Progetto di Simulazione di Sistemi
L.M. INFORMATICA A.A. 2019-2020

Heavy traffic limit for a tandem queue with identical service times

OMNeT++

Andrea Ercolessi (Matr. 0000953727)

Indice

1.Introduzione

Il modello analizzato rispecchia approssimativamente il lavoro che viene svolto nell'articolo di ricerca : Heavy traffic limit for a tandem queue with identical service times.

Uno dei modelli di accodamento è il cosiddetto, rete tandem costituito da due code FIFO (first-first-out-out), dove la prima è M / G / 1 con velocità di arrivo λ e ogni job riutilizza il tempo di servizio della prima coda quando viene processato nella coda di uscita. Tale studio è focalizzato sul comportamento che manifesta la coda di uscita.

2.Tool Utilizzati

Il framework utilizzato per la modellazione e la simulazione del sistema consiste in **Omnet++ versione 5.5.1** del quale viene riportata di seguito un immagine. Uno dei vantaggi principali di oment è che oltre ad essere un framework distribuito sotto la Licenza pubblica accademica, fornisce all'utilizzatore un ambiente di sviluppo già basato sulla tipologia di modello a code mettendo a disposizione alcune implementazioni software delle entità presenti nel modello, oltre che a numerosi tool grafici che facilitano la configurazione

Per meglio rendere interpretabili i dati ricavati dal software e quindi per effettuare in maniera più agevole delle analisi su quest'ultimi possiamo far affidamento su diverse librerie Python.

A tal proposito vengono descritte le principali librerie utilizzate per effettuare lo studio dei dati ed il Plot di essi :

1. **Pandas** è uno strumento di analisi e manipolazione dei dati open source veloce, potente, flessibile e facile che permette l'analisi dei dati. Inoltre mette a disposizione dei metodi per la manipolazione dei dati ".sca" ed ".vec" provenienti dalle simulazione di omnet

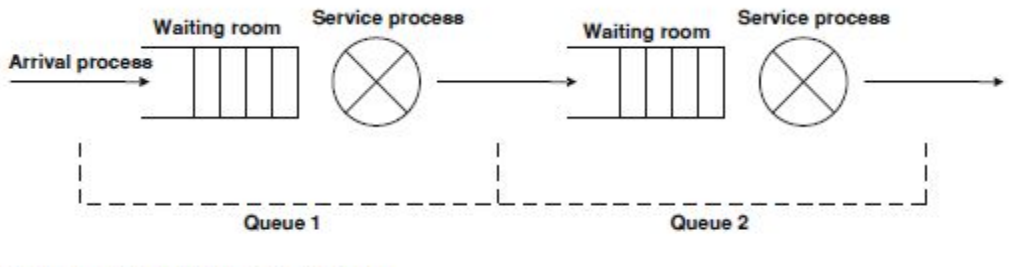
2. **Statistics** questo modulo fornisce funzioni per il calcolo di statistiche matematiche di dati numerici
3. **MatplotLib** è una libreria completa per la creazione di visualizzazioni statiche, animate e interattive

3.Modello

In questa sezione verrà descritto il modello implementato considerando due aspetti principali; la descrizione, ovvero la struttura del modello ed infine i parametri che sono stati utilizzati nello svolgimento della simulazione.

3.1 Descrizione

Il modello implementato prevede delle varianti rispetto all'articolo citato in precedenza, il sistema TandemQueue è schematicamente rappresentato nella successiva figura.



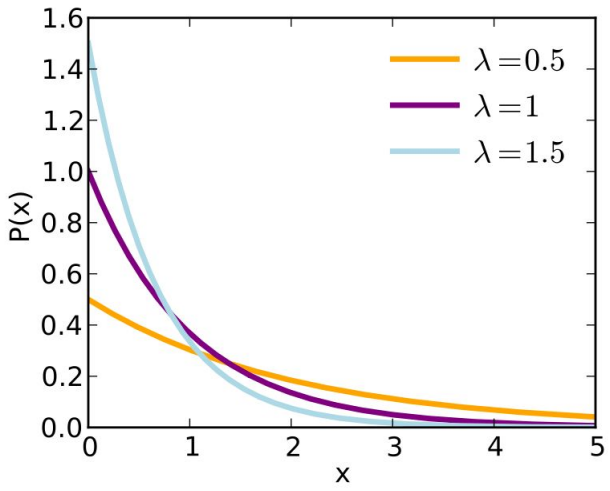
Il sistema è caratterizzato da due code e due serventi, nello specifico tali componenti sono così strutturati :

1. Per quanto riguarda la coda del primo servente, essa è a capacità illimitata e viene caratterizzata da una **policy** di tipo **SPTF**. Si tratta del Teorema noto come Teorema dei Tempi di Lavorazione più brevi / Shortest Processing Time elaborato nella Teoria della schedulazione che afferma che il tempo di attraversamento medio-average flow time, F , è minimizzato sequenziando i lotti in ordine non decrescente dei tempi di lavorazione $p_{[1]} \leq p_{[2]} \leq .... \leq p_{[n]}$. Il tempo di servizio al primo servente è caratterizzato da una distribuzione di probabilità che sarà descritta nel paragrafo successivo.
2. Mentre per quanto riguarda la coda del secondo servente è anch'essa a capacità illimitata gestita da una **policy FIFO(First-In-First-Out).** Una

particolarità di questo modello e che il tempo di servizio richiesto da un utente nel secondo servente è pari al tempo di servizio assegnato, allo stesso utente, nel primo servente.

3.2 Parametri

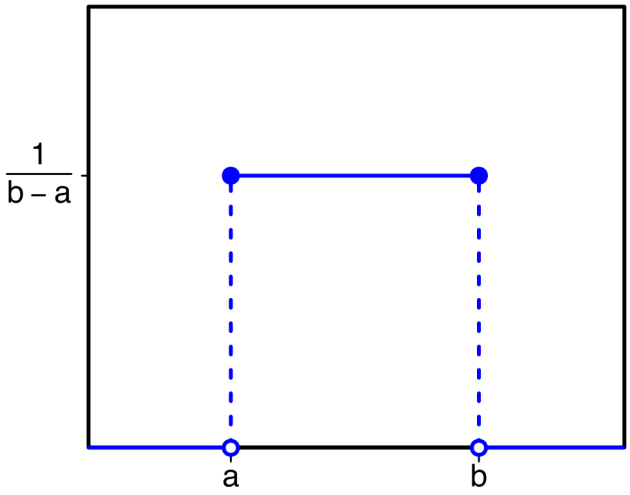
Analizzando nel particolare i parametri di questo sistema, gli “Arrival process”, i quali indicano i job modellati da tempi di interarrivo esponenziali di medie 1/λ sono rappresentati nella figura sottostante. La medesima distribuzione di tipo esponenziale, viene utilizzata per la definizione dei serventi considerando dei processi con media $1/\mu$.



I parametri della la prima simulazione utilizzati sono :

- λ=0.5 ----> μ=0.33; μ=0.25
- λ=0.7 ----> μ=0.33; μ=0.25
- λ=0.8 ----> μ=0.33; μ=0.25
- λ=1.0 ----> μ=0.33; μ=0.25

Per quanto riguarda la seconda configurazione esaminata, λ rimane una distribuzione di probabilità esponenziali, mentre i serventi sono caratterizzati da una distribuzione uniforme in cui il parametro μ varia in un intervallo [A;B]. Nella figura seguente riportiamo l'andamento della funzione densità di probabilità della distribuzione uniforme.



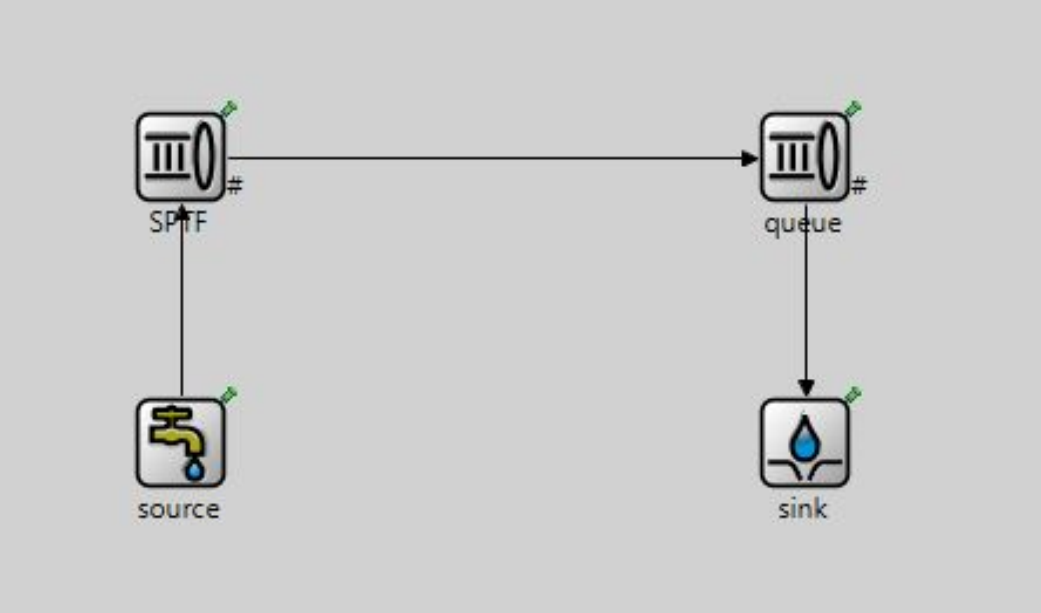
I parametri utilizzati per la seconda simulazione :

- λ=0.5 ----> μ=[0,16;1]; μ=[0.5;0.25]
- λ=0.7 ----> μ=[0,16;1]; μ=[0.5;0.25]
- λ=0.8 ----> μ=[0,16;1]; μ=[0.5;0.25]
- λ=1.0 ----> μ=[0,16;1]; μ=[0.5;0.25]

Considerando tutti i valori che i parametri da analizzare possono assumere, risultano possibili 16 configurazioni. In questo modo, è quindi possibile performare varie tipologie di analisi considerando diversi valori dei parametri che entrano in gioco nelle simulazioni in modo da analizzare un gran numero modelli.

4.Implementazione del modello

Per l'implementazione sono stati utilizzati i moduli implementati dalla libreria **Queuinglib**, apportando le opportune modifiche .

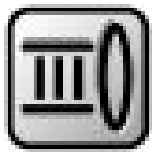


Utilizzando la precedente immagine come riferimento nei paragrafi seguenti andremo a descrivere i componenti che il software OMNet++ fornisce per la costruzione di un modello da analizzare.

4.1 Source

Il modulo Source si occupa di generare i job ad ogni interArrivalTime. Questo modulo, in funzione dei parametri selezionati per la descrizione del modello, genera i job necessari alle simulazioni.

4.2 Queue



Il modulo Queue standard si occupa di implementare un server con coda all'ingresso. Tale modulo può essere incontrato nel modello in due versioni una chiamata "SPTF " ed una "queue". Per quanto riguarda la prima è stato implementato il comportamento descritto dalla policy SPTF descritta precedentemente. Invece, per quanto riguarda le modifiche apportate alla classe "queue", per adattarla alle simulazioni considerate, possono essere riscontrata in ***handleMessage() della classe MyQueue.c*** .

Nella sezione successiva è possibile analizzare le modifiche apportate alla classe ***MyQueue.c :***

Nel codice sottostante viene inserito come parametro di un job il tempo di servizio richiesto da esso, dopo di che il job viene inserito nella coda.

```
    /*************************************************************************/
    simtime_t serviceTime =startService(jobServiced);
    //std::cout<<serviceTime<<endl ;
    double valore = (double)serviceTime.raw();
    cMsgPar * parametro = &(msg->addPar("Tempo di servizio"));
    parametro->setDoubleValue(valore);
    //std::cout<<"paramtro Fine di tutto : "<<valore<< std::endl ;
    serviceTimeVec.record(serviceTime);
    queue.insert(job);
    emit(queueLengthSignal, length());
    job->setQueueCount(job->getQueueCount() + 1);
}
```

Per effettuare l'insert del job, nella superclasse di ***queue,*** ossia ***cqueue,*** è presente un metodo chiamato ***comparator*** il quale detta la politica con il quale viene inserito il job nella coda. Per far si che i job inseriti rispettino la politica SPTF si è deciso di implementare una classe definita come ***Mycomaprator***. All'interno di questa classe viene ereditato il metodo ***comparator*** della superclasse ***cqueue*** e modificata in modo da avere il comportamento desiderato. Più precisamente vengono analizzati i tempi di servizio e viene effettuato uno swap dei job con i tempi di servizi minori in

modo da avere una lista di job ordinata in maniera non crescente. Il codice riportato nell'immagine sottostante descrive quanto appena descritto.

```
bool Mycomparator::less(omnetpp::cObject *a, omnetpp::cObject *b) {

        omnetpp::cMessage *msg = (omnetpp::cMessage *)(a);
        omnetpp::cMessage *mymsg = (omnetpp::cMessage *)(b);
        double val1=msg->par("Tempo di servizio").doubleValue();
        double val2=mymsg->par("Tempo di servizio").doubleValue();
        //std::cout<<"VALORE COMPARATOR : "<<val1<< std::endl;
        return val1<val2;
}

omnetpp::cQueue::Comparator* Mycomparator::dup() const {
    Mycomparator* p =new Mycomparator();
       return p;
}
```

Nel caso in cui siano, presenti dei job all'interno della coda, è stato implementato un metodo con il quale si considera il job ordinato e si processa.

```
jobServiced = getFromQueue();
emit(queueLengthSignal, length());
/*****************************************************************************/
simtime_t serviceTime;
cMsgPar * par;
try{
    double parServiceTime;
    par = &(jobServiced->par("Tempo di servizio"));
    parServiceTime =par->doubleValue();
    //int64_t converted =(int64_t)(parServiceTime);
    serviceTime.setRaw(parServiceTime);
    serviceTimeVec.record(serviceTime);
    scheduleAt(simTime()+serviceTime, endServiceMsg);
    return;
}catch(cRuntimeError e){
    serviceTime =startService(jobServiced);
    double valore = (double)serviceTime.raw();
    cMsgPar * parametro = &(msg->addPar("Tempo di servizio"));
    parametro->setDoubleValue(valore);
    std::cout<<"paramtro set 1: "<<valore<< std::endl ;
    serviceTimeVec.record(serviceTime);
    scheduleAt(simTime()+serviceTime, endServiceMsg);
}}
```

Nel modello descritto all'inizio della sezione 4 sono presenti due queue una con il comportamento che abbiamo descritto in precedenza mentre l'altra denominata

“queue” dove il comportamento di quest'ultima risulta essere più semplice di quanto descritto fino ad ora in quanto l’unico procedimento che viene effettuato è quello di modificare il metodo presente nella classe ***Queue.c*** in ***startService()*** per far si che il tempo di servizio richiesto da un utente nel secondo server è pari al tempo di servizio assegnato allo stesso utente, nel primo server

```
simtime_t Queue::startService(Job *job)
{
    // gather queueing time statistics
    simtime_t d = simTime() - job->getTimestamp();
    emit(queueingTimeSignal, d);
    job->setTotalQueueingTime(job->getTotalQueueingTime() + d);
    EV << "Starting service of " << job->getName() << endl;
    job->setTimestamp();
    return par("serviceTime").doubleValue();
}
```

4.3 Sink

Il modulo Sink si occupa di deallocare i Job usciti dal sistema e accogliere le statistiche sulla loro permanenza nel sistema utili all’analisi dei dati prodotti dalle simulazioni eseguite

4.4 Inserimento dei parametri

I parametri di configurazione della simulazione vengono definiti nel file con estensione ”.ini” . Dato il gran numero dei parametri da studiare si è deciso di dividere i parametri in base alle distribuzioni presenti e di conseguenza agli input da inserire all’interno della simulazione. Nella immagine sottostante viene mostrato, in che modo è stata effettuata la divisione dei parametri in modo da

lanciare le simulazione in base ad i nomi assegnati, che sono caratterizzati dall'uso delle parentesi quadre.

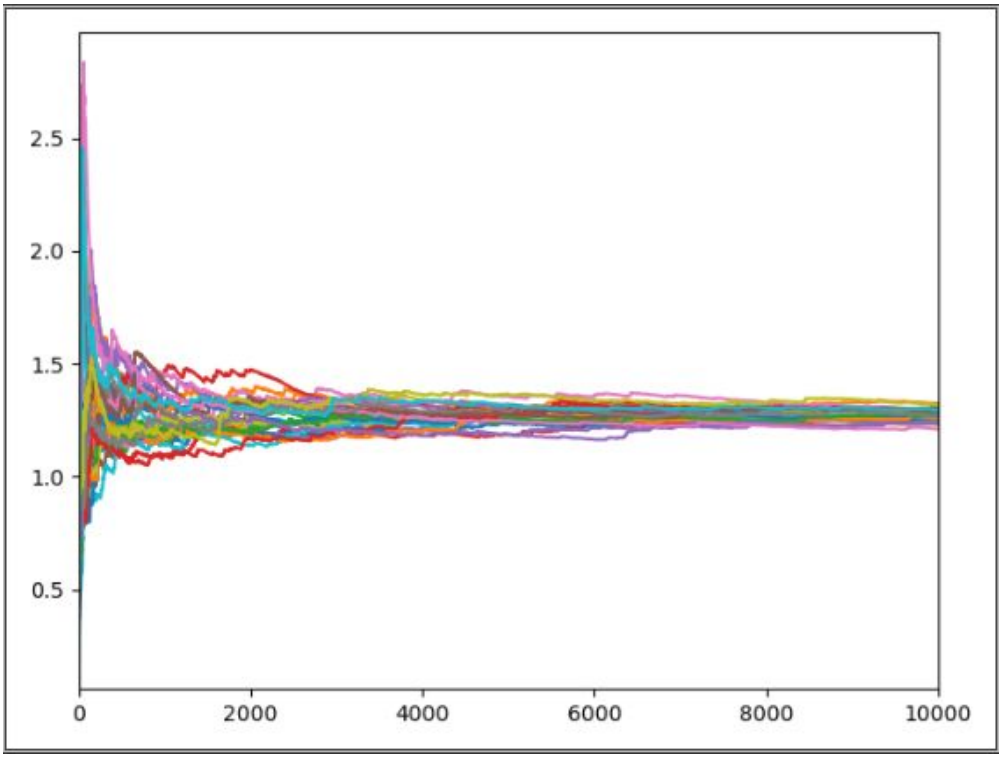
```
[General]
network = Network
[Config exponential]
warmup-period = 3500s
**.source.interArrivalTime = exponential(${lamb=0.5, 0.7, 0.8, 1.0}s)
**.queue.serviceTime =0s
**.SPTF.serviceTime = exponential(${expo=0.33,0.25}s)
sim-time-limit = 100000s
repeat = 30
[Config uniform]
warmup-period = 10000s
**.source.interArrivalTime = exponential(${lam=0.5, 0.7, 0.8, 1.0}s)
**.queue.serviceTime =0s
**.SPTF.serviceTime = uniform(1s,0.16s)
sim-time-limit = 100000s
repeat = 30
[Config uniformUno]
warmup-period = 10000s
**.source.interArrivalTime = exponential(${lam=0.5, 0.7, 0.8, 1.0}s)
**.queue.serviceTime =0s
**.SPTF.serviceTime = uniform(0.5s,0.25s)
sim-time-limit = 100000s
repeat = 30
```

5.Statistica

Per effettuare un buono studio della misurazione statistica sui dati sono state effettuate 30 ripetizioni per ogni configurazione riportata in precedenza per un tempo complessivo di 100000s

5.1 Transiente iniziale

Nel momento in cui un sistema inizia da poco ad essere operativo, questo risulta fortemente influenzato dallo stato iniziale e dal tempo trascorso prima dell'attivazione. Questa condizione del sistema è detta transitoria. Trascorso un lasso di tempo iniziale il sistema si stabilizza e raggiunge condizioni stazionarie. Durante gli esperimenti visualizzando i risultati ottenuti è stato notato un comportamento anomalo nella fase iniziale. Di seguito riportiamo una figura che mostra il fenomeno



È evidente che nelle prime fasi di simulazione il sistema è instabile. Valutando tutte le configurazioni si è deciso di eliminare questa fase e analizzare il sistema da quando va a regime. La fase transiente iniziale viene rimossa grazie alla funzionalità warmup-period di OMNeT++. La fase transiente individuata per questo sistema è di 3500 unità di tempo simulato, su un totale di 100000 unità di tempo totale della simulazione.

5.2 Intervalli di confidenza

In questa paragrafo verrà descritto un intervallo nel quale ci aspettiamo stia il parametro da stimare con un elevato grado di fiducia. Questa "fiducia" è assegnata in termini probabilistici e viene detta confidenza. Tale intervallo si dice intervallo di confidenza e la probabilità (che indicheremo con 1-α) assegnata viene detta livello di significatività (o livello di confidenza). Usualmente si sceglie come livello di significatività il 95% o il 99%.

Una stima puntuale del valore atteso μ è data dal valore $\bar{x}$ assunto dalla variabile $\overline{X}_n$ nel campione. Un intervallo di confidenza, con livello di significatività del 95%, è un intervallo aleatorio

$$(\overline{X}_n - \delta, \overline{X}_n + \delta)$$

con δ scelto in modo tale che

$$P\ (\overline{X}_n - \delta < \mu <, \overline{X}_n + \delta) = 0,95$$

ossia tale che la probabilità di sbagliare sia pari a α=0.05 e quindi bassa.

Per effettuare il calcolo dell'intervallo di confidenza e stato creato nello script python un metodo nel quale dato come input il vettore con i risultati da calcolare e la percentuale di intervallo di confidenza da calcolare restituisce l'intervallo di confidenza. Questo viene effettuato tramite una distribuzione di probabilità

T_Student con 29 gradi di libertà e la percentuale del 95%. Lo script implementato per l'analisi dei dati di output in particolare utilizza il metodo, già implementato nella libreria Python sciPy, stats.t.ppf() per ottenere il valore della tabella dei percentili con i gradi di libertà sopra specificati.

5.3 Risultati

Tutti i dati prodotti da OMNeT++ sono stati salvati su file di tipo *.vec, *.sca.Tramite scavetool sono stati estratti sia i vettori di tutte le osservazione simulate, sia i valori scalari generati dal sistema. I file generati sono stati analizzati successivamente tramite lo script python riportati sotto il nome di tandemQueue.py

5.3.1 Risultati

Di seguito vengono riportate tutte le analisi di prestazioni richieste da analizzare:

1. **Mediana della distribuzione del tempo di risposta del sistema:** viene effettuato lo studio dei ***lifeTime:vector*** che contiene i tempi di permanenza nel sistema di tutti gli utenti ordinati per tempo di arrivo all'elemento Sink Questo vettore è un vettore privo di transiente iniziale grazie alla funzione precedentemente descritta ed in seguito è stato estratto il valore di mediana.

2. **Mediana della distribuzione del tempo di risposta di ogni singolo servente:** per questo studio sono stati studiati i ***serviceTime:vector*** inerenti alla coda ***Network.SPTF*** di questi è stato calcolato il valore della mediana.

3. **Tempo medio di permanenza nel sistema dei job:** per valutare questo indice di prestazione è stato utilizzato il vettore ***lifeTime:vector*** descritto in precedenza però in questo caso è stata calcolata la semplice media.

4. **Tempo massimo (minimo) di permanenza nel sistema dei job :** per studiare questi due parametri si è ricorso all'uso del vettore ***lifeTime:vector*** ma con la differenza che è stato preso il valore massimo e minimo presenti all'interno di esso.

5. **Fattore di utilizzo dei singoli server:** riguardo questo indice di prestazione è stato utilizzato un valore scalare che viene già calcolato da OMNeT++ per ognuno dei serventi, denominato ***busy:timeavg.*** Ma data l'implementazione del modello basta studiare quello della coda SPTF in quanto risultano avere gli stessi tempi per quanto descritto nel paragrafo tre dove vengono spiegate le caratteristiche del modello.

A tal proposito nel file presente nella cartella sono disponibili i risultati ottenuti secondo gli indici di prestazioni sopra descritti e per ogni combinazione di parametro descritto nella sezione 3.2. Nell'immagine sottostante viene riportata una piccola parte dei risultati presenti nel file "Res.txt" per mostrare la struttura che essi compongono.

```
|-------------------------------------------------------------------------------
Exponetial---> λ= 2.0 μ=3.0:
Mediana della distriubuzione del tempo di risposta:
[0.4626±0.0008]
Mediana della distriubuzione del tempo di risposta del servente:
[0.2288±0.0003]
Tempo medio di permanenza nel sistema dei job:
[0.7122±0.0014]
Tempo Max di permanenza nel sistema dei job:
[42.7704±4.6190]
Tempo Min di permanenza nel sistema dei job
[0.0000±0.0000]
Fattore di utilizzo del server
[0.6607±0.0007]
Risultati analitici Rho =0.6667 Rho Omnett = 0.6607 Dovrebbe essere stabile
Risulta essere stabile  Risultato analitico Ws=1.0 Omnett Ws=0.7122


-------------------------------------------------------------------------------
```

Nelle ultime due righe testuali vengono riportati in ordine il primo risultato analitico che stabilisce una condizione di stabilità per il sistema che è data dalla formula :

$$\Phi = \frac{\lambda}{\mu} < 1$$

Con Φ viene indicato il fattore di utilizzo del sistema. Viene confrontano il valore calcolato con quello calcolato dalla simulazione OMNeT++. Se risulta essere un sistema stabile viene calcolato il tempo medio di permanenza dell'utente nel sistema :

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

Una volta calcolato ***Ws*** viene fatto il confronto con i tempi risultanti dalle simulazioni terminando la parte l'analisi richiesta nel progetto.

6. Bibliografia

1.Heavy traffic limit for a tandem queue with identical service times.H. Christian Gromoll, Bryce Terwilliger & Bert Zwart

2. OMNeT++ [https://omnetpp.org/intro]

3. Python [https://www.python.org/]