Simulazione PMCSN

Luca Mastrobattista, 0292461

Indice

1 Traccia della Simulazione

1.1 Caso di studio

Si vuole valutare l'idea di aprire un locale in un piccolo paese. L'attività dovrà offrire ai clienti servizi di bar e di pizzeria. Il locale è già provvisto di tutto l'arredamento e lo si affitterà per un costo 1500 € al mese. Il pizzaiolo scelto per i servizi di pizzeria ha comunicato che, nel forno presente, si possono preparare contemporaneamente al massimo 2 pizze, ognuna delle quali può essere preparata con un tempo medio di 3 minuti. Inoltre, si è già trovato un accordo con lui: lavorerà ogni giorno della settimana dalle ore 19:00 alle ore 23:00, percependo una paga di 50 € al giorno con il vincolo che tutte le ordinazioni arrivate prececedentemente alle 23:00 verranno sempre completate, anche se per farlo dovrà continuare a sfornare pizze oltre questo orario. Per quanto riguarda le richieste al bar, si vuole che "l'ultimo giro" venga chiamato alle ore 02:00, senza accettare altre richieste successive a quell'orario ma completando tutte quelle ancora presenti.
Dopo un'osservazione settimanale di altri locali che offrono servizi simili, si è notato che il numero di clienti che arrivano al locale si differenzia per fasce orarie della giornata diverse. Inoltre, nel fine settimana, la frequenza delle richieste nelle fasce orarie identificate è maggiore rispetto a quella settimanale. Infine, si è osservato che nella fascia oraria tra le 15:00 e le 18:00 le richieste sono talmente poche che non è conveninente mantenere il locale aperto. Si riportano di seguito delle tabelle riassuntive per le frequenze di arrivo:

Fascia oraria	$\lambda_{B,W}$	$\lambda_{P,W}$
07:00 → 11:00	30 j/h	✗
11:00 → 15:00	12.5 j/h	✗
15:00 → 18:00	✗	✗
18:00 → 19:00	25 j/h	✗
19:00 → 23:00	12.5 j/h	10 j/h
23:00 → 02:00	10 j/h	✗

Frequenze di arrivo settimanali

Fascia oraria	$\lambda_{B,WE}$	$\lambda_{P,WE}$
07:00 → 13:00	30 j/h	✗
13:00 → 18:00	20 j/h	✗
15:00 → 18:00	✗	✗
18:00 → 19:00	45 j/h	✗
19:00 → 23:00	22.5 j/h	30 j/h
23:00 → 02:00	20 j/h	✗

Frequenze di arrivo fine-settimanali

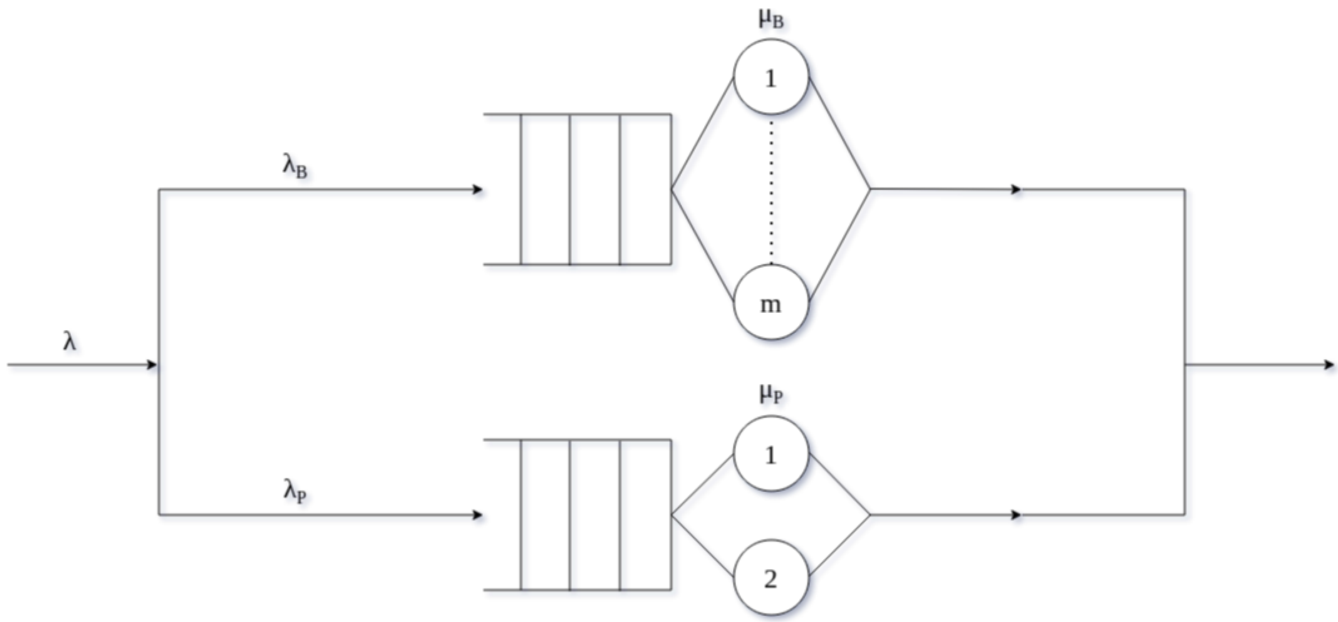
1.2 Obiettivi

Obiettivo dell'analisi è la valutazione del numero di baristi da assumere, col fine ultimo di massimizzare i guadagni. Si considera una paga di 40 € al giorno per ognuno di loro, assumendo per loro turni di 8 ore. Si assume che ogni barista sia in grado di servire un'ordinazione in 2 minuti, durante i quali si dedica esclusivamente a quella richiesta. Si assume inoltre che il prezzo medio delle richieste di tipo B sia di 5 €, mentre quello delle richieste di tipo P sia di 7 €. Si vuole, però, che i seguenti vincoli siano sempre rispettati:

- Ogni ordinazione al bar deve essere servita in un tempo strettamente minore di 3 minuti;
- Ogni ordinazione per la pizzeria sia servita in un tempo strettamente minore di 10 minuti

2 Modello concettuale

2.1 Visualizzazione grafica



La frequenza di arrivo λ si compone della frequenza di arrivo λ_{B} e λ_{P}, che sono rispettivamente i tassi di arrivo per richieste al bar e alla pizzeria. Una opportuna coda per ogni tipologia rappresenta la lista di attesa della tipologia stessa. Ogni servente di tipo B rappresenta un barista assunto, che lavora con una frequenza μ_{B}. Ogni servente di tipo P, invece, rappresenta una delle due richieste che il pizzaiolo è in grado di gestire contemporaneamente.

2.2 Eventi del sistema e variabili di stato

2.3 Eventi

Indice	Descrizione	Attributo 1	Attributo 2
0	Arrivo di tipo B	t	x
1	Completamento dal server B_1	t	x
..	..	t	x
m	Completamento dal server B_m	t	x
m + 1	Arrivo di tipo P	t	x
m + 2	Completamento dal server P_1	t	x
m + 3	Completamento dal server P_2	t	x
m + 4	Evento di campionamento	t	x

L'attributo t indentifica il tempo schedulato per la successiva occorrenza dell'evento di quel tipo; l'attributo x identifica lo stato di attività dell'evento.

2.4 Variabili di stato

- $l_{\mathrm{B}}(t)$: numero di richieste di tipo B al centro all'istante t
- $l_{\mathrm{P}}(t)$: numero di richieste di tipo P al centro all'istante t

- $X_s(t)$: stato del servente s all'istante t, con $s \in \mathcal{B} \cup \mathcal{P}$, dove $\mathcal{B} \cup \mathcal{P}$ è l'insieme dei serventi di tipo B unito all'insieme dei serventi di tipo P.

$$X_s(t) = \begin{cases} 1 & \text{se servente s è occupato} \\ 0 & \text{altrimenti} \end{cases}$$

3 Modello delle specifiche

3.1 Periodo di osservazione

Il periodo di osservazione è quello di un intero anno e ogni giorno si osserva l'intera giornata lavorativa costituita dalle due fasce orarie riportate precedentemente nelle *tabelle riassuntiva dei tassi di arrivo*.

3.2 Distribuzione degli arrivi

I valori dei vari tassi di arrivo sono stati raccolti analizzando un caso reale, anche se si tratta comunque di una stima. Per rappresentare il processo degli arrivi è stata utilizzata la distribuzione esponenziale, utilizzando λ diversi per ogni fascia oraria. Inoltre, all'interno della singola fascia oraria, gli arrivi potrebbero essere modellati come una distribuzione gaussiana, centrata attorno all'ora in cui le richieste sono più probabili. A partire da questa osservazione, si è scelto di utilizzare la distribuzione esponenziale per modellare gli arrivi, ma la media utilizzata è pesata opportunamente per una probabilità che è tanto più alta quanto più il tempo di simulazione è vicino all'ora di massima affluenza per quella fascia oraria. Per modellare questo, si definiscono delle frequenze di interarrivo medie per ogni fascia oraria, riportate qui in minuti:

Fascia oraria	$\lambda_{B,W}$	$\lambda_{P,W}$
07:00 → 11:00	0.5 j/min	✗
11:00 → 15:00	0.21 j/min	✗
18:00 → 19:00	0.42 j/min	✗
19:00 → 23:00	0.21 j/min	0.17 j/min
23:00 → 02:00	0.17 j/min	✗

Frequenze di arrivo settimanali

Fascia oraria	$\lambda_{B,WE}$	$\lambda_{P,WE}$
07:00 → 11:00	0.5 j/min	✗
11:00 → 15:00	0.34 j/min	✗
18:00 → 19:00	0.75 j/min	✗
19:00 → 23:00	0.375 j/min	0.5 j/min
23:00 → 02:00	0.34 j/min	✗

Frequenze di arrivo fine-settimanali

Per ogni fascia oraria si definisce una distribuzione di probabilità gaussiana:

Fascia oraria	μ	σ
07:00 → 11:00	8	1.2
11:00 → 15:00	13.5	2
18:00 → 19:00	18.5	0.4
19:00 → 23:00	22.5	2
23:00 → 02:00	24	0.9

Parametri delle gaussiane per richieste di tipo B

Fascia oraria	μ	σ
19:00 → 23:00	20.5	1

Parametri delle gaussiane per richieste di tipo P

La loro rappresentazione grafica è riportata in fondo al documento.

Ora, supponiamo di essere all'istante di simulazione t_0 di un giorno settimanale, nella prima fascia oraria; in questo caso $\lambda = 0.5\ j/min$. Per generare il prossimo tempo di interarrivo, si usa:

$$\texttt{Exponential(1/}\lambda\texttt{) * } f^n(t0)$$

dove $f^n(t_0)$ è il valore della distribuzione normale relativa alla fascia oraria valutata in t_0 e normalizzata rispetto alla fascia oraria. Nell'esempio:

$$f^n(t_0) = \frac{f(t_0)}{F(11) - F(7)}$$

con $F(x)$ funzione cumulativa della distribuzione gaussiana relativa alla fascia oraria 07:00 → 11:00.

Non si è usata una gaussiana direttamente come distribuizione del tempo di interarrivo perché avrebbe modellato una cosa diversa: in quel caso i tempi sarebbero molto più vicini al valor medio della distribuzione all'interno dell'intera fascia in esame, invece si vuole modellare che il tempo di interarrivo diminuisce in un intorno di un certo tempo.

La simulazione permette di disabilitare questa opzione inserendo l'opportuno *flag* nel lancio del programma.

3.3 Assunzioni

- Stato iniziale vuoto:
$$l_{\mathrm{B}}(0) + l_{\mathrm{P}}(0) = P(0) + B(0) = 0$$
Come conseguenza, il primo evento deve essere necessariamente un arrivo e, in particolare, è un arrivo di tipo B: la pizzeria apre alle 19.

- Stato finale di ogni giorno vuoto:
$$X_{\mathrm{s}}(T) = 0 \quad \forall s \in \mathcal{B} \cup \mathcal{P}$$
Con T tempo di chiusura giornaliero e $\mathcal{B} \cup \mathcal{P}$ l'unione dell'insieme dei serventi di tipo B e P. Come conseguenza, l'ultimo evento non può essere un arrivo, e sarà quindi o una partenza o un campionamento.

- I tempi di servizio di ognuno dei serventi si assumono esponenziali e indipendenti dalla fascia oraria. In particolare, ogni servente di tipo B lavora con frequenza media pari a $\mu_{\mathrm{B}} = \frac{1}{2}\ j/min$; ogni servente di tipo P lavora con frequenza media pari a $\mu_{\mathrm{P}} = \frac{1}{3}\ j/min$.

- l'evento di campionamento non deve seguire un evento di campionamento: non ha molto senso raccogliere due volte le statistiche se nel mezzo non è accaduto niente. Anzi: un campionamento incontrollato di questo tipo darebbe un peso maggiore al valore delle statistiche raccolte successivamente.

3.4 Guadagni e costi

4 Modello analitico

Di seguito sono rappresentate le formule utilizzate per l'analisi teorica del sistema. In particolare, tutti i centri sono stati modellati come M/M/m per i quali si hanno le seguenti formule teoriche:

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

$$P_Q = \frac{(m\rho)^m}{m!(1-\rho)} \cdot p(0)$$

$$E(TQ) = \frac{P_Q E(S)}{1-\rho}$$

$$E(TS) = E(TQ) + E(S_i)$$

$$E(NQ)_{Erlang} = \frac{PQ\rho}{1-\rho}$$

$$E(NS) = E(NQ) + m\rho$$

5 Modello computazionale

Il modello computazionale è stato sviluppato in *Python* ed è il programma `simulation.py`; i parametri configurabili sono definiti invece nel file `configurations/Config.py`. Il programma è lanciabile da riga comando con diversi *flag* che ne influenzano il comportamento, listabili con il comando: `python simulation.py -h` (oppure `python simulation.py -help`), ottenendo il seguente messaggio:

```
~/Scrivania/PMCSN/PMCSN-simulation (main*) » python simulation.py -h                                    luca@nixos
usage: simulation.py [-h] [-cf FILEPATH] [-scf FILEPATH] [-fh] [-ih] [-cc OPTION VALUE] [-fb THRESHOLD] [-s SEED] [-ngf] [-of FILE] [-wed]

PMCSN project command line interface

options:
  -h, --help            show this help message and exit
  -cf FILEPATH, --configFile FILEPATH
                        specify a configuration file to load
  -scf FILEPATH, --storeConfigFile FILEPATH
                        specify an output file where to store config
  -fh, --finite_horizont
                        simulate a finite horizont case
  -ih, --infinite_horizont
                        simulate an infinite horizont case. Using this, gaussian factor is automatically disabled
  -cc OPTION VALUE, --change_config OPTION VALUE
                        specify configuration to change
  -fb THRESHOLD, --find_b_value THRESHOLD
                        find the value of b such tath autocorellation lag j=1 is <= THRESHOLD
  -s SEED, --seed SEED  use the given SEED as random seed. if SEED = 0 then the initial seed is to be supplied interactively; if SEED < 0
                        then the initial seed is obtained from the system clock; if SEED > 0 > 0 then it is the initial seed (unless too
                        large). default value is 0
  -ngf, --no_gaussian_factor
                        don't use the gaussian probability value to weight interarrival times
  -of FILE, --output_file FILE
                        save output in ./output/OUTPUTFILE.csv. The file format .csv is added if not already present
  -wed, --weekend_day   simulate a weekend day, with the proper system variables. Default this option is disabled, meaning that a week day
                        is simulate
```

In particolare, col flag `-cc` è possibile cambiare una qualsiasi impostazione presente nel file di configurazione specificandone il valore a riga comando, mentre col flag `-ngf` (cioè `-no_gaussian_factor`) si disabilita la normalizzazione del tempo di interarrivo col fattore gaussiano determinato secondo le modalità descritte nella sezione dedicata alla distribuzione degli arrivi.

Il programma è realizzato seguendo l'approccio della *next-event simulation* e perciò sono state create opportune classi python, memorizzate ognuna in un file diverso.

Le funzioni per processare gli arrivi sono `processArrivalB()` e `processArrivalP()` al cui interno è implementatata la logica per recuperare il λ corretto in base alla fascia oraria, il cui inverso va usato come parametro per la funzione `getArrivalB(m)` e `getArrivalP(m)` dove viene invocata la

`Exponential(m)` dopo aver selezionato opportunamente uno *stream* diverso. Le funzioni per processare le partenze sono `processDepartureB()`, `processDepartureP()`.

La funzione per la selezione del servente è diversa per richieste di tipo B e P: si tratta di `FindOneB()` e `FindOneP()`. Queste funzioni utilizzano politiche di assegnazione ai server diverse: nel primo caso la politica è di *equity*, ricercando quindi il server che risulta libero da più tempo, nel secondo, invece, dovendo simulare un pizzaiolo che può infornare contemporaneamente 2 pizze, la richiesta è assegnata al primo server libero di tipo P che si trova, scandendo la lista in ordine crescente.

L'evento di campionamento viene schedulato aggiungendo un valore di tempo costante generato da una chiamata a `Uniform(config.SAMPLING_UNIFORM_A, config.SAMPLING_UNIFORM_B)` ad il tempo minimo tra quello dei prossimi eventi schedulati:

simulation.py

```python
times = []
for index, ev in enumerate(stats.events):
    if index != e and ev.x == 1:
        times.append(ev.t)
stats.events[e].t = min(times) + samplingInterarrivalTime
```

I vari *sample* che vengono raccolti durante la simulazione vengono memorizzati in una opportuna istanza di `SamplingList`, il cui metodo `append()`, oltre ad aggiungere il nuovo campione in fondo alla lista, implementa il *one-pass alghoritm* secondo *Welford* per ognuna delle grandezze raccolte; è quindi necessario, a fine simulazione, invocare i metodi `makeCorrectVariance` che divide per il numero di campioni tutte le varianze e `computeConfidenceInterval` per calcolare l'intervallo di confidenza per ognuna delle grandezze.

5.1 Makefile

Nella directory principale è presente un `Makefile` che aiuta ad eseguire correttamente il programma. Usando le configurazioni predisposte, nessun parametro di configurazione viene modificato: il flag `-cc` non viene mai usato. In particolare, il comando `make clean` elimina tutti i file di output `.csv` generati da precedenti esecuzioni. Il comando `make`, che corrisponde a `make all`, esegue, in ordine:

```
rm -f ./output/*.csv
rm -f ./output/finite/*.csv
rm -f ./output/infinite/*.csv
python3 simulation.py -ih -s 123 -of week
python3 simulation.py -ih -s 123 -wed -of weekend
python3 simulation.py -fh -s 123 -of week_gauss
python3 simulation.py -fh -s 123 -of week -ngf
python3 simulation.py -fh -s 123 -of weekend_gauss -wed
python3 simulation.py -fh -s 123 -of weekend -wed -ngf
```

Questo comando è quello utilizzato per generaer gli output che verranno successivamente discussi.

5.2 Dipendenze

Le librerie esterne importate, di cui il programma necessita per funzionare, sono le seguenti e sono riportate anche nel file `dipendenze.txt`:

- `copy`
- `argparse`
- `importlib`
- `ast`
- `math`
- `scipy.stats`
- `time`

5.3 Script di supporto

6 Verifica

La fase di verifica serve a capire se, rispetto al modello che io avevo, il programma implementato è corretto. Si è quindi confrontato il valore delle statistiche restituito dal simulatore con quelle teoriche e perciò si è fatto riferimento ai valori ottenuti da una esecuzione senza il peso gaussiano. I risultati ottenuti dall'esecuzione di `python simulation.py -s 123 -ngf [-wed]` differiscono leggermente da quelli teorici:

B type	Week		Weekend	
Statistica	Risultato teorico	Risultato sperimentale	Risultato teorico	Risultato sperimentale
avgInterarrivals	3.471 min	2.984 ± 0.076 min	2.413 min	2.494 ± 0.025 min
avgWaits	2.251 min	2.284 ± 0.047 min	2.524 min	2.266 ± 0.036 min
avgNumNodes	0.682 j	0.788 ± 0.029 j	1.102 j	0.801 ± 0.021 j
avgDelays	0.251 min	0.268 ± 0.014 min	0.524 min	0.286 ± 0.010 min
avgNumQueues	0.106 j	0.091 ± 0.008 j	0.273 j	0.103 ± 0.005 j
avgService (1)	2 min	1.724 ± 0.047 min	2 min	1.772 ± 0.035 min
avgService (2)	2 min	2.367 ± 0.039 min	2 min	2.216 ± 0.036 min

P type	Week		Weekend	
Statistica	Risultato teorico	Risultato sperimentale	Risultato teorico	Risultato sperimentale
avgInterarrivals	5.882 min	4.408 ± 0.160 min	2.000 min	1.846 ± 0.036 min
avgWaits	3.209 min	3.465 ± 0.116 min	6.857 min	5.667 ± 0.151 min
avgNumNodes	0.545 j	0.761 ± 0.042 j	3.429 j	2.962 ± 0.046 j
avgDelays	0.209 min	0.050 ± 0.008 min	3.857 min	2.514 ± 0.083 min
avgNumQueues	0.035 j	0.013 ± 0.003 j	1.929 j	1.316 ± 0.037 j
avgService (1)	3 min	2.918 ± 0.136 min	3 min	2.947 ± 0.116
avgService (2)	3 min	4.722 ± 0.201 min	3 min	3.361 ± 0.069 min

Il motivo di tale differenza va ricercato nella bassa frequenza di arrivo: infatti, all'interno di un singolo run di tipo *week*, i job processati di tipo B non arrivano a 300, mentre quelli di tipo P superano di poco i 50, rendendo le statistiche poco rappresentative. Perciò, si è definito un nuovo file di configurazione,

Fascia oraria	$\lambda_{B,W}$	$\lambda_{P,W}$
07:00 → 11:00	4.2 j/s	✗
11:00 → 15:00	4.2 j/s	✗
18:00 → 19:00	2.1 j/s	✗
19:00 → 23:00	2.1 j/s	2 j/s
23:00 → 02:00	2.1 j/s	✗

6.1 `verify1.py`

Per ovviare al problema del basso tasso di interarrivo, si è costruito il file di configurazione `verify1.py` che cerca di aumentare il numero di job di ogni tipologia processati nel singolo *run*. Per farlo, si è definito le seguenti frequenze di interarrivo, cercando di abbattere anche il contributo di una eccessiva variabilità tra le fasce:
Altre configurazioni cambiate sono le seguenti:

`configurations/verify1.py`

```
# per esprimere i tempi in secondi:
SLOTSTIME = [ (i * 3600) for i in [7, 11, 15, 18, 19, 23] ]
STOP_B = 26 * 3600
# per rendere il sistema stabile anche dal punto di vista
   analitico
MEAN_SERVICE_TIME_B = 0.5
MEAN_SERVICE_TIME_P = 0.5
# i lambda:
WEEK_LAMBDA_B = [2, 2, 0, 3, 3, 3]
WEEK_LAMBDA_P = 1.7
```

Eseguendo il comando `python simulation.py -s 123 -ngf -cf configurations/verify1.py`, si ottengono i seguenti risultati:

B type	Week	
Statistica	Risultato teorico	Risultato sperimentale
avgInterarrivals	0.317 s	0.466 ± 0.002 s
avgWaits	0.415 s	0.337 ± 0.000 s
avgNumNodes	1.394 j	0.739 ± 0.008 j
avgDelays	0.115 s	0.037 ± 0.000 s
avgNumQueues	0.449 j	0.083 ± 0.000 j
avgService (1)	0.3 s	0.300 ± 0.000 s
avgService (2)	0.3 s	0.300 ± 0.000 s

P type	Week	
Statistica	Risultato teorico	Risultato sperimentale
avgInterarrivals	0.588 s	0.593 ± 0.000 s
avgWaits	0.610 s	0.603 ± 0.000 s
avgNumNodes	1.037 j	1.017 ± 0.001 j
avgDelays	0.110 s	0.105 ± 0.000 s
avgNumQueues	0.187 j	0.176 ± 0.000 j
avgService (1)	0.5 s	0.496 ± 0.000 s
avgService (2)	0.5 s	0.501 ± 0.000 s

Si può osservare come i valori teorici per le richieste di tipo P, che sono state in tutto 24407, cominciano a convergere ai valori teorici. Per le richieste di tipo B, nonostante siano state 181785, i valori ottenuti sono ancora lontani da quelli teorici.

Si può però osservare che la frequenza di interarrivo media nella prima metà della giornata è più piccola, e perciò il numero di job processati e di campioni fatti nelle prime 8 ore sarà minore rispetto a quelli fatti nelle seconde 8. Per questo motivo, nel calcolo della media globale, i dati delle statistiche raccolti nella prima metà della giornata tende a pesare di meno rispetto a quelli raccolti nella seconda metà. Alla luce di questo comportamento, è stata aggiunta una nuova opzione, attiva per default, che serve a *spezzare* l'analisi media nelle due fasce orarie. In particolare, si raccolgono i dati in entrambe le metà separatamente, calcolando per ogni statistica la sua media e varianza per poi mediarle col numero di job processati:

$$\mu_{glob} = \frac{n_1\mu_i + n_2\mu_2}{n_1 + n_2}$$

Dove n_1 ed n_2 sono rispettivamente il numero di job processati nella prima e nella seconda metà, mentre μ_1 e μ_2 sono le medie raccolte nelle due metà. Per disabilitare questa funzione, bisogna specificare a riga comando il flag `-ns`.

I risultati ottenuti dal comando `python simulation.py -s 123 -ngf -cf configurations/verify1.py` senza aver cambiato nulla nel file di configurazione rispetto al caso precedente sono i seguenti:

B type	Week	
Statistica	Risultato teorico	Risultato sperimentale
avgInterarrivals	0.400 s	0.399 ± 0.000 s
avgWaits	0.353 s	0.355 ± 0.000 s
avgNumNodes	0.894 j	0.938 ± 0.000 j
avgDelays	0.053 s	0.055 ± 0.000 s
avgNumQueues	0.144 j	0.156 ± 0.000 j
avgService (1)	0.3 s	0.300 ± 0.000 s
avgService (2)	0.3 s	0.299 ± 0.000 s

Alla luce degli esperimenti fatti, si può concludere che il simulatore risponde bene ai cambiamenti di configurazione e che, con un numero adeguato di job, produce risultati fedeli a quelli teorici dimostrando così di essere fedele al modello di sistema in esame.

7 Validazione

Nella fase di validazione il quesito fondamentale è: "Stiamo costruendo il prodotto giusto?" Tuttavia, durante il processo di validazione, si sono riscontrate delle sfide significative nell'utilizzo dell'analisi a orizzonte finito, principalmente dovute al numero limitato di job processati. Un aspetto critico che ha reso l'analisi a orizzonte finito inadeguata è il basso numero di job processati in ciascun *run* di simulazione, una caratteristica intrinseca del caso di studio in esame. Questa limitazione ha comportato un campione statistico ridotto e, di conseguenza, una maggiore variabilità nei risultati osservati. In altre parole, la variabilità è emersa come una problematica principale a causa della limitata quantità di dati disponibili.

È importante notare che, in un contesto in cui avessimo processato un numero significativamente maggiore di job in ogni run, la variabilità non avrebbe avuto un impatto significativo.

Per affrontare questa sfida, si è deciso di adottare un'analisi a orizzonte infinito in modo da ottenere risultati più stabili. In questa fase di validazione, verifichiamo se, per ciascuna fascia oraria, le statistiche generate dal nostro simulatore convergono ai valori teorici, considerati come punti di riferimento per il nostro sistema. Questo approccio ci assicura che il sistema software risponda alle caratteristiche previste e ai requisiti desiderati.

7.1 Analisi a orizzonte infinito

Per condurre un'analisi a orizzonte infinito, è necessario utilizzare i seguenti comandi:

- `make infinite-week`: esegue `python3 simulation.py -ih -s 123 -of week`. Effettua l'analisi per il giorno lavorativo della settimana.
- `make infinite-weekend`: esegue `python3 simulation.py -ih -s 123 -wed -of weekend`. Effettua l'analisi per il fine settimana.

È fondamentale notare che, prima di eseguire questi comandi, è consigliabile eliminare eventuali file di output CSV già presenti al fine di evitare confusione nei risultati.

Per determinare i parametri dell'analisi, è possibile specificare il flag `-fb THRESHOLD`, il quale consente di determinare dinamicamente il numero di campioni da raccogliere per ogni batch. Il parametro `THRESHOLD` specifica che il valore di autocorrelazione per lag `j = 1` di ogni statistica non deve superare quel valore. Utilizzando `k = 128` batches, ciascuno con `b = 1024` campioni, l'autocorrelazione per lag `j = 1` è inferiore a `0.2`, contribuendo così alla stabilità e all'affidabilità dei risultati.

L'analisi viene condotta per ciascun tipo di richiesta e per ciascuna statistica di interesse, coprendo tutte le fasce orarie sia nei giorni lavorativi che nei giorni del fine settimana. I risultati vengono presentati in forma tabellare e grafica; in particolare, nelle tabelle si riportano i risultati sperimentali ottenuti mediando su tutti i batch con un intervallo di confidenza al 95%.

	B type - Interarrivals					
	Slot	Risultato teorico	Risultato sperimentale	Media nell'intervallo	Errore	Link all'immagine
Week	0	2.000 min	2.000 ± 0.014 min	✓		link
	1	4.762 min	4.950 ± 0.389 min	✓		link
	3	2.381 min	2.614 ± 0.477 min	✓		link
	4	4.762 min	5.154 ± 0.811 min	✓		link
	5	5.882 min	6.217 ± 0.074 min	✓		link
Weekend	0	2.000 min	2.000 ± 0.014 min	✓		link
	1	2.941 min	3.091 ± 0.322 min	✓		link
	3	1.333 min	1.460 ± 0.259 min	✓		link
	4	2.667 min	2.794 ± 0.268 min	✓		link
	5	2.941 min	3.065 ± 0.285 min	✓		link

	B type - Waits						
	Slot	Risultato teorico	Risultato sperimentale	Media nell'intervallo	Errore	Link all'immagine	Rispetta *QoS*
Week	0	2.667 min	2.665 ± 0.035 min	✓		link	✓
	1	2.092 min	2.083 ± 0.018 min	✓		link	✓
	3	2.428 min	2.440 ± 0.027 min	✓		link	✓
	4	2.092 min	2.086 ± 0.021 min	✓		link	✓
	5	2.060 min	2.067 ± 0.023 min	✓		link	✓
Weekend	0	2.667 min	2.665 ± 0.035 min	✓		link	✓
	1	2.261 min	2.257 ± 0.024 min	✓		link	✓
	3	4.571 min	4.499 ± 0.115 min	✓		link	✗
	4	2.327 min	2.339 ± 0.031 min	✓		link	✓
	5	2.261 min	2.246 ± 0.025 min	✓		link	✓

	B type - Num. in the nodes					
	Slot	Risultato teorico	Risultato sperimentale	Media nell'intervallo	Errore	Link all'immagine
Week	0	1.333 min	1.336 ± 0.021 min	✓		link
	1	0.439 min	0.440 ± 0.006 min	✓		link
	3	1.020 min	1.031 ± 0.016 min	✓		link
	4	0.439 min	0.440 ± 0.007 min	✓		link
	5	0.350 min	0.355 ± 0.005 min	✓		link
Weekend	0	1.333 min	1.336 ± 0.021 min	✓		link
	1	0.769 min	0.773 ± 0.011 min	✓		link
	3	3.429 min	3.399 ± 0.095 min	✓		link
	4	0.873 min	0.881 ± 0.014 min	✓		link
	5	0.769 min	0.769 ± 0.011 min	✓		link

	B type - Delays					
	Slot	Risultato teorico	Risultato sperimentale	Media nell'intervallo	Errore	Link all'immagine
Week	0	0.667 min	0.671 ± 0.027 min	✓		link
	1	0.092 min	0.100 ± 0.006 min	✗	0.002	link
	3	0.428 min	0.455 ± 0.019 min	✗	0.008	link
	4	0.092 min	0.098 ± 0.007 min	✓		link
	5	0.060 min	0.072 ± 0.007 min	✗	0.005	link
Weekend	0	0.667 min	0.671 ± 0.027 min	✓		link
	1	0.261 min	0.268 ± 0.014 min	✓		link
	3	2.571 min	2.519 ± 0.108 min	✓		link
	4	0.327 min	0.342 ± 0.020 min	✓		link
	5	0.261 min	0.265 ± 0.014 min	✓		link

	B type - Num. in the queue					
	Slot	Risultato teorico	Risultato sperimentale	Media nell'intervallo	Errore	Link all'immagine
Week	0	0.333 min	0.339 $\pm$ 0.015 min	✓		link
	1	0.019 min	0.021 $\pm$ 0.001 min	✗	0.001	link
	3	0.180 min	0.194 $\pm$ 0.005 min	✗	0.009	link
	4	0.019 min	0.021 $\pm$ 0.001 min	✗	0.001	link
	5	0.010 min	0.013 $\pm$ 0.001 min	✗	0.002	link
Weekend	0	0.333 min	0.339 $\pm$ 0.015 min	✓		link
	1	0.089 min	0.093 $\pm$ 0.005 min	✓		link
	3	1.929 min	1.910 $\pm$ 0.087 min	✓		link
	4	0.123 min	0.130 $\pm$ 0.008 min	✓		link
	5	0.089 min	0.092 $\pm$ 0.005 min	✓		link

	P type - all statistics in the slot						
	Statistica	Risultato teorico	Risultato sperimentale	Media nell'intervallo	Errore	Link all'immagine	Rispetta QoS
Week	Interarrivo	5.882 min	6.145 $\pm$ 0.499 min	✓		link	
	Attesa	3.209 min	3.223 $\pm$ 0.038 min	✓		link	✓
	Num. nel nodo	0.545 min	0.548 $\pm$ 0.009 min	✓		link	
	Ritardo	0.209 min	0.229 $\pm$ 0.017 min	✗	0.003	link	
	Num. in coda	0.035 min	0.040 $\pm$ 0.003 min	✗	0.002	link	
Weekend	Interarrivo	2.000 min	2.114 $\pm$ 0.228 min	✓		link	
	Attesa	6.867 min	6.979 $\pm$ 0.281 min	✓		link	✓
	Num. nel nodo	3.429 min	3.505 $\pm$ 0.150 min	✓		link	
	Ritardo	3.857 min	3.985 $\pm$ 0.267 min	✓		link	
	Num. in coda	1.929 min	2.010 $\pm$ 0.139 min	✓		link	

L'analisi a orizzonte infinito ha dimostrato che la maggior parte dei risultati converge in modo affidabile ai valori teorici previsti per il caso di studio in esame, confermando così l'accuratezza del nostro simulatore nel rappresentare il comportamento del sistema in uno stato stazionario. Tuttavia, un aspetto interessante emerge quando ripetiamo l'analisi utilizzando un seme iniziale diverso, specificamente il seme `12345`, con il comando `make clean infinite-weekend SEED=12345`. In questo caso, notiamo che la maggior parte degli errori viene azzerata, e gli altri risultano notevolmente ridotti. Questo suggerisce che il caso con il seme iniziale pari a `123` potrebbe essere considerato come un caso particolarmente sfortunato per l'analisi delle statistiche.

È interessante notare anche che nei casi in cui la frequenza di arrivo è leggermente più alta, come nel fine settimana, questi errori risultano azzerati anche con lo stesso seme `123`. Ciò evidenzia come la bassa frequenza di arrivo possa influenzare la precisione delle stime di queste grandezze, persino nell'analisi a orizzonte infinito.

8 Analisi dei costi e dei guadagni

Per effettuare l'analisi dei costi e dei guadagni sul caso di studio in esame è stata eseguita un'analisi a orizzonte finito che, seppur poco rappresentativa in ambito di validazione, è quella che permette effettivamente di valutare le entrate e le uscite per un sistema con queste specifiche.

8.1 Analisi a orizzonte finito

Con questo metodo, il sistema viene simulato per una durata pari a 16 ore lavorative effettive, prendendo in considerazione la variazione dei tassi di arrivo durante il passaggio da una fascia oraria all'altra. Per garantire la validità dell'analisi a orizzonte infinito, è essenziale che lo stato iniziale e lo stato finale del sistema siano identici. Si iniza ogni simulazione con il sistema in uno stato vuoto e ci si assicura che, al termine di ogni simulazione, il sistema sia nuovamente nello stato vuoto.

Simuliamo la giornata lavorativa per 1024 volte e, in ogni esecuzione, partiamo da uno stato vuoto, reimpostiamo tutte le statistiche e continuiamo senza alterare lo stato del generatore dei numeri casuali. Questo approccio garantisce l'indipendenza tra le diverse esecuzioni e ci consente di ottenere risultati affidabili per l'analisi a orizzonte infinito.

Anche qui l'analisi viene condotta per ciascun tipo di richiesta limitandoci però ai soli tempi di risposta, sia introducendo il fattore gaussiano che non. I risultati vengono presentati in forma tabellare e grafica; in particolare, nelle tabelle si riportano i risultati sperimentali ottenuti mediando su tutti i *run* con un intervallo di confidenza al 95%.

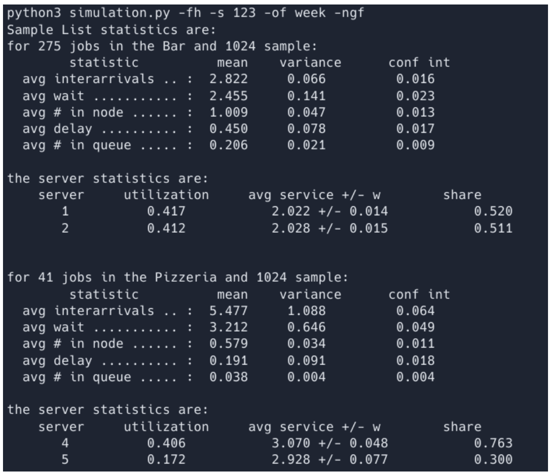
	Analisi senza fattore gaussiano						
	Statistica	Risultato teorico	Risultato sperimentale	Media nell'intervallo	Errore	Link all'immagine	Rispetta QoS
Week	Attesa di tipo B	2.251 min	2.455 ± 0.023 min	✗	0.181	link	✓
Week	Attesa di tipo P	3.209 min	3.212 ± 0.049 min	✓		link	✓
Weekend	Attesa di tipo B	2.524 min	2.761 ± 0.032 min	✗	0.205	link	✓
Weekend	Attesa di tipo P	6.857 min	5.813 ± 0.150 min	✗	0.894	link	✓

	Analisi con fattore gaussiano						
	Statistica	Risultato teorico	Risultato sperimentale	Media nell'intervallo	Errore	Link all'immagine	Rispetta QoS
Week	Attesa di tipo B	2.251 min	2.184 ± 0.027 min	✗	0.040	link	✓
Week	Attesa di tipo P	3.209 min	3.082 ± 0.078 min	✗	0.049	link	✓
Weekend	Attesa di tipo B	2.524 min	2.987 ± 0.075 min	✗	0.388	link	✓
Weekend	Attesa di tipo P	6.857 min	3.209 ± 0.056 min	✗	3.592	link	✓

Come ci aspettavamo dalle precedenti osservazioni, il valore teorico non cade quasi mai all'interno dell'intervallo di confidenza della statistica sperimentale. D'altra parte, i QoS vengono rispettati in ogni situazione con la configurazione che prevede $m_B = 2$. Si può notare che la scelta di $m_B = 1$ non rende il sistema stabile nella prima fascia oraria, dove $\lambda_B = 0.5j/min$, infatti, per un servente singolo:

$$\rho = \lambda E(S) = 0.5\ j/min \cdot 2\ min = 1$$

La scelta di $m_B = 2$ è quindi la più logica: è il minor valore di m che rispetta i QoS. Anche le utilizzazioni dei serventi sono accettabili e vengono riportati insieme ai valori delle altre grandezze:



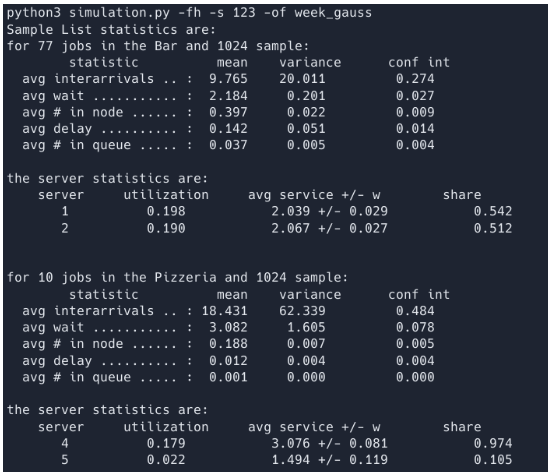
```
python3 simulation.py -fh -s 123 -of week -ngf
Sample List statistics are:
for 275 jobs in the Bar and 1024 sample:
        statistic           mean    variance      conf int
  avg interarrivals .. :  2.822     0.066          0.016
  avg wait ........... :  2.455     0.141          0.023
  avg # in node ...... :  1.009     0.047          0.013
  avg delay .......... :  0.450     0.078          0.017
  avg # in queue ..... :  0.206     0.021          0.009

the server statistics are:
    server     utilization      avg service +/- w        share
       1          0.417          2.022 +/- 0.014           0.520
       2          0.412          2.028 +/- 0.015           0.511


for 41 jobs in the Pizzeria and 1024 sample:
        statistic           mean    variance      conf int
  avg interarrivals .. :  5.477     1.088          0.064
  avg wait ........... :  3.212     0.646          0.049
  avg # in node ...... :  0.579     0.034          0.011
  avg delay .......... :  0.191     0.091          0.018
  avg # in queue ..... :  0.038     0.004          0.004

the server statistics are:
    server     utilization      avg service +/- w        share
       4          0.406          3.070 +/- 0.048           0.763
       5          0.172          2.928 +/- 0.077           0.300
```

Week day - senza fattore gaussiano



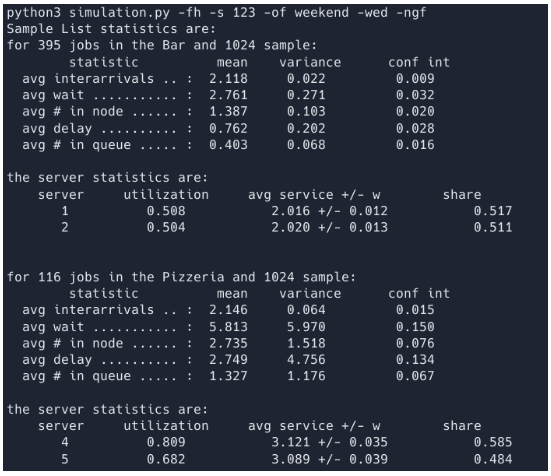
```
python3 simulation.py -fh -s 123 -of week_gauss
Sample List statistics are:
for 77 jobs in the Bar and 1024 sample:
        statistic           mean    variance      conf int
  avg interarrivals .. :  9.765    20.011          0.274
  avg wait ........... :  2.184     0.201          0.027
  avg # in node ...... :  0.397     0.022          0.009
  avg delay .......... :  0.142     0.051          0.014
  avg # in queue ..... :  0.037     0.005          0.004

the server statistics are:
    server     utilization      avg service +/- w        share
       1          0.198          2.039 +/- 0.029           0.542
       2          0.190          2.067 +/- 0.027           0.512


for 10 jobs in the Pizzeria and 1024 sample:
        statistic           mean    variance      conf int
  avg interarrivals .. : 18.431    62.339          0.484
  avg wait ........... :  3.082     1.605          0.078
  avg # in node ...... :  0.188     0.007          0.005
  avg delay .......... :  0.012     0.004          0.004
  avg # in queue ..... :  0.001     0.000          0.000

the server statistics are:
    server     utilization      avg service +/- w        share
       4          0.179          3.076 +/- 0.081           0.974
       5          0.022          1.494 +/- 0.119           0.105
```

Week day - con fattore gaussiano



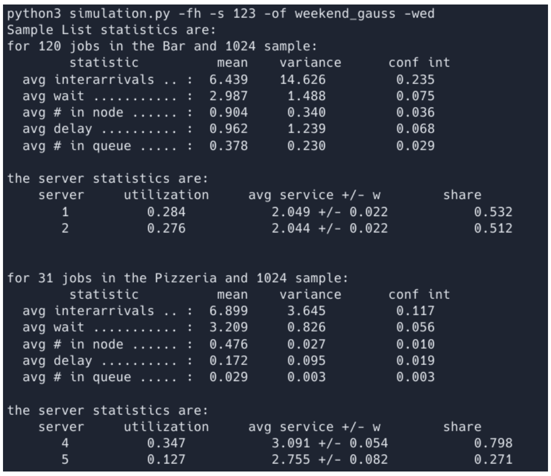
```
python3 simulation.py -fh -s 123 -of weekend -wed -ngf
Sample List statistics are:
for 395 jobs in the Bar and 1024 sample:
        statistic           mean    variance      conf int
  avg interarrivals .. :  2.118     0.022          0.009
  avg wait ........... :  2.761     0.271          0.032
  avg # in node ...... :  1.387     0.103          0.020
  avg delay .......... :  0.762     0.202          0.028
  avg # in queue ..... :  0.403     0.068          0.016

the server statistics are:
    server     utilization      avg service +/- w        share
       1          0.508          2.016 +/- 0.012           0.517
       2          0.504          2.020 +/- 0.013           0.511


for 116 jobs in the Pizzeria and 1024 sample:
        statistic           mean    variance      conf int
  avg interarrivals .. :  2.146     0.064          0.015
  avg wait ........... :  5.813     5.970          0.150
  avg # in node ...... :  2.735     1.518          0.076
  avg delay .......... :  2.749     4.756          0.134
  avg # in queue ..... :  1.327     1.176          0.067

the server statistics are:
    server     utilization      avg service +/- w        share
       4          0.809          3.121 +/- 0.035           0.585
       5          0.682          3.089 +/- 0.039           0.484
```

Weekend day - senza fattore gaussiano



```
python3 simulation.py -fh -s 123 -of weekend_gauss -wed
Sample List statistics are:
for 120 jobs in the Bar and 1024 sample:
        statistic           mean    variance      conf int
  avg interarrivals .. :  6.439    14.626          0.235
  avg wait ........... :  2.987     1.488          0.075
  avg # in node ...... :  0.904     0.340          0.036
  avg delay .......... :  0.962     1.239          0.068
  avg # in queue ..... :  0.378     0.230          0.029

the server statistics are:
    server     utilization      avg service +/- w        share
       1          0.284          2.049 +/- 0.022           0.532
       2          0.276          2.044 +/- 0.022           0.512


for 31 jobs in the Pizzeria and 1024 sample:
        statistic           mean    variance      conf int
  avg interarrivals .. :  6.899     3.645          0.117
  avg wait ........... :  3.209     0.826          0.056
  avg # in node ...... :  0.476     0.027          0.010
  avg delay .......... :  0.172     0.095          0.019
  avg # in queue ..... :  0.029     0.003          0.003

the server statistics are:
    server     utilization      avg service +/- w        share
       4          0.347          3.091 +/- 0.054           0.798
       5          0.127          2.755 +/- 0.082           0.271
```

Weekend day - con fattore gaussiano

Il numero di job riportati nelle immagini precedenti corrisponde alla media dei job processati nei vari *run*, ed è il valore che verrà utilizzato successivamente come numero di richieste giornaliere nell'analisi dei costi. Il calcolo è stata effettuato su base mensile (4 settimane), e perciò le grandezze giornaliere sono state convertite; le configurazioni usate sono quelle di default presenti nel file `configurations/Config.py`.

8.2 Analisi delle spese

Spesa	Valore	Contributo mensile
Baristi	40 € al giorno per barista	$40 \cdot 28 \cdot 2 = 2240$ € al mese
Pizzaiolo	50 € al giorno	$50 \cdot 28 = 1400$ € al mese
Bollette	2750 € al mese	2750 € al mese
Affitto	1500 € al mese	1500 € al mese
Fornitori	2000 € al mese	2000 € al mese
Totale		12970 €

8.3 Analisi dei guadagni

Tipo richiesta	Richieste *week*	Richieste *weekend*	Guadagno	Contributo mensile
Tipo B	275	5 € a richiesta	$275 \cdot 5$ € $\cdot 5 \cdot 4 =$ 27500 € al mese	
Tipo P	41	10 € a richiesta	$41 \cdot 10$ € $\cdot 2 \cdot 4 =$ 820 €	
Totale				12970 €

9 Conclusioni

Come si nota dai risultati ottenuti, a parte per qualche errore di approssimazione, i risultati della simulazione tendono a quelli teorici.
I risultati dell'analisi mostrano che il numero migliore di baristi da assumere è 2: infatti questo è il numero minimo con cui si riesce a rispettare il vincolo sul tempo di risposta, anche se il guadagno sarebbe stato maggiore con $m = 1$. Continuando invece ad aumentare il numero di baristi, il guadagno diminuisce sempre di più, mentre migliorano i tempi di risposta:

m	$E[T_S]$	$r(\tau)$
1	5.53 min	3070.10 € al mese
2	2.18 min	1853.43 € al mese
3	2.02 min	636.76 € al mese
4	2.00 min	-579.90 € al mese

Si può notare che, al crescere di m, la differenza dei tempi con il caso $m-1$ è sempre minore: questo si può spiegare considerando che i tassi di arrivo non sono stati cambiati: aumentando m, quindi, si va a diminuire il tempo di coda di ogni job, che tende quindi a 0.