PROGETTO DI MODELLISTICA, SIMULAZIONE E VALUTAZIONE DELLE PRESTAZIONI

Sistema di accesso GreenPark

Abstract

Il Progetto è stato caricato al seguente link:

https://github.com/IlConteCvma/ProjectPMCSN

Marco Calavaro

0295233

Tabella dei contenuti

Contents

Capitolo 1

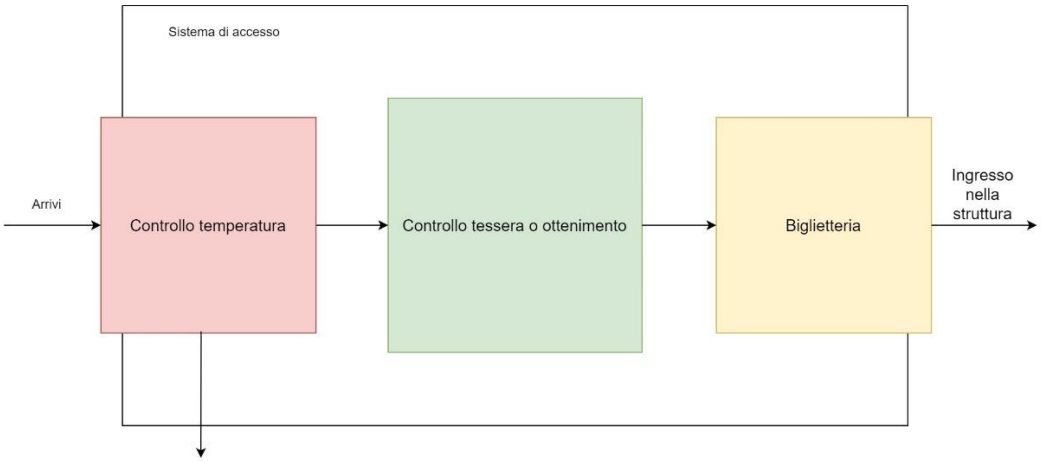
1.1 Sintesi del caso di studio

Per lo sviluppo di un modello di simulazione è stato preso come caso di studio l'associazione culturale "GreenParkGrottaferrata", in particolare il suo sistema di accesso.

Per poter accedere giornalmente alla struttura è necessario:

- Avere la tessera di socio, ottenibile tramite registrazione.
- Versare la quota giornaliera.
- Effettuare controllo della temperatura e sanificazione, meccanismo introdotto a seguito delle direttive sanitarie del governo per la pandemia globale Covid-19.

Si può quindi rappresentare il sistema di accesso tramite una visione a macro-livello che lo suddivide in tre fasi, come riportato nella figura seguente.



1. Il blocco di controllo temperatura offre dei macchinari (termoscanner) in grado di rilevare la temperatura del cliente e segnala coloro che presentano una temperatura superiore ai 37 gradi.

2. Il blocco di controllo tessera o ottenimento offre diversi servizi in base alle necessità del cliente, in particolare:
    a. Postazioni automatiche per il controllo della tessera, in grado anche di direzionare il cliente verso l'apposita coda in caso di mancanza.
    b. Postazioni con dipendenti per la compilazione del modulo necessario all'ottenimento della tessera.
    c. Postazioni con dipendenti per completamento iscrizione, nelle quali vengono effettuati opportuni controlli sui moduli e il consecutivo rilascio della nuova tessera al cliente.

In questo blocco si possono quindi individuare le seguenti categorie di clienti:

- *Cliente socio*: è un cliente che ha già sottoposto l'iscrizione all'associazione, deve essere verificata solo la sua identità per poter poi procedere al pagamento.
- *Nuovo cliente con modulo online*: cliente che ha già effettuato la compilazione del modulo online, necessità apporre la firma in loco per poter completare la registrazione (operazione che necessita di supervisione di un operatore), riceverà la tessera di socio al completamento di tale fase.
- *Nuovo cliente senza modulo*: è un cliente che deve registrarsi all'associazione ma deve effettuare la procedura di registrazione completa.

Bisogna distinguere a sua volta la tipologia di cliente per la consegna del corretto modulo:
    - *Cliente Maggiorenne*.
    - *Famiglia*: rappresenta la presenza di minori che devono avere necessariamente un accompagnatore maggiorenne (che può firmare come tutore).

3. Il blocco biglietteria gestisce il versamento della quota giornaliera in base alla tipologia di cliente che rientrano nelle seguenti categorie:
    - *Cliente singolo*: cliente che deve versare la quota giornaliera direttamente in cassa.
    - *Cliente in gruppo*: cliente che fa parte di un gruppo, non deve versare una quota ma effettuare solo il controllo in lista, il pagamento sarà effettuato dal capogruppo. Non vi è necessità che il capogruppo paghi all'ingresso. Tale cliente ha priorità rispetto i singoli.

Tale sistema di ingresso consente l'arrivo di nuovi clienti nella fascia giornaliera che va dalle 10:00 alle 22:00.

1.2 Obiettivi dello studio

L'obiettivo dello studio è quello di organizzare il meccanismo di accesso che minimizza il numero di postazioni per ciascuna fase, con il fine di ridurre i costi mantenendo però dei buoni tempi di "attraversamento" del sistema per i vari clienti.

Per definire la bontà dei tempi vengono stabiliti i seguenti criteri di qualità (QoS) legati anche alla tipologia di cliente:

Cliente non socio:

- Senza modulo:
  1. Deve sperimentare un tempo nel sistema di non più di $15\ minuti$.
- Con modulo:
  2. Poiché ha già compilato il modulo deve attendere di meno di un cliente senza modulo, deve sperimentare un tempo nel sistema di $7\ minuti$.

Cliente socio:

3. Deve sperimentare un tempo nel sistema di non più di $3\ minuti$.

Budget mensile stanziato per il sistema di ingresso:

4. Il costo complessivo tra dipendenti e macchinari impiegati deve essere inferiore a $15000\ €$

Capitolo 2

2 Modello concettuale

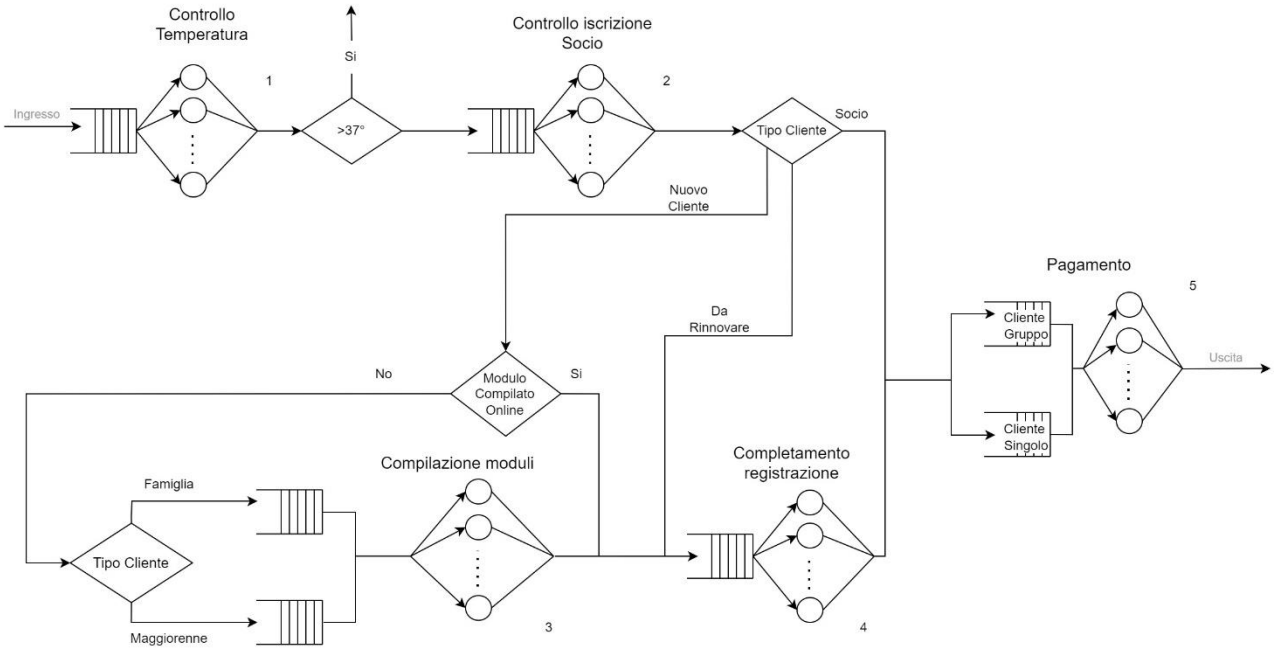


Figura 2.1 Diagramma meccanismo di accesso.

La seguente figura descrive il funzionamento del sistema ad alto livello, non rappresenta quindi lo scheduling effettivo di esso. Tale diagramma utilizza le seguenti notazioni per poter descrivere il comportamento e l'incanalamento del flusso di ingresso:

- Il rombo rappresenta un branch decisionale.
- Le frecce possono assumere un significato di risposta ad una decisione se è presente un testo sopra di esse.
- I cerchi rappresentano i serventi.
- I rettangoli rappresentano le code di attesa.

2.1 Descrizione dettagliata del diagramma

Nella figura si individuano 5 insiemi di postazioni e sono rispettivamente:

1. *Controllo temperatura*: postazioni che offrono servizi legati alle normative Covid.
2. *Controllo iscrizione socio*: postazioni per individuare il tipo di cliente.
3. *Compilazione moduli*.
4. *Completamento registrazione*.
5. *Pagamento*: postazioni per effettuare il versamento della quota giornaliera

Ciascuno di questi insiemi è modellato con una coda del tipo M/M/m in quanto non esiste alcun tipo di limite sulla coda. Per l'insieme "Pagamento" è presente una doppia coda che rappresenta una priorità di servizio.

Per differenziare le coda con priorità nell'insieme di pagamento, con le code logiche nell'insieme di compilazione moduli, si è scelto di non usare un rombo decisionale, ma di marcare direttamente le code con il tipo di priorità così da rimarcare tale differenza.

Nella figura sono presenti 4 branch:

- *>37*: in base alla temperatura osservata vengono scartati i clienti con valore superiore a 37.
- *Due Tipo Clienti*: utilizzati per ridirezionare il flusso in base alla categoria del cliente.
- *Modulo compilato Online*: per ridirezionare il flusso di clienti in base all'aver compilato il modulo online o meno.

2.2 Definizioni variabili di stato

Di seguito saranno elencate e descritte le variabili che definiscono lo stato del sistema ad un dato istante temporale (non rappresentato nella definizioni delle variabili).

Nelle seguenti definizioni l'apice (i) indica l'insieme di postazioni considerato.

- Per ciascun insieme deve essere definita una variabile che identifica il numero di postazioni:

$$N_s^{(i)} \in \mathbb{N}, i \in \{1, \cdots, 5\}$$

In base all'insieme considerato sono stati definiti i seguenti valori di stato:

ServerState 1
IDLE
BUSY
Famiglia

ServerState 2
IDLE
BUSY
Famiglia

ServerState 3
IDLE
Maggiorenne
Famiglia

ServerState 4
IDLE
Completamento Registrazione

ServerState 5
IDLE
Singolo
Gruppo

- Per ciascuna postazione di ogni insieme viene definita una variabile che ne identifica lo stato:

$$Server_j^{(i)} \in ServerState^{(i)}, j \in \left\{1,2,\ldots,N_s^{(i)}\right\}$$

Si definisce $Client\_type$ l'insieme di tutte le possibili coppie (tipo client, tipo pagamento) che il sistema deve gestire.

Tipo cliente "T"
Socio
Rinnovo
Nuovo con modulo
Nuovo senza modulo Maggiorenne
Nuovo senza modulo Famiglia

Tipo pagamento "P"
Singolo
Gruppo

$$Client\_type \stackrel{\text{def}}{=} T \times P = \{(t,p) | t \in T, p \in P\}$$
$$\#\{Client\_type\} = 10$$

A seguito della precedente definizione sono state individuate le seguenti variabili di stato:

- La variabile $Cliente_c^{(i)}$ mantiene il numero di clienti in base al tipo che si trovano nell'insieme $i-esimo$:

$$Cliente_c^{(i)} \in \mathbb{N}, \forall c \in Client\_type\ , i \in \{1,\dots,5\}$$

- La variabile $ClientiServiti_c^{(i)}$ mantiene il numero di clienti in base al tipo che sono attualmente serviti da una postazione dell'insieme $i-esimo$:

$$ClientiServiti_c^{(i)} \in \mathbb{N}, \forall c \in Client\_type\ , i \in \{1,\dots,5\}$$

Dalle precedenti variabili è possibile derivare ulteriori informazioni sul sistema quali:

- Il numero dei clienti che si trovano nell'insieme $i-esimo$ è dato da:

$$Clienti^{(i)} = \sum_{c \in Client\_type} Cliente_c^{(i)}(t)\ , i \in \{1,\dots,5\}$$

- Il totale di clienti nel sistema in base al tipo è dato da:

$$Tot\_Clienti_c = \sum_{i \in \{1,\dots,5\}} Cliente_c^{(i)}(t)\ , \forall c \in Client\_type$$

- Il totale dei clienti nel sistema è dato da:

$$Tot\_Clienti = \sum_{i \in \{1,\dots,5\}} Clienti^{(i)}$$

2.3 Eventi del sistema

Gli eventi che possono avvenire nel sistema sono:

- Arrivo di un cliente.
- Completamento di un servizio ad una specifica postazione.

2.4 Assunzioni nel modello

- Lo scheduling prevede classi di priorità per la coda di pagamento poiché bisogna favorire il cliente in gruppo.
- I clienti hanno un comportamento one-step: ad ogni istante di tempo può spostarsi un solo cliente in ogni insieme.
- Si assume che il cliente di tipo famiglia è composto da coppia minore, tutore.
- Tutte le code seguono la politica FIFO.
- Non è possibile avere una postazione libera se è presente un cliente in coda.

Capitolo 3

3 Modello delle specifiche

3.1 Distribuzione dei dati

Non essendo disponibile un archivio dei dati sulle distribuzioni di interesse, ci si è basati su osservazioni e misurazioni approssimative durante il periodo estivo 2021, sulla base di esse si è ricavati un valore medio per le distribuzioni.

Le medie osservate legate a fattori temporali hanno come unità di misura il minuto.

La giornata lavorativa va dall'orario di apertura 10:00 alla chiusura 22:00, ma è stato notato che il flusso di clienti è maggiore nella fascia 12:00-19:00 pertanto si è ridotto il periodo di osservazione a 7 ore (420 minuti).

La probabilità di perdita dovuta ai controlli della temperatura è $p_{>37} = 0.01$

3.2.1 Probabilità Clienti

Mediamente si sono osservate le seguenti probabilità per i clienti:

Tipologia Cliente:

- Probabilità cliente socio: $p_s = 0.05$
- Probabilità cliente da rinnovare: $p_r = 0.1$
- Probabilità nuovo cliente: $p_n = 1 - (p_s + p_r) = 0.85$
  - Con modulo: $p_{nmo} = 0.2$
  - Senza modulo maggiorenne: $p_{nma} = 0.25$
  - Senza modulo famiglia: $p_{nf} = 0.4$

Tipologia pagamento:

- Probabilità singolo: $p_{ps} = 0.35$
- Probabilità gruppo: $p_{pg} = 1 - p_{ps} = 0.65$

Per ottenere le probabilità della combinazione del tipo cliente e tipo di pagamento basta effettuare il prodotto tra le due probabilità in interesse.

3.2.2 Tasso arrivi

Per poter ricavare il tasso di arrivi giornaliero si è stimato che:

- Giornalmente vengono accolti nella struttura mediamente 450 clienti.
- Periodo di osservazione di 420 minuti.

$$\lambda = \frac{\mathbb{E}[\#clientigiornalieri]}{tempo_{osservazione}} = \frac{450\ clienti}{420\ minuti} \approx 1.071429\ {}^{clienti}/_{minuto}$$

Si è assunto che i clienti arrivino a istanti di tempo casuali, comporta una distribuzione di Poisson per gli arrivi.

3.2.3 Tempi di servizio

I tempi di servizio dipendono da diversi fattori:

- Operazione richiesta.
- Tipo di cliente che richiede tale operazione, la stessa operazione ha tempi diversi in base all'utente, si è usata la notazione descritta nel capitolo del modello concettuale per tipo cliente, si ricorda inoltre che il simbolo "T" indica tutto l'insieme del tipo e il simbolo "P" tutto l'insieme dei pagamenti.

La tabella riassume i valori medi dei tempi di servizio osservati

Insieme server	Stato	Operazione	id	Client_type (t,p)	Tempo servizio
1	*BUSY*	Controllo temperatura	1	$t \neq$ Nuovo senza modulo Famiglia	0.5 min
1	*Famiglia*	Controllo temperatura	2	$t =$ Nuovo senza modulo Famiglia	1 min
2	*BUSY*	Controllo iscrizione socio	3	$t \neq$ Nuovo senza modulo Famiglia	0.5 min
2	*Famiglia*	Controllo iscrizione socio	4	$t =$ Nuovo senza modulo Famiglia	1 min
3	*Maggiorenne*	Compilazione modulo	5	$t =$ Nuovo senza modulo Maggiorenne	4 min
3	*Famiglia*	Compilazione modulo	6	$t =$ Nuovo senza modulo Famiglia	6 min
4	*Completamento*	Completamento registrazione	7	$t \neq$ Socio	3 min
5	*Singolo*	Pagamento Singolo	8	$p =$ Singolo	1.5 min
5	*Gruppo*	Pagamento Gruppo	9	$p =$ Gruppo	0.5 min

Tabella 3.4 Tempi di servizio medi

Di seguito saranno descritte le distribuzioni che caratterizzano i tempi di servizio delle operazioni riportate nella tabella precedente, si farà riferimento ad esse tramite id (aggiunto per soli motivi di trattazione).

1. L'operazione "1" può essere rappresentata da due sotto-processi random (effettivo controllo e igienizzazione mani) si è scelto di modellare il tempo di servizio con:
$$Erlang\left(2, b = \frac{0.5}{2}\right)$$
2. A seguito delle assunzioni fatte per la precedente operazione si modella tale tempo di servizio con:
$$Erlang\left(2, b = \frac{1}{2}\right)$$
3. Si è osservato che l'operazione "3" ha un tempo di servizio random tra i $20\ secondi\ e\ 40\ secondi$, pertanto si è modellato il tempo di servizio con:
$$Uniform\left(\frac{1}{3}, \frac{2}{3}\right)$$
4. A seguito delle assunzioni fatte per la precedente operazione si modella tale tempo di servizio con:
$$Uniform\left(\frac{2}{3}, \frac{4}{3}\right)$$
5. Si è modellato il tempo di servizio dell'operazione "5" tramite una distribuzione normale troncata tra 2 e 10 minuti, con parametri:
$$Normal(4{,}2), truncation\ (a = 2, b = 10)$$
6. Si è modellato il tempo di servizio dell'operazione "6" tramite una distribuzione normale troncata tra 2 e 10 minuti, con parametri:
$$Normal(6{,}2), truncation\ (a = 2, b = 10)$$
7. Si è modellato il tempo di servizio dell'operazione "7" tramite una distribuzione normale troncata tra 1 e 5 minuti, con parametri:
$$Normal(3{,}1), truncation\ (a = 1, b = 5)$$

8. Si è modellato il tempo di servizio dell'operazione "8" tramite una distribuzione normale troncata tra 1 e 2 minuti, con parametri:
$$Normal(1.5,0.2), truncation\ (a = 1, b = 2)$$
9. Si è osservato che l'operazione "9" ha un tempo di servizio random tra i $20\ secondi\ e\ 40\ secondi$, pertanto si è modellato il tempo di servizio con:
$$Uniform\left(\frac{1}{3}, \frac{2}{3}\right)$$

Nota:

Le versioni troncate delle distribuzioni sono state ottenute tramite il metodo del "constrained inversion" descritto nel paragrafo 6.3 del libro.

3.2.4. Costi legati agli impianti

Ciascuna postazione può rappresentare o un macchinario o un dipendente, di seguito saranno riportati dei valori approssimativi dei costi settimanali, mensili o giornalieri in base alle informazioni ricavate sulla rete.

Nel primo insieme le postazioni rappresentano un macchinario di rilevamento temperatura automatico (termo scanner), ciascuno di essi ha un costo settimanale di affitto pari a 60 €.

Nel secondo insieme le postazioni rappresentano un macchinario di scansione tessera e validazione dal costo mensile di 200 €.

Le restanti postazioni impiegano un dipendente con uno stipendio giornaliero di 40 €.

Capitolo 4

4 Modello computazionale

Tale modello è stato realizzato seguendo l'approccio next-event simulation descritto nel libro Discrete-Event Simulation: A First Course (Lawrence), di seguito saranno trattate le componenti necessarie a tele sviluppo.

Si è scelto di utilizzare un linguaggio di alto livello orientato a oggetti per meglio rappresentare le varie componenti di sistema, nello specifico il linguaggio adottato è il *"Python".* Inoltre il codice è stato sviluppato prediligendo la lingua inglese nel nominare le varie componenti del codice.

Il codice di sviluppo è stato suddiviso nei seguenti file:

- Main.py: file principale per il lancio della simulazione.
- SimCore.py: file che contiene le classi che implementano la logica del simulatore.
- SystemConfiguration.py: file che contiene le varie strutture di configurazione del sistema.
- TimeDef.py: file che contiene le classi che implementano logiche temporali.
- Utility.py: file che contiene alcuni funzioni di utility per il sistema.

4.1 Stato del sistema

Ciascun insieme di postazioni (nel codice rinominati ServerSet) ha associato una classe che rappresenta il suo stato (ServerSetStatus). Le variabili di stato mantenute possono essere osservate nel seguente screenshot:

```
class ServerSetStatus:

    def __init__(self) -> None:
        self.clients = [0] * CLIENTTYPENUM              # Number of client for each type in set
        self.servedClients = [0] * CLIENTTYPENUM        # Served number of client for each type
        self.number = 0                                 # Number of job in set
        self.completed = [0] * CLIENTTYPENUM            # Number of completed job
```

Screenshot 4.1 inizializzazione delle variabili di ServerSetStatus

- clients[CLIENTTYPENUM[1]]: rappresenta una lista che mantiene il numero di clienti nel set in base al tipo.
- servedClients[CLIENTTYPENUM]: rappresenta una lista che mantiene il numero di clienti attualmente serviti in base al tipo.
- completed[CLIENTTYPENUM]: rappresenta una lista che mantiene il numero di clienti che sono stati serviti in base al tipo.
- number: rappresenta il numero di job attualmente presenti nel set.

Ciascun server mantiene delle variabile che ne rappresentano lo stato. Di seguito è riportato lo screenshot che rappresenta l'inizializzazione generica di un server.

[1] Variabile che rappresenta il numero totale di tipologie di clienti

```python
class Server:
    def __init__(self,IDset,id,) -> None:
        self.setID = IDset                  # Set identifier
        self.identifier = id                # Server identifier
        self.state = ServerStateType.IDLE   # server state
        self.client = None                  # actual served client
```

Screenshot 4.2 Inizializzazione della classe Server

- setID: identificatore numerico del set.
- identifier: identificatore numerico del server.
- state: variabile che mantiene lo stato del server.
- client: variabile che mantiene l'eventuale cliente che è in servizio in accordo alla tipologia del cliente.

La tipologia del cliente ($Client\_type$) trattata nei precedenti capitoli, a livello computazionale è stata trasformata in una classe che mantiene un identificare univoco per ciascun tipo, oltre ad altre informazioni legate ad esso.

Il seguente screenshot riporta la trasformazione sopra citata:

```python
# Client type according to documentation
class ClientType(Enum):
    SS      = {"index" : 0 , "type": ClientTV.SOCIO    , "pay": ClientPV.SINGOLO , "prob": ClientTProb.SOCIO     .value * ClientPProb.SINGOLO.value }
    SG      = {"index" : 1 , "type": ClientTV.SOCIO    , "pay": ClientPV.GRUPPO  , "prob": ClientTProb.SOCIO     .value * ClientPProb.GRUPPO .value }
    RS      = {"index" : 2 , "type": ClientTV.RINNOVO  , "pay": ClientPV.SINGOLO , "prob": ClientTProb.RINNOVO   .value * ClientPProb.SINGOLO.value }
    RG      = {"index" : 3 , "type": ClientTV.RINNOVO  , "pay": ClientPV.GRUPPO  , "prob": ClientTProb.RINNOVO   .value * ClientPProb.GRUPPO .value }
    NMOS    = {"index" : 4 , "type": ClientTV.NEWMODULO, "pay": ClientPV.SINGOLO , "prob": ClientTProb.NEWMODULO .value * ClientPProb.SINGOLO.value }
    NMOG    = {"index" : 5 , "type": ClientTV.NEWMODULO, "pay": ClientPV.GRUPPO  , "prob": ClientTProb.NEWMODULO .value * ClientPProb.GRUPPO .value }
    NMAS    = {"index" : 6 , "type": ClientTV.NEWMAGG  , "pay": ClientPV.SINGOLO , "prob": ClientTProb.NEWMAGG   .value * ClientPProb.SINGOLO.value }
    NMAG    = {"index" : 7 , "type": ClientTV.NEWMAGG  , "pay": ClientPV.GRUPPO  , "prob": ClientTProb.NEWMAGG   .value * ClientPProb.GRUPPO .value }
    NFS     = {"index" : 8 , "type": ClientTV.NEWFAMILY, "pay": ClientPV.SINGOLO , "prob": ClientTProb.NEWFAMILY .value * ClientPProb.SINGOLO.value }
    NFG     = {"index" : 9 , "type": ClientTV.NEWFAMILY, "pay": ClientPV.GRUPPO  , "prob": ClientTProb.NEWFAMILY .value * ClientPProb.GRUPPO .value }
```

Screenshot 4.3 Classe ClientType

4.2 Eventi

Gli eventi del sistema possono essere suddivisi in due macro tipologie, arrivi ad un certo set o completamento di un server di un set.

- L'arrivo di un cliente di un certo tipo (con valore i corrispondente a $ClientType[\text{“}index\text{”}]$) comporta l'incremento di due variabili di stato del set:

$$client[i] = client[i] + 1$$
$$\text{number} = \text{number} + 1$$

- Il completamento di un server che serviva un cliente di un certo tipo (con valore i corrispondente a $ClientType[\text{“}index\text{”}]$) comporta il passaggio del suo stato a "*IDLE*" e il decremento delle variabili del set:

$$client[i] = client[i] - 1$$
$$servedClient[\text{i}] = servedClient[\text{i}] - 1$$
$$number = number - 1$$

- Quando è disponibile un server per il lavoro viene incrementato il valore della variabile:

$$servedClient[\text{i}] = servedClient[\text{i}] + 1$$

  La tipologia di cliente viene selezionata rispettando la politica Fifo.

Ciascun evento è stato realizzato tramite la classe Event, riportata nell'immagine sottostante:

```python
class Event:
    def __init__(self,typ:EventType,id,client,time) -> None:
        self.time = time        # Occurence of event
        self.typ = typ          # Event type
        self.client = client    # Client
        self.identifier = id    # Identifier of set and server if completation
```

Screenshot 4.4 Classe Event

- Time: rappresenta l'occorrenza dell'evento.
- Typ: la tipologie dell'evento.
- Client: il tipo di cliente associato all'evento.
- Identifier: identificatore del set o del server in cui l'evento avrà luogo.

4.2.1 Lista degli Eventi

Per poter mantenere gli eventi ciascun set mantiene una list Python a cui saranno appesi di volta in volta i nuovi eventi generati per tale set. Per mantenere l'ordinamento degli eventi si è sfruttata la funzione "sort" di Python passando come regola il check sull'attributo time della classe, in questo modo il primo elemento della lista corrisponde al next-event da schedulare per il set.

4.2.2 Generazione degli eventi:

- Possiamo distinguere due tipologie di arrivi:
  1. Esterni: sono gli arrivi nel sistema, essi vengono generati allo startup del sistema e ogni qual volta viene processato un evento di tale tipo. Gestiti dalla funzione "GetArrival".
  2. Interni: sono gli arrivi ad un set causati da un completamento ad un set precedente. Gestiti dalla funzione "StaticArrival"
1. Per quanto riguarda i completamenti si è utilizzata la funzione "GetService":

```python
def GetService(time,clientType : ClientType ,Identifier,serverState) -> Event :

    if (Identifier[0] == 1):
        return __GetServiceSet1(time,clientType,Identifier,serverState)

    if (Identifier[0] == 2):
        return __GetServiceSet2(time,clientType,Identifier,serverState)

    if (Identifier[0] == 3):
        return __GetServiceSet3(time,clientType,Identifier,serverState)

    if (Identifier[0] == 4):
        return __GetServiceSet4(time,clientType,Identifier,serverState)

    if (Identifier[0] == 5):
        return __GetServiceSet5(time,clientType,Identifier,serverState)
```

Screenshot 4.5 Funzione GetService

Come si può notare dall'immagine tale funzione è una interfaccia per invocare la corrispettiva funzione di servizio in base al set che l'ha invocata. Ciascuna di queste funzioni ha un comportamento simile:

- Setta la tipologia di evento a completamento.
- In base allo stato del server genera l'opportuno valore medio.

- Invoca la funzione che genera il tempo di servizio in base alla corretta distribuzione.
- Restituisce l'oggetto evento.

4.3 Clock

Ciascun set mantiene lo scorrere del tempo tramite la variabile timer che punta ad un oggetto di tipo Timer, tale oggetto mantiene:

```
class Timer:
    def __init__(self) -> None:
        self.current        = START                                 # Current time
        self.arrival        = [INFINITY] * SystemConfiguration.CLIENTTYPENUM      # Last Arrival time
        self.completation   = [INFINITY] * SystemConfiguration.CLIENTTYPENUM      # Last Completation time
```

Screenshot 4.8 Classe Timer e sua inizializzazione

- Current : valore attuale dell'orologio del set.
- Arrival : lista che mantiene per ciascuna tipologia di cliente il valore temporale dell'ultimo arrivo.
- Completation: lista che mantiene per ciascuna tipologia di cliente il valore temporale dell'ultimo completamento.

4.4 Scheduler

Per come sono state implementate le liste di eventi per poter garantire il corretto scorrere del tempo nei vari timer dei set vengono gestiti due tipi di aggiornamento:

1. Quando viene selezionato un nuovo evento per un certo set, si aggiorna il valore current del timer al valore dell'evento selezionato.
2. Quando viene aggiunto un evento alla lista si aggiorna il valore di arrival o completation in base al suo tipo.

4.5 Algoritmo next-event

Di seguito sarà riportata la logica dell'algoritmo next-event

4.5.1 Inizializzazione

Viene creato un oggetto Simulation che è il core della logica del simulatore, nella funzione di init vengono inizializzati i suoi attributi e vengono creati i 5 set di server. Ciascun set alla creazione invocherà la sua funzione di init che preparerà a sua volta il corretto stato iniziale.

Per procedere al run della simulazione si invoca la funzione "startSimulation" che come primo step genera gli arrivi sul primo set di server e poi passa ad un loop per la selezione e processamento del next-event.

4.5.2 Logica del loop

Il loop che gestisce gli eventi avviene seguendo questa logica:

- Fino a quando il tempo di esecuzione è minore di un valore "close-the-door" vengono eventualmente generati nuovi arrivi dall'esterno.
- Se vi sono ancora eventi da processare in uno dei set lo scheduling continua a lavorare anche oltre il tempo di simulazione.
- Viene selezionato il prossimo evento osservando gli istanti temporali del primo elemento della lista degli eventi di ciascun set.
- Si processa opportunamente l'evento.
- L'esecuzione termina quando non vi sono più eventi nelle varie liste.

4.5.3 Processamento dell'evento

Il processamento dell'evento avviene seguendo tale logica:

- Viene determinata la tipologia di evento e il set di appartenenza.
- Si aggiorna l'area del set. (sarà descritta nel prossimo paragrafo)
- Si aggiorna il valore corrente del Timer del set.
- In base ai valori determinati nel primo punto si processa opportunamente l'evento:
    - Si aggiornano le variabili di stato come descritto nei precedenti paragrafi.
    - Se l'evento è un arrivo ed è disponibile un server viene schedulato un job disponibile in una coda (si genera quindi l'evento di completamento cambiando lo stato del server).
    - Se l'evento è un completamento si genera l'evento di arrivo al set opportuno.

4.6 Statistiche

Seguendo gli studi del libro e le caratteristiche del sistema sono state calcolate diverse statistiche la loro descrizione è riportata all'interno di tale paragrafo.

4.6.1 Job-averaged statistic

In accordo alle definizioni del libro di testo (1.2.4 e 1.2.5) sono state definite le seguenti statistiche:

- Average interarrival time in base al tipo di cliente.
- Average service time.
- Average delay in queue.
- Average wait time nel set.

4.6.2 Time-averaged

Seguendo le definizione (1.2.6 del libro di testo) sono stati calcolati i valori medi nell'intervallo di simulazione $(0, \tau)$ per le seguenti statistiche:

- Time-averaged number in set.
- Time-averaged number in queue.
- Time-averaged number in service.

4.6.3 Implementazione del calcolo

Data la tipologia di algoritmo stepwise è possibile calcolare le aree necessarie alla stima delle sopra citate statistiche mediante la funzione "UpdateArea" che computa i valori ad ogni ciclo dell'algoritmo.

```python
class Area:
    def __init__(self,nQueue) -> None:

        self.clients        = 0.0
        self.queue          = 0.0
        self.service        = 0.0


    def UpdateArea(self,globalTime,current,number,service):
        self.clients    += (globalTime - current) * number
        self.queue      += (globalTime - current) * (number - service)
        self.service    += (globalTime - current) * service
```

Screenshot 4.9 Classe Area

```python
def getStatistics(self) -> dict:
    stat = {}
    temp = {}
    completations = sum(self.status.completed)
    current = self.timer.current
    stat ["time"]           = current
    stat ["completations"]  = completations
    for elem in list(ClientType):
        idx = elem.value["index"]
        div = self.status.completed[idx]
        if div != 0 :
            temp[elem] = self.timer.arrival[idx] / div
        else:
            temp[elem] = 0

    # From book notation
    stat ["r"]  = temp

    stat ["s"]  = self.area.service / completations
    stat ["d"]  = self.area.queue   / completations
    stat ["w"]  = self.area.clients / completations

    stat ["l"]  = self.area.clients / current
    stat ["q"]  = self.area.queue   / current
    stat ["x"]  = self.area.service / current

    return stat
```

Screenshot 4.10 Funzione GetStatistics

Capitolo 5

5 Verifica

In questo capitolo sono stati riportati alcuni degli strumenti utilizzati per il controllo della correttezza del codice sviluppato. Per poter visualizzare gli output basta lanciare il codice “main.py” senza il parametro “-O” che disabilita il debug.

5.1 Verifica con debug attivo

Per effettuare la verifica del sistema si è osservato lo stato di esso ad ogni iterazione di simulazione, mediante l’utilizzo della funzione “__printDebugUpdate” che, come si può osservare nella figura 5.1, riporta informazioni sull’evento corrente selezionato dall’algoritmo e sul set che lo sta gestendo.

```
def __printDebugUpdate(self,event:Event,set:ServerSet):
    print("\n-------------------Event info-------------------")
    typ = event.typ
    if (typ == EventType.ARRIVAL):
        print("\nArrival event at set: {} \n\tclient: {}\n".format(event.identifier,event.client))
    else:
        serveridx = event.identifier[1]
        print("\nCompletation event at set: {}\n\t server: {}\n\t client: {}\n\t status: {}\n".format(event.identifier[0],serveridx,event.client,set.servers[serveridx].state)

    print("           ------------Set info------------")
    print("Set time:\n\tCurrent:\t{}\n\tArrival:\t{}\n\tCompletation:\t{}\n".format(set.timer.current,set.timer.arrival,set.timer.completation))
    statusStats = set.status.GetStats()
    setId = set.identifier
    print("Set [{}] numer of job: {} ".format(setId,statusStats["Total job"]))


    for elem in list(ClientType):
        # TODO better print
        print("\t{}".format(statusStats[elem]))



    print("\n-------------------End info-------------------\n")
    #time.sleep(1)
```

Figura 5.1 Funzione DebugUpdate

Oltre alla stampa su i singoli eventi generati, è stata anche verificata la consistenza sulle statistiche di output mediante la funzione “__printStatistics”

```
def __printStatistics(self):
    print("\n-------------------Simulation stats-------------------")
    print("\n SISTEM DISCARDED clients are : {}".format(self.discarded))

    print("\n.....................Set stats.....................\n")
    set:ServerSet = None
    for set in self.serverSets:
        populationStats = set.getStatistics()
        statusStats = set.status.GetStats()

        print("Set time:\n\tCurrent:\t{}\n\tArrival:\t{}\n\tCompletation:\t{}\n".format(set.timer.current,set.timer.arrival,set.timer.completation))

        for elem in list(ClientType):
        # TODO better print
            print("\t{}".format(statusStats[elem]))

        print("\nSET [{}] STATISTICS REPORT".format(set.identifier))
        print("\n\tCompletation time          :{:10.2f}".format(populationStats["time"]))
        print("\tNumber of completation     :{:10.2f}".format(populationStats["completations"]))
        print("")
        for elem in list(ClientType):
            temp = populationStats["r"][elem]
            print("\tAvarage interarrival time for client ({:4})  :{:7.2f}".format(elem.name,temp))
        print("")
        print("\tAvarage service time       :{:10.2f}".format(populationStats["s"]))
        print("\tAvarage delay              :{:10.2f}".format(populationStats["d"]))
        print("\tAvarage wait               :{:10.2f}".format(populationStats["w"]))
        print("\tAvarage number in set      :{:10.2f}".format(populationStats["l"]))
        print("\tAvarage number in queue    :{:10.2f}".format(populationStats["q"]))
        print("\tAvarage number in service  :{:10.2f}".format(populationStats["x"]))
        print("\n.....................................................\n")
```

Figura 5.2 Funzione printStatistics

Si è verificato che in ogni set:

$$\bar{w} = \bar{d} + \bar{s}$$

$$\bar{l} = \bar{q} + \bar{x}$$

Si è verificato che il numero di completamenti sia conforme alla progettazione del sistema.

Di seguito sono riportate le immagini di un run di simulazione.

```
SISTEM DISCARDED clients are : 3
```

```
SET [1] STATISTICS REPORT

        Completation time          :    422.61
        Number of completation     :    460.00

        Avarage interarrival time for client (SS  )  :  29.72
        Avarage interarrival time for client (SG  )  :  44.55
        Avarage interarrival time for client (RS  )  :  25.66
        Avarage interarrival time for client (RG  )  :  11.53
        Avarage interarrival time for client (NMOS)  :  12.14
        Avarage interarrival time for client (NMOG)  :   6.74
        Avarage interarrival time for client (NMAS)  :  13.11
        Avarage interarrival time for client (NMAG)  :   5.63
        Avarage interarrival time for client (NFS )  :   5.75
        Avarage interarrival time for client (NFG )  :   3.41

        Avarage service time       :      0.70
        Avarage delay              :      0.07
        Avarage wait               :      0.77
        Avarage number in set      :      0.83
        Avarage number in queue    :      0.07
        Avarage number in service  :      0.76
```

```
SET [2] STATISTICS REPORT

        Completation time          :    422.79
        Number of completation     :    457.00

        Avarage interarrival time for client (SS  )  :  29.72
        Avarage interarrival time for client (SG  )  :  44.55
        Avarage interarrival time for client (RS  )  :  25.67
        Avarage interarrival time for client (RG  )  :  11.55
        Avarage interarrival time for client (NMOS)  :  12.16
        Avarage interarrival time for client (NMOG)  :   6.74
        Avarage interarrival time for client (NMAS)  :  13.13
        Avarage interarrival time for client (NMAG)  :   5.79
        Avarage interarrival time for client (NFS )  :   5.84
        Avarage interarrival time for client (NFG )  :   3.44

        Avarage service time       :      0.71
        Avarage delay              :      0.05
        Avarage wait               :      0.76
        Avarage number in set      :      0.82
        Avarage number in queue    :      0.06
        Avarage number in service  :      0.77
```

```
SET [3] STATISTICS REPORT

        Completation time          :    760.04
        Number of completation     :    298.00

        Avarage interarrival time for client (SS  )  :   0.00
        Avarage interarrival time for client (SG  )  :   0.00
        Avarage interarrival time for client (RS  )  :   0.00
        Avarage interarrival time for client (RG  )  :   0.00
        Avarage interarrival time for client (NMOS)  :   0.00
        Avarage interarrival time for client (NMOG)  :   0.00
        Avarage interarrival time for client (NMAS)  :  13.13
        Avarage interarrival time for client (NMAG)  :   5.80
        Avarage interarrival time for client (NFS )  :   5.85
        Avarage interarrival time for client (NFG )  :   3.44

        Avarage service time       :      5.04
        Avarage delay              :    175.53
        Avarage wait               :    180.57
        Avarage number in set      :     70.80
        Avarage number in queue    :     68.82
        Avarage number in service  :      1.98
```

```
SET [4] STATISTICS REPORT

        Completation time          :    760.29
        Number of completation     :    439.00

        Avarage interarrival time for client (SS  )  :   0.00
        Avarage interarrival time for client (SG  )  :   0.00
        Avarage interarrival time for client (RS  )  :  25.73
        Avarage interarrival time for client (RG  )  :  11.56
        Avarage interarrival time for client (NMOS)  :  12.16
        Avarage interarrival time for client (NMOG)  :   6.75
        Avarage interarrival time for client (NMAS)  :  23.69
        Avarage interarrival time for client (NMAG)  :  10.44
        Avarage interarrival time for client (NFS )  :  10.62
        Avarage interarrival time for client (NFG )  :   6.18

        Avarage service time       :      3.02
        Avarage delay              :     26.04
        Avarage wait               :     29.06
        Avarage number in set      :     16.78
        Avarage number in queue    :     15.04
        Avarage number in service  :      1.74
```

```
SET [5] STATISTICS REPORT

        Completation time          :    760.49
        Number of completation     :    457.00

        Avarage interarrival time for client (SS  )  :  29.74
        Avarage interarrival time for client (SG  )  :  44.60
        Avarage interarrival time for client (RS  )  :  34.06
        Avarage interarrival time for client (RG  )  :  15.60
        Avarage interarrival time for client (NMOS)  :  16.58
        Avarage interarrival time for client (NMOG)  :   9.63
        Avarage interarrival time for client (NMAS)  :  23.72
        Avarage interarrival time for client (NMAG)  :  10.45
        Avarage interarrival time for client (NFS )  :  10.68
        Avarage interarrival time for client (NFG )  :   6.18

        Avarage service time       :      0.82
        Avarage delay              :      0.05
        Avarage wait               :      0.88
        Avarage number in set      :      0.53
        Avarage number in queue    :      0.03
        Avarage number in service  :      0.49
```

Figur3 5.3 Statistiche finali di un run di simulazione

Capitolo 6

6 Confronto modello analitico

Per poter ulteriormente verificare la correttezza del simulatore sviluppato si è impostato uno studio che ha permesso un confronto con il modello analitico.

Per poter ottenere tale confronto sono state assunte delle semplificazioni sulle distribuzioni dei tempi di servizio che vengono considerate come esponenziali, di media pari al valore medio nel set dei tempi riportati in tabella, nel capitolo delle specifiche.

Sono state effettuate sia analisi a orizzonte finito che infinito, in particolare:

- Per l'orizzonte finito sono state effettuate 100 ripetizioni del test, inizialmente si è impostato il seed al valore "123456789", per ogni replica si è utilizzata la funzione "getSeed" per ottenere il nuovo valore del seed.
- Per l'orizzonte infinito si è utilizzato il metodo delle batch mean impostando $(b, k) = (256{,}64)$. Tali parametri sono stati individuati fissando il valore k e ricercando il valore di b che permetteva di avere il lag 1 di autocorrelazione minore di 0.2. (Seguendo quindi le indicazioni di Banks, Carson, Nelson e Nicol). Il computo di tale valore è stato effettuato tramite il programma acs.py.

Per ottenere il valore medio e l'intervallo di confidenza al 95% si è utilizzato il programma "reader.py".

In questa fase dello studio non si è ancora interessati a rispettare i QoS pertanto si è impostata una configurazione che potesse garantire a livello teorico utilizzazione inferiore a uno nei vari set.

	Set 1	Set 2	Set 3	Set 4	Set 5
Numero server	2	2	4	4	2

Le formule teoriche utilizzate per il confronto sono:

- Erlang-C:

$$E[T_{Q,set}] = P_Q * \frac{\rho}{\lambda(1-\rho)} \;, P_Q = \frac{(m\rho)^m}{m!\,(1-\rho)} * p(0) \;, p(0) = \frac{1}{\sum_{i=0}^{m-1} \frac{(m\rho)^i}{i!} + \frac{(m\rho)^m}{m!\,(1-\rho)}}$$

- Tempo attesa nel set:

$$E[T_{S,set}] = E[T_{Q,set}] + E[S_i]$$

- Abstract priority whitout preemption multiserver:

$$E[T_{Q,set5}] = p_1 \frac{P_{Q,1} * E[S]}{(1-\rho_1)} + p_2 \frac{P_Q * E[S]}{(1-\rho)(1-\rho_1)}$$

Di seguito saranno calcolate le statistiche teoriche e il relativo confronto con il risultato di simulazione per ciascun set:

6.1 Confronto per set 1

- Tasso di arrivo $\lambda_1 = \frac{450\ clienti}{420\ minuti} \approx 1.071429\ {}^{clienti}/_{minuto}$
- Tasso di servizio $\mu_1 = \frac{1}{E[S_i]} = \frac{1}{\frac{1+0.5}{2}} = 1.\bar{3}\ {}^{clienti}/_{minuto}$
- Utilizzazione globale $\rho_1 = \frac{\lambda_1}{m\mu_1} = \frac{1.071429}{2 * 1.\bar{3}} = 0.401786$

Statistica in analisi	Risultato teorico	Risultato simulazione (orizzonte finito)	Risultato simulazione (orizzonte infinito)
$\boldsymbol{\rho}$	0.401786	0.402 +/- 0.005	0.395 +/- 0.014
$\boldsymbol{E[T_{Q,1}]}$	0.1443819	0.14 +/- 0.01	0.137 +/- 0.013
$\boldsymbol{E[S_i]}$	0.75	0.75 +/- 0.01	0.743 +/- 0.023
$\boldsymbol{E[T_{S,1}]}$	0.8943819	0.89 +/- 0.01	0.880 +/- 0.031

6.2 Confronto per set 2

- Tasso di arrivo $\lambda_2 = \lambda_1 * (1 - p_{>37}) \approx 1.060715 \ {}^{clienti}/_{minuto}$
- Tasso di servizio $\mu_2 = \frac{1}{E[S_i]} = \frac{1}{\frac{1+0.5}{2}} = 1.\overline{3} \ {}^{clienti}/_{minuto}$
- Utilizzazione globale $\rho_2 = \frac{\lambda_2}{m\mu_2} = \frac{1.060715}{2 * 1.\overline{3}} = 0.397768$

Statistica in analisi	Risultato teorico	Risultato simulazione (orizzonte finito)	Risultato simulazione (orizzonte infinito)
$\boldsymbol{\rho}$	0.397768	0.395 +/- 0.005	0.390 +/- 0.014
$\boldsymbol{E[T_{Q,2}]}$	0.140968	0.13 +/- 0.01	0.140 +/- 0.013
$\boldsymbol{E[S_{i,2}]}$	0.75	0.75 +/- 0.01	0.746 +/- 0.012
$\boldsymbol{E[T_{S,2}]}$	0.890968	0.88 +/- 0.01	0.885 +/- 0.021

6.3 Confronto per set 3

- Probabilità clienti verso questo centro:

$$p_{c3} = p_{nma} + p_{nf} = 0.65$$

- Tasso di arrivo $\lambda_3 = \lambda_2 * p_{c3} \approx 0.68946475 \ {}^{clienti}/_{minuto}$
- Tasso di servizio $\mu_3 = \frac{1}{E[S_i]} = \frac{1}{5} = 0.2 \ {}^{clienti}/_{minuto}$
- Utilizzazione globale $\rho_3 = \frac{\lambda_3}{m\mu_3} = \frac{0.68946475}{4 * 0.2} = 0.861831$

Statistica in analisi	Risultato teorico	Risultato simulazione (orizzonte finito)	Risultato simulazione (orizzonte infinito)
$\boldsymbol{\rho}$	0.861831	0.83 +/- 0.013	0.853 +/- 0.035
$\boldsymbol{E[T_{Q,3}]}$	6.552017	5.27 +/- 0.90	6.549 +/- 1.443
$\boldsymbol{E[S_{i,3}]}$	5	5.03 +/- 0.06	5.073 +/- 0.111
$\boldsymbol{E[T_{S,3}]}$	11.552017	10.29 +/- 0.94	11.621 +/- 1.496

6.4 Confronto per set 4

- Probabilità clienti verso questo centro:

$$p_{c4} = p_{nmo} + p_r = 0.3$$

- Tasso di arrivo $\lambda_4 = \lambda_2 * p_{c4} + \lambda_3 \approx 1.00767925 \ {}^{clienti}/_{minuto}$
- Tasso di servizio $\mu_4 = \frac{1}{E[S_i]} = \frac{1}{3} = 0.\overline{3} \ {}^{clienti}/_{minuto}$
- Utilizzazione globale $\rho_4 = \frac{\lambda_4}{m\mu_4} = \frac{1.0076795}{4 * 0.\overline{3}} = 0.7557594$

Statistica in analisi	Risultato teorico	Risultato simulazione (orizzonte finito)	Risultato simulazione (orizzonte infinito)
$\boldsymbol{\rho}$	0.7557594	0.719 +/- 0.01	0.744 +/- 0.028
$\boldsymbol{E[T_{Q,4}]}$	1.594159	1.37 +/- 0.15	1.411 +/- 0.224
$\boldsymbol{E[S_{i,4}]}$	3	3.01 +/- 0.03	3.005 +/- 0.051
$\boldsymbol{E[T_{S,4}]}$	4.594159	4.38 +/- 0.17	4.416 +/- 0.252

6.5 Confronto per set 5

- Tasso di arrivo $\lambda_5 = \lambda_2 \approx 1.060715\ {clienti}/{minuto}$
- Tasso di servizio $\mu_5 = \frac{1}{E[S_i]} = \frac{1}{1} = 1\ {clienti}/{minuto}$
- Utilizzazione globale $\rho_5 = \frac{\lambda_5}{m\mu_5} = \frac{1.060715}{2*1} = 0.5303575$
- Utilizzazione classe 1 $\rho_{5,1} = \rho_5 * p_{pg} = 0.5303575 * 0.65 = 0.344732375$

Statistica in analisi	Risultato teorico	Risultato simulazione (orizzonte finito)	Risultato simulazione (orizzonte infinito)
$\boldsymbol{\rho}$	0.5303575	0.507 +/- 0.006	0.527 +/- 0.020
$\boldsymbol{E[T_{Q,5}]}$	0.449281	0.37 +/- 0.02	0.404 +/- 0.038
$\boldsymbol{E[S_{i,5}]}$	1	1.01 +/- 0.01	1.014 +/- 0.015
$\boldsymbol{E[T_{S,5}]}$	1. 449281	1.38 +/- 0.03	1.418 +/- 0.047

6.6 Confronto per tempi risposta globali:

Il tempo di risposta globale può essere calcolato tramite la seguente formula:

$$\nu_i = \frac{\lambda_i}{\gamma}, E[T_S] = \sum_{i=1}^{5} \nu_i * E[T_{S,i}]$$

In tabella sono riportati tali tempi computati:

Statistica in analisi	Risultato teorico	Risultato simulazione (orizzonte finito)	Risultato simulazione (orizzonte infinito)
$\boldsymbol{E[T_S]}$	21.6038733	17.82 +/- 1.16	19.201 +/- 1.546

Capitolo 7

7 Validazione

Non avendo a disposizione un dataset con i dati reali la validazione è stata effettuata tramite controlli di consistenza, con l'obiettivo di verificare che il simulatore abbia un comportamento aspettato.

- Il precedente capitolo ha messo in evidenza che il simulatore, configurato con le opportune assunzioni, si comporta come il modello analitico corrispondente, soprattutto quando si vanno ad analizzare le statistiche a orizzonte infinito.

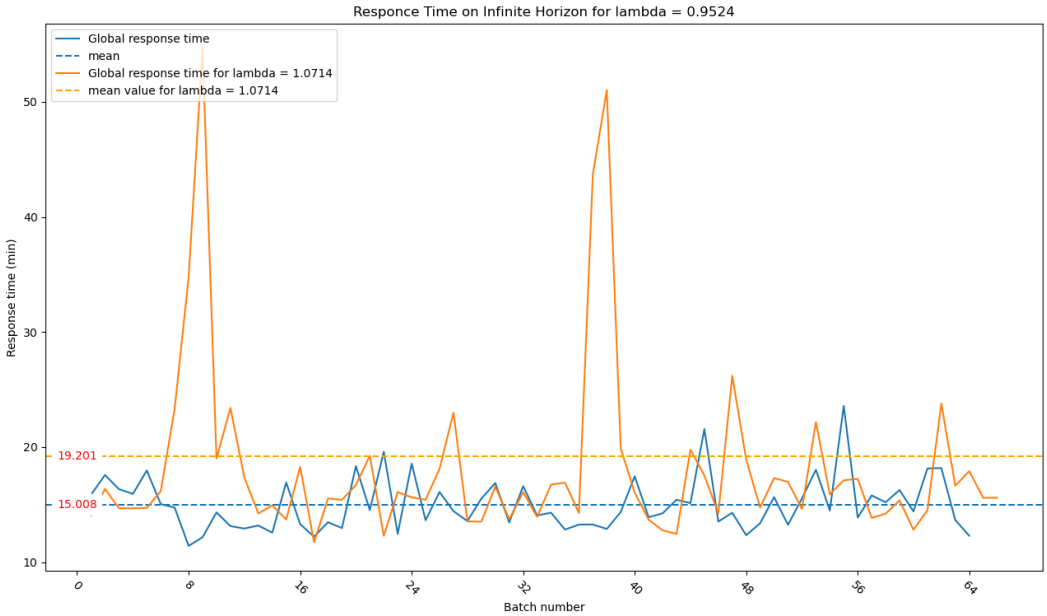
7.1 Ulteriori analisi partendo dal modello analitico

Mantenendo le assunzioni del capitolo precedente sui servizi, stessa configurazione dei server e probabilità di routing, si è verificato che al crescere di lambda il tempo di risposta del sistema cresce e viceversa.

- Lo studio è stato effettuato a orizzonte infinito, impostando $(b, k) = (256,64)$.
- Il valore iniziale dei seed è stato fissato a 72392 per ogni test.
- In particolare osservando le utilizzazioni dei server ci si aspetta che il sistema diventi instabile con un tasso degli arrivi $\lambda > 1.2432$.

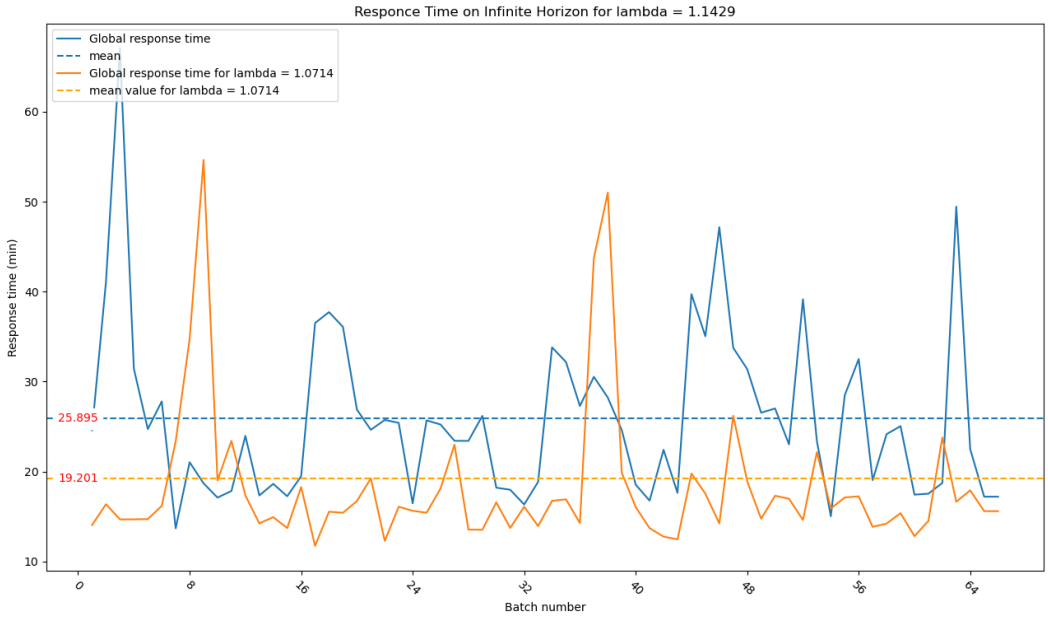
Di seguito saranno riportati grafici dei tempi di risposta degli studi effettuati.

Confronto con tasso di arrivi più piccolo



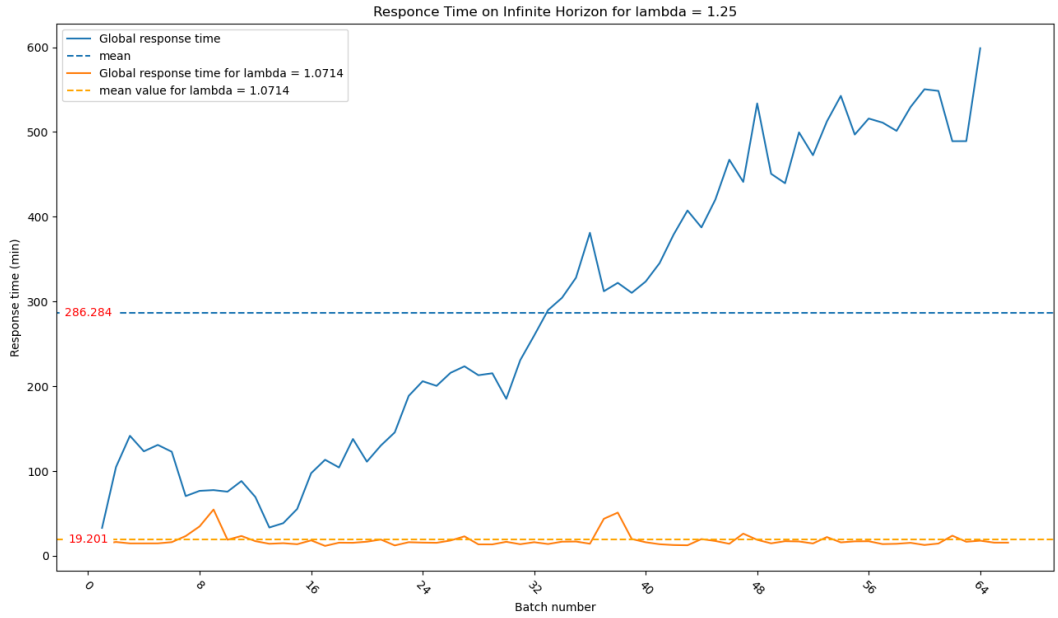
	Set 1	Set 2	Set 3	Set 4	Set 5
ρ	0.345	0.343	0.750	0.653	0.457

Confronto con tasso di arrivi più grande



	Set 1	Set 2	Set 3	Set 4	Set 5
ρ	0.415	0.412	0.899	0.784	0.550

Tempo di risposta con tasso arrivi $\lambda > 1.2432$



	Set 1	Set 2	Set 3	Set 4	Set 5
ρ	0.469	0.465	1	0.876	0.615

7.1.1 Osservazioni

Dalle immagini sopra riportate possiamo osservare che il simulatore al variare di lambda ha un comportamento aspettato. Inoltre dall'ultima immagine si può notare come l'utilizzazione nel set 3 porti il tempo di risposta a crescere in modo indefinito non raggiungendo la stazionarietà.

Si è ripetuto lo stesso studio anche per effettuare controlli di consistenza con il simulatore impostando i tempi di servizio in accordo alle distribuzioni del capitolo delle specifiche. Per brevità di trattazione non saranno riportati i risultati, poiché il comportamento osservato è lo stesso del caso in cui è stata utilizzata la distribuzione esponenziale.

Capitolo 8

8 Esperimenti di Simulazione

In questo capitolo saranno trattati gli esperimenti che sono stati progettati con lo scopo di rispettare i requisiti di qualità descritti nel capitolo degli obiettivi.

Impostazione degli esperimenti:

- È stata realizzata una analisi a orizzonte finito considerando l'intervallo temporale di 7 ore, che corrisponde a 420 minuti. (Valore impostato come STOP per gli arrivi)
- Inizialmente il simulatore è vuoto e la simulazione termina quando si raggiunge nuovamente lo stato iniziale.
- I tempi di servizio e il tasso di arrivi del simulatore sono stati impostati in accordo al capitolo delle specifiche.
- Si è interessati a campionare, a intervalli di 1 minuto, le statistiche associate a ciascuno dei 5 insiemi di postazioni (Set). Tali statistiche saranno salvate su un file csv per successiva analisi.
- Per poter ottenere una maggior quantità di dati si è utilizzato il metodo "replication" effettuando 100 repliche per ogni esperimento, per poter ottenere la media e l'intervallo di confidenza al 95% si è sfruttato una modifica del codice "estimate.py". (Gli intervalli di confidenza sono stati graficati come una zona in trasparenza mentre la curva rappresenta l'andamento della media).
- Il numero di repliche è stato scelto non eccessivamente grande per avere un certo grado di variabilità nel dataset risultante.
- Il seed iniziale viene impostato fuori dal loop di replicazione tramite la funzione "PlantSeeds", esso non verrà aggiornato per ogni replica, in questo modo si utilizza lo stato finale di ogni stream rng come stato iniziale per la successiva replica. Questo permette di evitare sovrapposizioni tra gli eventi generati nelle singole repliche.

8.1 Ricerca del valore ottimale di postazioni nel sistema

Gli esperimenti sono stati progettati con la seguente logica:

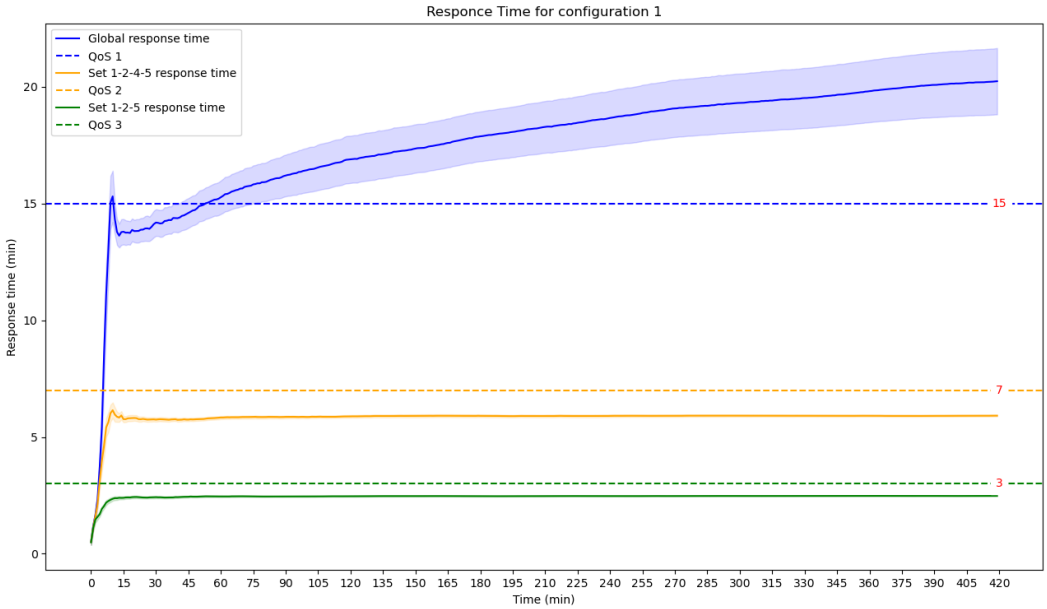
- Si sceglie una configurazione che indica il numero di postazioni per ciascun set.
- Si effettua la simulazione.
- Si verifica che i risultati rispettino i QoS temporali.
- In caso negativo si aumenta il numero di postazioni di uno assegnandolo al set che non permette il raggiungimento dei QoS.

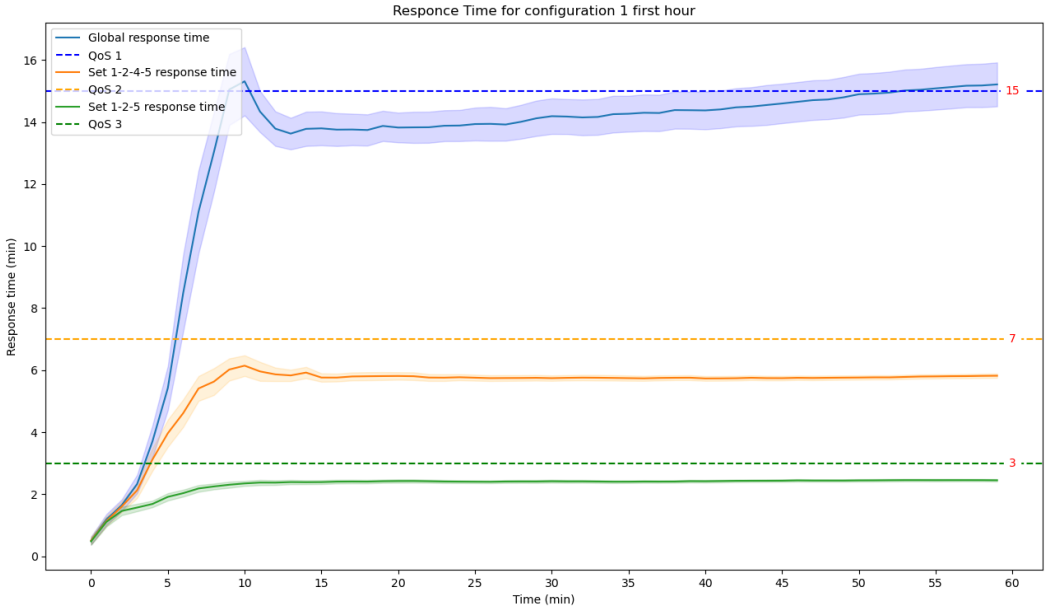
Per maggior chiarezza i QoS temporali possono essere tradotti nelle seguenti considerazioni:

1. QoS 1: il tempo di risposta del sistema deve essere minore di $15\ minuti$.
2. QoS 2: Il tempo di risposta tra i set 1-2-4-5 deve essere minore di $7\ minuti$
3. QoS 3: il tempo di risposta tra i set 1-2-5 deve essere minore di $3\ minuti$

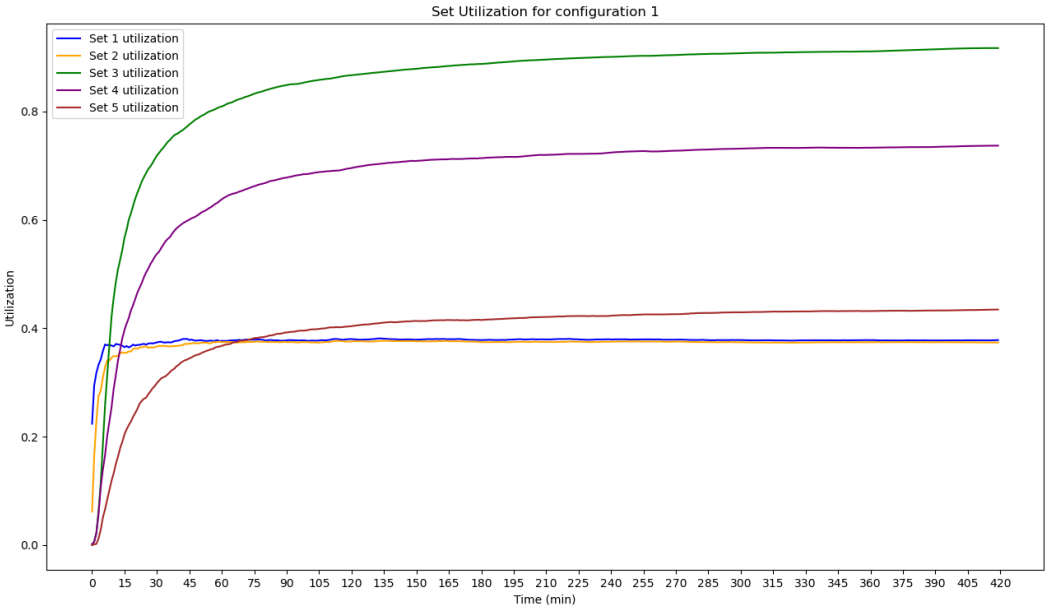
Configurazione 1:

Numero server (M)	Set 1	Set 2	Set 3	Set 4	Set 5
Configurazione 1 (14)	2	2	4	4	2





- Osservando il primo grafico si può facilmente notare come il tempo di risposta globale non rispetti il requisito di qualità, mentre i rimanenti requisiti di qualità vengono rispettati.
- Dal primo grafico si è osservato che il tempo di risposta tende a crescere notevolmente in un arco temporale ridotto, pertanto per fornire una visione più dettagliata, è stato realizzato un secondo grafico che mostra l'andamento dei tempi di risposta nella prima ora di attività.

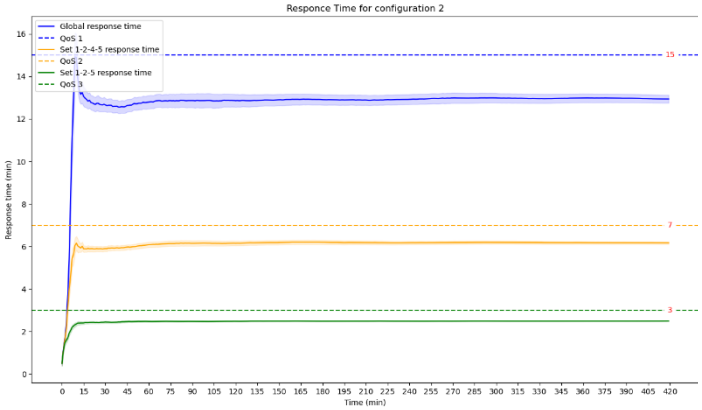


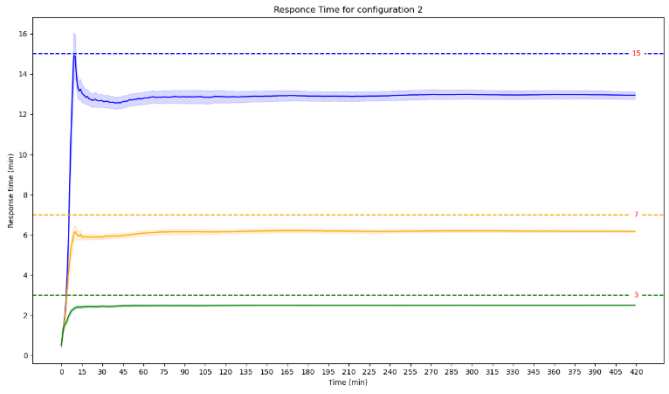
Il grafico dell'utilizzazione ci mostra come ci aspettavamo dai precedenti che il set con utilizzazione elevata è il numero 3. Inoltre possiamo osservare come l'utilizzazione cresca notevolmente nella prima ora e mezza per poi raggiungere un valore stazionario.

Pertanto da questo studio si è deciso di andare a modificare la configurazione dei server aggiungendone uno nel set 3.

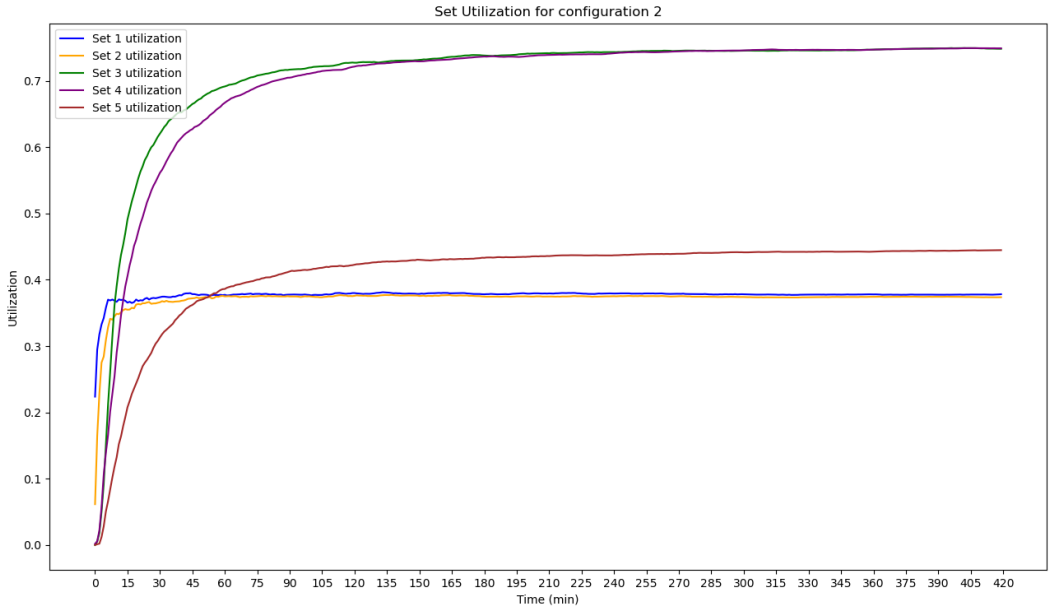
Configurazione 2:

Numero server (M)	Set 1	Set 2	Set 3	Set 4	Set 5
Configurazione 1 (15)	2	2	5	4	2



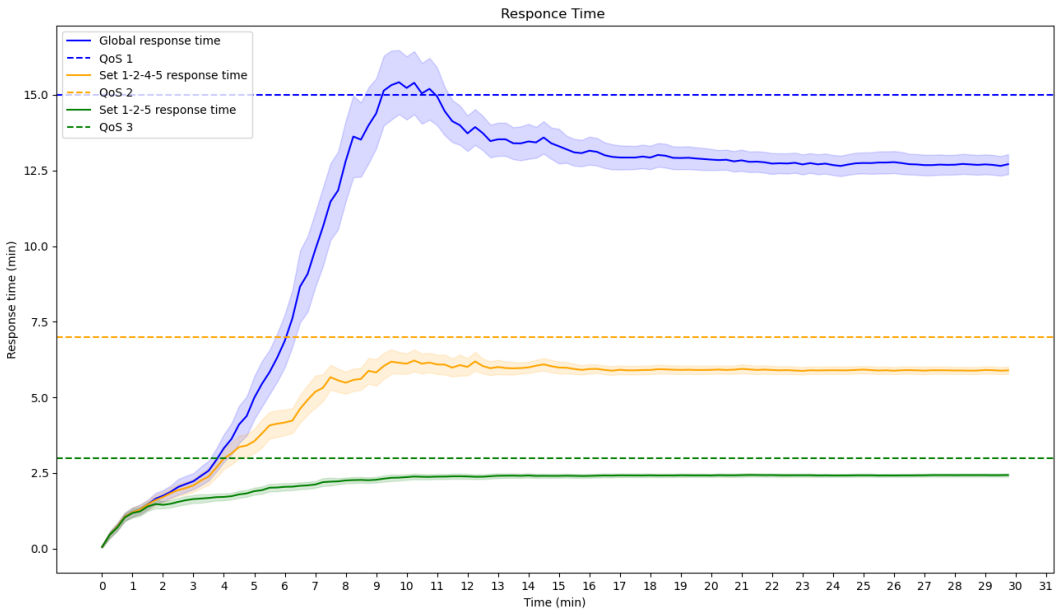


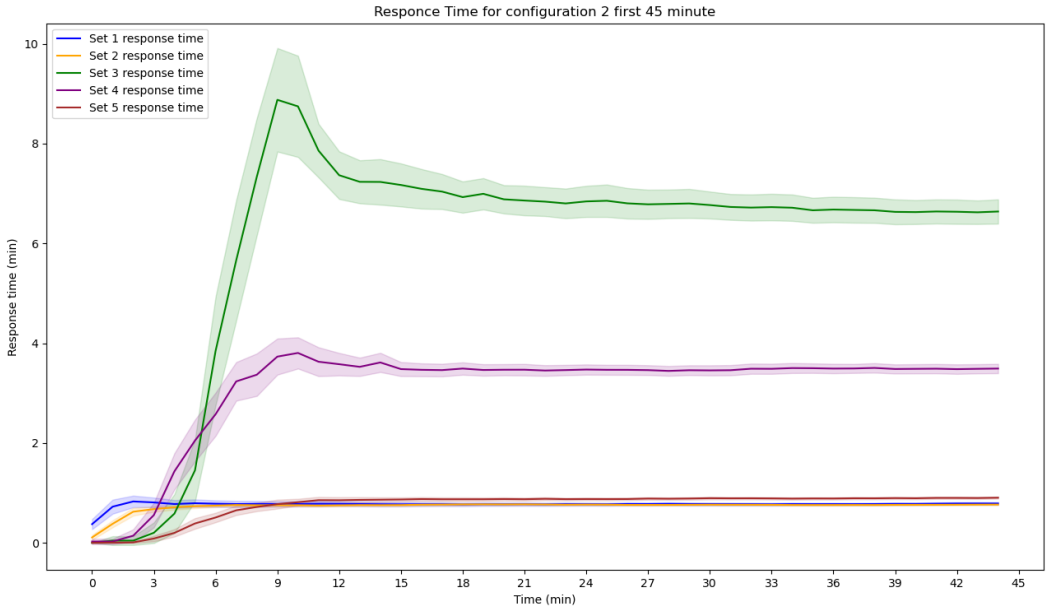
L'aggiunta di un server nel set 3 ha permesso di raggiungere i tre requisiti di qualità nel lungo periodo della giornata come messo in evidenza dal grafico sopra riportato.

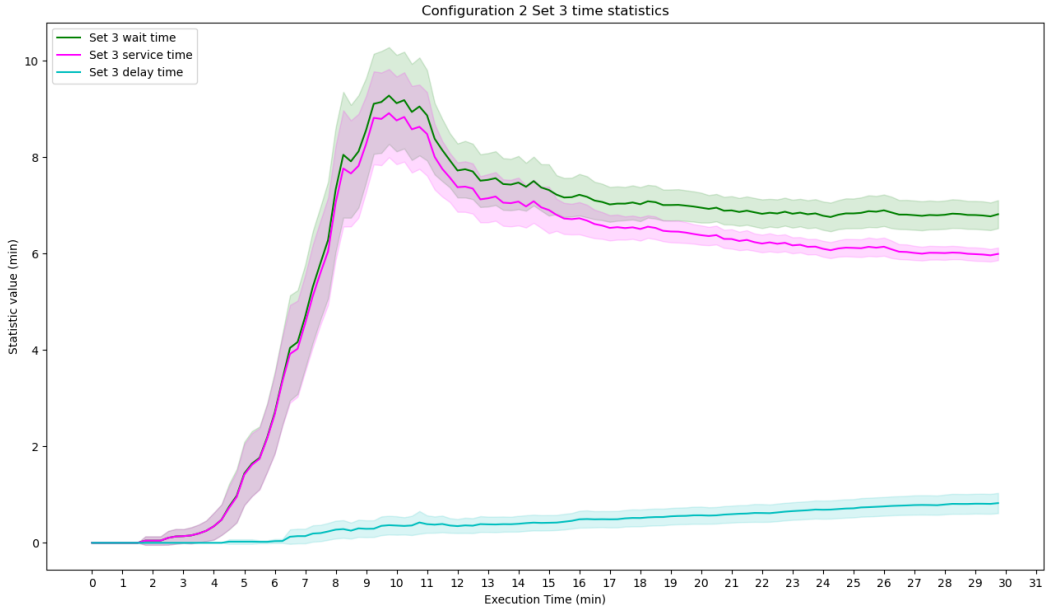


Intorno ai 10 minuti di attività è stato notato un picco nel quale il tempo di risposta globale potrebbe superare il requisito di qualità numero 1, ma l'attuale intervallo di campionamento non lo mette in risalto, pertanto si è deciso di ripetere la simulazione con un intervallo di campionamento inferiore, nello specifico pari a 25 secondi. Di seguito sono riportate le immagini rilevanti a tale studio, in particolare:

1. Tempi di risposta secondo i QoS nei primi 30 minuti.
2. Tempi di risposta dei singoli set nei primi 45 minuti.
3. Statistiche temporali per il set 3 nei primi 45 minuti.







Dalle immagini sopra riportate possiamo evidenziare che il periodo in cui il tempo di risposta globale, in media, non rispetta il requisito di qualità ha un arco di durata breve (circa 2 minuti), buona parte dell'intervallo di confidenza si trova però al di sotto della "linea di QoS". Pertanto in vista di uno studio legato anche ai costi si è deciso di non aggiungere un server ma di tollerare il picco.

La seconda e terza immagine mettono in risalto la causa del picco che dipende dal comportamento del set 3, in particolare dal suo tempo di servizio.

Computo dei costi

Il costo dei server impiegati per questa configurazione è:

- Set 1: $2\ server\ *\ (4 * 60€)\ =\ 480\ €$
- Set 2: $2\ server\ *\ (200€)\ =\ 400\ €$
- Set 3-4-5 : $11\ server\ *\ (30 * 40€)\ =\ 13200\ €$

Spesa totale pari a $14080\ €$ che rispetta il requisito di qualità sul budget mensile

Osservazioni su configurazione 2

Dai grafici dell'utilizzazione e dei tempi di risposta dei singoli set si è portati a pensare che si potrebbe diminuire il numero di server nei set non considerati, cosi da ridurre i costi. Questo porterebbe però a far esplodere i tempi di risposta di tali set e quindi al non raggiungimento dei requisiti di qualità. Nel paragrafo successivo saranno riportate due configurazioni (4 e 5), che dimostrano questo aspetto.

Pertanto la configurazione 2 può essere considerata minimale, cioè quella che impiega il minor numero di server e dal costo più basso per il raggiungimento degli obiettivi.

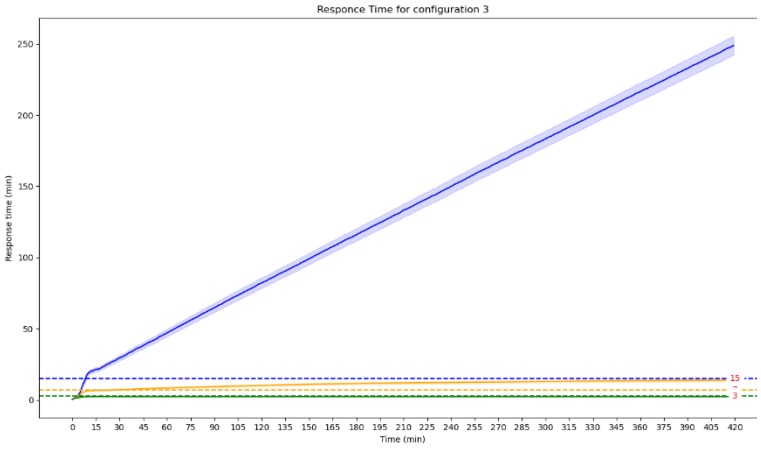
8.3 Alcune configurazioni non ottimali

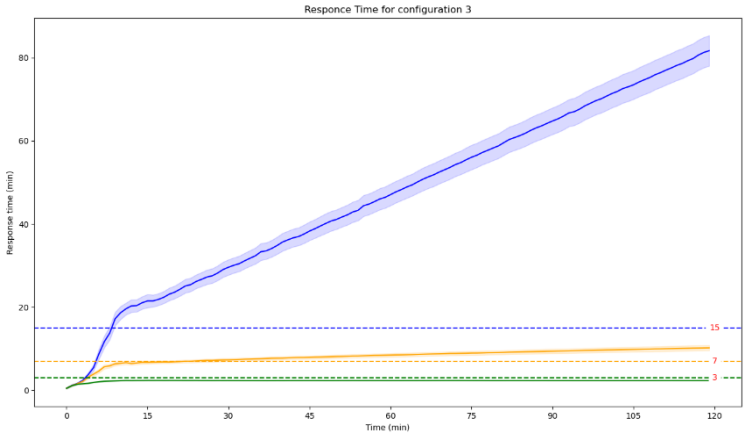
In questo breve paragrafo sono state riportate alcune configurazioni non ottimali studiate con lo scopo di osservare il comportamento del simulatore in tali configurazioni.

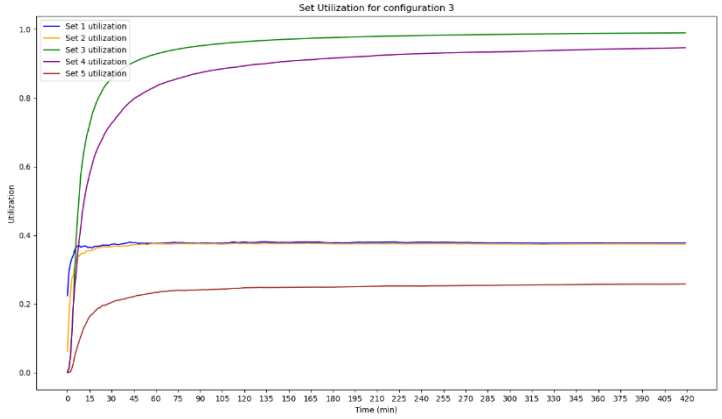
Configurazione 3:

Numero server (M)	Set 1	Set 2	Set 3	Set 4	Set 5
Configurazione 3 (10)	2	2	2	2	2

Questa configurazione mostra come i set che hanno un maggior impatto sui tempi di risposta sono i set 3 e 4, inoltre si può facilmente notare che il sistema in tale configurazione è altamente instabile.

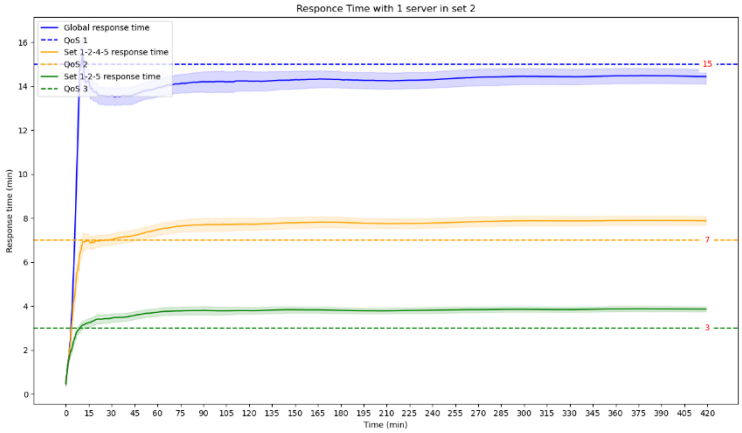


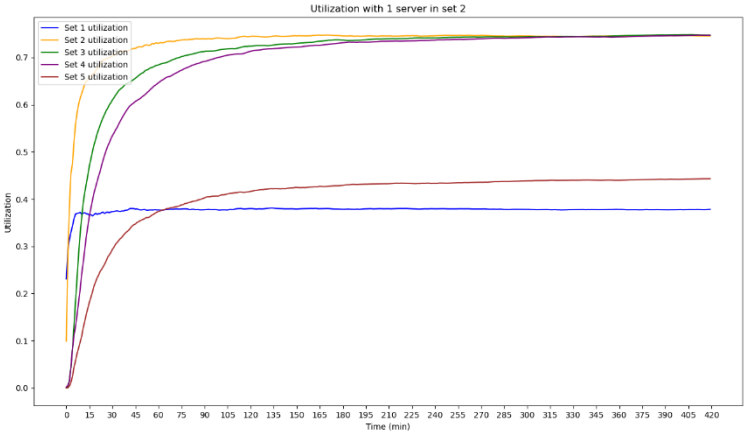




Configurazione 4:

Numero server (M)	Set 1	Set 2	Set 3	Set 4	Set 5
Configurazione 4 (14)	2	1	5	4	2





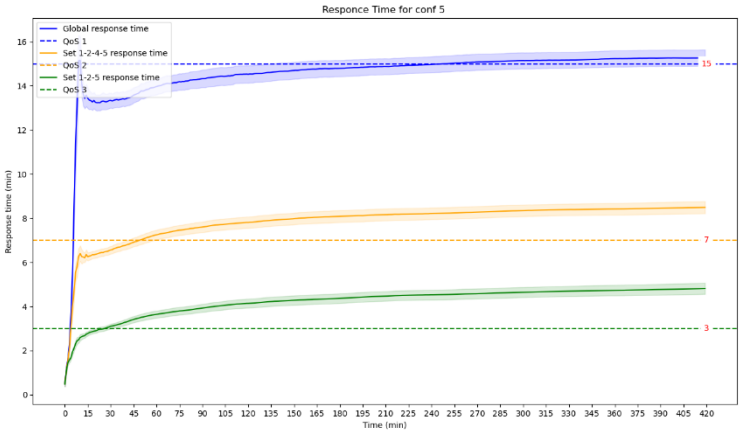
Questa configurazione studia l'idea di ridurre il numero di server in set con utilizzazione bassa rispetto la configurazione 2. In particolare è stata scelto di ridurre il numero di server nel set 2 poiché era quello con tempi di risposta e utilizzazione inferiore.

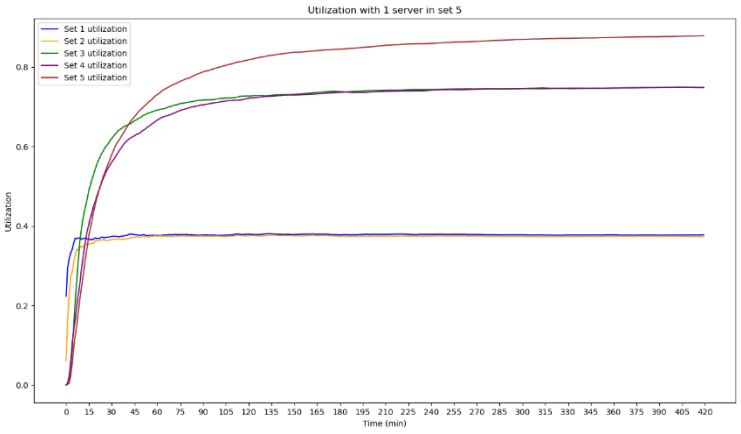
Si può facilmente notare che questa modifica fa alzare i tempi di risposta e non permette di rispettare le QoS 2 e 3.

Configurazione 5:

Numero server (M)	Set 1	Set 2	Set 3	Set 4	Set 5
Configurazione 5 (14)	2	2	5	4	1

La configurazione 5 ha lo scopo di verificare un riduzione dei server rispetto alla configurazione 2, come per la precedente configurazione. Ma in questo caso la scelta del set è stata ponderata in base alla riduzione dei costi. In particolare si è quindi scelto il set 5.





Anche in questo caso i tempi di risposta aumentano, e inoltre non si è in grado di rispettare i requisiti di qualità.

Capitolo 9

9 Conclusioni

In questo capitolo saranno riportate delle brevi osservazioni e possibili migliorie del sistema sviluppato.

9.1 Possibili migliorie

Inizialmente la selezione del prossimo job da schedulare era stata sviluppata tramite un meccanismo size-based, che però non era conforme agli obiettivi e descrizione del sistema. In particolare questo errore è stato messo in evidenza durante con il confronto del modello analitico, in quanto i risultati ottenuti in simulazione si discostavano di molto da quelli teorici.

Tale errore nonostante, sia stato corretto e il modello opportunamente verificato, ha evidenziato che uno scheduling size-based permetterebbe di migliorare i tempi di attesa nei set con tale miglioria, permettendo eventualmente di ridurre il numero di server complessivo.

Per brevità e poiché tali dati raccolti sono fuori dall'obbiettivo di progetto non sono state riportate prove nella relazione in confutazione delle precedenti affermazioni. Nelle cartelle di output sono comunque presenti le statiche computate sia nel caso stazionario che a orizzonte finito.

9.2 Osservazioni

L'affrontare questo progetto mi ha permesso di capire l'importanza di seguire dei passi ben strutturati per poter realizzare un simulatore. Una buona progettazione di un modello concettuale e delle specifiche mi ha permesso di realizzare il codice in maniera chiara e con delle strutture ben prefissate, nonostante ciò nel corso dello sviluppo del codice sono stati commessi degli errori ma, grazie alla fase di verifica e validazione, sono stati portati alla luce, permettendomi di correggere e ottenere un simulatore conforme al modello concettuale e specifiche del caso di studio.

Inoltre la scelta di Python come linguaggio di programmazione, mi ha permesso di velocizzare la raccolta e l'analisi delle statistiche di output.

In particolare è stata utilizzata la libreria "Pandas" per poter gestire la grande quantità di statistiche, effettuare delle organizzazioni logiche di esse per effettuare gli studi e di poterle eventualmente salvare comodamente in file csv. Per la realizzazione dei grafici è stata invece utilizzata la libreria "Matplotlib".