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

1.1 Sintesi del caso di studio 3
1.2 Obiettivi dello studio 5
2 Modello concettuale 6
2.1 Descrizione dettagliata del diagramma 6
2.2 Definizioni variabili di stato 7
2.2 Assunzioni nel modello 7
3 Modello delle specifiche 8
3.1 Variabili matematiche 8
3.2 Distribuzione dei dati 9
3.2.1 Probabilità Clienti 9
3.2.2 Tasso arrivi e Throughput 10
3.2.3 Tempi di servizio 10
3.2.4. Costi legati agli impianti 10
4 Modello computazionale 11
4.1 Stato del sistema 11
4.2 Eventi 12
4.2.1 Lista degli Eventi 13
4.2.2 Generazione degli eventi: 13
4.3 Clock 15
4.4 Scheduler 15
4.5 Algoritmo next-event 15
4.5.1 Inizializzazione 15
4.5.2 Logica del loop 15
4.5.3 Processamento dell'evento 16
4.6 Statistiche 16
4.6.1 Job-averaged statistic 16
4.6.2 Time-averaged 16
4.6.3 Implementazione del calcolo 17
5 Verifica 18
6 Validazione 20
6.1 Validazione set 1 20
6.2 Validazione set 2 21
6.3 Validazione set 3 21

6.4 Validazione set 4 21
6.5 Validazione set 5 22
6.6 Tempi risposta per la tipologia di clienti: 22
6.7 Osservazioni 22
7 Esperimenti di Simulazione 23
7.1 Ricerca valore ottimale di sever nel sistema 23
Configurazione 1: 24
Configurazione 2: 24
Configurazione 3: 24
7.2.1 Analisi configurazioni 24
8 Conclusioni 25
8.1 Possibili migliorie 25
8.2 Osservazioni 25

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. Esso prevede che per accedere alla struttura è obbligatorio avere la tessera di socio, ottenibile tramite registrazione (compilazione del modulo di iscrizione e apposizione firma in sede), e effettuare il versamento della quota giornaliera.

Per far ciò, all'ingresso, vi è una o più postazioni che effettuano:

- Servizi di iscrizione:
    - Servizio controllo iscrizione: avviene tramite riconoscimento di tessera socio o facciale (apposito sistema digitale di riconoscimento), in grado di capire se un cliente è già iscritto all'associazione (l'iscrizione dura un anno, potrebbe essere necessario effettuare nuovamente l'iscrizione al termine del controllo).
    - Servizio iscrizione all'associazione: permette ai nuovi clienti o a chi deve rinnovare l'iscrizione di ottenere la tessera di socio.
- Servizio di pagamento (Biglietteria).

Poiché per le caratteristiche dell'associazione è obbligatorio essere soci, il meccanismo di accesso prevede che venga prima effettuata una fase per controllo/ compilazione dell'iscrizione e una seconda fase in cui viene effettuato il pagamento.

Il Servizio iscrizione all'associazione è composto da due fasi:

1. Compilazione moduli:
    a. Online: il cliente ha già compilato il modulo online e può passare alla seconda fase.
    b. In sede: il cliente necessita di compilare il modulo in sede.
2. Completamento registrazione: fase nella quale sono controllati i dati inseriti nel modulo di iscrizione e validati, con successiva apposizione della firma e rilascio tessera.

La fase di compilazione moduli in sede deve garantire la consegna dell'apposito modulo che si distingue in:

- Per famiglie.
- Modulo standard per Maggiorenni.

Sono state individuate delle fasce orarie con maggiore affluenza di clienti, nelle quali è necessario prevedere almeno una postazione per il completamento della registrazione e una separata per la compilazione dei moduli.

Al di fuori di queste fasce è possibile ridurre le postazioni necessarie accorpando quelle che si occupano della compilazione del modulo con quelle di completamento della registrazione.

Orario di servizio (10:00 / 20:00)

Fasce orarie di maggiore affluenza:

- 12:00 – 14:00
- 16:00 – 19:00

Per ogni fase sono state individuate diverse categorie di clienti in base alle operazioni che devono effettuare.

Nella fase di iscrizione i clienti possono essere distinti nelle seguenti categorie:

- *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 non ha compilato il modulo, deve effettuare quindi la procedura di registrazione completa.
  Bisogna distinguere a sua volta la tipologia di cliente per la consegna del corretto modulo:
  - *Cliente Maggiorenne*.
  - *Famiglia*: insieme che prevede la presenza di minore che devono avere necessariamente un accompagnatore maggiorenne (che può firmare come tutore).

Nella fase di pagamento i clienti vengono distinti nelle seguenti categorie:

- *Cliente singolo*: cliente che deve pagare la sua quota.
- *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.

I dipendenti dell'associazione sono assegnati alle postazioni di compilazione moduli, completamento registrazione e pagamento.

Inoltre a seguito della pandemia per Covid-19, è stato necessario introdurre un meccanismo di controllo temperatura e sanificazione con lo scopo di prevenire la diffusione del virus. La procedura di ingesso deve quindi prevedere almeno una postazione con termometro e igienizzante. Ogni tipologia di cliente deve effettuare tale controllo.

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 mentendo 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 socio:

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

Cliente non socio:

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

Budget mensile sistema di ingresso:

4. Il costo complessivo tra dipendenti e macchinari impiegati nel sistema di ingresso deve essere inferiore a $16000\ €$

Lo studio deve prevedere un eventuale aumento dei clienti dovuto ad una futura espansione dell'associazione, non potendo però impiegare ulteriori fondi.

Capitolo 2

2 Modello concettuale

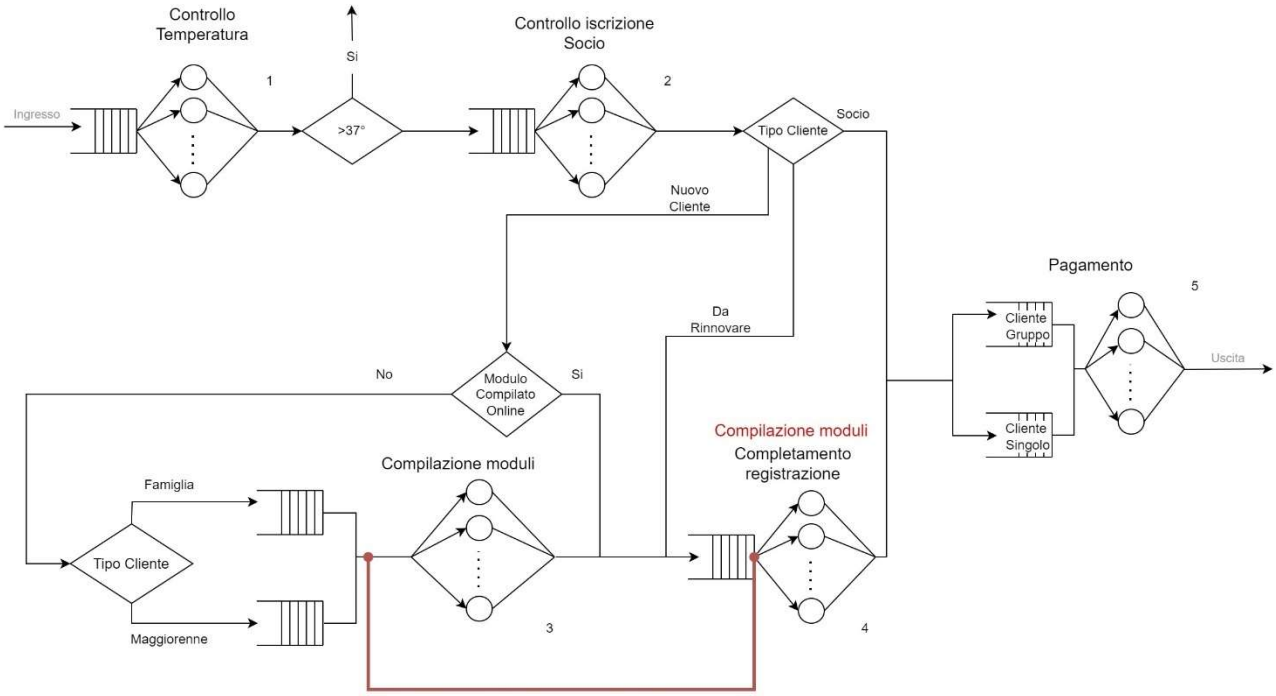


Figura 2.1 Diagramma meccanismo di accesso.

La seguente figura descrive il funzionamento del sistema ad alto livello, non rappresenta quindi lo scheduling effettivo di esso. Tali diagrammi utilizzano notazioni UML 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.

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 moduli*: postazioni che permettono di completare o rinnovare la registrazione alla società, con richiesta firma e rilascio successivo di tessera.
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.

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.

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.

Le parti in rosso descrivono le modifiche al sistema che possono essere apportate al di fuori delle fasce orarie di maggiore affluenza. Tali modifiche sono state rappresentate in questo modello e non apportate nei successivi perché tendono a complicare lo sviluppo del simulatore e i dati raccolti non portano a sostanziale differenziazione dei flussi di arrivo nelle fasce orarie. Pertanto ai fini didattici si è considerato il modello per la sola fascia ad alta intensità per l'intera giornata lavorativa.

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.

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

- Per ciascun insieme deve essere definita una variabile che identifica il numero di Serventi, il valore tra parentesi indica l'insieme:

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

- Ciascun servente può trovarsi in uno dei seguenti stati in base all'insieme di appartenenza:

$$State_k^{(i)} \in \{Occupato, Inattivo\}, i \in \{1,2\} \wedge \forall k = 1, \cdots, N_s^{(i)}$$

$$State_k^{(3)} \in \{Famiglia, Maggiorenne, Inattivo\}, \forall k = 1, \cdots, N_s^{(3)}$$

$$State_k^{(4)} \in \{Completamento\ registrazione, Famiglia, Maggiorenne, Inattivo\}, \forall k = 1, \cdots, N_s^{(4)}$$

$$State_k^{(5)} \in \{Singolo, Gruppo, Inattivo\}, \forall k = 1, \cdots, N_s^{(5)}$$

- Ciascun tipo di cliente è rappresentato da una coppia (t, p) determinata dopo il controllo iscrizione in cui :

$$t \in \{Socio, Rinnovo, NuovoConModulo, NuovoSenzaModuloFamiglia, NuovoSenzaModuloMaggiorenne\}$$

$$p \in \{Gruppo, Singolo\}$$

- Il numero di clienti in base al tipo è modellato dalle variabili:

$$N_c^{(t,p)} \in \mathbb{N} \ con\ t, p\ sopra\ descritti$$

2.2 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.
- Alcuni tempi di servizio variano in base al tipo cliente che la effettua.
- 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 Variabili matematiche

Di seguito saranno presentate le variabili matematiche che definiscono lo stato del sistema ad un certo istante di tempo t.

Per una maggiore chiarezza logica sono stati riportati i valori della tipologia clienti e stato dei server in tabelle presentate all'interno di questo capitolo.

- 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

Tabelle 3.1 Tabelle di conversione per cliente

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

- Per quanto riguarda lo stato dei serventi esso dipende dall'insieme di appartenenza, come descritto nel modello concettuale vi sono 5 insiemi, di seguito sono riportate le tabelle in accordo alla descrizione nel capitolo precedente.

Stato insieme 1
IDLE
BUSY
Famiglia

Stato insieme 2
IDLE
BUSY
Famiglia

Stato insieme 3
IDLE
Maggiorenne
Famiglia

Stato insieme 4
IDLE
Completamento Registrazione
Maggiorenne
Famiglia

Stato insieme 5
IDLE
Singolo
Gruppo

Tabelle 3.2 Tabelle di conversione per server

- Ponendo i l'indice che rappresenta un insieme di server, j il pedice che indica il server considerato, $ServerState$ rappresenta l'insieme di valori in accordo alla tabella i-esima e $N_s^{(i)}$ il numero di server nell'insieme (i), tramite tale presupposti posso descrivere lo stato dei server mediante la seguente rappresentazione all'istante t:

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

- Utilizzando le precedenti notazioni e considerando i vari insiemi di server la variabile di stato che mantiene il numero di clienti viene così rappresentata all'istante di tempo t:

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

- I clienti che si trovano nell'insieme di server i-esimo è dato da:

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

- Il totale dei clienti nel sistema è dato da:

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

- Dalle variabili sopra descritte non è possibile ricavare gli utenti in coda in base al loro tipo per poter implementare questa feature è necessario mantenere una variabile che tiene traccia del numero di clienti in base al tipo che sono in servizio:

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

- Gli utenti in coda in base al tipo possono ora essere calcolati:

$$Queue_c^{(i)}(t) = Cliente_c^{(i)}(t) - ClientiServiti_c^{(i)}(t)$$

3.2 Distribuzione dei dati

Per recuperare i dati necessari a questo modello sono state sfruttate osservazioni e misurazioni del flusso di clienti nel periodo estivo che corrisponde al periodo di maggior attività dell'associazione.

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

La giornata lavorativa considerata va dall'orario di apertura 10:00 alla chiusura 20:00, è stato osservato 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 $p_{>37} = 0.01$

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

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: $p_{pg} = 1\text{-}p_{ps} = 0.65$

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 e Throughput

Per poter ricavare il tasso di arrivi giornaliero tramite le varie osservazioni, si è stimato un throughput del sistema e si è assunta l'ipotesi di job flow balance.

- Giornalmente vengono accolti nella struttura mediamente 450 clienti.
- Periodo di osservazione di 840 minuti

$$X = \frac{\mathbb{E}[\#\{\text{clienti giornalieri}\}]}{\text{tempo\_osservazione}} = \frac{450 \text{ clienti}}{420 \text{ minuti}} \approx 1.071429 \ {}^{\text{clienti}}/_{\text{minuto}}$$

3.2.3 Tempi di servizio

I tempi di servizio dipendono da diversi fattori:

- Insieme di appartenenza del server e suo stato (vedi tabella 3.2).
- Tipo di cliente che richiede tale operazione, la stessa operazione ha tempi diversi in base all'utente, si è usata la notazione descritta nella tabella 3.1 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.
- Operazione richiesta (In base al cliente il server accede uno stato diverso).

La tabella riassume i valori medi dei tempi di servizio

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

Tabella 3.4 Tempi di servizio medi

3.2.4. Costi legati agli impianti

Ciascuna postazione può essere implementata da un macchinario o da un dipendente, di seguito saranno riportati dei valori approssimativi dei costi settimanali o mensili 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 30 €.

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:

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

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:

Gli eventi di arrivo del sistema sono generati in due modi a seconda del set considerato, nello specifico:

1. Il set uno che riceve clienti dall'esterno sfrutta la funzione "GetArrival" per generare il nuovo evento di arrivo dato la tipologia del cliente, l'implementazione è riportata nella figura sottostante.

```python
def GetArrival(clientType : ClientType , setId , currentTime) -> Event:
    event = None
    typ = EventType.ARRIVAL

    mVal = (1 / (clientType.value["prob"] * arrivalRate) ) # compute value based on client type

    if (clientType == ClientType.SS):
        event = __EventCreationExp(typ,__SS_ARR_STREAM,mVal,currentTime,clientType,setId)

    elif (clientType == ClientType.SG):
        event = __EventCreationExp(typ,__SG_ARR_STREAM,mVal,currentTime,clientType,setId)

    elif (clientType == ClientType.RS):
        event = __EventCreationExp(typ,__RS_ARR_STREAM,mVal,currentTime,clientType,setId)

    elif (clientType == ClientType.RG):
        event = __EventCreationExp(typ,__RG_ARR_STREAM,mVal,currentTime,clientType,setId)

    elif (clientType == ClientType.NMOS):
        event = __EventCreationExp(typ,__NMOS_ARR_STREAM,mVal,currentTime,clientType,setId)

    elif (clientType == ClientType.NMOG):
        event = __EventCreationExp(typ,__NMOG_ARR_STREAM,mVal,currentTime,clientType,setId)

    elif (clientType == ClientType.NMAS):
        event = __EventCreationExp(typ,__NMAS_ARR_STREAM,mVal,currentTime,clientType,setId)

    elif (clientType == ClientType.NMAG):
        event = __EventCreationExp(typ,__NMAG_ARR_STREAM,mVal,currentTime,clientType,setId)

    elif (clientType == ClientType.NFS):
        event = __EventCreationExp(typ,__NFS_ARR_STREAM,mVal,currentTime,clientType,setId)

    elif (clientType == ClientType.NFG):
        event = __EventCreationExp(typ,__NFG_ARR_STREAM,mVal,currentTime,clientType,setId)

    return event
```

Screenshot 4.5 Funzione GetArrival

La funzione “GetArrival” è una interfaccia per invocare la funzione “__EventCreationExp” con gli opportuni parametri di stream e valore medio dipendenti dalla tipologia di cliente, di seguito l’immagine dell’implementazione della funzione:

```
def __EventCreationExp(typ:EventType,stream,expM,currentTime,cl,id) ->Event:
    selectStream(stream)
    time = currentTime + Exponential(expM)
    return Event(typ,id,cl,time)
```

Screenshot 4.6 Funzione EventCreationExp

Tale funzione seleziona lo stream e calcola l’istante del successivo arrivo sommando al tempo dell’ultimo arrivo un valore pseudocasuale generato tramite funzione esponenziale con media dipendente dalla tipologia di cliente (calcolato nella funzione “GetArrival”). In fine restituisce l’oggetto evento con gli opportuni parametri

2. Gli arrivi ai restanti set sono invece deterministici e dipendono da completamenti rispetto alla configurazione della rete, pertanto è stata realizzata una funzione chiamata “StaticArrival” che opera nel seguente modo:
    - Genera un nuovo evento ti tipo arrivo con cliente e tempo pari al completamento dell’evento in un server del set precedente.
    - Aggiunge l’evento alla lista del set.

Per quanto riguarda i completamenti si è utilizzata la funzione “GetService”:

```
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 4.7 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 “__EventCreationExp” passando l’opportuno stream.
- 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 (pedice c)

$$\bar{r}_c = \frac{1}{n_c}\sum_{i}^{nc} r_{c,i} = \frac{a_{n_c}}{n_c}\text{, dove } a_{n_c} \text{ è il tempo di arrivo dell'ultimo cliente di tipo c}$$

- Average service time

$$\bar{s} = \frac{1}{n}n$$

- Average delay in queue

$$\bar{d} = \frac{1}{n}\sum_{i}^{n} d_i$$

- Average wait time nel set

$$\bar{w} = \frac{1}{n}\sum_{i}^{n} w_i$$

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

$$\bar{l} = \frac{1}{\tau}\int_0^{\tau} l(t)\,dt$$

- Time-averaged number in queue

$$\bar{q} = \frac{1}{\tau}\int_0^{\tau} q(t)\,dt$$

- Time-averaged number in service

$$\bar{x} = \frac{1}{\tau}\int_0^{\tau} x(t)\,dt$$

4.6.3 Implementazione del calcolo

Utilizzando il teorema di Little (Theorem 1.2.1) possiamo definire i seguenti calcoli integrali con $c_n = t$:

$$\int_0^{c_n} l(t)\,dt = \sum_i^n w_i \,,\ \int_0^{c_n} q(t)\,dt = \sum_i^n w_i \,,\ \int_0^{c_n} x(t)\,dt = \sum_i^n w_i$$

Inoltre per la tipologia di algoritmo stepwise è possibile calcolare tali integrali come riportato nella figura sottostante, la funzione "UpdateArea" computa i valori ad ogni ciclo dell'algoritmo.

```
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

Tramite queste definizioni e l'orologio è possibile computare le statistiche per ciascun set come riportato nella figura seguente.

```
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 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

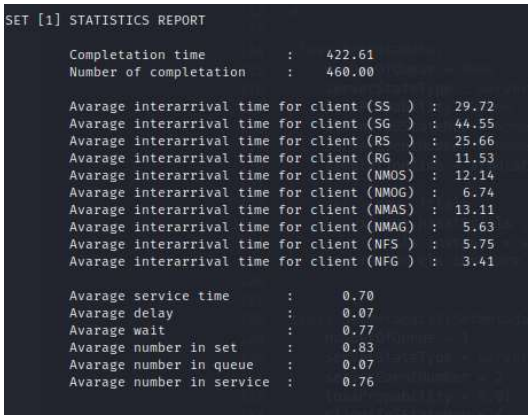
Si è verificato che in ogni set:

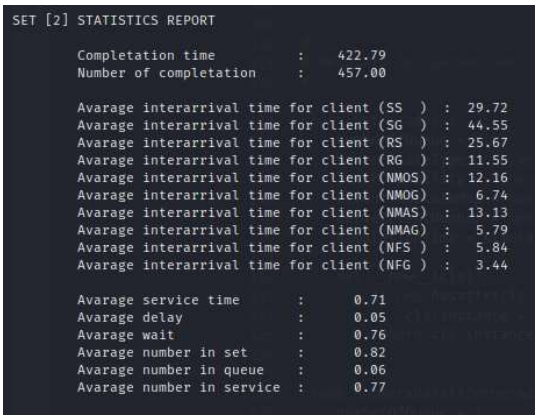
$$\bar{w} = \bar{d} + \bar{s}$$

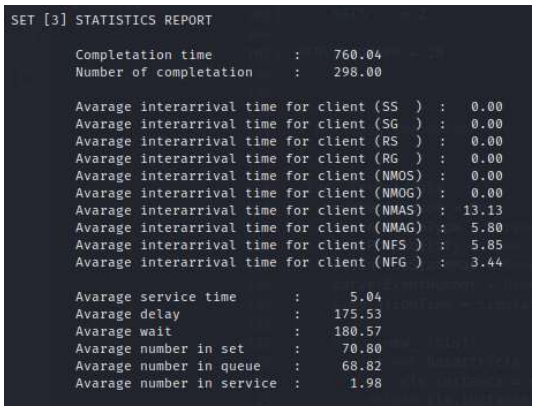
$$\bar{l} = \bar{q} + \bar{x}$$

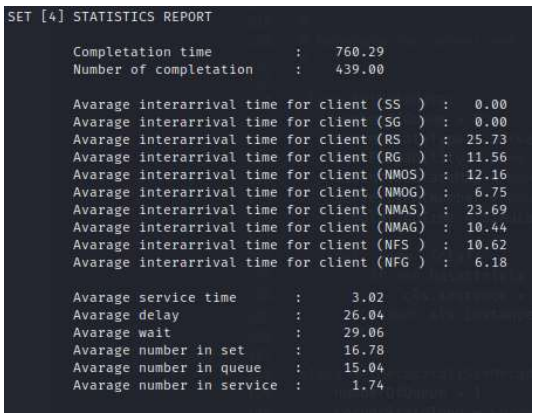
Si è verificato che il numero di completamenti sia conforme alla progettazione del sistema.

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









Figur3 5.3 Statistiche finali di un run di simulazione

Capitolo 6

6 Validazione

Nel seguente capitolo sarà validato il modello andando a confrontare le statistiche ottenute nei vari set con un opportuno modello teorico.

Si assumono delle semplificazioni per poter calcolare i vari modelli teorici, quali i tassi di servizio non tengono conto delle probabilità che i clienti vengano serviti con tempi diversi ma si utilizza un valore medio. Si assume quindi un servizio esponenziale.

È stata effettuata sia una analisi a orizzonte finito, effettuando 256 ripetizioni del test con diversi seed, sia una analisi a orizzonte infinito con il metodo delle batch mean impostando $(b, k) = (256,64)$.

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

Si è impostata una configurazione del numero di server 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 analitico per ciascun set:

6.1 Validazione set 1

- Tasso di arrivo $\lambda_1 = \frac{450 \text{ clienti}}{420 \text{ minuti}} \approx 1.071429 \text{ clienti/minuto}$
- Tasso di servizio $\mu_1 = \frac{1}{E[S_i]} = \frac{1}{\frac{1+0.5}{2}} = 1.\bar{3} \text{ clienti/minuto}$
- 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 analitico (orizzonte finito)	Risultato analitico (orizzonte infinito)
$E[T_{Q,1}]$	0.1443819	0.14 +/- 0.01	0.13 +/- 0.02
$E[S_i]$	0.75	0.75 +/- 0.01	0.75 +/- 0.02
$E[T_{S,1}]$	0.8943819	0.89 +/- 0.01	0.88 +/- 0.03

6.2 Validazione set 2

- Tasso di arrivo $\lambda_2 = \lambda_1 * (1\text{-}p_{>37}) \approx 1.060715 \ {}^{\text{clienti}}/_{\text{minuto}}$
- Tasso di servizio $\mu_2 = \frac{1}{E[S_i]} = \frac{1}{\frac{1+0.5}{2}} = 1.\overline{3} \ {}^{\text{clienti}}/_{\text{minuto}}$
- Utilizzazione globale $\rho_2 = \frac{\lambda_2}{m\mu_2} = \frac{1.06071}{2 * 1.\overline{3}} = 0.397768$

Statistica in analisi	Risultato teorico	Risultato analitico (orizzonte finito)	Risultato analitico (orizzonte infinito)
$E[T_{Q,2}]$	0.140968	0.15 +/- 0.01	0.14 +/- 0.02
$E[S_{i,2}]$	0.75	0.74 +/- 0.01	0.74 +/- 0.02
$E[T_{S,2}]$	0.890968	0.88 +/- 0.01	0.88 +/- 0.03

6.3 Validazione set 3

- Probabilità clienti verso questo centro:

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

- Tasso di arrivo $\lambda_3 = \lambda_2 * p_{c3} \approx 0.68946475 \ {}^{\text{clienti}}/_{\text{minuto}}$
- Tasso di servizio $\mu_3 = \frac{1}{E[S_i]} = \frac{1}{3.4} = 0.2941176 \ {}^{\text{clienti}}/_{\text{minuto}}$
- Utilizzazione globale $\rho_3 = \frac{\lambda_3}{m\mu_3} = \frac{0.68946475}{4 * 0.2941176} = 0.81113487$

Statistica in analisi	Risultato teorico	Risultato analitico (orizzonte finito)	Risultato analitico (orizzonte infinito)
$E[T_{Q,3}]$	9.794606	5.14 +/- 0.54	11.01 +/- 2.89
$E[S_{i,3}]$	3.4	3.42 +/- 0.03	3.65 +/- 0.09
$E[T_{S,3}]$	13.194606	8.56 +/- 0.56	14.66 +/- 2.94

6.4 Validazione set 4

- Probabilità clienti verso questo centro:

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

- Tasso di arrivo $\lambda_4 = \lambda_2 * p_{c4} + \lambda_3 \approx 1.00767925 \ {}^{\text{clienti}}/_{\text{minuto}}$
- Tasso di servizio $\mu_4 = \frac{1}{E[S_i]} = \frac{1}{3} = 0.\overline{3} \ {}^{\text{clienti}}/_{\text{minuto}}$
- 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 analitico (orizzonte finito)	Risultato analitico (orizzonte infinito)
$E[T_{Q,4}]$	6.376636797	1.33 +/- 0.10	1.72 +/- 0.27
$E[S_{i,4}]$	3	2.84 +/- 0.02	3.02 +/- 0.06
$E[T_{S,4}]$	10.376636797	4.17 +/- 0.12	4.74 +/- 0.31

6.5 Validazione set 5

- Tasso di arrivo $\lambda_5 = \lambda_2 \approx 1.060715 \ {}^{\text{clienti}}/_{\text{minuto}}$
- Tasso di servizio $\mu_5 = \frac{1}{E[S_i]} = \frac{1}{1} = 1 \ {}^{\text{clienti}}/_{\text{minuto}}$
- Utilizzazione globale $\rho_5 = \frac{\lambda_5}{m\mu_5} = \frac{1.06071}{2 * 1} = 0.5303575$
- Utilizzazione classe 1 $\rho_{5,1} = \rho_5 * p_{pg} = 0.5303575 * 0.65 = 0.344732375$

Statistica in analisi	Risultato teorico	Risultato analitico (orizzonte finito)	Risultato analitico (orizzonte infinito)
$E[T_{Q,5}]$	0.593406	0.24 +/- 0.01	0.31 +/- 0.05
$E[S_{i,5}]$	1	0.75 +/- 0.01	0.91 +/- 0.02
$E[T_{S,5}]$	1.593406	0.85 +/- 0.01	1.22 +/- 0.07

6.6 Tempi risposta per la tipologia di clienti:

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 analitico (orizzonte finito)	Risultato analitico (orizzonte infinito)
$E[T_S]$	21.6038733	15.35 +/ 0.71	22.38 +/- 3.38

Il tempo di risposta per ciascuna tipologia di clienti dipende dai set di server attraversati.

6.7 Osservazioni

Dalle seguenti misurazioni è possibile osservare come una simulazione ad orizzonte finito si avvicina maggiormente ai risultati teorici, inoltre possiamo affermare che:

- Il comportamento per il set 1 e il set 2 rispecchia i risultati teorici.
- Il set 3 a nel complesso una media maggiore per il tempo di attesa, anche se il risultato teorico è nell'intervallo di confidenza, questo comporta un allontanamento delle statistiche per i set 4 e 5 dal loro valore teorico.
- Complessivamente i tempi di servizio rispettano il valore teorico.
- I set 4 e 5 sperimentano mediamente una attesa minore rispetto al valore teorico.

Nel complesso il simulatore ha, ad orizzonte infinito, il comportamento aspettato.

Capitolo 7

7 Esperimenti di Simulazione

Seguendo gli obiettivi dello studio, sono stati realizzati alcuni esperimenti di simulazione che saranno trattati nel seguente capitolo.

Si è scelto di adottare uno studio a orizzonte finito, poiché rispecchia maggiormente le caratteristiche del sistema, in particolare si è fissato lo STOP time a 420 corrispondente alle due fasce con maggiore affluenza di clienti. Come osservato nella fase di validazione questo vincolo temporale non è in grado di portare il sistema ad uno stato stazionario.

Per avere una maggiore quantità di dati sono state effettuate, per ogni esperimento, 500 run ponendo però un unico seed iniziale per il generatore pseudocasuale con lo scopo di rimuovere la dipendenza da esso.
Si è poi utilizzato il programma "reader.py" per calcolare le medie e i relativi intervalli di confidenza al 95% per le statistiche di interesse.

7.1 Ricerca valore ottimale di sever nel sistema

Osservando i risultati ottenuti durante la fase di validazione si può facilmente affermare che i tempi nel sistema in base alla tipologia di cliente non rispettano tutti i QoS, nello specifico non sono rispettati Qos 1 e QoS 2.

La configurazione utilizzata per validare il sistema ha comunque permesso di inferire alcune assunzioni per poter scegliere una configurazione ottimale dei set, nello specifico:

- La configurazione con 14 server può essere considerata una base di partenza vicino all'ottimo, poiché i QoS non sono rispettati, ma di poco.
- La QoS 1 dipende dai tempi di risposta dei set 1,2 e 3, tale vincolo viene rispettato con un margine minimo, pertanto non è possibile diminuire il numero di server in tali set.
- Per ottenere una utilizzazione inferiore ad 1 il numero di server nei set 3 e 4 non può essere inferiore a 4.
- Le tipologie di clienti che non rispettano i vincoli QoS 2 e QoS 3 sono influenzate in particolare dai tempi di risposta dei server nei set 3 e 4, pertanto incrementare il numero di server in questi set potrebbe migliorare i tempi di risposta globali.
- Bisogna migliorare il tempo nel sistema per l'attraversamento di:
    - Tutti i set per i clienti nuovi sprovvisti di modulo.
    - Attraversamento sei set (1-2-4-5) per i nuovi clienti con modulo.

Varie configurazioni testate:

Numero server (M)	Set 1	Set 2	Set 3	Set 4	Set 5
Configurazione validazione (14)	2	2	4	4	2
Configurazione 1 (15)	2	2	5	4	2
Configurazione 2 (15)	2	2	4	5	2
Configurazione 3 (16)	2	2	5	5	2

Configurazione 1:

Configurazione 1	Set 1	Set 2	Set 3	Set 4	Set 5
Tempi nel set	0.90 +/- 0.01	0.89 +/- 0.01	4.66 +/- 0.10	4.38 +/- 0.10	1.10 +/- 0.01

- Tempo medio di risposta dei clienti per QoS2: $11.93\ minuti$
- Tempo medio di risposta dei clienti per QoS3: $7.27\ minuti$

Questa configurazione non rispetta il QoS3.

Configurazione 2:

Configurazione 1	Set 1	Set 2	Set 3	Set 4	Set 5
Tempi nel set	0.90 +/- 0.01	0.89 +/- 0.01	9.29 +/- 0.44	3.19 +/- 0.03	1.08 +/- 0.01

- Tempo medio di risposta dei clienti per QoS2: $15.35\ minuti$
- Tempo medio di risposta dei clienti per QoS3: $6.06\ minuti$

Questa configurazione non rispetta il QoS2.

Configurazione 3:

Configurazione 1	Set 1	Set 2	Set 3	Set 4	Set 5
Tempi nel set	0.90 +/- 0.1	0.89 +/- 0.01	4.66 +/- 0.01	3.26 +/- 0.04	1.11 +/- 0.01

- Tempo medio di risposta dei clienti per QoS2: $10.82\ minuti$
- Tempo medio di risposta dei clienti per QoS3: $6.16\ minuti$

Questa configurazione rispetta tutte e tre i vincoli di qualità descritti nel capitolo degli obiettivi.

7.2.1 Analisi configurazioni

Utilizzando questo approccio incrementale si è arrivati a poter dire che il numero minimo di Server su tutto il sistema è 16, in particolare la configurazione 3 mostra la suddivisione nei vari set.

Inoltre la spesa mensile complessiva della configurazione è:

- Set 1: 2 server * (4*60€) = 480 €
- Set 2: 2 server * (200€) = 400 €
- Set 3-4-5 : 14 server * (30*35€) = 14700 €

Costo mensile complessivo 15580 €

Viene quindi rispettato il QoS4.

Capitolo 8

8 Conclusioni

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

8.1 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 la fase di Validazione del modello, in quanto i risultati analitici si discostavano da quelli teorici.

Tale errore nonostante, sia stato corretto e il modello opportunamente validato, ha evidenziato che uno scheduling size-based permette di migliorare i tempi di attesa e dai dati ottenuti ne inficiava anche la risposta globale. Per brevità e poiché tali dati raccolti sono fuori dall'obbiettivo 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.

8.2 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, ma nel corso dello sviluppo sono stati commessi anche degli errori. Ma la fase di verifica e validazione mi ha permesso di rilevarli e costruire quindi un simulatore conforme agli obiettivi e al modello del caso in studio.