Progetto di Simulazione di Sistemi

Andrea Schinoppi [0001097628]

A.A. 2022/2023

Il progetto

Scopo di questo progetto era creare una variante del modello di simulazione descritto nelle sezioni 1 e 2 dell'articolo *"Alternating server with non-zero switch-over times and opposite-queue threshold-based switching policy"* [1] mediante piattaforma Omnet++.

Questa presentazione riassumerà la creazione del modello di simulazione, i risultati ottenuti e il processo di convalida del modello stesso.

Il modello

Il modello da realizzare prevede che un **servente** serva per un certo service time una di **due code**, secondo una politica basata su **threshold**: quando la coda non servita raggiunge la soglia impostata, in generale, il server dovrebbe eseguire uno **switch-over** con una certa durata per passare a servirla.

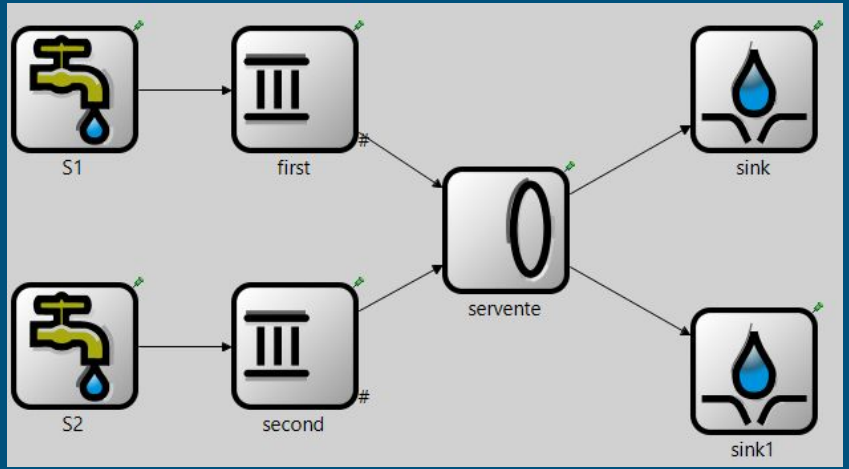
Tuttavia se durante uno switch-over, anche la coda da cui il server proviene raggiunge la propria soglia, il server dovrà continuare a servirla senza effettuare alcun cambio.
Inoltre, nel caso in cui entrambe le code superino il threshold contemporaneamente, lo switch-over **non** avverrà.
Infine, nel caso in cui le code fossero vuote, il servente **attenderà** che una abbia almeno un elemento per iniziare (o passare) a servirla.

Le due code sono M/M/1 e M/M/1/C_1, ciò significa che la prima ha capacità infinita mentre la seconda ha capacità C_1.
Raggiunta la propria capacità, la seconda coda comincerà quindi a **perdere** i job ricevuti.
I tempi di interarrivo dei job alle code sono indicati da λ_1 e λ_2 e sono distribuzioni esponenziali con una certa media $1 \setminus \lambda$.
I tempi di servizio del servente in base alla coda di provenienza dei job sono distribuzioni **uniformi** di valori in due differenti intervalli.
Infine, i threshold delle due code sono definiti rispettivamente dai parametri **K ed N** e la durata di uno switch-over è determinata dal parametro α.

Rete realizzata

La rete realizzata per questo progetto consiste in due sorgenti (”**S1**” ed ”**S2**”) che inviano job rispettivamente a due code (”**first**” e ”**second**”).

È presente un servente che secondo la politica basata su threshold precedentemente descritta serve una delle due code, per poi mandare i job serviti in un diverso sink (”**sink**” e ”**sink1**”) a seconda della provenienza.



Implementazione - 1

Dopo aver apportato le modifiche necessarie ai file **.ned** per aggiungere tutti i parametri necessari è iniziata la fase implementativa.

Per prima cosa è stata aggiunta alla classe SelectionStrategy, ovvero quella contenente le diverse strategie con cui il servente può scegliere quale coda servire, la classe **ThresholdSelectionStrategy**.
L'implementazione è basata sulla verifica di tutti i possibili casi per decidere se continuare a servire una coda o avviare uno switch-over, in particolare:

1. Se entrambe le code sono vuote il servente attende;
    a. Se le code erano vuote in precedenza viene verificato se lo siano ancora, controllando per prima quella che il server stava servendo ed avviando eventualmente uno switch-over.
2. Se solo la coda che il server sta attualmente servendo è vuota, avviene uno switch-over;
3. Se la coda non attualmente servita supera il threshold e l'altra no, avviene uno switch-over;
4. Se entrambe le code superano la propria soglia, il server continua a servire la stessa coda.

Inoltre ogni prima di ogni switch-over vengono nuovamente verificate tutte le condizioni per cui potrebbe dover essere annullato, in particolare:

- Viene verificato che la coda di destinazione non sia vuota;
- Viene verificato che la coda di destinazione non abbia superato la propria soglia nello stesso momento di quella di partenza.

Se una di queste verifiche non va a buon fine, lo switch-over è **annullato**.

Implementazione - 2

Un'altra modifica eseguita è stata quella alla funzione **Server::handleMessage**.
Porzioni di codice differente vengono eseguite in base al tipo di messaggio ricevuto dal servente:

1. Se il messaggio è un **job**, viene impostato un tempo di servizio in base alla sua provenienza, viene allocato e viene programmato un endServiceMsg al termine del service time.
2. Se il messaggio è un **endServiceMsg**, viene selezionato un gate di uscita in base alla provenienza del job attualmente gestito che viene poi deallocato e inviato al sink corretto. In seguito viene chiamata la selection strategy per sapere da quale coda prelevare il prossimo job. Da questo punto sono possibili **3 alternative**:
    a. Tutte le code sono **vuote**: il servente **attende**;
    b. La coda restituita dalla selezione è la **stessa** che stava venendo servita in precedenza: viene richiesto un nuovo job dalla stessa coda;
    c. La coda restituita dalla selezione è **diversa** da quella che stava venendo servita: **switch-over**. In questo caso il server si invia un self-message chiamato Begin switchMsg.
3. Se il messaggio è un **Begin switchMsg**, lo stato del server viene impostato a occupato e viene programmato dopo α secondi un messaggio End switchMsg.
4. Se il messaggio è un **End switchMsg**, lo stato del server torna libero e viene richiamata la strategia di selezione in modo da confermare o meno lo switch-over.

Simulazioni

Sono state eseguite due simulazioni con parametri differenti.
In tutti i casi sono stati eseguiti **20** esperimenti per ogni configurazione.

Il **seed** del generatore di numeri casuali è stato lasciato impostato a ${runnumber}, ovvero al numero della run corrente.

Ogni esperimento è stato impostato per terminare automaticamente dopo **5000** secondi di tempo simulato.

Per quanto riguarda gli scalari, è stato impostato un warmup-period di 1000s di tempo simulato per eliminare il transiente iniziale.

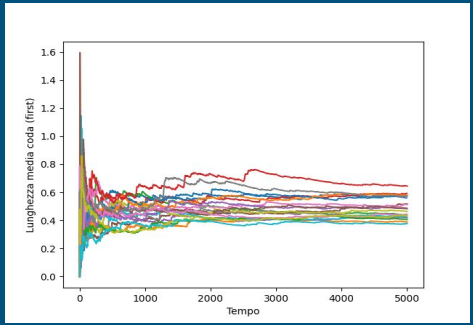
Configurazioni

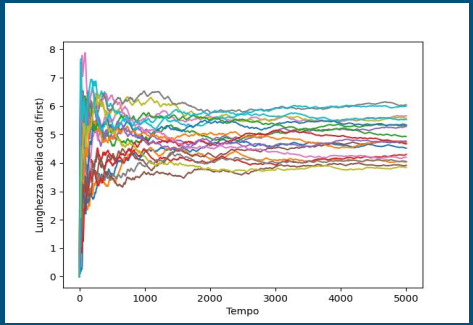
Parametro	Configurazione 1	Configurazione 2
λ_1	1.0, 2.0, 4.0, 10.0, 20.0	1.0, 2.0, 3.0, 3.5
λ_2	1.0, 2.0	1.5, 2.0
Service time 1	[0.11, 0.55]	[0.22, 0.44]
Service time 2	[0.1, 0.4]	[0.2, 0.3]
α	[0.1, 0.3]	[0.1, 0.3]
K	10	10
Capacità coda 1	10	10
N	3	3
Capacità coda 2	infinita	infinita

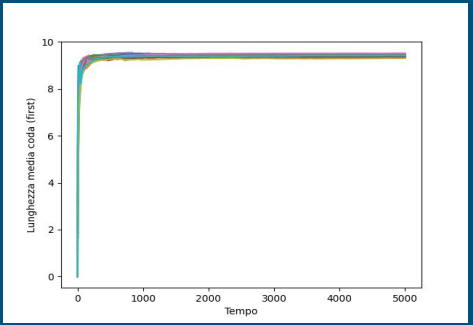
Vettori - configurazione 1

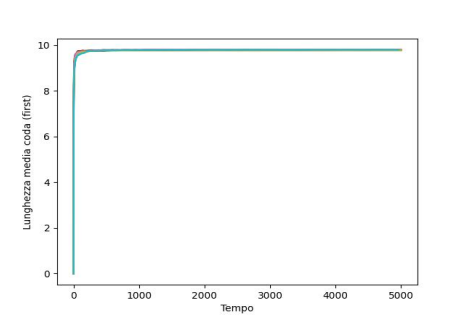
Conf. 1 - Lunghezza della coda “first”

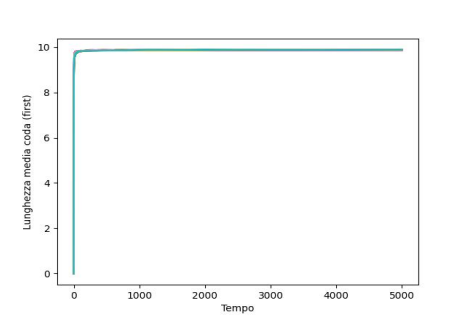
Da sinistra verso destra aumenta il valore di λ_1 (1.0, 2.0, 4.0; 10.0, 20.0); λ_2 = 1.0





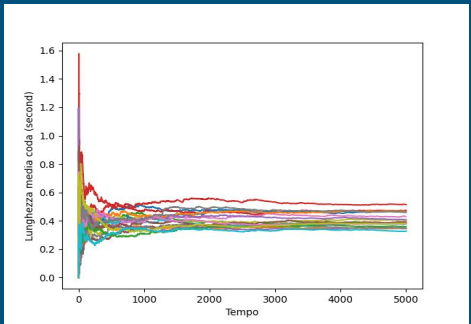


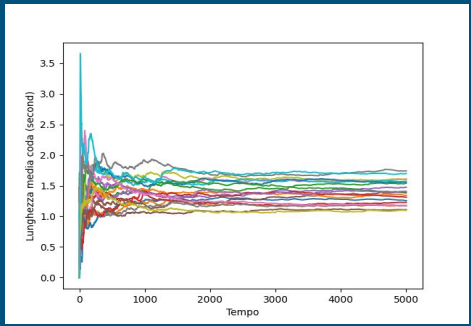


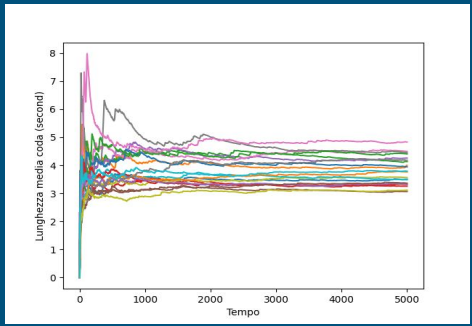


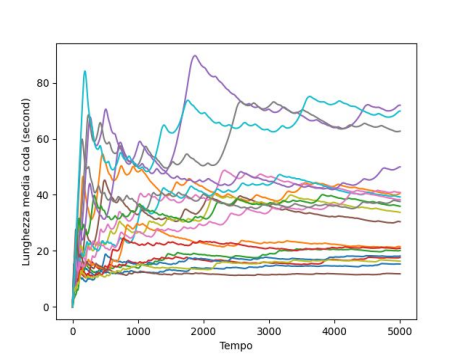
Conf. 1 - Lunghezza della coda “second”

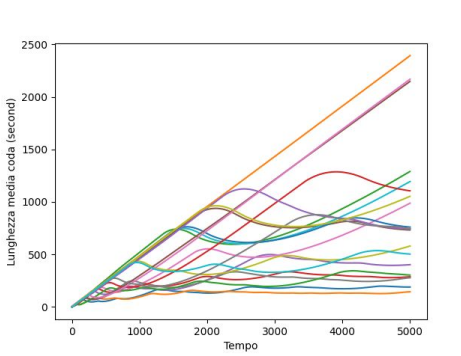
Da sinistra verso destra aumenta il valore di λ_1 (1.0, 2.0, 4.0; 10.0, 20.0); $\lambda_2 = 1.0$





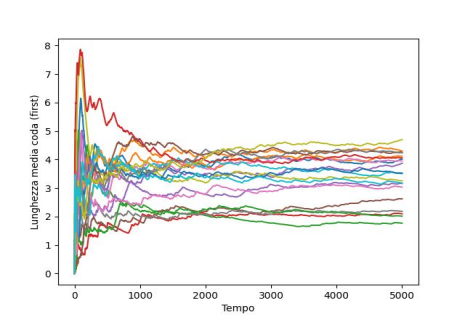


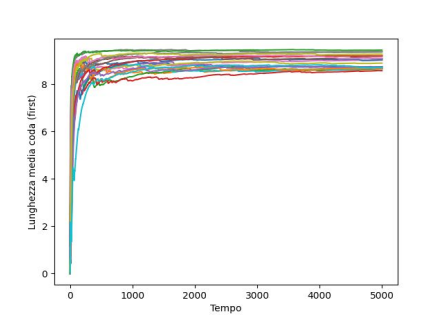


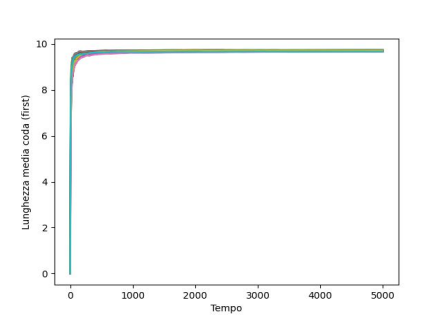


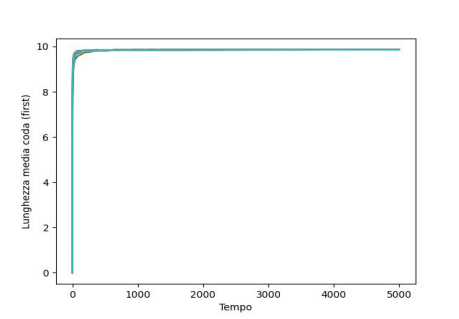
Conf. 1 - Lunghezza della coda “first”

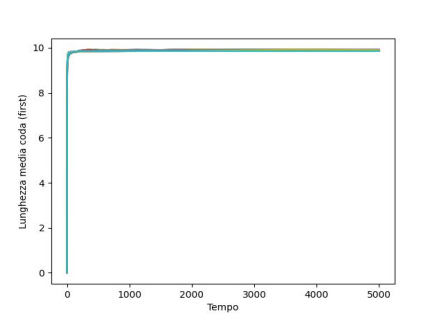
Da sinistra verso destra aumenta il valore di λ_1 (1.0, 2.0, 4.0; 10.0, 20.0); λ_2 = 2.0





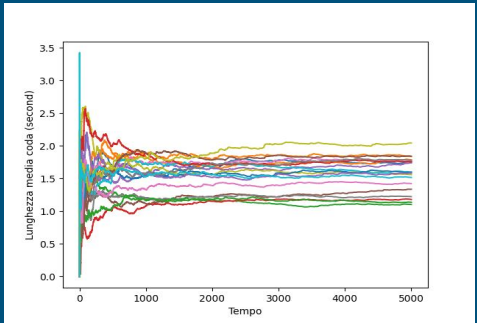


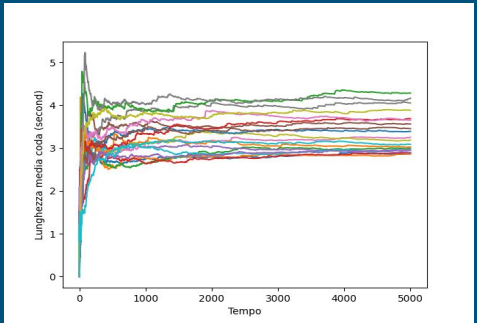


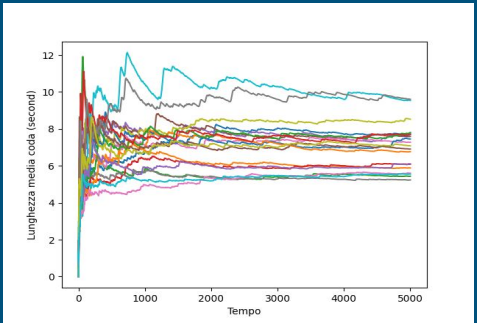


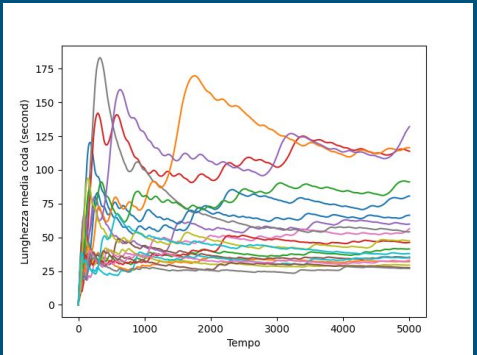
Conf. 1 - Lunghezza della coda “second”

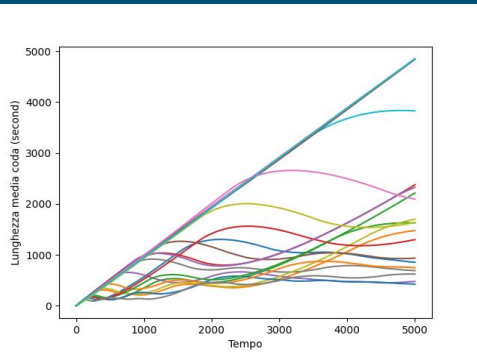
Da sinistra verso destra aumenta il valore di λ_1 (1.0, 2.0, 4.0; 10.0, 20.0); $\lambda_2 = 2.0$







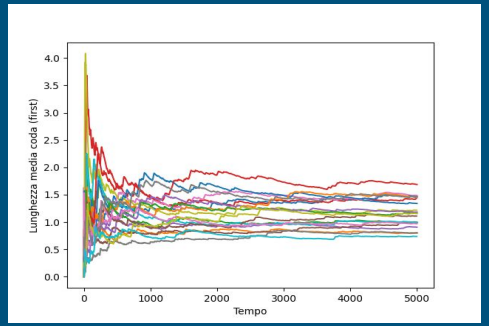


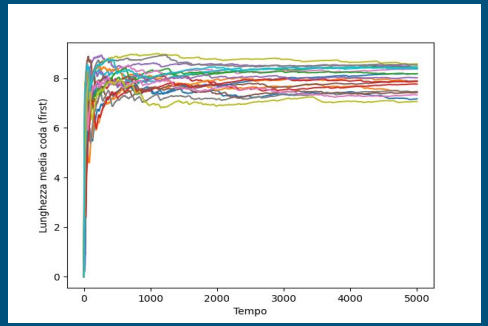


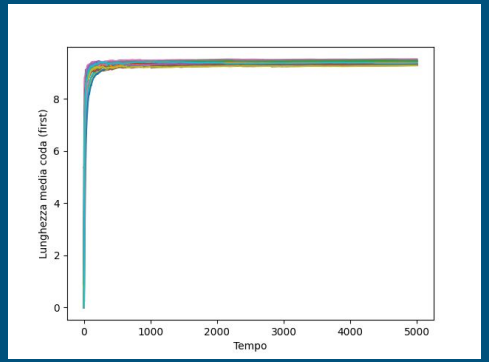
Vettori - configurazione 2

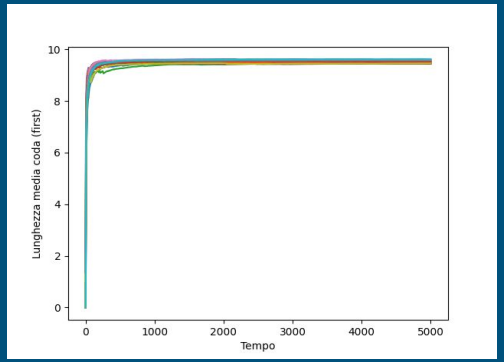
Conf. 2 - Lunghezza della coda “first”

Da sinistra verso destra aumenta il valore di λ_1 (1.0, 2.0; 3.0, 3.5); λ_2 = 1.5



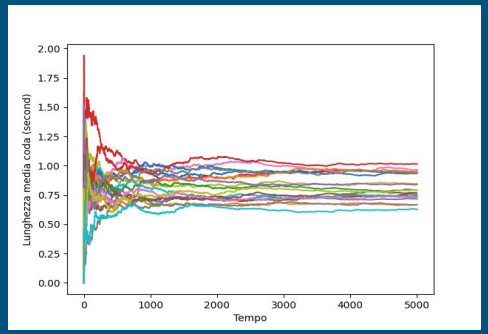


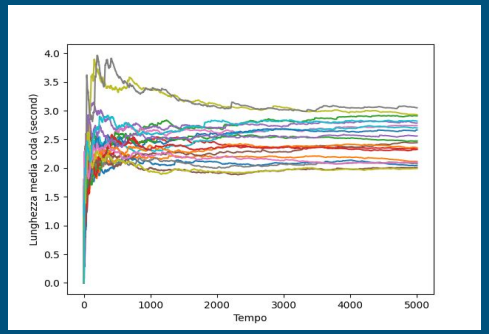


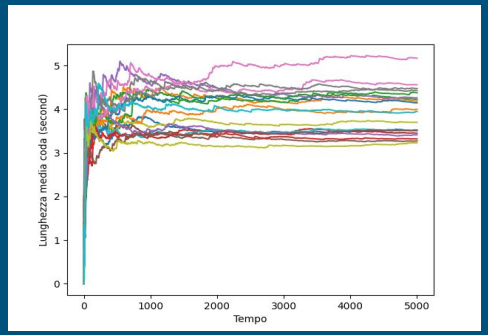


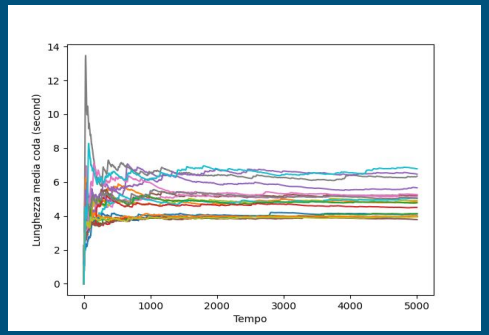
Conf. 2 - Lunghezza della coda “second”

Da sinistra verso destra aumenta il valore di λ_1 (1.0, 2.0, 3.0, 3.5); λ_2 = 1.5



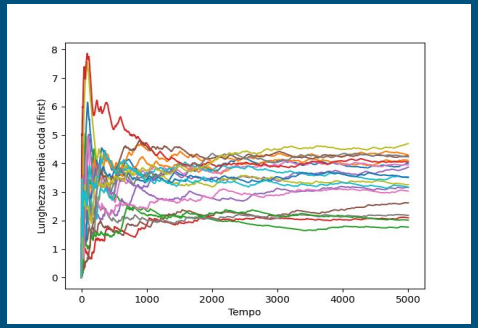


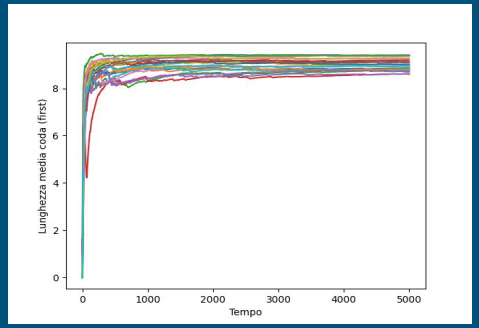


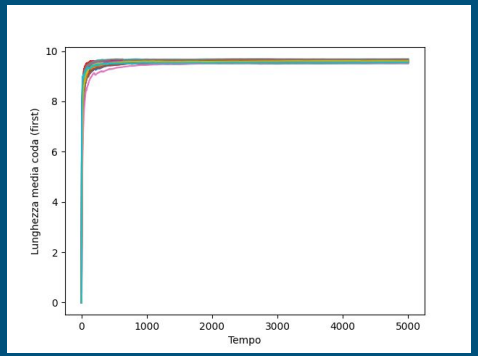


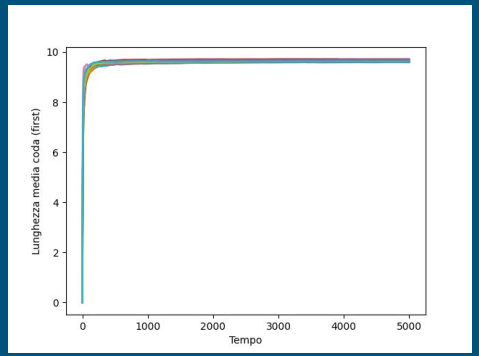
Conf. 2 - Lunghezza della coda “first”

Da sinistra verso destra aumenta il valore di λ_1 (1.0, 2.0; 3.0, 3.5); λ_2 = 2.0



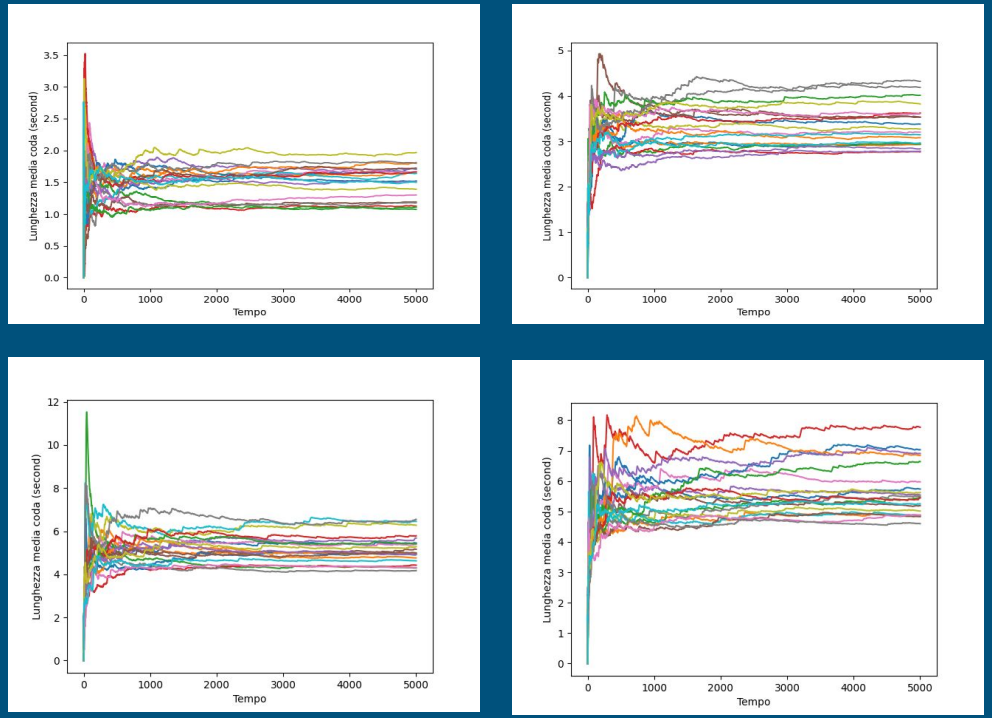






Conf. 2 - Lunghezza della coda “second”

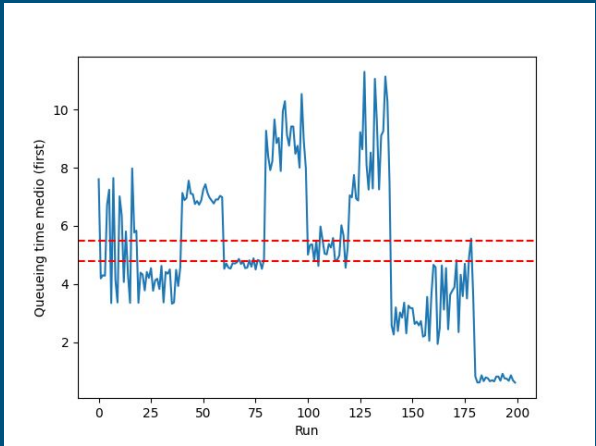
Da sinistra verso destra aumenta il valore di λ_1 (1.0, 2.0; 3.0, 3.5); λ_2 = 2.0

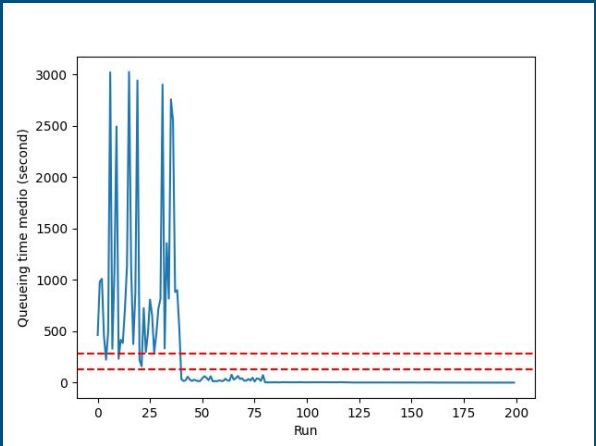


Scalari - configurazione 1

Conf 1 - tempo medio di permanenza

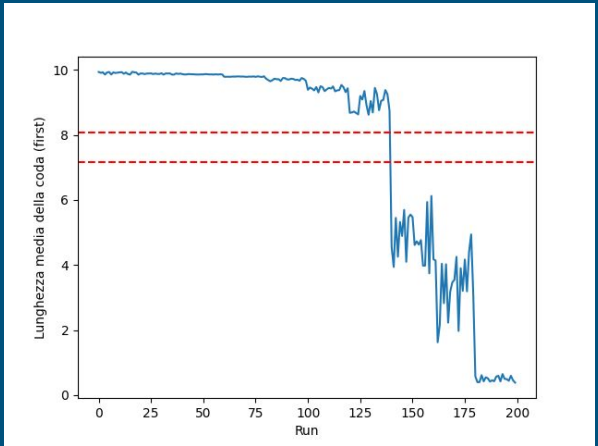
Modulo	Media	Intervallo di confidenza
Network.first	5.12 s	(4.77, 5.47)
Network.second	209.77 s	(131.87, 287.66)
Entrambe le code	107.44 s	(67.23, 147.66)

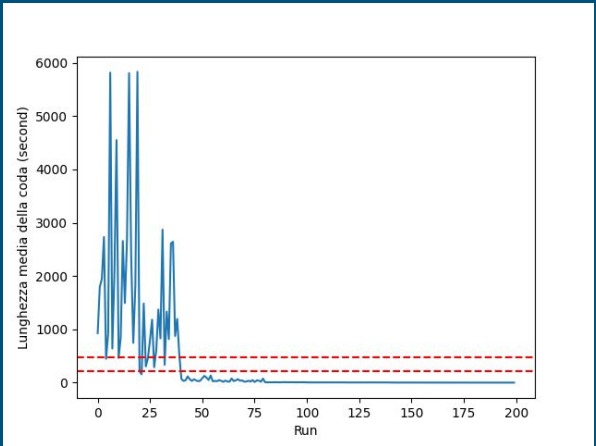




Conf 1 - numero medio di utenti nelle code

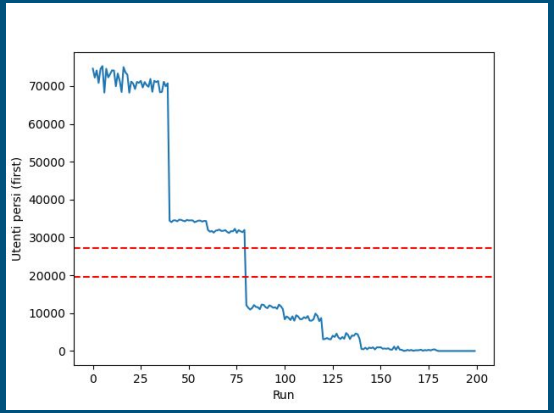
Modulo	Media	Intervallo di confidenza
Network.first	7.62 (8)	(7.18, 8.08)
Network.second	348.47 (348)	(216.75, 480,19)
Entrambe le code	178.05 (178)	(110.10, 246.00)





Conf 1 - numero medio di utenti persi

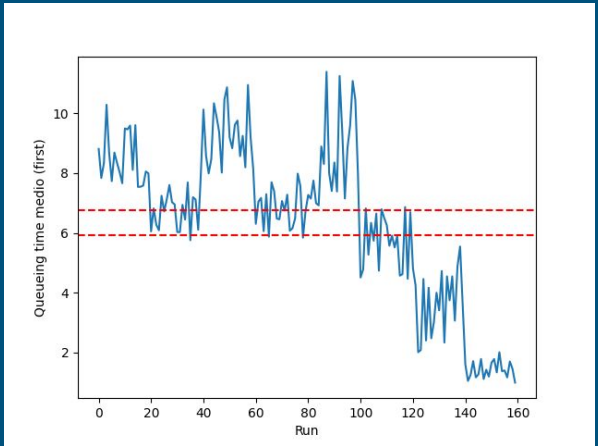
Modulo	Media	Intervallo di confidenza
Network.first	23397.16 (23397)	(19688.32, 27106.00)
Network.second	0	/
Entrambe le code	11698.5800 (11699)	(9518.40, 13878.76)

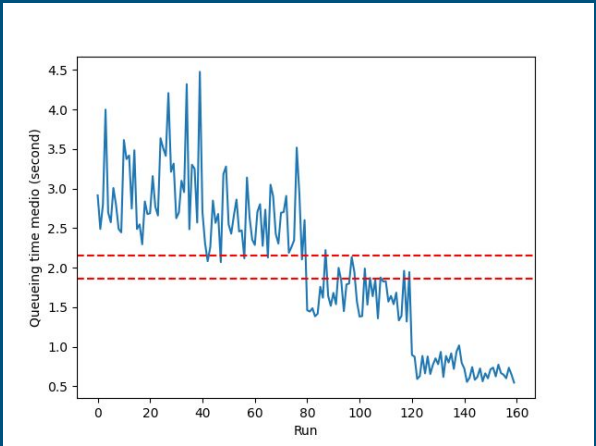


Scalari - configurazione 2

Conf 2 - tempo medio di permanenza

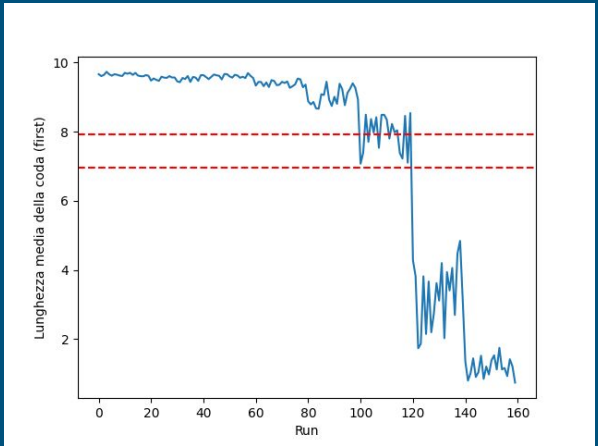
Modulo	Media	Intervallo di confidenza
Network.first	6.34 s	(5.93, 6.75)
Network.second	2.01 s	(1.86, 2.16)
Entrambe le code	4.17 s	(3.85, 4.50)

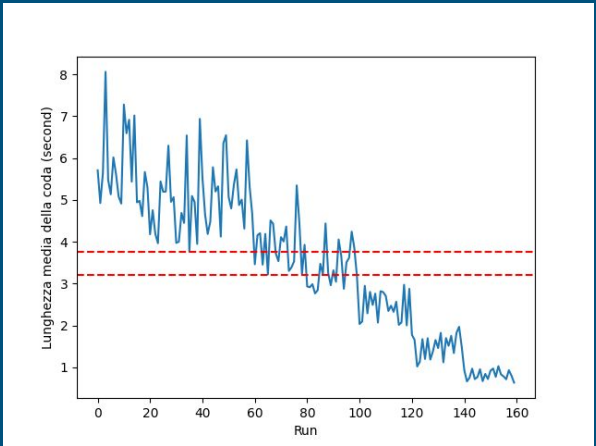




Conf 2 - numero medio di utenti nelle code

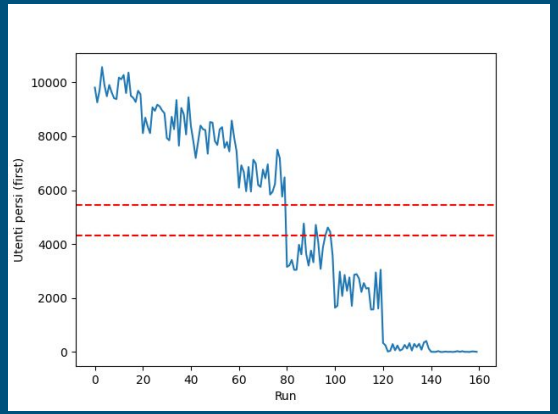
Modulo	Media	Intervallo di confidenza
Network.first	7.45 (7)	(6.97, 7.93)
Network.second	3.49 (3)	(3.22, 3.77)
Entrambe le code	5.47 (5)	(5.12, 5.82)





Conf 2 - numero medio di utenti persi

Modulo	Media	Intervallo di confidenza
Network.first	4890.83 (4891)	(4330.1817, 5451.4933)
Network.second	0	/
Entrambe le code	2445.42 (2445)	(2057.6406, 2833.1969)



Convalida del modello

Processo di convalida

Per la convalida sono stati confrontati i risultati presenti nel capitolo 5 dell'articolo [1] (in particolare nelle tabelle 5.1 - 5.5) con quelli ottenuti nelle stesse condizioni dal modello realizzato.

I parametri analizzati sono stati $E[L_i]$, ovvero il numero medio di utenti nella coda *i* e $E[W_i]$, ovvero il tempo medio passato nella coda *i* dagli utenti.

Per ogni esperimento sono state effettuate 20 ripetizioni della durata di 5000 secondi di tempo simulato l'una.

Risultati della convalida

La convalida ha dato esito positivo, mostrando solo poche differenze tra i risultati ottenuti in [1] e quelli del modello realizzato.

In particolare le differenze più significative sono legate al fatto che, quando i due λ sono molto diversi tra loro e uno è molto maggiore dell'altro, una delle due code si trova sempre più spesso ad essere satura, impedendo al servente di effettuare lo switch over e accumulando di conseguenza utenti nell'altra coda.

Fine della presentazione,
grazie per l’attenzione.

Bibliografia

[1] **Amit Jolles, Efrat Perel, and Uri Yechiali**. *Alternating server with non-zero switch-over times and opposite-queue threshold-based switching policy*. Performance Evaluation, 126:22–38, 2018.