Progetto di Performance Modeling of Computer Systems and Networks

Anno accademico 2021-2022

Adrian Petru Baba
Sara Da Canal
Matteo Federico

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, che rischiano la vita, 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.

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 120 minuti
- I codici bianchi di 240 minuti

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 mantenendo gli stessi tempi. I codici arancioni dovrebbero avere attesa di 15 minuti e blu di 45 minuti.

Modello concettuale

Stato:

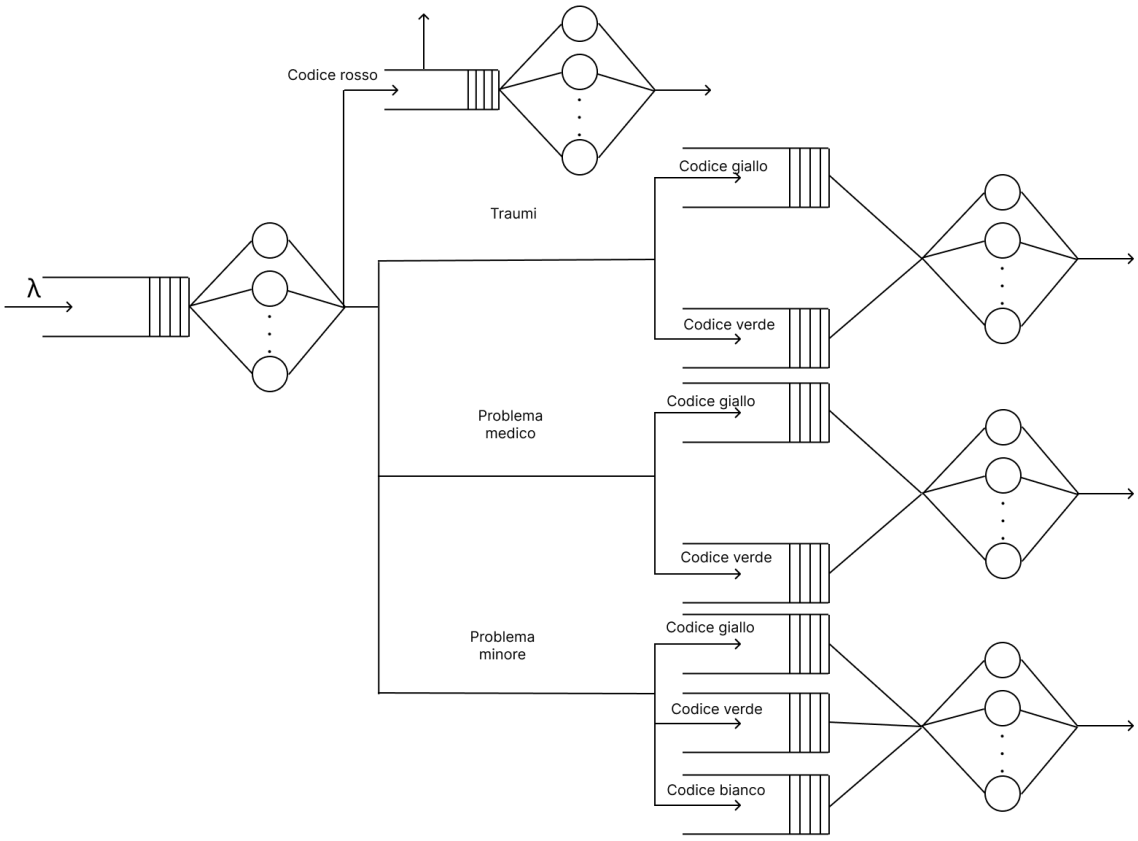
Per ogni istante di tempo t

- 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 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%.

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 nella coda dei codici rossi, 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, ha una probabilità del 5,2% 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 soglia di 5,2 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
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 e il 9 per stabilire in quale nodo andrà un codice verde. 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. 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.

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 512, 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.

Verifica

La verifica implica andare a controllare che il simulatore produca i risultati corretti per il modello considerato. Per fare questo bisogna andare a produrre dei risultati teorici e controllare che il simulatore produca risultati abbastanza simili, ovviamente con un margine di errore. Per la verifica, considerando che un margine di errore è accettabile, abbiamo considerato una singola run con un tempo lungo (500000 minuti), cosa non accettabile negli esperimenti. I parametri teorici considerati sono stati l'utilizzazione, il tempo di attesa in coda, per singola coda e medio nel caso del multi-coda, il tempo totale nel nodo e il tempo di servizio, che non è necessario calcolare a livello teorico essendo tra i valori di input, ma è comunque sensato controllare che il valore empirico sia vicino al valore teorico medio. Per effettuare la verifica, sono stati scelti dei valori fissi per il numero di serventi, che non è detto sia il valore ottimale, è stato semplicemente scelto un valore abbastanza grande da ottenere un sistema stazionario. I calcoli teorici sono stati effettuati per nodo:

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 esattamente 0.45

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

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

$E(T_q) = \frac{0.27931 * 5}{1-0.45} = 2.5392\ min$ Il risultato del simulatore è 2.72 minuti

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

I risultati del simulatore sono 10.04 minuti per il tempo di servizio e 12.76 minuti per il tempo del sistema.

Codici rossi

Dato che il triage è stazionario ed ha throughput λ,

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

Questo valore non è l'arrival rate effettivo dato che va tenuto conto degli abbandoni:

$\lambda' = \lambda * p_{dec} = 9.36 * 10^{-4} * (1 - 0.052) = 8.87328 * 10^{-4}\ job/min$

$E(S_i) = 225.5\ min$

$N = 2$

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

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

$\rho = \frac{\lambda}{\mu} = \frac{8.87328 * 10^{-4}}{8.86918 * 10^{-4}} = 0.10$ Il valore ottenuto dal simulatore è 0.11

$P_q = \frac{(N\rho)^N}{N!(1-\rho)} * (\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!} + \frac{(N\rho)^N}{N!(1-\rho)})^{-1} = \frac{(2*0.1)^2}{2*(1-0.1)} * (\sum_{i=0}^{1} \frac{(2*0.1)^i}{i!} + \frac{(2*0.1)^2}{2*(1-0.1)})^{-1} = 0.01818$

$E(T_q) = \frac{P_q E(S)}{1-\rho} = \frac{0.01818 * 112.75}{1-0.1} = 2.27991\ min$ Il risultato del simulatore è 2.89 minuti.

$E(T_s) = E(T_q) + E(S_i) = 2.27991 + 225.5 = 227.77991\ min$

I valori ottenuti dal simulatore per il tempo di servizio sono di 219.31 min e per il tempo nel sistema di 222.20 minuti. L'errore che abbiamo in questo caso è leggermente più elevato rispetto al triage, ma questo è giustificato dal fatto che i job in questa coda sono molto di meno a parità di tempo di esecuzione.

Traumatologia

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

$E(S_i) = 93.4\ min$

$N = 3$

$E(S) = \frac{E(S_i)}{N} = \frac{93.4}{3} = 31.3333\ min \quad \mu = \frac{1}{E(S)} = \frac{1}{31.3333} = 0.03191\ job/min$

$\rho = \frac{\lambda}{\mu} = \frac{0.01901}{0.03191} = 0.59565$ Il valore ottenuto con il simulatore è 0.58

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.

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

$$\rho_1 = \rho * p_1 = 0.59565 * 0.23259 = 0.13854$$ (Simulatore 0.13)

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

$$\rho_2 = \rho * p_2 = 0.59565 * 0.76741 = 0.45711$$ (Simulatore 0.45)

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

$$P_q = \frac{(N\rho)^N}{N!(1-\rho)} * (\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!} + \frac{(N\rho)^N}{N!(1-\rho)})^{-1} = \frac{(3 * 0.59565)^3}{3! * (1 - 0.59565)} * (\sum_{i=0}^{2} \frac{(3 * 0.59565)^i}{i!} + \frac{(3 * 0.59565)^3}{3! * (1 - 0.59565)})^{-1} = 0.34919$$

$$E(T_{q_1}) = \frac{P_q E(S)}{1 - \rho_1} = \frac{0.34919 * 31.3333}{1 - 0.13854} = 12.70077\ min$$ (Simulatore 11.54)

$$E(T_{q_2}) = \frac{P_q E(S)}{(1 - \rho_1)(1 - \rho)} = \frac{0.34919 * 31.3333}{(1 - 0.13854)(1 - 0.59565)} = 31.41034\ min$$

(Simulatore 28.22)

$$E(T_{s_1}) = E(T_{q_1}) + E(S_i) = 12.70077 + 93.4 = 106.10077\ min$$

(Simulatore 104.28)

$$E(T_{s_2}) = E(T_{q_2}) + E(S_i) = 31.41034 + 93.4 = 122.81034\ min$$

(Simulatore 119.70)

$$E(T_q) = p_1 E(T_{q_1}) + p_2 E(T_{q_2}) = 27.05871\ min$$

$$E(T_s) = E(T_q) + E(S_i) = 120.45871\ min$$

I valori del simulatore per il tempo in coda medio e il tempo totale medio sono di 24.42 e 116.19, abbastanza più bassi rispetto a quelli teorici, ma dato che l'utilizzazione è leggermente più bassa del valore teorico e il tempo di servizio (91.77) è nuovamente leggermente più basso, per quanto questi siano valori accettabili considerando che il simulatore usa una probabilità che non è detto sia identica al valore teorico, è abbastanza sensato che la somma di questi errori porti a errori maggiori nei tempi in coda e nel sistema.

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 0.72

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

$$\rho_1 = p_1 * \rho = 0.23259 * 0.73029 = 0.16986$$ (Simulatore 0.17)

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

$$\rho_2 = 0.76741 * 0.73029 = 0.56043$$ (Simulatore 0.55)

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

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

(Simulatore 20.13)

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

(Simulatore 86.79)

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

(Simulatore 186.24)

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

(Simulatore 251.57)

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

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

Anche in questo caso i risultati del simulatore sono leggermente più bassi ma coerenti, con 71.11 come tempo in coda e 236.20 come tempo nel sistema.

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 0.80)

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 0.13)

$$p_1 = \frac{p_{verdi} * p_{min}}{(p_{gialli} + p_{verdi}) * p_{min} + p_{bianchi}} = \frac{60.71 * 0.486}{(18.4 + 60.71) * 0.486 + 19.85} = 0.50611$$

$$\rho_2 = p_2 * \rho = 0.50611 * 0.79137 = 0.40052$$ (Simulatore 0.40)

$$p_1 = \frac{p_{bianchi}}{(p_{gialli} + p_{verdi}) * p_{min} + p_{bianchi}} = \frac{19.85}{(18.4 + 60.71) * 0.486 + 19.85} = 0.34050$$

$$\rho_3 = p_3 * \rho = 0.34050 * 0.79137 = 0.26946$$ (Simulatore 0.27)

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

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

(Simulatore 8.18)