, maggiore per i casi rossi e minore per i gialli, che si alza al variare di quanto devono attendere.

Obiettivi

Il nostro obiettivo è modellare il sistema trovando la distribuzione ottima di serventi per garantire che non vengano superati i seguenti tempi di attesa:

- I codici rossi dovrebbero essere presi in carico immediatamente
- I codici gialli l'attesa dovrebbe essere di non più di 30 minuti
- I codici verdi di 60 minuti
- I codici bianchi di 120 minuti

A questo va aggiunto che l'accettazione non dovrebbe superare i quindici minuti totali tra attesa e servizio.

Per avvicinarci il più possibile a questi tempi abbiamo un budget limitato. I diversi lavoratori hanno stipendi differenti in base al centro in cui lavorano, dato che ricoprono mansioni differenti:

- I medici responsabili di trattare i codici rossi costano 7.000€/mese
- I medici che si occupano di traumatologia e primo intervento costano 4.200€/mese
- I problemi di minore entità possono essere gestiti da infermieri con costo 2.800€/mese
- I lavoratori dell'accettazione costano 1.700€/mese

Il budget che consideriamo è di 65.000 €/mese.

Oltre al budget, anche un altro vincolo va rispettato, ovvero mantenere il numero di morti al minimo possibile, l'ideale sarebbe sotto lo 0.5%.

Successivamente si cambia il modello per separare i codici gialli in due code diverse, codici arancioni e blu, e si vede se il sistema migliora e se è possibile diminuire il numero di server senza aumentare la probabilità di morte. I codici arancioni dovrebbero avere attesa di 15 minuti e blu di 45 minuti.

Modello concettuale

Stato:

Per ogni istante di tempo t

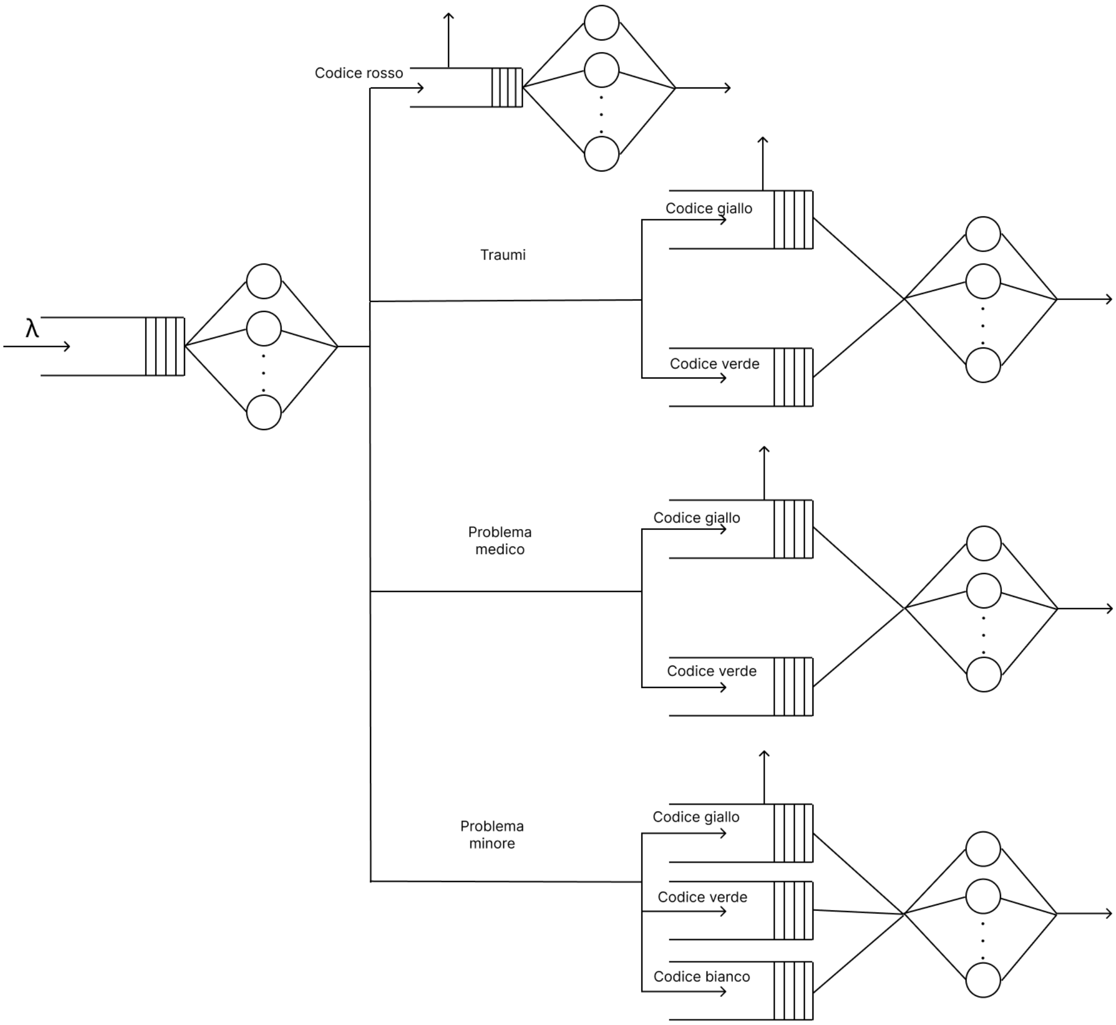
- Numero di persone nel sistema per ogni nodo divise per coda
- Numero di persone in servizio per ogni nodo, anche queste divise per codice di provenienza
- Stato di ogni server, se occupato o meno

Assunzioni:

- Non preemptive
- Conservativo (se un job è in attesa e il servente è libero, il servente esegue subito il job)

- Sistema stazionario

Il sistema può essere modellato nel seguente modo:



Ogni coda è FIFO e i tempi di arrivo e servizio sono tutti esponenziali, abbiamo quindi tutti server M/M/m, alcuni a coda singola altri con una multi-coda a priorità astratta. Il primo server dove passa ogni job è il triage, e poi i job vengono divisi. Dalla coda del codice rosso e dalle code gialle ci potrebbero essere abbandoni che non entrano in servizio, a causa della probabilità di morte.

TRIAGE

Le variabili di stato sono l'occupazione dei server e il numero di job nel sistema, triageNumber. Il numero di job in servizio, avendo un solo tipo di job, si può trovare come il numero di server del sistema se triageNumber è maggiore dei server totali e triageNumber se invece è minore.

CODICI ROSSI

Il funzionamento per i codici rossi è uguale a quello del triage, l'unica variabile di stato è redNumber, e i job in servizio possono essere calcolati usando il numero totale di server per questo nodo analogamente a quanto detto prima.

TRAUMI

In questo caso le variabili di stato sono quattro, due, traumaYellowNumber e traumaGreenNumber, rappresentano i job nel sistema con codice giallo o verde, le altre due, traumaInServiceYellow e traumaInServiceGreen rappresentano invece i job in servizio per ogni codice.

PROBLEMI MEDICI
Anche in questa situazione abbiamo quattro variabili di stato analoghe a quelle di traumatologia, medicalYellowNumber e medicalGreenNumber per i job nel sistema; medicalInServiceYellow e medicalInServiceGreen per i job in servizio.

PROBLEMI MINORI
Per questo nodo abbiamo sei variabili di stato, che rappresentano i job nel sistema e in servizio per ognuna delle tre code. Le variabili del sistema sono minorYellowNumber, minorGreenNumber e minorWhiteNumber, le variabili del servizio minorInServiceYellow, minorInServiceGreen e minorInServiceWhite.

Modello delle specifiche

I parametri di input necessari al nostro simulatore sono gli arrivi medi, le probabilità di finire in ogni coda, il tempo di servizio medio per ogni nodo e la probabilità che ci siano decessi nella coda dei codici rossi. Abbiamo trovato dei dati molto puntuali per quanto riguarda gli accessi e le prestazioni effettuate per tutti i pronti soccorsi della regione Veneto relativi all'anno 2013, e abbiamo deciso di selezionare un pronto soccorso tra quelli presenti e basarci sui dati relativi ad esso per quanto possibile. Il pronto soccorso selezionato è stato l'ospedale Borgo Roma di Verona, con 49.600 accessi annuali. La scelta è ricaduta su questo ospedale poiché il numero di accessi è molto vicino alla media tra tutti gli ospedali, ed è quindi sembrata una buona scelta per il nostro sistema che simula un pronto soccorso di medie dimensioni.
Dagli accessi abbiamo ottenuto il primo parametro di input, il tasso di arrivo medio λ, pari a 0,09 job/min.
Tra i dati trovati erano presenti le probabilità di assegnazione per ogni codice. Nei dati reali un 3,8% dei casi non ha avuto un codice assegnato oppure l'informazione non è disponibile. Abbiamo deciso di ridistribuire questo 3,8% tra gli altri codici in modo proporzionale, calcolando ogni percentuale con la formula 3,8*percentuale originaria/(100-3,8). Le probabilità ottenute sono:

- Codice rosso: 1,04%
- Codice giallo: 18,40%
- Codice verde: 60,71%
- Codice bianco: 19,85%

Per quanto riguarda le probabilità dei codici rossi e bianchi non sono necessarie altre suddivisioni, mentre i codici gialli e verdi hanno bisogno di un'ulteriore suddivisione per essere divisi nei tre reparti dei traumi, problemi medici e problemi di entità minore. I dati trovati contenevano le probabilità divise in cinque reparti, traumi, problemi medici, intossicazioni, assistenza medico legale e problemi minori, le probabilità di problemi medici, intossicazioni e assistenza medico legale sono state aggregate per ottenere solo tre reparti, dato che le probabilità per quanto riguarda le intossicazioni e l'assistenza medico legale sono abbastanza basse e dedicargli un nodo apposito avrebbe prodotto dei nodi con utilizzazione molto bassa e privi di coda, oltretutto non sono presenti informazioni riguardo ai tempi di servizio per questi reparti, al contrario degli altri. I valori ottenuti sono i seguenti: il 26,7% dei codici gialli e verdi finiscono in traumatologia, il 24,7% hanno dei problemi medici e il 48,6% hanno problemi minori. Usando queste probabilità abbiamo ottenuto le probabilità per ogni coda:

- Coda dei codici rossi: 1,04%
- Coda dei codici gialli in traumatologia: 4,98%
- Coda dei codici verdi in traumatologia: 16,03%
- Coda dei codici gialli con problemi medici: 4,53%
- Coda dei codici verdi con problemi medici: 14,95%
- Coda dei codici gialli con problemi minori: 8,93%
- Coda dei codici verdi con problemi minori: 29,46%
- Coda dei codici bianchi: 19,85%

L'unica probabilità mancante a questo punto è quella di avere decessi tra i codici rossi, che è del 5,2%, e dei decessi tra i codici gialli. Non abbiamo trovato informazioni al riguardo, quindi è stato necessario inventare questo dato. Abbiamo selezionato una probabilità di circa il 2%.

Per quanto riguarda i tempi di servizio, abbiamo i tempi di servizio per i singoli serventi e non per nodo. L'unico tempo di servizio mancante è quello del triage, che abbiamo supposto di dieci minuti. Per quanto riguarda gli altri valori, che sono stati ricavati dai dati presenti, abbiamo un tempo di 105,6 minuti per i problemi minori, 93,4 minuti per traumatologia, 165,9 minuti per i problemi medici e 225,5 minuti per i codici rossi.

Per quanto riguarda invece la simulazione è stata pensata come una Next Event Simulation, quindi è necessario stabilire i diversi eventi da usare per l'avanzamento del clock. Gli eventi stabiliti sono stati i seguenti:

- Arrivo
- Completamento del triage
- Completamento di traumatologia
- Completamento del centro per i problemi medici
- Completamento del centro per problemi minori
- Completamento nel centro che gestisce i casi più gravi

L'ultimo possibile evento sarebbe stato una morte, ma abbiamo deciso di vederlo invece come una probabilità di perdita, quindi, nel momento in cui un evento viene indirizzato verso la coda dei codici rossi o gialli, se non entra in servizio immediatamente ha una probabilità di uscire prima di arrivare al servente, facendo si che gli arrivi effettivi siano minori del λ.

L'evento di arrivo è uno dei più facili a livello algoritmico:

- Inserimento di un nuovo job nel triage
- Generazione di un nuovo arrivo
- Se sono presenti server liberi

    Cercare il primo server disponibile

    Impostare lo stato del server ad occupato e generare un tempo di completamento per quel job

Il completamento del triage è invece un evento molto più complicato, dato che è l'evento responsabile di distribuire i job nei vari nodi della rete:

- Svuotamento di un server e diminuzione dei job nel centro
- Se sono presenti job in coda inserimento di un job del server appena liberato con quindi generazione di un nuovo tempo di completamento
- Assegnazione di un codice al job terminato
- In base al codice assegnato c'è una diversa gestione:

    CODICE ROSSO

    Inserimento di un nuovo job nel nodo dei codici rossi

Se è presente un server libero
occupazione del server e generazione del tempo di completamento
Altrimenti
generazione di una percentuale: se ci troviamo al di sopra della probabilità di morte inserimento del job in coda, se siamo al di sotto c'è un decesso prima che il job sia preso in carico
CODICE GIALLO
Scelta del nodo che se ne occuperà, tra i tre possibili
Inserimento di un nuovo job nel nodo selezionato, nella coda a priorità maggiore
Se è presente un server libero nel nodo
inserimento del job in servizio e generazione di un tempo di completamento per quel job
Altrimenti
generazione di una percentuale: se ci troviamo al di sopra della probabilità di morte inserimento del job in coda, se siamo al di sotto c'è un decesso prima che il job sia preso in carico
CODICE VERDE
Gestione identica a quella dei codici gialli, ma il job viene inserito nella coda a probabilità minore in traumatologia o per problemi medici, nella coda a priorità intermedia per i problemi minori
CODICI BIANCHI
Inserimento di un nuovo job nel nodo dei problemi minori, nella coda a priorità minore
Se è presente un server libero, occupazione del server e generazione del tempo di completamento per il job entrato in servizio

Gli eventi di completamento in traumatologia e problemi medici hanno una gestione identica tra di loro a meno delle variabili usate, dato che presentano le stesse code:

- Svuotare il server
- Controllare la presenza di job nella coda dei codici gialli
- Se presenti
  Inserimento di un job dalla coda nel server appena svuotato e generazione del tempo di completamento successivo
- Altrimenti
  Controllare la presenza di un job nella coda dei codici verdi
  Se presenti
  Inserimento di un job dalla coda nel server appena svuotato e generazione del tempo di completamento successivo

Il completamento nel centro dei problemi minori prende lo stesso algoritmo usato per traumatologia e problemi medici, ma lo allunga, perché, nel caso anche la coda dei codici verdi sia vuota, invece che lasciare il centro vuoto va a controllare la presenza di codici bianchi e se sono presenti li inserisce nel centro

Il completamento nel centro dei codici rossi è nuovamente molto facile, non avendo code di priorità:

- Svuotamento del server che ha terminato
- Controllare se ci sono job in coda
- Se presenti inserirli nel server appena liberato

Con la gestione di questi eventi è possibile simulare l'intero sistema. Il clock funzionerà nel seguente modo: sarà inizializzato ad un certo valore di START, e saranno inizializzati gli eventi, usando infinito per gli eventi impossibili: all'inizio l'unico evento possibile è un arrivo, dato che il sistema è vuoto. Ogni volta, si calcolerà il next event, ovvero l'evento più vicino nel tempo, e si avanzerà il clock fino al tempo del next event. Si processerà l'evento e si andrà a calcolare il prossimo next event, fino a che non sarà raggiunta una certa condizione di stop.

Modello computazionale

Per quanto riguarda il modello computazionale, abbiamo deciso di scrivere il nostro simulatore in C, facendo sì che i valori ottenuti venissero scritti su dei file .csv e poi usando Python e la libreria mathplotlib.py per la realizzazione di grafici che permettano di visualizzare i risultati ottenuti.
Abbiamo iniziato il progetto scrivendo il codice di un simulatore capace di svolgere una singola run: partendo da un sistema vuoto, simulare il comportamento del sistema fino a un certo tempo di stop e poi fermare i nuovi arrivi e continuare fino allo svuotamento del sistema. In questo modello, ad ogni avanzamento del clock siamo andati ad incrementare i valori medi delle statistiche che ci interessa ottenere dalla simulazione.
Per mantenere queste statistiche abbiamo creato la seguente struct:

```
struct nodeData{
    double node;
    double queue;
    double service;
    int index;
    double current;
    int serverNumber;
}
```

Questa struct contiene, nei primi tre parametri, quanti job si trovano nel nodo, in coda e in servizio, integrando sul tempo. Per ottenere questo calcolo, ogni volta che il c'è un avanzamento del clock siamo andati a sommare al valore precedente di node, queue e service, rispettivamente i valori delle persone presenti nel centro, nella coda e nel sistema moltiplicati per la differenza tra il tempo precedente e il tempo corrente. Index è un campo usato per contare il numero di job completati e current è usato per salvarsi il tempo corrente. Queste due variabili servono per poter ottenere statistiche mediate sul numero di job e nel tempo rispettivamente. L'ultimo campo, serverNumber, è necessario per il calcolo dell'utilizzazione, dato che rappresenta quanti server ci sono in un nodo e ci permette di passare dal tempo di servizio $E(S_i)$ del singolo server a $E(S)$, calcolo del servizio del sistema.
Una volta arrivati al termine della simulazione, da questi campi siamo andati a calcolare le statistiche vere e proprie, che in questa fase in realtà sono state soltanto stampate ma che in seguito sono state salvate in una struct:

```
struct output{
    double wait;
    double delay;
    double service;
    double numberNode;
    double numberQueue;
    double utilization;
    double job;
```

}

Wait sarebbe il tempo medio passato da un job nel sistema (riferito sempre al singolo nodo e non alla rete), ed è stato calcolato come nodeData.node/nodeData.index, che corrisponde a $\bar{w} = \frac{1}{N}\int_0^\tau l(t)dt$.

Delay è il tempo medio passato da un job in coda ed è calcolato come nodeData.queue/nodeData.index, in termini matematici corrisponde a $\bar{d} = \frac{1}{N}\int_0^\tau q(t)dt$.

Service è il tempo medio di servizio, è calcolato come nodeData.service/nodeData.index e corrisponde a $\bar{s} = \frac{1}{N}\int_0^\tau s(t)dt$.

Queste sono le tre statistiche relative al tempo di nostro interesse, poi ci sono tre statistiche relative al numero medio di job.

NumberNode rappresenta il numero di job medio nel sistema, ed è calcolato come nodeData.node/nodeData.current, matematicamente sarebbe $\bar{l} = \frac{1}{\tau}\int_0^\tau l(t)dt$

NumberQueue rappresenta il numero di job medio in coda, calcolato come nodeData.queue/nodeData.current, ovvero $\bar{q} = \frac{1}{\tau}\int_0^\tau q(t)dt$.

Utilization è l'utilizzazione del sistema, questa è calcolata in modo leggermente diverso dagli altri parametri dato che per l'utilizzazione serve calcolare $\bar{x} = \frac{1}{\tau * m}\int_0^\tau s(t)dt$, dove m è il numero di server, in questo modo l'utilizzazione, che nel codice viene calcolata come (nodeData.service/nodeData.current)/nodeData.serverNumber, è riferita all'intero sistema e non al singolo server.

L'ultimo campo, job, salva soltanto quanti job hanno attraversato il nodo ed equivale a nodeData.index.

Queste statistiche sono state calcolate per ogni nodo, e poi nei nodi di traumatologia, problemi medici e problemi minori, che sono sistemi multi-coda, le stesse statistiche sono state calcolate non per il sistema ma per la singola coda, dato che avere l'attesa o l'utilizzazione per coda risulta molto più utile che avere quelle medie del sistema, meno rappresentative del funzionamento del sistema.

Per quanto riguarda il simulatore vero e proprio, per la generazione di numeri casuali abbiamo usato la libreria rngs.h, in particolare le sue funzioni di PlantSeed(SEED) per inizializzare tutti gli stream, Random() per la generazione di un numero casuale e SelectStream(stream) per cambiare stream tra generazioni di variabili diverse. Abbiamo usato dieci stream per le seguenti variabili random:

gli stream da 0 a 4 sono usati per la generazione dei tempi di completamento di triage, codici rossi, traumatologia, problemi medici e problemi minori rispettivamente, lo stream 5 è usato per la generazione del prossimo arrivo, il 6 per stabilire quale codice viene assegnato a un job, il 7 per stabilire se nella coda dei codici rossi ci sarà una morte, l'8 per stabilire in quale nodo andrà un codice giallo o verde, il nove per stabilire se ci sarà un decesso tra i codici gialli. Le variabili di arrivo e completamenti sono state generate come delle esponenziali, quindi come $-\lambda * log(1 - Random())$, dove λ è la media, quindi il tempo di servizio $E(S)$ nel caso dei completamenti e il tempo di interarrivo medio per gli arrivi. Per tutte le altre variabili abbiamo usato variabili uniformi, quindi generate come $a + (b - a) * Random()$, dove a e b sono gli estremi entro cui vogliamo far cadere la

nostra variabile. Dato che tutte le nostre variabili uniformi portano a prendere diversi rami di computazione in base a se il numero generato cade sopra o sotto una certa percentuale, a e b sono sempre stati posti a 0 e 100 rispettivamente, per semplicità di calcolo.
A questo punto è stato necessario andare a calcolare la probabilità di morte, e abbiamo deciso di implementare la cosa con due funzioni, create in modo da avere la probabilità leggermente inferiore a quella trovata dal modello delle specifiche per una persona in coda e leggermente superiore per due. In realtà in questa situazione siamo andati a ragionare già nell'ottica di avere in futuro il sistema migliorato: quando andremo a dividere tra codici blu e arancioni, solo gli arancioni avranno una probabilità di decesso. Abbiamo deciso di scrivere le funzioni per i decessi nei codici rossi e arancioni, e poi nella coda gialla moltiplicare la probabilità per la probabilità di avere un codice arancione. In questo modo riusciamo ad assicurarci che il confronto sia sensato, perché la probabilità di morte nei due sistemi sarà la stessa. Sarebbe stato migliore gestirlo evitando di inserire la probabilità di avere un codice arancione nel simulatore prima di inserire le code arancioni, ma questo ci assicura che i due sistemi funzionino nello stesso modo. Per fare questo controllo, siamo andati ad usare l'undicesimo stream per andare a calcolare la probabilità che un codice giallo sia più o meno grave.
Una volta generate le variabili random, implementata la gestione degli eventi come descritto nelle specifiche e calcolate le statistiche, il simulatore ha iniziato a funzionare. Arrivati a questo punto abbiamo anche potuto svolgere una prima fase di verifica, anche se non troppo precisa, andando a produrre dei risultati teorici e controllando che le statistiche di una run con tempi molto lunghi non divergessero di molto dai risultati attesi. Anche se a questo punto non avevamo ancora intervalli di confidenza o diverse run su cui fare una media, questo controllo ci ha permesso di effettuare una fase di debug prima di andare a complicare ulteriormente il codice.

SIMULAZIONE A ORIZZONTE FINITO

Una volta ottenuto il simulatore funzionante abbiamo per prima cosa deciso di implementare un'analisi dello stato transiente, che è stata effettuata tramite il metodo della replicazione: abbiamo effettuato 64 run consecutive, senza mai reinizializzare gli stream di numeri random per ottenere risultati diversi tra run diverse, del simulatore, e usato questi risultati per calcolare un valore medio e un intervallo di confidenza per tutte le statistiche di nostro interesse. Queste run sono state effettuate con un tempo di stop abbastanza limitato, in particolare abbiamo deciso di simulare il sistema per una giornata, quindi ponendo lo stop a 1440.0 (minuti). Abbiamo inizializzato una matrice dove salvare su ogni riga i risultati di una run, usando ogni colonna per un centro diverso. Gli elementi di questa matrice sono di tipo struct output, in modo da contenere tutte le statistiche necessarie. Per far questo abbiamo leggermente modificato il programma di simulazione in modo che prendesse in input la matrice e l'indice della riga da scrivere e terminasse scrivendo i dati ottenuti sulla riga. Una volta effettuate le 64 run del simulatore e riempita la matrice, possiamo andare a calcolare le statistiche finali, che sono di due tipi:

1. Calcolare, per ogni statistica di ogni nodo, la media su tutte le simulazioni e l'intervallo di confidenza
2. Scrivere su un file le medie incrementali (quindi la media tra le prime due run, poi le prime tre…) per poter realizzare grafici della simulazione

Per il primo punto abbiamo realizzato una funzione che calcola la media, salvato le medie, le abbiamo usate per calcolare la varianza e, a partire dalla varianza, l'intervallo di confidenza.

MEDIA

Il calcolo della media inizialmente è sembrato abbastanza banale: scorrere le colonne della matrice, per ogni colonna effettuare la somma di tutti i valori sulle righe relativi a quella colonna, arrivati alla fine dividere per il numero di righe e salvare ogni risultato in un vettore lungo quanto il numero di colonne. Dopo aver testato il programma però ci siamo resi conto che questo non era sufficiente: infatti, effettuando le run per un tempo limitato, capitano alcune run dove in qualche centro non entra nessun job. Questo porta ad un problema per quanto riguarda i tempi di servizio, wait e delay: infatti, dato che vengono calcolati dividendo per il numero di job, effettuare una divisione per zero porta a dei valori NAN, e nella somma, andare a sommare un NAN a un numero porta ad avere un NAN. Quindi, per ogni centro dove una run ha prodotto un NAN, non avevamo statistiche usabili. Abbiamo deciso di risolvere questo problema andando a controllare che il valore fosse diverso da NAN prima di effettuare la somma e contare su quante run stavamo effettivamente calcolando la media per la divisione finale, dato che non sono più 64 ma qualcuna in meno.

INTERVALLO DI CONFIDENZA

L'intervallo di confidenza si calcola con la seguente formula: $\frac{t^\star \sigma}{\sqrt{n-1}}$, che rappresenta di quanto è possibile scostarsi dalla media in positivo o in negativo. σ è la deviazione standard, n è il numero di ripetizioni considerate e $t^\star$ viene calcolato come l'inverso di una distribuzione di Student con parametro n-1. In particolare, presa una variabile $T = Student(n-1)$, $t^\star$ deve essere tale che $P(-t^\star \leq T \leq t^\star) = 1 - \alpha$. α è un numero compreso tra 0 e 1 che può essere definito come un "parametro di confidenza", e solitamente è una buona scelta prenderlo del 5%. A livello di codice, abbiamo calcolato $t^\star$ con l'ausilio della libreria rvms.h, che contiene le funzioni di densità, distribuzione e distribuzione inversa per diverse variabili random. In particolare abbiamo usato la funzione idfStudent(long n, double u), che calcola la distribuzione inversa di una variabile di Student, passandogli come parametri $n-1$ e $1-\alpha/2$. Una volta ottenuto $t^\star$, che è uguale per ogni statistica, essendo il numero di ripetizioni sempre uguale, è necessario calcolare la deviazione standard. Per far questo, abbiamo scansionato la matrice come per il calcolo della media, solo che invece che sommare le medie delle singole run, abbiamo sommato le differenze tra le medie delle singole run e la media delle medie. Ottenuti questi valori è stato possibile calcolare e stampare l'intervallo di confidenza per ogni media.

MEDIE INCREMENTALI

Per realizzare dei grafici che mostrino l'andamento della simulazione all'aumentare delle run abbiamo deciso di realizzare un file .csv dove salvare di volta in volta la media aggregata fino alla run i. Il file è costituito da 64*12 righe, che contengono le medie per i dodici nodi, ognuna ripetuta per 64 iterazioni. Sulle colonne, abbiamo salvato gli indici del nodo e dell'iterazione per semplicità di lettura, e poi la media per quel centro e quell'iterazione di ognuno dei campi della struct output, che ci interessano per i grafici.

Un'altra media di cui abbiamo tenuto conto è la percentuale di decessi, calcolata usando un array che contenga la percentuale per ogni iterazione e poi facendo la media sugli elementi dell'array. La percentuale è calcolata come il numero di decessi, da qualsiasi coda, diviso per il numero di accessi con codice rosso o giallo.

SIMULAZIONE A ORIZZONTE INFINITO

Mentre la simulazione a orizzonte finito va ad analizzare il comportamento in uno stato transiente con tempo limitato, questo tipo di simulazione va ad esaminare il comportamento del simulatore in un regime stazionario. Per fare questo tipo di simulazione siamo andati ad utilizzare il metodo delle batch means, che consiste nell'eseguire una run molto lunga, dividerla in batch in base al numero di job e calcolare le statistiche di interesse come media tra le medie delle varie batch. Sono necessari due parametri, il numero di batch e il numero di job in ogni batch. Per il numero di batch è stato scelto 64, che la letteratura in merito indica come numero ideale e che offre anche simmetria con la simulazione a orizzonte finito in cui abbiamo effettuato 64 run. In questo modo abbiamo la stessa quantità di dati aggregati per ogni simulazione. Per quanto riguarda il numero di job abbiamo scelto 1024, che non è altissimo ma è raggiungibile anche da centri con meno affluenza, come il centro responsabile dei codici rossi, in tempi ragionevoli. Per effettuare questo tipo di simulazione siamo dovuti andare a modificare il codice del simulatore vero e proprio, a cui per prima cosa abbiamo passato un valore booleano per andare a discriminare se ci troviamo in una simulazione a orizzonte finito o infinito. Due sono le differenze fondamentali rispetto alla simulazione a orizzonte finito: lo stop del sistema è calcolato in modo differente, e la matrice che contiene i dati va riempita totalmente dall'unica run che effettuiamo, mentre prima ogni run riempiva solo una riga.

Per quanto riguarda il momento in cui il sistema si ferma, in questo caso non è più basato sul tempo: il sistema si ferma quando tutte le batch di tutti i centri sono state riempite. Questo ci ha portato un problema a livello tecnico: finché avevamo un tempo di stop avevamo calcolato la costante INF, usata come instante di tempo per gli eventi impossibili, come 100*STOP. Ovviamente, non avendo più un tempo di STOP, effettuare questo calcolo non è più possibile, e per evitare di incorrere in problemi abbiamo definito INF come il più grande double rappresentabile, che è probabilmente molto più che eccessivo ma sicuramente non corriamo il rischio che il current time superi INF, andando a falsare i risultati della simulazione.

Abbiamo inizializzato due vettori, responsabili rispettivamente di mantenere le informazioni relative alla batch corrente e al numero corrente di job nella batch per ogni centro. Le batch non vengono riempite in modo simmetrico tra di loro: ogni volta che un job passa nel centro viene contato verso la batch corrente di quel centro, quindi alcune batch si riempiranno più velocemente di altre. Quando un centro ha riempito tutte le sue batch, anche se i job continuano ad arrivare non vengono più contati ne vengono salvate statistiche relative a quei job. In questo modo, le statistiche che andiamo ad analizzare sono relative a tempi diversi ma numero uguale di job per ogni centro.

L'altra differenza fondamentale è la scrittura delle statistiche: ogni volta che avviene un evento e quindi le statistiche della run vengono aggiornate si va a controllare se qualcuna delle batch è stata riempita: in caso affermativo, le statistiche raccolte fino a quel momento per il centro vengono salvate nella casella (o nelle caselle in caso di centri multi-coda) corrispondente della matrice, che contiene le batch sulle righe e i centri/code sulle colonne, poi la struct nodeData responsabile viene re-inizializzata così come il numero di job nella batch per quel centro, e l'indice delle batch viene incrementato. Andare a calcolare le statistiche per ogni batch ha fatto si che fosse necessario salvare un campo aggiuntivo nella struct nodeData. Infatti, mentre nell'orizzonte finito, facendo partire la simulazione al tempo zero è corretto usare currentTime per le statistiche mediate sul tempo, in questa situazione va usata la differenza tra il currentTime nel momento in cui andiamo a calcolare la statistica e il momento in cui è stata cambiata batch, dato che in quel momento abbiamo azzerato il conto dei job.

Finita la simulazione, avendo ottenuto una matrice molto simile a quella scritta dalle diverse run necessarie per la simulazione ad orizzonte finito, possiamo usare lo stesso codice scritto in precedenza per ottenere media, intervallo di confidenza e file con le medie incrementali. Nello stesso modo, possiamo usare un array come quello usato nella simulazione a orizzonte finito per la percentuale di decessi, dove ogni posizione rappresenta i decessi su una batch diversa.

Verifica

Per quanto riguarda la verifica, abbiamo calcolato i valori teorici delle statistiche che stiamo raccogliendo, quindi utilizzazione, numero di persone in coda, numero di persone nel sistema, tempo totale nel sistema, tempo in coda e tempo di servizio, che non è effettivamente da calcolare dato che appartiene ai dati di input ma è comunque da controllare che il nostro simulatore calcoli il valore corretto. Per la verifica usiamo le medie prodotte dalla simulazione a orizzonte infinito, dato che i calcoli teorici si riferiscono a un sistema stazionario. Non sappiamo ancora qual è la configurazione ottima di serventi, quindi abbiamo scelto dei valori di N per ogni nodo in modo abbastanza casuale, assicurandoci soltanto che siano valori abbastanza alti da garantire un sistema stazionario. Non siamo effettivamente in grado di calcolare in modo teorico l'utilizzazione effettiva data dalle code gialle e rosse, dato che non abbiamo un modo preciso per calcolare le perdite. Abbiamo deciso di effettuare una verifica senza il codice che gestisce le morti, poi inserirlo e verificare se le utilizzazioni rilevanti diminuiscono leggermente dopo questo inserimento.

Triage

$\lambda = 0.09\ job/min$

$E(S_i) = 10\ min$

$N = 2$

$E(S) = \frac{E(S_i)}{N} = \frac{10}{2} = 5\ min$ $\qquad \mu = \frac{1}{E(S)} = \frac{1}{5} = 0.2\ job/min$

$\rho = \frac{\lambda}{\mu} = \frac{0.09}{0.2} = 0.45$ Il valore ottenuto dal simulatore è 0.455278 ± 0.004452, quindi abbastanza accurato

Essendo un sistema M/M/m a coda singola, il tempo di attesa in coda va calcolato come $E(T_q) = \frac{P_q E(S)}{1-\rho}$.

$$P_q = \frac{(N\rho)^N}{N!(1-\rho)} * \left(\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!} + \frac{(N\rho)^N}{N!(1-\rho)}\right)^{-1} = \frac{(2*0.45)^2}{2*(1-0.45)} * \left(\sum_{i=0}^{1} \frac{(2*0.45)^i}{i!} + \frac{(2*0.45)^2}{2*(1-0.45)}\right)^{-1} = 0.27931$$

$E(T_q) = \frac{0.27931 * 5}{1 - 0.45} = 2.5392\ min$ Il risultato del simulatore è 2.671317 ± 0.149609, anche questo è verificato.

$E(T_s) = E(T_q) + E(S_i) = 2.5392 + 10 = 12.5392\ min$

I risultati del simulatore sono 10.043568 ± 0.078102 minuti per il tempo di servizio e 12.714884 ± 0.199692 minuti per il tempo del sistema, quindi anche questi valori sono verificati.

$E(N_q) = \lambda E(T_q) = 0.09 * 2.5392 = 0.22853$ Il risultato del simulatore è 0.242585 ± 0.013994

$E(N_s) = \lambda E(T_s) = 0.09 * 12.5392 = 1.12853$ Il risultato del simulatore è 1.153140 ± 0.020755

Codici rossi

Dato che il triage è stazionario ed ha throughput λ,

$\lambda = \lambda_{triage} * p = 0.09 * 0.0104 = 9.36 * 10^{-4}\ job/min$

$E(S_i) = 225.5\ min$

$N = 2$

$E(S) = \dfrac{E(S_i)}{N} = \dfrac{225.5}{2} = 112.75\ min$

$\mu = \dfrac{1}{E(S)} = \dfrac{1}{112.75} = 8.86918 * 10^{-3}\ job/min$

$\rho = \dfrac{\lambda}{\mu} = \dfrac{9.36 * 10^{-4}}{8.86918 * 10^{-3}} = 0.10553$ Il valore ottenuto dal simulatore è 0.105893 ± 0.001063. Aggiungendo la possibilità di decessi il rho diventa 0.105768 ± 0.001066, quindi leggermente inferiore anche se di poco.

$P_q = \dfrac{(N\rho)^N}{N!(1-\rho)} * (\sum_{i=0}^{N-1} \dfrac{(N\rho)^i}{i!} + \dfrac{(N\rho)^N}{N!(1-\rho)})^{-1} = \dfrac{(2 * 0.10553)^2}{2 * (1 - 0.10553)} * (\sum_{i=0}^{1} \dfrac{(2 * 0.10553)^i}{i!} + \dfrac{(2 * 0.10553)^2}{2 * (1 - 0.10553)})^{-1} = 0.02015$

$E(T_q) = \dfrac{P_q E(S)}{1-\rho} = \dfrac{0.02015 * 112.75}{1 - 0.10553} = 2.53958\ min$ Il risultato del simulatore è 2.444438 ± 0.218717.

$E(T_s) = E(T_q) + E(S_i) = 2.53958 + 225.5 = 228.03958\ min$

I valori ottenuti dal simulatore per il tempo di servizio sono di 225.417495 ± 1.819388 min e per il tempo nel sistema di 227.861934 ± 1.886351 minuti, anche in questo caso i numeri sono sufficientemente precisi da considerare la simulazione funzionante.

$E(N_q) = \lambda E(T_q) = 9.36 * 10^{-4} * 2.53958 = 2.37705 * 10^{-3}$ Il risultato del simulatore è 0.002302 ± 0.000211

$E(N_s) = \lambda E(T_s) = 9.36 * 10^{-4} * 228.03958 = 0.21345$ Il risultato del simulatore è 0.214087 ± 0.002216, appena più alto del valore teorico ma abbastanza vicino da essere accettabile.

Traumatologia

$\lambda = \lambda_{triage} * p_{trauma} * (p_{giallo} + p_{verde}) = 0.09 * 0.267 * (0.184 + 0.6071) = 0.01901\ job/min$

$E(S_i) = 93.4\ min$

$N = 3$

$E(S) = \dfrac{E(S_i)}{N} = \dfrac{93.4}{3} = 31.13333\ min$

$\mu = \dfrac{1}{E(S)} = \dfrac{1}{31.13333} = 0.03212\ job/min$

$\rho = \dfrac{\lambda}{\mu} = \dfrac{0.01901}{0.03212} = 0.59184$ Il valore ottenuto con il simulatore è 0.587792 ± 0.006014

In questo nodo abbiamo due code, la coda 1 dei codici gialli e la coda 2 dei codici verdi, vanno quindi calcolate le probabilità e l'utilizzazione, così come il tempo in coda ed in servizio, per ogni coda.

$$p_1 = \frac{p_{gialli}}{p_{gialli} + p_{verdi}} = \frac{18.4}{18.4 + 60.71} = 0.23259$$

$\rho_1 = \rho * p_1 = 0.59184 * 0.23259 = 0.13766$ (Simulatore 0.135904 ± 0.003358)

Aggiungendo la possibilità di decessi diventa 0.135899 ± 0.003064, anche questo leggermente inferiore.

$$p_2 = \frac{p_{verdi}}{p_{gialli} + p_{verdi}} = \frac{60.71}{18.4 + 60.71} = 0.76741$$

$\rho_2 = \rho * p_2 = 0.59184 * 0.76741 = 0.45418$ (Simulatore 0.452168 ± 0.005193)

Per poter calcolare i tempi in coda, essendo un multiserver, è comunque necessario il valore di P_q

$$P_q = \frac{(N\rho)^N}{N!(1-\rho)} * (\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!} + \frac{(N\rho)^N}{N!(1-\rho)})^{-1} = \frac{(3 * 0.59184)^3}{3! * (1 - 0.59184)} * (\sum_{i=0}^{2} \frac{(3 * 0.59184)^i}{i!} + \frac{(3 * 0.59184)^3}{3! * (1 - 0.59184)})^{-1} = 0.34435$$

$E(T_{q_1}) = \frac{P_q E(S)}{1 - \rho_1} = \frac{0.34435 * 31.13333}{1 - 0.13766} = 12.43217\ min$ (Simulatore 12.190407 ± 0.715426)

$$E(T_{q_2}) = \frac{P_q E(S)}{(1 - \rho_1)(1 - \rho)} = \frac{0.34435 * 31.13333}{(1 - 0.13766)(1 - 0.59184)} = 30.45907\ min$$

(Simulatore 29.662747 ± 2.498538)

$E(T_{s_1}) = E(T_{q_1}) + E(S_i) = 12.43217 + 93.4 = 105.83217\ min$

(Simulatore 104.482441 ± 1.773818)

$E(T_{s_2}) = E(T_{q_2}) + E(S_i) = 30.45907 + 93.4 = 123.85907\ min$

(Simulatore 122.347703 ± 3.015000)

$E(T_q) = p_1 E(T_{q_1}) + p_2 E(T_{q_2}) = 26.26619\ min$

$E(T_s) = E(T_q) + E(S_i) = 119.66619\ min$

I valori del simulatore per il tempo in coda medio e il tempo totale medio sono di 25.614632 ± 2.014997 e 118.223520 ± 2.504792. I tre tempi di servizio, per le due code e medio, sono 92.292035 ± 1.576546, 92.684955 ± 0.917884 e 92.608888 ± 0.838055, tutti corrispondenti all'E(S).

$E(N_{q_1}) = \lambda * p_1 * E(T_{q_1}) = 0.01901 * 0.23259 * 12.43217 = 0.05497$ Il valore del simulatore è 0.054053 ± 0.003579

$E(N_{q_2}) = \lambda * p_2 * E(T_{q_2}) = 0.01901 * 0.76741 * 30.45907 = 0.44435$ Il valore del simulatore è 0.434459 ± 0.036840

$E(N_{s_1}) = \lambda * p_1 * E(T_{s_1}) = 0.01901 * 0.23259 * 105.83217 = 0.46794$ Il valore del simulatore è 0.461765 ± 0.011856

$E(N_{s_2}) = \lambda * p_2 * E(T_{s_2}) = 0.01901 * 0.76741 * 123.85907 = 1.80691$ Il valore del simulatore è 1.790963 ± 0.046091

$E(N_q) = \lambda * E(T_q) = 0.01901 * 26.26619 = 0.49932$ Il valore del simulatore è 0.488386 ± 0.039025

$E(N_s) = \lambda * E(T_s) = 0.01901 * 119.66619 = 2.27485$ Il valore del simulatore è 2.251760 ± 0.050860

Problemi medici

$$\lambda = \lambda_{triage} * p_{pmed} * (p_{giallo} + p_{verde}) = 0.09 * 0.247 * (0.184 + 0.6071) = 0.0176\ job/min$$

$E(S_i) = 165.9\ min$

$N = 4$

$E(S) = \frac{E(S_i}{N} = \frac{165.9}{4} = 41.475\ min$ $\mu = \frac{1}{E(S)} = \frac{1}{41.475} = 0.0241\ job/min$

$\rho = \frac{\lambda}{\mu} = \frac{0.0176}{0.0241} = 0.73029$ Il valore ottenuto dal simulatore è di 0.730926 ± 0.008591

$p_1 = \frac{p_{gialli}}{p_{gialli} + p_{verdi}} = \frac{18.4}{18.4 + 60.71} = 0.23259$

$\rho_1 = p_1 * \rho = 0.23259 * 0.73029 = 0.16986$ (Simulatore 0.169359 ± 0.004108)

Aggiungendo la possibilità di decessi diventa 0.170057 ± 0.004103, che sembra non variare ma se consideriamo che la variazione è molto piccola, ha senso non sia visibile su una media.

$p_2 = \frac{p_{verdi}}{p_{gialli} + p_{verdi}} = \frac{60.71}{18.4 + 60.71} = 0.76741$

$\rho_2 = 0.76741 * 0.73029 = 0.56043$ (Simulatore 0.561953 ± 0.006784)

$P_q = \frac{(N\rho)^N}{N!(1-\rho)} * (\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!} + \frac{(N\rho)^N}{N!(1-\rho)})^{-1} = \frac{(4 * 0.73029)^4}{4! * (1 - 0.73029)} * (\sum_{i=0}^{3} \frac{(4 * 0.73029)^i}{i!} + \frac{(4 * 0.73029)^4}{4! * (1 - 0.73029)})^{-1} = 0.47683$

$E(T_{q_1}) = \frac{P_q E(S)}{(1 - \rho_1)} = \frac{0.47683 * 41.475}{(1 - 0.16986)} = 23.82312\ min$

(Simulatore 22.746423 ± 1.209670)

$E(T_{q_2}) = \frac{P_q E(S)}{(1 - \rho_1)(1 - \rho)} = \frac{0.47683 * 41.475}{(1 - 0.16986)(1 - 0.73029)} = 88.32865\ min$

(Simulatore 85.844821 ± 9.552533)

$E(T_{s_1}) = E(T_{q_1}) + E(S_i) = 23.82312 + 165.9 = 189.72312\ min$

(Simulatore 188.014844 ± 3.595199)

$E(T_{s_2}) = E(T_{q_2}) + E(S_i) = 88.32865 + 165.9 = 254.22865\ min$

(Simulatore 252.245868 ± 10.142855)

$E(T_q) = p_1 E(T_{q_1}) + p_2 E(T_{q_2}) = 73.32531\ min$

$E(T_s) = E(T_q) + E(S_i) = 239.22531\ min$

Anche in questo caso i risultati del simulatore sono coerenti, con 71.106294 ± 7.439697 come tempo in coda e 237.219357 ± 8.151894 come tempo nel sistema, nonostante la varianza, e quindi l'intervallo di confidenza, siano molto maggiori. I tre tempi di servizio, per coda e totali, sono 165.268422 ± 3.10204, 166.401046 ± 1.406778 e 166.113063 ± 1.337167, tutti e tre coerenti.

$E(N_{q_1}) = \lambda * p_1 * E(T_{q_1}) = 0.0176 * 0.23259 * 23.82312 = 0.09752$ Il valore del simulatore è 0.093927 ± 0.006000

$E(N_{q_2}) = \lambda * p_2 * E(T_{q_2}) = 0.0176 * 0.76741 * 88.32865 = 1.19300$ Il valore del simulatore è 1.166760 ± 0.135459

$E(N_{s_1}) = \lambda * p_1 * E(T_{s_1}) = 0.0176 * 0.23259 * 189.72312 = 0.77665$ Il valore del simulatore è 0.771364 ± 0.020665

$E(N_{s_2}) = \lambda * p_2 * E(T_{s_2}) = 0.0176 * 0.76741 * 254.22865 = 3.43372$ Il valore del simulatore è 3.414572 ± 0.152504

$E(N_q) = \lambda * E(T_q) = 0.0176 * 73.32531 = 1.29053$ Il valore del simulatore è 1.260468 ± 0.139443

$E(N_s) = \lambda * E(T_s) = 0.0176 * 239.22531 = 4.21037$ Il valore del simulatore è 4.184172 ± 0.165639

Problemi minori

$\lambda = \lambda_{triage} * (p_{min} * (p_{gialli} + p_{verdi}) + p_{bianchi}) = 0.09 * (19.85 + (18.40 + 60.71) * 48.6) = 0.05246\ job/min$

$E(S_i) = 105.6\ min$

$N = 7$

$$E(S) = \frac{E(S_i)}{N} = \frac{105.6}{7} = 15.08571\ min$$

$$\mu = \frac{1}{E(S)} = \frac{1}{15.08571} = 0.06629\ job/min$$

$$\rho = \frac{\lambda}{\mu} = \frac{0.05246}{0.06629} = 0.79137$$ (Il valore del simulatore è 0.797714 ± 0.009969)

In questo nodo abbiamo tre code e quindi tre diversi valori di probabilità, 1 per i gialli, 2 per i verdi e 3 per i bianchi

$$p_1 = \frac{p_{gialli} * p_{min}}{(p_{gialli} + p_{verdi}) * p_{min} + p_{bianchi}} = \frac{18.4 * 0.486}{(18.4 + 60.71) * 0.486 + 19.85} = 0.15339$$

$\rho_1 = p_1 * \rho = 0.15339 * 0.79137 = 0.12139$ (Simulatore 0.122785 ± 0.003473)

Aggiungendo la possibilità di decessi diventa 0.120729 ± 0.003978

$$p_2 = \frac{p_{verdi} * p_{min}}{(p_{gialli} + p_{verdi}) * p_{min} + p_{bianchi}} = \frac{60.71 * 0.486}{(18.4 + 60.71) * 0.486 + 19.85} = 0.50611$$

$\rho_2 = p_2 * \rho = 0.50611 * 0.79137 = 0.40052$ (Simulatore 0.407976 ± 0.007249)

$$p_3 = \frac{p_{bianchi}}{(p_{gialli} + p_{verdi}) * p_{min} + p_{bianchi}} = \frac{19.85}{(18.4 + 60.71) * 0.486 + 19.85} = 0.34050$$

$\rho_3 = p_3 * \rho = 0.34050 * 0.79137 = 0.26946$ (Simulatore 0.267270 ± 0.004272)

$$P_q = \frac{(N\rho)^N}{N!(1-\rho)} * \left(\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!} + \frac{(N\rho)^N}{N!(1-\rho)}\right)^{-1} = \frac{(7 * 0.79137)^7}{7! * (1 - 0.79137)} * \left(\sum_{i=0}^{6} \frac{(7 * 0.79137)^i}{i!} + \frac{(7 * 0.79137)^7}{7! * (1 - 0.79137)}\right)^{-1} = 0.4652$$

$$E(T_{q_1}) = \frac{P_q * E(S)}{1 - \rho_1} = \frac{0.4652 * 15.08571}{1 - 0.15339} = 8.28938$$

(Simulatore 8.420890 ± 0.459342)

$$E(T_{q_2}) = \frac{P_q * E(S)}{(1 - \rho_1)(1 - \rho_1 - \rho_2)} = \frac{0.4652 * 15.08571}{(1 - 0.15339)(1 - 0.15339 - 0.40052)} = 18.58230$$

(Simulatore 17.384608 ± 1.431516)

$$E(T_{q_3}) = \frac{P_q * E(S)}{(1 - \rho)(1 - \rho_1 - \rho_2)} = \frac{0.4652 * 15.08571}{(1 - 0.79137)(1 - 0.15339 - 0.40052)} = 75.40605$$

(Simulatore 75.866191 ± 11.390080)

$E(T_{s_1}) = E(T_{q_1}) + E(S) = 8.28938 + 105.6 = 113.88938$

(Simulatore 115.041293 ± 1.956712)

$E(T_{s_2}) = E(T_{q_2}) + E(S) = 18.58230 + 105.6 = 124.18230$

(Simulatore 123.331316 ± 2.340810)

$E(T_{s_3}) = E(T_{q_3}) + E(S) = 75.40605 + 105.6 = 181.00605$

(Simulatore 180.430495 ± 11.660854)

$E(T_q) = p_1 * E(T_{q_1}) + p_2 * E(T_{q_2}) + p_3 * E(T_{q_3}) = 0.15339 * 8.28938 + 0.50611 * 18.58230 * 0.34050 * 75.40605 = 36.35195$

(Simulatore 35.642855 ± 4.385445)

$E(T_s) = E(T_q) + E(S) = 36.35195 + 105.6 = 141.95195$

(Simulatore 141.205007 ± 4.947116)

Per quanto riguarda i tempi di servizio, che questa volta sono quattro data la coda in più, dal simulatore abbiamo ottenuto i valori: 106.620403 ± 1.811982, 105.946709 ± 1.192823, 104.564304 ± 1.266096 e 105.562152 ± 0.869603, tutti coerenti con il valore teorico.

$E(N_{q_1}) = \lambda * p_1 * E(T_{q_1}) = 0.05246 * 0.15339 * 8.28938 = 0.06670$ Il valore del simulatore è 0.068214 ± 0.004332

$E(N_{q_2}) = \lambda * p_2 * E(T_{q_2}) = 0.05246 * 0.50611 * 18.58230 = 0.49337$ Il valore del simulatore è 0.471631 ± 0.041554

$E(N_{q_3}) = \lambda * p_3 * E(T_{q_3}) = 0.05246 * 0.34050 * 75.40605 = 1.34695$ Il valore del simulatore è 1.363993 ± 0.21375

$E(N_{s_1}) = \lambda * p_1 * E(T_{s_1}) = 0.05246 * 0.15339 * 113.88938 = 0.916450$ Il valore del simulatore è 0.927711 ± 0.026963

$E(N_{s_2}) = \lambda * p_2 * E(T_{s_2}) = 0.05246 * 0.50611 * 124.18230 = 3.29711$ Il valore del simulatore è 3.327460 ± 0.085850

$E(N_{s_3}) = \lambda * p_3 * E(T_{s_3}) = 0.05246 * 0.34050 * 181.00605 = 3.23324$ Il valore del simulatore è 3.234883 ± 0.224144

$E(N_q) = \lambda * E(T_q) = 0.05246 * 36.35195 = 1.90701$ Il valore del simulatore è 1.903301 ± 0.249057

$E(N_s) = \lambda * E(T_s) = 0.05246 * 141.95195 = 7.44680$ Il valore del simulatore è 7.487296 ± 0.307365

Validazione

Fare una validazione accurata richiederebbe avere dati affidabili sui tempi di attesa medi per i nostri reparti e codici e per la nostra configurazione, e questi dati non sono stati reperibili. Infatti, anche se abbiamo trovato dei dati sui tempi di servizio e sulle attese medie, non abbiamo informazioni su come è configurato l'ospedale abbastanza da rendere significativi i dati sulle attese.

Per effettuare la validazione, quindi, si è scelto un approccio diverso: sono stati elaborati diversi casi di test con configurazioni diverse del sistema in termini di numeri di server per nodo e λ in ingresso per valutare se il comportamento del sistema, e quindi i risultati ottenuti, sono conformi a quanto ci aspetteremmo.

In particolare, a parità di numero di server utilizzati, all'aumentare del lambda ci aspettiamo un aumento di tempi medi di attesa su tutte le code, con un' influenza maggiore sui tempi del triage e minore sui tempi dei codici rossi.

Mantenendo il tasso di ingresso costante, invece, e aumentando il numero di server su un qualsiasi nodo oppure mantenendo costante il numero ma aumentando la qualità dei

server, diminuendo quindi il tempo medio di servizio, ci aspettiamo che i parametri sui tempi medi di attesa diminuiscano sul nodo modificato.
Di conseguenza sono stati scelti i seguenti casi di validazione: nel primo è stata scelta la configurazione iniziale di server e la stessa configurazione con tasso di ingresso aumentato del 50%, situazione in cui il sistema non è nemmeno più stazionario.
Nel secondo caso di test il tasso di ingresso è stato mantenuto costante ma in un caso il numero di server è (-boh inserire numeri bassi) e in un altro è (-boh inserire numeri alti).
Di conseguenza si può vedere come per tutti e due i casi le aspettative sono rispettate.

Progettazione degli esperimenti

Il nostro scopo è cercare la combinazione di server che, usandone il minor numero possibile, garantisca i tempi prefissati per le diverse code del sistema. Come configurazione iniziale abbiamo deciso di calcolare quanti serventi sono necessari per ogni centro per garantire la stazionarietà del sistema, andando da lì a migliorare dove necessario per ridurre i tempi di attesa.
La configurazione minima di stazionarietà possiamo calcolarla a partire dai λ e dai tempi di servizio. Usiamo per ogni centro la seguente formula: $\frac{\lambda}{m * \mu} < 1$, dove m è il numero di server che stiamo cercando, e prendiamo il minimo m possibile. Per i centri i risultati sono i seguenti:

- TRIAGE: $\frac{0.09}{0.1 * m} < 1, \quad m > 0.9 \rightarrow m = 1$
- CODICI ROSSI: $\frac{8.87329 * 10^{-4}}{4.4845 * 10^{-3} * m} < 1, \quad m > 0.20 \rightarrow m = 1$
- TRAUMATOLOGIA: $\frac{0.01901}{0.01071 * m} < 1, \quad m > 1.8 \rightarrow m = 2$
- PROBLEMI MEDICI: $\frac{0.0176}{6.02773 * 10^{-3} * m} < 1, \quad m > 2.9 \rightarrow m = 3$
- PROBLEMI MINORI: $\frac{0.05246}{9.4697 * 10^{-3}} < 1, \quad m > 5.5398 \rightarrow m = 6$

Partiamo quindi da una configurazione con tredici serventi. Questi serventi vanno a costare totalmente $1 * c_{triage} + 1 * c_{rossi} + 6 * c_{inf} + 5 * c_{med} = 1700 + 7000 + 6 * 2800 + 5 * 4200 = 46500$ da sottrarre al budget. Abbiamo dei limiti sul delay da rispettare per ognuno di questi centri, quindi dobbiamo andare a vedere i delay ottenuti da questa situazione e aumentare i serventi dove necessario, andando quindi a vedere dove i tempi non vengono rispettati. Il triage, con un'utilizzazione del 90%, ha tempi nel sistema assolutamente inaccettabili, con tempi superiori ai 100 minuti, così come i codici rossi, che dovrebbero avere attesa quasi nulla ma così hanno un attesa di quasi un'ora. Per gli altri tre centri la situazione non è così critica, ma comunque non tutti i tempi vengono rispettati.

L'aumento del numero di serventi dovrebbe comunque non causare costi maggiori del budget.
Il primo parametro da rispettare è mantenere molto bassa la probabilità che un codice rosso non venga servito immediatamente e quindi si verifichi un decesso. Con un solo servente questa probabilità è leggermente superiore all'1%, valore decisamente elevato dato che tiene conto delle morti che si verificano in attesa, che dovrebbero tendere allo zero. Quindi, qualsiasi configurazione dovrebbe considerare almeno 2 serventi per i codici rossi, con un abbassamento della percentuale di decessi al 0,075%, valore che, essendo minore dello 0,1%, può essere considerato accettabile.
Questo valore, che deve essere fissato, porta ad altri 7000€ spesi del budget, portandolo a 53500 €. Ora va deciso come distribuire il resto del budget per ottenere tempi di attesa accettabili anche nelle altre code.
Abbiamo deciso di misurare le differenze tra i delay medi dei diversi nodi e i valori target, sommandole poi con peso diverso in base al codice, per cercare di dare una sorta di "punteggio" alle diverse soluzioni possibili mantenendoci nel budget. Siamo poi andati comunque a valutare nelle configurazioni che abbiamo ritenuto migliori dove fossero i ritardi più elevati per cercare di ottenere una soluzione più sensata possibile. Le configurazioni più promettenti che abbiamo testato sono state:

- Triage: 2
- Codici rossi: 2
- Traumatologia: 3
- Problemi medici: 3
- Problemi minori: 8

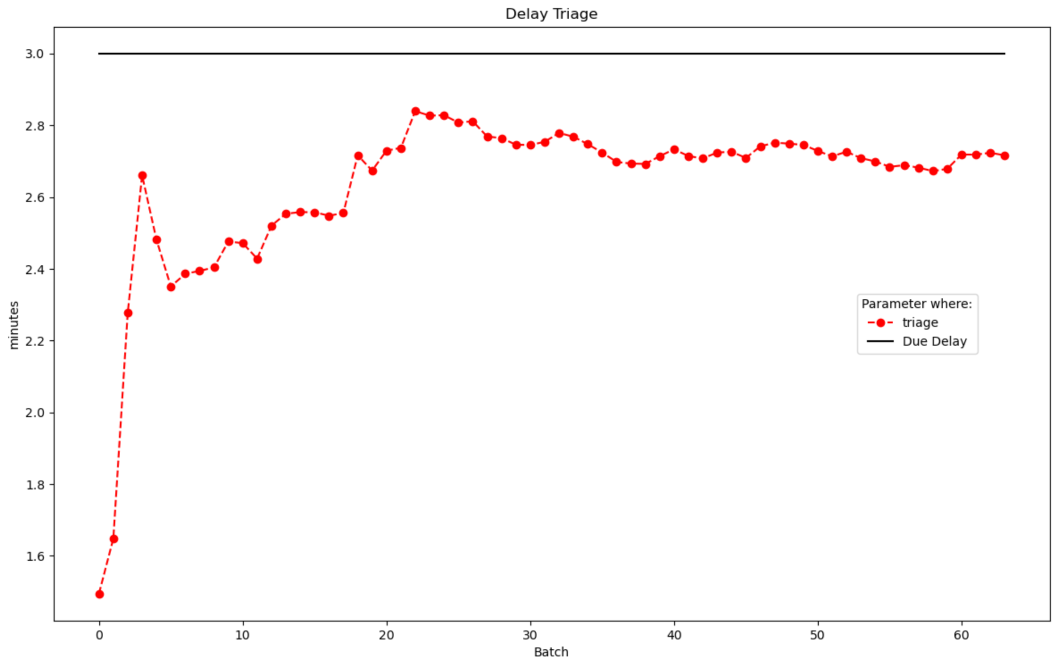
Questa soluzione utilizza tutto il budget, rispetta gli obiettivi per il triage e i codici rossi, così come per i problemi minori e traumatologia, ma il ritardo accumulato dai problemi medici è molto elevato, con la coda dei codici gialli che raddoppia i tempi di attesa rispetto all'obiettivo e la coda dei codici verdi che richiede tempi di attesa di quasi due giorni. Questa soluzione presenta un punteggio molto basso e non è decisamente una soluzione buona per il sistema.
Abbiamo provato ad aumentare di uno i problemi medici andando a diminuire sul triage e i problemi minori:

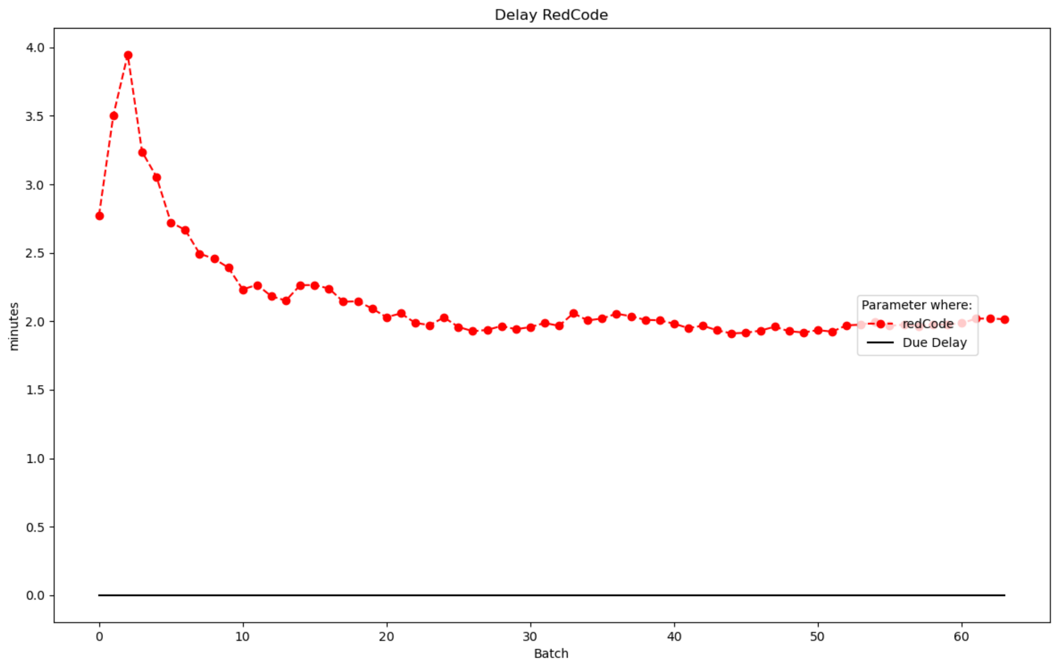
- Triage: 1
- Codici rossi: 2
- Traumatologia: 3
- Problemi medici: 4
- Problemi minori: 7

Questa soluzione rientra nel budget usandone 300€ di meno, le code dei problemi medici sembrano avere meno problemi, con i codici gialli che rientrano nei tempi e i codici verdi che sforano di una mezz'ora, ma il triage risulta problematico con più di 100 minuti di attesa, quando ci aspettavamo non più di 15 minuti di attesa totali. Possiamo aumentare nuovamente il triage diminuendo i serventi sui problemi minori, che per ora hanno molto margine, circa 20 minuti per i codici gialli, 40 minuti per i codici verdi e 50 minuti per i codici bianchi, che hanno tempi di attesa molto minori rispetto all'obiettivo.
La nuova soluzione può essere quindi:

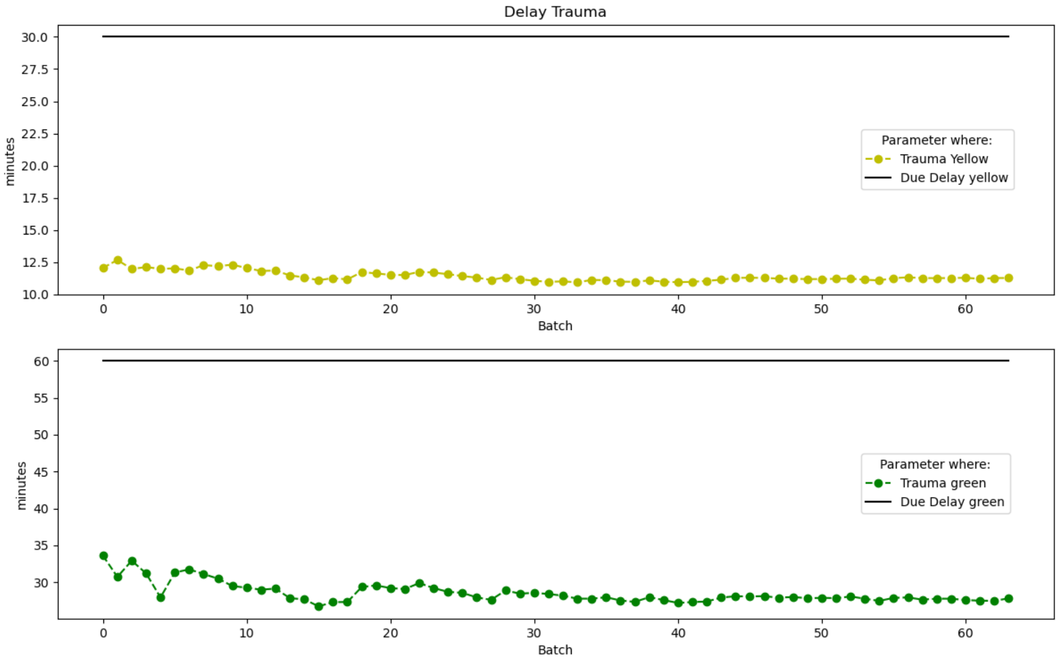
- Triage: 2
- Codici rossi: 2
- Traumatologia: 3
- Problemi medici: 4
- Problemi minori: 6

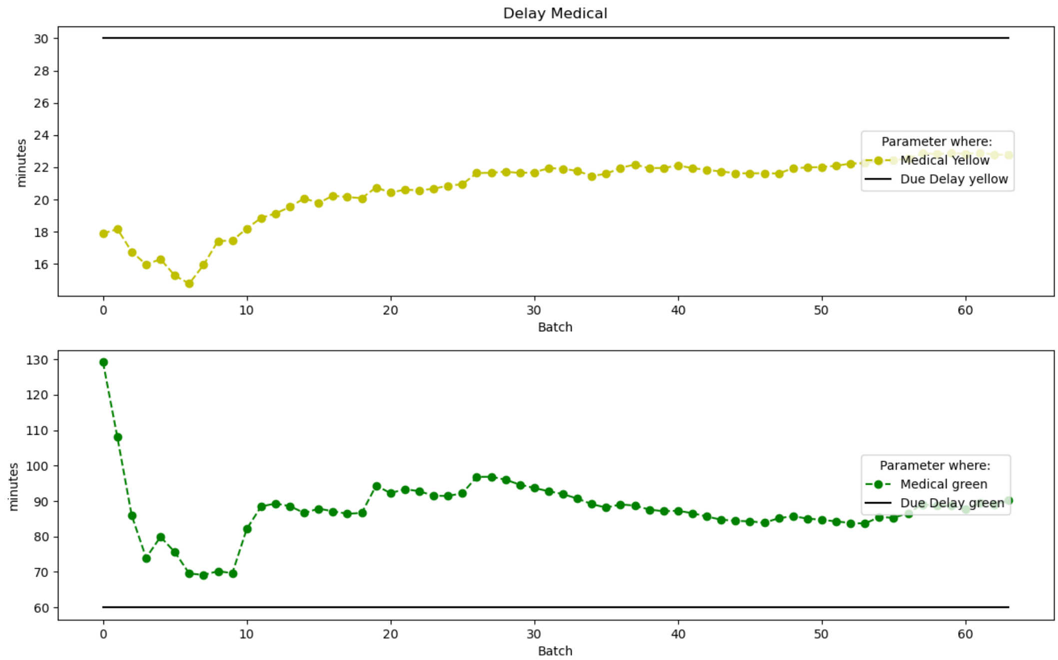
Questa soluzione per ora utilizza la percentuale minore di budget, con un avanzo di 1400€, il triage rispetta i tempi, i codici rossi restano invariati, traumatologia e problemi medici rispettano i tempi per i codici gialli, c'è un ritardo di una mezz'ora per i codici verdi

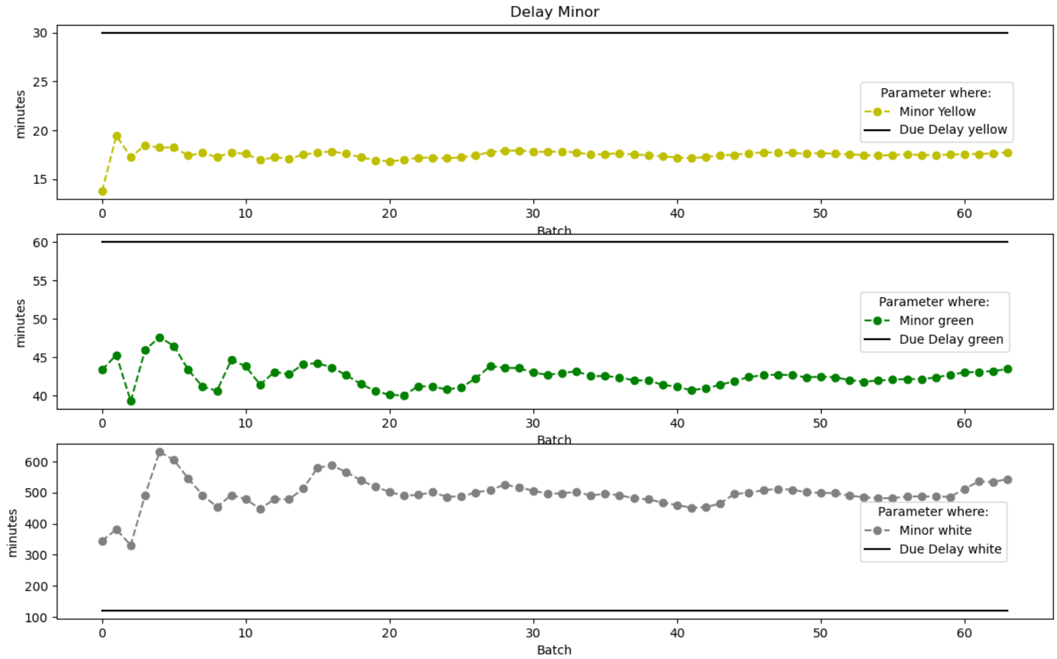




nei problemi medici. I problemi minori rispettano i tempi tranne che per i codici bianchi. Questa sembra la soluzione migliore dato che le uniche code che subiscono ritardi sono quelle a bassa priorità, dove tempi lunghi sono molto più accettabili.
I grafici sottostanti mostrano l'andamento del delay per i diversi centri, e il loro posizionamento rispetto alla soglia limite, segnata in nero. Si può vedere come anche il grafico dei codici rossi sembra accumulare ritardi, nonostante non fosse stato nominato. Questo è dovuto al fatto che la soglia per questo servente è zero, e portare a zero il ritardo richiede un numero estremamente elevato di serventi. Il ritardo medio si aggira sui due minuti che è comunque accettabile.


Delay Trauma
minutes
Batch
Parameter where:
Trauma Yellow
Due Delay yellow
minutes
Batch
Parameter where:
Trauma green
Due Delay green


Delay Medical
minutes
Batch
Parameter where:
Medical Yellow
Due Delay yellow
minutes
Batch
Parameter where:
Medical green
Due Delay green



La soluzione con meno serventi possibili che rispetta tutti i requisiti è:
- Triage: 2
- Codici rossi: 2
- Traumatologia: 3
- Problemi medici: 5
- Problemi minori: 7

Realizzabile soltanto con un aumento di budget di 5600€, quindi non applicabile con il budget a nostra disposizione ma che non richiede aumenti troppo sostanziali.
Per migliorare ancora di più avrebbe senso aumentare il numero di serventi per i codici rossi a quattro, in modo che la probabilità di morte sia praticamente dello 0% e l'attesa in coda media di qualche millisecondo, quindi praticamente zero. Questo miglioramento, che non diminuirebbe però di molto le attese, sarebbe realizzabile con un aumento del budget di 14.000€, portando l'aumento totale per il sistema ottimo a più di 20.000€ che inizia ad essere decisamente sostanziale.

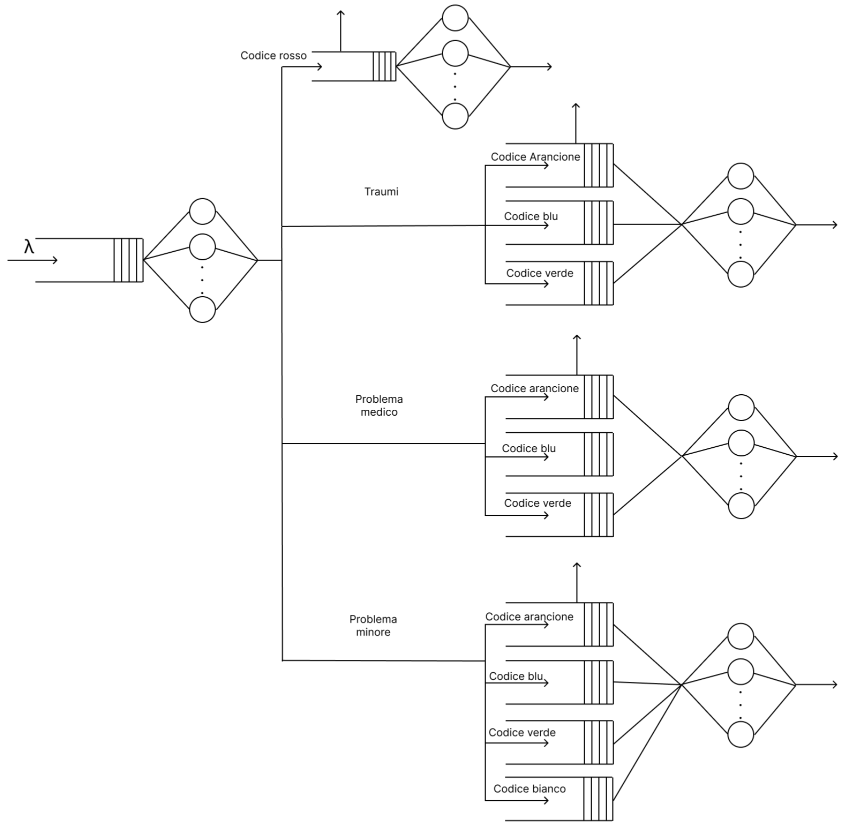
Una volta ottenuta questa configurazione con la simulazione a orizzonte infinito, siamo andati ad analizzare i dati prodotti dalla simulazione a orizzonte finito, che simuliamo sulle 24 ore di una giornata (1440 minuti). Avendo tassi di arrivo non troppo elevati, non arriviamo ad una situazione di convergenza per tutti i centri, possiamo però andare a controllare che i delay restino sotto la soglia che ci siamo prefissati per la configurazione scelta.

- TRIAGE: il delay medio delle varie repliche è 2.423332 ± 0.297579, che resta inferiore rispetto al limite prefissato.
- CODICI ROSSI: il delay medio è 0.643967 ± 1.159758, sempre leggermente superiore rispetto allo zero prefissato ma migliore rispetto ai due minuti dell'orizzonte infinito
- CODICI GIALLI: in traumatologia il delay dei codici gialli è di 9.190521 ± 3.469533, nei problemi medici di 13.535712 ± 5.700493 e nei problemi minori di 11.692292 ± 2.542884, in tutti i casi sotto i 30 minuti prefissati
- CODICI VERDI: in traumatologia il delay dei codici verdi è di 26.536057 ± 11.119662, nei problemi medici di 34.503581 ± 19.414968 e nei problemi minori di 27.339114 ± 8.309207, in tutti i casi di molto sotto i 60 minuti previsti. Rispetto alla simulazione a tempo infinito, anche i codici verdi nei problemi medici sembrano rispettare i requisiti se si va a considerare soltanto un tempo limitato
- CODICI BIANCHI: i codici bianchi subiscono un delay medio di 109.451330 ± 45.078767, che, anche se al limite superiore dell'intervallo di confidenza supera il valore obiettivo di 120 minuti, mediamente vi rientra, mostrando anche qui che se viene considerato solo un tempo limitato la combinazione scelta per la distribuzione dei serventi risulta sicuramente ancora adeguata, e anzi sembra non richiedere altri miglioramenti.

Modello migliorativo

Descrizione

Il sistema migliorato resta molto simile a quello attuale ma va a dividere i codici gialli in due code differenti, coda arancione e coda blu. Il criterio per la divisione è mandare coloro che hanno un rischio di decesso nella coda arancione e gli altri nella coda blu. Non abbiamo dati per come potrebbe essere effettuata questa divisione, ma dato che i dati evidenziano come avere casi gravi sia più raro, sembra sensato effettuare una divisione che non divida i codici gialli mandandone la metà in ognuna delle nuove code, ma avere una percentuale maggiore di codici blu e minore di codici arancioni. Un'idea potrebbe essere fare delle prove mandandone lo 0.36788 nei codici arancioni e lo 0.63212 nei codici blu, dividendo come una funzione esponenziale. L'obiettivo è andare a vedere se è possibile diminuire le percentuali di morte andando ad inserire questo miglioramento.
Il sistema diventa il seguente:



Il simulatore a livello di modello concettuale, delle specifiche o computazionale va ad a mantenersi molto simile a quello esistente con un codice di meno, andiamo ad aggiungere ai centri di traumatologia, problemi medici e problemi minori le variabili per mantenere quanti codici arancioni e blu ci sono nel sistema e in servizio andando a rimuovere quelle relative ai codici gialli. A livello computazionale, abbiamo fatto in modo che, se il codice assegnato risulta essere giallo, questo viene immediatamente ridiviso tra arancioni e blu, permettendoci di poter andare a modificare più velocemente le percentuali di divisione tra arancioni e blu per verificare se un cambiamento nel criterio di suddivisione porta a miglioramenti o peggioramenti. A livello di dati, tolta la percentuale di divisione non abbiamo bisogno di altri dati, a livello algoritmico, bisogna cambiare i completamenti del triage e dei centri affetti dalla nuova suddivisione. Il completamento del triage diventa il seguente:

- Svuotamento di un server e diminuzione dei job nel centro
- Se sono presenti job in coda inserimento di un job del server appena liberato con quindi generazione di un nuovo tempo di completamento
- Assegnazione di un codice al job terminato
- In base al codice assegnato c'è una diversa gestione:

CODICE ROSSO
Inserimento di un nuovo job nel nodo dei codici rossi
Se è presente un server libero
occupazione del server e generazione del tempo di completamento
Altrimenti
generazione di una percentuale: se ci troviamo al di sopra della soglia di decesso inserimento del job in coda, se siamo al di sotto c'è un decesso prima che il job sia preso in carico
CODICE GIALLO
Suddivisione tra codici arancioni e blu

Scelta del nodo che se ne occuperà, tra i tre possibili
Inserimento di un nuovo job nel nodo selezionato,
Nella coda a priorità maggiore se arancione
Nella coda di secondo livello se blu
Se è presente un server libero nel nodo, inserimento del job in servizio e generazione di un tempo di completamento per quel job
Se non c'è un server disponibile e il codice è arancione
generazione di una percentuale: se ci troviamo al di sopra della soglia di decesso inserimento del job in coda, se siamo al di sotto c'è un decesso prima che il job sia preso in carico
CODICE VERDE
Gestione identica a quella dei codici gialli, ma il job viene inserito nella coda a probabilità minore in traumatologia o per problemi medici, nella coda a priorità intermedia per i problemi minori
CODICI BIANCHI
Inserimento di un nuovo job nel nodo dei problemi minori, nella coda a priorità minore
Se è presente un server libero, occupazione del server e generazione del tempo di completamento per il job entrato in servizio

Dove ciò che cambia è la gestione dell'arrivo di un codice giallo. Anche nei completamenti di traumatologia, problemi medici e problemi minori cambia solo la gestione dei codici gialli. I completamenti in traumatologia e problemi medici diventano i seguenti:

- Svuotare il server
- Controllare la presenza di job nella coda dei codici arancioni
- Se presenti
  Inserimento di un job dalla coda nel server appena svuotato e generazione del tempo di completamento successivo
- Altrimenti
  Controllare la presenza di un job nella coda dei codici blu
  Se presenti
  Inserimento di un job dalla coda nel server appena svuotato e generazione del tempo di completamento successivo
  Altrimenti
  Controllare la presenza di un job nella coda dei codici verdi
  Se presenti
  Inserimento di un job dalla coda nel server appena svuotato e generazione del tempo di completamento successivo

La gestione dei problemi minori è la stessa di traumi e problemi medici salvo che per la coda bianca alla fine.

A livello computazionale, come già detto, abbiamo bisogno di uno stream di numeri random aggiuntivo per la divisione tra arancioni e blu, per cui abbiamo usato lo stream 10 dalla libreria rngs.h e anche in questo caso abbiamo implementato la probabilità come una variabile uniforme tra zero e cento, se il numero ottenuto è sotto la percentuale dei codici arancioni diventa arancione, altrimenti blu.

Verifica

Dopo aver scritto il codice del nuovo simulatore è stato necessario andare a verificare nuovamente le statistiche per i centri toccati dal cambiamento, quindi traumatologia, problemi medici e problemi minori. Per gli altri due centri l'unica cosa necessaria è vedere che i valori ottenuti non cambino tra il simulatore e il simulatore migliorato. Usiamo la stessa combinazione di serventi usata per la verifica del simulatore precedente, quindi due per triage e codici rossi, tre per traumatologia, quattro per i problemi medici e sette per i problemi minori.

Traumatologia

$\lambda = \lambda_{triage} * p_{trauma} * (p_{giallo} + p_{verde}) = 0.09 * 0.267 * (0.184 + 0.6071) = 0.01901\ job/min$

$E(S_i) = 93.4\ min$

$N = 3$

$E(S) = \frac{E(S_i)}{N} = \frac{93.4}{3} = 31.13333\ min$

$\mu = \frac{1}{E(S)} = \frac{1}{31.13333} = 0.03212\ job/min$

$\rho = \frac{\lambda}{\mu} = \frac{0.01901}{0.03212} = 0.59184$ Il valore ottenuto con il simulatore è 0.587792 ± 0.006014

In questo nodo abbiamo tre code, la coda 1 dei codici arancioni la coda 2 dei codici blu e la coda 3 dei codici verdi, vanno quindi calcolate le probabilità e l'utilizzazione, così come il tempo in coda ed in servizio, per ogni coda.

$p_1 = \frac{p_{gialli}}{p_{gialli} + p_{verdi}} * p_{arancioni} = \frac{18.4}{18.4 + 60.71} * 0.367879 = 0.085564$

$\rho_1 = \rho * p_1 = 0.59184 * 0.085564 = 0.05064$ (Simulatore 0.041186 ± 0.001957)

$p_2 = \frac{p_{gialli}}{p_{gialli} + p_{verdi}} * p_{blue} = \frac{18.4}{18.4 + 60.71} * 0.632120 = 0.147023$

$\rho_2 = \rho * p_2 = 0.59184 * 0.147023 = 0.08701$ (Simulatore 0.095350 ± 0.002523)

$p_3 = \frac{p_{verdi}}{p_{gialli} + p_{verdi}} = \frac{60.71}{18.4 + 60.71} = 0.76741$

$\rho_3 = \rho * p_3 = 0.59184 * 0.76741 = 0.45418$ (Simulatore 0.452168 ± 0.005193)

Per poter calcolare i tempi in coda, essendo un multiserver, è comunque necessario il valore di P_q

$$P_q = \frac{(N\rho)^N}{N!(1-\rho)} * \left(\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!} + \frac{(N\rho)^N}{N!(1-\rho)}\right)^{-1} = \frac{(3*0.59184)^3}{3!*(1-0.59184)} * \left(\sum_{i=0}^{2} \frac{(3*0.59184)^i}{i!} + \frac{(3*0.59184)^3}{3!*(1-0.59184)}\right)^{-1} = 0.34435$$

$E(T_{q_1}) = \frac{P_q E(S)}{1 - \rho_1} = \frac{0.34435 * 31.13333}{1 - 0.05064} = 11.29262\ min$ (Simulatore 11.397954 ± 1.034238)

$E(T_{q_2}) = \frac{P_q E(S)}{(1-\rho_1)(1-\rho_1-\rho_2)} = \frac{0.34435 * 31.13333}{(1-0.05064)(1-0.05064-0.08701)} = 13.09517\ min$
(Simulatore 12.537875 ± 0.714752)

$$E(T_{q_3}) = \frac{P_q E(S)}{(1-\rho)(1-\rho_1-\rho_2)} = \frac{0.34435 * 31.13333}{(1-0.59184)(1-0.05064-0.08701)} = 30.45872\ min$$

(Simulatore 29.662747 ± 2.498538)

$$E(T_{s_1}) = E(T_{q_1}) + E(S_i) = 11.29262 + 93.4 = 104.69262\ min$$

(Simulatore 101.785651 ± 2.797280)

$$E(T_{s_2}) = E(T_{q_2}) + E(S_i) = 13.09517 + 93.4 = 106.49517\ min$$

(Simulatore 105.596730 ± 1.806268)

$$E(T_{s_3}) = E(T_{q_2}) + E(S_i) = 30.45872 + 93.4 = 123.85872\ min$$

(Simulatore 122.347703 ± 3.015000)

$$E(T_q) = p_1 E(T_{q_1}) + p_2 E(T_{q_2}) + p_3 E(T_{q_3}) = 26.26619\ min$$

$$E(T_s) = E(T_q) + E(S_i) = 119.66619\ min$$

I valori del simulatore per il tempo in coda medio e il tempo totale medio sono di 25.614632 ± 2.014997 e 118.223520 ± 2.504792. I tre tempi di servizio, per le tre code e medio, sono 90.387697 ± 2.613558, 93.058856 ± 1.675123, 92.684955 ± 0.917884 e 92.608888 ± 0.83805, tutti corrispondenti all'E(S).

$E(N_{q_1}) = \lambda * p_1 * E(T_{q_1}) = 0.01901 * 0.085564 * 11.213880 = 0.01824$ Il valore del simulatore è 0.015569 ± 0.001553

$E(N_{q_2}) = \lambda * p_2 * E(T_{q_2}) = 0.01901 * 0.147023 * 13.09517 = 0.03660$ Il valore del simulatore è 0.038730 ± 0.002564

$E(N_{q_3}) = \lambda * p_3 * E(T_{q_3}) = 0.01901 * 0.76741 * 30.45872 = 0.44435$ Il valore del simulatore è 0.434459 ± 0.036840

$E(N_{s_1}) = \lambda * p_1 * E(T_{s_1}) = 0.01901 * 0.085564 * 104.69262 = 0.17029$ Il valore del simulatore è 0.139127 ± 0.0064553

$E(N_{s_2}) = \lambda * p_2 * E(T_{s_2}) = 0.01901 * 0.147023 * 106.49517 = 0.29764$ Il valore del simulatore è 0.324779 ± 0.008826

$E(N_{s_3}) = \lambda * p_2 * E(T_{s_3}) = 0.01901 * 0.76741 * 123.85872 = 1.80691$ Il valore del simulatore è 1.790963 ± 0.046091

$E(N_q) = \lambda * E(T_q) = 0.01901 * 26.26619 = 0.49932$ Il valore del simulatore è 0.488386 ± 0.039025

$E(N_s) = \lambda * E(T_s) = 0.01901 * 119.66619 = 2.27485$ Il valore del simulatore è 2.251760 ± 0.050860

Problemi medici

$$\lambda = \lambda_{triage} * p_{pmed} * (p_{giallo} + p_{verde}) = 0.09 * 0.247 * (0.184 + 0.6071) = 0.0176\ job/min$$

$$E(S_i) = 165.9\ min$$

$$N = 4$$

$$E(S) = \frac{E(S_i}{N} = \frac{165.9}{4} = 41.475\ min \qquad \mu = \frac{1}{E(S)} = \frac{1}{41.475} = 0.0241\ job/min$$

$\rho = \frac{\lambda}{\mu} = \frac{0.0176}{0.0241} = 0.73029$ Il valore ottenuto dal simulatore è di 0.730926 ± 0.008591

$$p_1 = \frac{p_{gialli}}{p_{gialli} + p_{verdi}} * p_{arancione} = \frac{18.4}{18.4 + 60.71} * 0.367879 = 0.085564$$

$\rho_1 = p_1 * \rho = 0.085564 * 0.73029 = 0.06249$ (Simulatore 0.053509 ± 0.002217)

$$p_2 = \frac{p_{gialli}}{p_{gialli} + p_{verdi}} * p_{blu} = \frac{18.4}{18.4 + 60.71} * 0.632121 = 0.147023$$

$\rho_2 = p_1 * \rho = 0.147023 * 0.73029 = 0.10737$ (Simulatore 0.116575 ± 0.003276)

$$p_3 = \frac{p_{verdi}}{p_{gialli} + p_{verdi}} = \frac{60.71}{18.4 + 60.71} = 0.76741$$

$\rho_3 = 0.76741 * 0.73029 = 0.56043$ (Simulatore 0.561953 ± 0.006784)

$$P_q = \frac{(N\rho)^N}{N!(1-\rho)} * (\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!} + \frac{(N\rho)^N}{N!(1-\rho)})^{-1} = \frac{(4*0.73029)^4}{4!*(1-0.73029)} * (\sum_{i=0}^{3} \frac{(4*0.73029)^i}{i!} + \frac{(4*0.73029)^4}{4!*(1-0.73029)})^{-1} = 0.47683$$

$$E(T_{q_1}) = \frac{P_q E(S)}{(1-\rho_1)} = \frac{0.47683 * 41.475}{(1-0.06249)} = 21.07483\ min$$

(Simulatore 20.126731 ± 1.318164)

$$E(T_{q_2}) = \frac{P_q E(S)}{(1-\rho_1)(1-\rho_1-\rho_2)} = \frac{0.47683 * 41.475}{(1-0.06249)(1-0.06249-0.10737)} = 25.38707\ min$$

(Simulatore 23.954403 ± 1.403811)

$$E(T_{q_3}) = \frac{P_q E(S)}{(1-\rho_1-\rho_2)(1-\rho)} = \frac{0.47683 * 41.475}{(1-0.16986)(1-0.73029)} = 88.32865\ min$$

(Simulatore 85.844821 ± 9.552533)

$$E(T_{s_1}) = E(T_{q_1}) + E(S_i) = 21.07483 + 165.9 = 186.97483\ min$$

(Simulatore 185.678985 ± 5.029863)

$$E(T_{s_2}) = E(T_{q_2}) + E(S_i) = 25.38707 + 165.9 = 191.28707\ min$$

(Simulatore 189.241681 ± 4.286611)

$$E(T_{s_3}) = E(T_{q_3}) + E(S_i) = 88.32865 + 165.9 = 254.22865\ min$$

(Simulatore 252.245868 ± 10.142855)

$$E(T_q) = p_1 E(T_{q_1}) + p_2 E(T_{q_2}) + p_3 E(T_{q_3}) = 73.32531\ min$$

$$E(T_s) = E(T_q) + E(S_i) = 239.22531\ min$$

Anche in questo caso i risultati del simulatore sono coerenti, con 71.106294 ± 7.439697 come tempo in coda e 237.219357 ± 8.151894 come tempo nel sistema, nonostante la varianza, e quindi l'intervallo di confidenza, siano molto maggiori. I quattro tempi di servizio, per coda e totali, sono 165.552254 ± 4.778051, 165.287278 ± 3.780323, 166.401046 ± 1.406778 e 166.113063 ± 1.337167, tutti e tre coerenti.

$E(N_{q_1}) = \lambda * p_1 * E(T_{q_1}) = 0.0176 * 0.085564 * 21.07483 = 0.03174$ Il valore del simulatore è 0.026195 ± 0.002028

$E(N_{q_2}) = \lambda * p_2 * E(T_{q_2}) = 0.0176 * 0.147023 * 25.38707 = 0.06569$ Il valore del simulatore è 0.068096 ± 0.004718

$E(N_{q_3}) = \lambda * p_3 * E(T_{q_3}) = 0.0176 * 0.76741 * 88.32865 = 1.19300$ Il valore del simulatore è 1.166760 ± 0.135459

$E(N_{s_1}) = \lambda * p_1 * E(T_{s_1}) = 0.0176 * 0.085564 * 186.97483 = 0.28158$ Il valore del simulatore è 0.240232 ± 0.009953

$E(N_{s_2}) = \lambda * p_2 * E(T_{s_2}) = 0.0176 * 0.147023 * 191.28707 = 0.49394$ Il valore del simulatore è 0.534396 ± 0.016090

$E(N_{s_3}) = \lambda * p_3 * E(T_{s_3}) = 0.0176 * 0.76741 * 254.22865 = 3.43372$ Il valore del simulatore è 3.414572 ± 0.152504

$E(N_q) = \lambda * E(T_q) = 0.0176 * 73.32531 = 1.29053$ Il valore del simulatore è 1.260468 ± 0.139443

$E(N_s) = \lambda * E(T_s) = 0.0176 * 239.22531 = 4.21037$ Il valore del simulatore è 4.184172 ± 0.165639

Problemi minori

$\lambda = \lambda_{triage} * (p_{min} * (p_{gialli} + p_{verdi}) + p_{bianchi}) = 0.09 * (19.85 + (18.40 + 60.71) * 48.6) = 0.05246\ job/min$

$E(S_i) = 105.6\ min$

$N = 7$

$$E(S) = \frac{E(S_i)}{N} = \frac{105.6}{7} = 15.08571\ min$$

$$\mu = \frac{1}{E(S)} = \frac{1}{15.08571} = 0.06629\ job/min$$

$\rho = \frac{\lambda}{\mu} = \frac{0.05246}{0.06629} = 0.79137$ (Il valore del simulatore è 0.797714 ± 0.009969)

In questo nodo abbiamo quattro code e quindi quattro diversi valori di probabilità, 1 per gli arancioni, 2 per i blu, 3 per i verdi e 4 per i bianchi

$$p_1 = \frac{p_{gialli} * p_{min}}{(p_{gialli} + p_{verdi}) * p_{min} + p_{bianchi}} * p_{arancioni} = \frac{18.4 * 0.486}{(18.4 + 60.71) * 0.486 + 19.85} * 0.367879 = 0.056430$$

$\rho_1 = p_1 * \rho = 0.056430 * 0.79137 = 0.04466$ (Simulatore 0.037023 ± 0.001958)

$$p_2 = \frac{p_{gialli} * p_{min}}{(p_{gialli} + p_{verdi}) * p_{min} + p_{bianchi}} * p_{blu} = \frac{18.4 * 0.486}{(18.4 + 60.71) * 0.486 + 19.85} * 0.632121 = 0.096962$$

$\rho_2 = p_2 * \rho = 0.096962 * 0.79137 = 0.07673$ (Simulatore 0.086473 ± 0.002592)

$$p_3 = \frac{p_{verdi} * p_{min}}{(p_{gialli} + p_{verdi}) * p_{min} + p_{bianchi}} = \frac{60.71 * 0.486}{(18.4 + 60.71) * 0.486 + 19.85} = 0.50611$$

$\rho_3 = p_3 * \rho = 0.50611 * 0.79137 = 0.40052$ (Simulatore 0.407976 ± 0.007249)

$$p_4 = \frac{p_{bianchi}}{(p_{gialli} + p_{verdi}) * p_{min} + p_{bianchi}} = \frac{19.85}{(18.4 + 60.71) * 0.486 + 19.85} = 0.34050$$

$\rho_4 = p_4 * \rho = 0.34050 * 0.79137 = 0.26946$ (Simulatore 0.267270 ± 0.004272)

$$P_q = \frac{(N\rho)^N}{N!(1-\rho)} * (\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!} + \frac{(N\rho)^N}{N!(1-\rho)})^{-1} = \frac{(7 * 0.79137)^7}{7! * (1 - 0.79137)} * (\sum_{i=0}^{6} \frac{(7 * 0.79137)^i}{i!} + \frac{(7 * 0.79137)^7}{7! * (1 - 0.79137)})^{-1} = 0.4652$$

$$E(T_{q_1}) = \frac{P_q * E(S)}{1 - \rho_1} = \frac{0.4652 * 15.08571}{1 - 0.04466} = 7.34594$$

(Simulatore 8.060445 ± 0.618247)

$E(T_{q_2}) = \frac{P_q * E(S)}{(1-\rho_1)(1-\rho_1-\rho_2)} = \frac{0.4652 * 15.08571}{(1-0.04466)(1-0.04466-0.07673)} = 8.36087$
(Simulatore 8.553777 ± 0.491161)
$E(T_{q_3}) = \frac{P_q * E(S)}{(1-\rho_1-\rho_2)(1-\rho_1-\rho_2-\rho_3)} = \frac{0.4652 * 15.08571}{(1-0.15339)(1-0.15339-0.40052)} = 18.58230$
(Simulatore 17.384608 ± 1.431516)
$E(T_{q_4}) = \frac{P_q * E(S)}{(1-\rho)(1-\rho_1-\rho_2-\rho_3)} = \frac{0.4652 * 15.08571}{(1-0.79137)(1-0.15339-0.40052)} = 75.40605$
(Simulatore 75.866191 ± 11.390080)
$E(T_{s_1}) = E(T_{q_1}) + E(S) = 7.34594 + 105.6 = 112.94594$
(Simulatore 112.323495 ± 3.551211)
$E(T_{s_2}) = E(T_{q_2}) + E(S) = 8.36087 + 105.6 = 113.96087$
(Simulatore 116.279544 ± 2.356286)
$E(T_{s_3}) = E(T_{q_3}) + E(S) = 18.58230 + 105.6 = 124.18230$
(Simulatore 123.331316 ± 2.340810)
$E(T_{s_4}) = E(T_{q_4}) + E(S) = 75.40605 + 105.6 = 181.00605$
(Simulatore 180.430495 ± 11.660854)
$E(T_q) = p_1 * E(T_{q_1}) + p_2 * E(T_{q_2}) + p_3 * E(T_{q_3}) + p_4 * E(T_{q_4}) = 36.35195$
(Simulatore 35.642855 ± 4.385445)
$E(T_s) = E(T_q) + E(S) = 36.35195 + 105.6 = 141.95195$
(Simulatore 141.205007 ± 4.947116)
Per quanto riguarda i tempi di servizio, che questa volta sono cinque data la coda in più, dal simulatore abbiamo ottenuto i valori: 104.263050 ± 3.479213, 107.725767 ± 2.221386, 105.946709 ± 1.192823, 104.564304 ± 1.266096 e 105.562152 ± 0.869603, tutti coerenti con il valore teorico.
$E(N_{q_1}) = \lambda * p_1 * E(T_{q_1}) = 0.05246 * 0.056430 * 7.34594 = 0.02175$ Il valore del simulatore è 0.020089 ± 0.001840
$E(N_{q_2}) = \lambda * p_2 * E(T_{q_2}) = 0.05246 * 0.096962 * 8.36087 = 0.04259$ Il valore del simulatore è 0.048473 ± 0.003344
$E(N_{q_3}) = \lambda * p_3 * E(T_{q_3}) = 0.05246 * 0.50611 * 18.58230 = 0.49337$ Il valore del simulatore è 0.471631 ± 0.041554
$E(N_{q_4}) = \lambda * p_4 * E(T_{q_4}) = 0.05246 * 0.34050 * 75.40605 = 1.34695$ Il valore del simulatore è 1.363993 ± 0.21375
$E(N_{s_1}) = \lambda * p_1 * E(T_{s_1}) = 0.05246 * 0.056430 * 112.94594 = 0.33436$ Il valore del simulatore è 0.279249 ± 0.014591
$E(N_{s_2}) = \lambda * p_2 * E(T_{s_2}) = 0.05246 * 0.096962 * 113.96087 = 0.57968$ I l valore del simulatore è 0.653786 ± 0.02033
$E(N_{s_3}) = \lambda * p_3 * E(T_{s_3}) = 0.05246 * 0.50611 * 124.18230 = 3.29711$ Il valore del simulatore è 3.327460 ± 0.085850
$E(N_{s_4}) = \lambda * p_4 * E(T_{s_4}) = 0.05246 * 0.34050 * 181.00605 = 3.23324$ Il valore del simulatore è 3.234883 ± 0.224144
$E(N_q) = \lambda * E(T_q) = 0.05246 * 36.35195 = 1.90701$ Il valore del simulatore è 1.903301 ± 0.249057

$E(N_s) = \lambda * E(T_s) = 0.05246 * 141.95195 = 7.44680$ Il valore del simulatore è 7.487296 ± 0.307365