Progettazione di una Architettura a Microservizi

Progettazione, simulazione e valutazione delle prestazioni

Jacopo Fabi 0293870

Davide Bianchi XXXXXX

AA 2022/2023

Indice

[1 Introduzione 3](#_Toc139275176)

[2 Obiettivo 3](#_Toc139275177)

[3 Modello Concettuale 4](#_Toc139275178)

[4 Modello di Speciﬁche 6](#_Toc139275179)

[4.1 Dati di Input 6](#_Toc139275180)

[4.2 Probabilità di Routing 6](#_Toc139275181)

[5 Modello Computazionale 7](#_Toc139275182)

[5.1 Strutture Dati 7](#_Toc139275183)

[5.2 Gestione degli eventi 8](#_Toc139275184)

[5.2.1 Arrivo 8](#_Toc139275185)

[5.2.2 Completamento 8](#_Toc139275186)

[5.3 Simulazione ad Orizzonte Inﬁnito 9](#_Toc139275187)

[5.4 Simulazione ad Orizzonte Finito 10](#_Toc139275188)

[6 Verifica 11](#_Toc139275189)

[7 Validazione 11](#_Toc139275190)

[8 Analisi dei Risultati 11](#_Toc139275191)

# Introduzione

Il continuo sviluppo in ambito tecnologico, con il passare del tempo, ha generato soluzioni innovative per le aziende, molte delle quali si sono adeguate così da organizzare e gestire al meglio la loro infrastruttura IT.

Il caso di studio in esame prende in considerazione un’architettura a micro-servizi di proprietà di un’azienda che offre al pubblico la possibilità di effettuare prenotazioni per viaggi verso mete turistiche.

L’azienda offre più alternative ai clienti, per questo motivo l’architettura presenta più micro-servizi, ognuno dei quali si occupa di gestire una precisa prenotazione: hotel, volo e automobile.

* Gli utenti che vogliono prenotare un biglietto aereo hanno la possibilità di prenotare un pacchetto completo aggiungendo un soggiorno in hotel e il noleggio di un’automobile per tutto il periodo di residenza.
* Gli utenti che vogliono prenotare un soggiorno in hotel hanno solamente la possibilità di aggiungere il noleggio di un’automobile per tutto il periodo di residenza.

Per completare la prenotazione del pacchetto vacanza è necessario che il pagamento vada a buon fine, per questo motivo ogni utente verrà re-direzionato, come ultimo step, verso un micro-servizio che si occupa della convalida finale: la prenotazione verrà confermata solamente se il pagamento dell’utente non presenta problemi.

# Obiettivo

L’obiettivo dello studio è quello di minimizzare i costi di gestione dell’architettura, in particolare si vuole individuare il numero ottimale di serventi ed operatori per ogni sottosistema dell’infrastruttura.

I due **QoS** che si vogliono rispettare sono i seguenti:

* Mantenere il tempo di risposta complessivo del sistema sotto i 12 secondi, senza considerare i tempi di percorrenza tra i vari micro-servizi della architettura.
* Convalidare il pagamento almeno per il 95% degli utenti che devono portare a termine una prenotazione.

A tale scopo si eﬀettua uno studio sia dello stato stazionario che del transiente, variando le configurazioni dei serventi per cercare di individuare quale di queste rispetta i due vincoli fissati, mantenendo il minor costo totale.

# Modello Concettuale

L’architettura esistente dell’azienda è stata modellata secondo la rete in figura, ogni micro-servizio è in esecuzione su un server dedicato che presenta molteplici repliche, ciascuna eseguita su un container.

Dalla rete si possono evidenziare quattro *nodi* (o sottosistemi) differenti:

* Nodo 1: Prenotazione Volo
* Nodo 2: Prenotazione Hotel
* Nodo 3: Prenotazione Noleggio
* Nodo 4: Convalida Pagamento

Un cliente ha la possibilità di effettuare prenotazioni per pacchetti differenti in base alle sue necessità: la prenotazione del volo permette di includere nel pacchetto un soggiorno in hotel e il noleggio di un’automobile, la prenotazione dell’hotel è invece dedicata a quei clienti che non hanno la necessità di acquistare biglietti aerei ma che in caso di necessità possono noleggiare un’automobile.

I primi tre nodi sono modellati come una **M/M/k**, le code sono inﬁnite proprio perché non si vuole limitare il numero di clienti che possono usufruire del servizio di prenotazione: all’arrivo di una richiesta su un server, si veriﬁca se c’è una replica libera che può gestirla (in tal caso si sceglie il primo servente libero), altrimenti la richiesta va in coda.

Il nodo per la convalida del pagamento è modellato come un **M/M/k/k**, quindi se tutti i serventi sono occupati la richiesta di prenotazione verrà semplicemente scartata senza neanche provare a convalidare il pagamento: proprio per questo, l’obiettivo dello studio è quello di dimensionare la rete in modo tale che il numero di clienti che non riescono a confermare la prenotazione sia al più del 5%.

Per lo studio in esame non interessa il numero di clienti per cui il pagamento è accettato o viceversa, l’analisi si focalizza sulla riduzione al minimo degli utenti che non riescono ad effettuare l’operazione di convalida, a prescindere dall’esito.

Un’architettura a micro-servizi offre al pubblico un servizio web che è disponibile 24H al giorno, con un numero di clienti che può variare nell’arco della giornata:

* Un’infrastruttura di questo tipo non viene dimensionata su diverse fasce orarie ma si utilizzano tecniche come auto-scaling e load-balancing che permettono al sistema di reagire a variazioni improvvise del carico adattando a run-time il numero di serventi sui diversi nodi.
* Per questo motivo, lo studio si basa su un dimensionamento nelle 24 ore, considerando tassi di arrivo simili a quelli che si avrebbero in presenza di picchi del traffico (caso peggiore).

Le **variabili di stato** considerate sono:

* Numero di serventi disponibili nei diversi nodi
* Stato di un servente (libero/busy)

Gli **eventi** che possono verificarsi sono:

* Arrivo di una richiesta
* Servizio di una richiesta presso il servente assegnato

Gli **utenti** in ingresso al sistema sono di due tipi e questa distinzione viene utilizzata solamente all’interno dell’algoritmo migliorativo:

* Iscritti
* Visitatori

Come già detto, per un’infrastruttura che offre un servizio web sempre disponibile ha poco senso effettuare un dimensionamento per fasce orarie nella singola giornata, d’altro canto risulta molto complesso simulare l’auto-scaling dovendo costantemente analizzare il tasso di arrivo dell’ultimo periodo (es. ultimi 30 secondi) e adattare il sistema, per questo si è scelto di dimensionare l’architettura a livello della singola giornata considerando un tasso di arrivo superiore alla media.

Ogni servente attivo implica avere un maggiore costo di manutenzione/gestione, per cui l’obiettivo è quello di identificare il numero minimo di serventi che sia in grado di soddisfare anche un carico di lavoro molto alto, in modo da minimizzare le spese rispettando comunque i due QoS.

# Modello delle Speciﬁche

## Dati di Input

Per la scelta dei parametri di input del sistema non è stato trovato alcun dato disponibile pubblicamente sul numero di richieste che un generico servizio web riceve in un certo periodo, informazioni che avrebbero permesso di produrre una stima verosimile del tasso di arrivo medio in uno scenario di questo tipo.

I tempi di interarrivo ed i tempi di servizio sono modellati secondo una distribuzione **Esponenziale** che, in generale, rappresenta un’ottima scelta per il sistema in esame:

* Numero di clienti al giorno: 240.000 circa
  + 1,9 arrivi medi al secondo per prenotazioni di voli
  + 0,8 arrivi medi al secondo per prenotazioni di hotel
* Tempo di prenotazione volo: 2 secondi

o 0.5 prenotazioni al secondo

* Tempo di prenotazione hotel: 3,2 secondi
  + 0,3 prenotazioni al secondo
* Tempo di prenotazione noleggio: 2,5 secondi
  + 0,4 prenotazioni al secondo
* Tempo di convalida pagamento: 1,3 secondi
  + 0,77 convalide di prenotazioni al secondo
* Percentuale di utenti che prenotano soltanto il volo: 𝑝1 = 65%
* Percentuale di utenti che prenotano oltre al volo un hotel: 𝑝2 = 20%
* Percentuale di utenti che prenotano oltre all’hotel un noleggio: 𝑝3 = 40%

## Probabilità di Routing

Il ﬂusso totale in ingresso viene distribuito verso i vari blocchi tramite le **probabilità di routing**, definite sfruttando le percentuali descritte in precedenza:

* Probabilità di uscita dalla prenotazione del volo

o Convalida Pagamento: 𝑝1 = 0,65

o Prenotazione Hotel: 𝑝2 = 0.20

o Prenotazione Noleggio: 1-𝑝1-𝑝2 = 0.15

* Probabilità di uscita dalla prenotazione dell’hotel

o Prenotazione Noleggio: 𝑝3 = 0,40

o Convalida Pagamento: 1-𝑝3 = 0.60

* La probabilità di uscita dai nodi 3-4 è pari a 1 perché la destinazione è unica, tenendo conto di una percentuale 𝑝loss scartata dal nodo 4 per la coda finita.

# Modello Computazionale

Per la simulazione della rete è stato utilizzato l’approccio di tipo **Next-Event Simulation**, in cui l’avanzamento del tempo si eﬀettua tramite processamento dell’evento successivo.

Il modello è sviluppato in linguaggio C sfruttando le librerie di Larry Leemis descritte al link [*http://www.math.wm.edu/~leemis/simtext.code.c*](http://www.math.wm.edu/%7Eleemis/simtext.code.c): in particolare si utilizzano i files *rngs.h*, *rvgs.h* e *rvms.h* rispettivamente per la generazione di numeri random e di variabili aleatorie.

Il codice della simulazione è implementato nei files *microservices\_base.c*, *microservices\_resized.c* e *microservices\_improved.c*, il codice è molto simile masono necessari più files per differenziare gli scenari base, ridimensionato e migliorato, che presentano configurazioni differenti.

Nella directory /lib troviamo le librerie descritte in precedenza con l’aggiunta della libreria custom *utils.h* che contiene diverse funzioni user-defined utilizzate all’interno dell’implementazione per avere un codice più pulito.

Infine, nel file *config.h* troviamo la definizione delle strutture utilizzate all’interno della simulazione e i parametri di configurazione globale del sistema.

## Strutture Dati

Descriviamo le principali strutture dati utilizzate all’interno della simulazione:

* Struttura *node\_stats* che mantiene le informazioni sul singolo nodo.
  + Mantiene il riferimento ai server associati e informazioni sulle statistiche individuali (jobs in coda, jobs in servizio, jobs processati, …).
  + La coda del nodo è implementata come una *linked list* di job in cui ogni job punta a quello successivo.
* Struttura *server\_stats* che mantiene le informazioni sul singolo servente.
  + Mantiene il riferimento al job in servizio (se presente) e informazioni sulle statistiche individuali (stato, jobs processati, ultima uscita, …).
* Struttura *event* che rappresenta la event\_list, mantiene il riferimento ai prossimi istanti di occorrenza, uno per ogni tipo di evento.
  + Nel caso specifico abbiamo un tipo di evento per ogni nodo: arrivo nel sistema, prenotazione taxi, controllo pagamento, uscita dal sistema.
* Struttura *analysis* che mantiene le statistiche medie temporanee prodotte da una singola run della simulazione (replica/batch).
* Struttura *statistic\_analysis* che mantiene le statistiche medie finali prodotte dall’esecuzione dell’intera simulazione.
* Per mantenere il tempo di simulazione si usa la variabile globale *current\_time.*

## Gestione degli eventi

Gli eventi di arrivo e di completamento vengono gestiti in maniera diﬀerente, per questo motivo si verifica sempre se il prossimo evento è un arrivo o un completamento.  
Per l’implementazione della event\_list, all’estrazione di un evento questo viene direttamente eliminato dalla lista:

1. Si estrae il primo evento disponibile dalla event\_list tramite ExtractEvent().
2. Si controlla il tipo di evento e si processa un arrivo (*job\_arrival*) o una partenza (*job\_departure*) sul nodo corrispondente.

### Arrivo

La event\_list viene inizializza generando degli eventi iniziali tramite GenerateEvent(), questi sono arrivi dall’esterno con tempo di arrivo ottenuto da GetInterArrival() che utilizza la funzione Exponential() di rvgs.h.

Ogni arrivo dall’esterno viene gestito dal nodo per la prenotazione del volo o per la prenotazione dell’hotel, se il numero di jobs nel nodo (coda + servizio) è minore dei servers totali si assegna l’evento al primo servente libero tramite SelectServer().

Se c’è un servente libero quindi:

* + - 1. Si genera un job per il processamento dell’evento di arrivo tramite GenerateJob() con tempo di servizio esponenziale tramite GetService().
      2. Si registra il job nel nodo assegnandolo a un servente libero tramite SelectServer().
      3. Si setta lo stato del server come BUSY e si assegna il job attualmente in servizio.
      4. Si genera un evento di completamento per il servente in questione tramite GenerateEvent() e si inserisce nella event\_list tramite InsertEvent().

Se non c’è alcun servente libero nel nodo il job viene inserito nella coda del nodo (lo spazio c’è sempre avendo code infinite sui due nodi di ingresso).

### Completamento

Quando viene processato un completamento, sì aggiornano le statistiche del servente in questione (jobs processati, jobs nel nodo, …) e si rimuove il job che era in servizio.

A questo punto, se ci sono jobs in coda, si estrae il più vecchio e si mette subito in servizio, altrimenti si setta lo stato del server come IDLE.

Il completamento di un servente equivale ad un arrivo sul nodo successivo:

* + - 1. Tramite SwitchNode() si seleziona il nodo successivo che deve processare l’evento in questione secondo le probabilità di routing.
         * Si genera un valore casuale tra 0 e 100 tramite Random() e si confronta con le diverse probabilità, individuando il blocco di destinazione.
      2. Si genera un evento di arrivo sul nodo selezionato tramite GenerateEvent() e si inserisce nella event\_list tramite InsertEvent().

## Simulazione ad Orizzonte Inﬁnito

Nella simulazione ad orizzonte inﬁnito il sistema viene simulato per un tempo di simulazione “inﬁnito”, ciò vuol dire per un tempo di gran superiore al tempo del caso di studio reale: in questo modo si producono le cosiddette statistiche a stato stazionario del sistema.

Per ricavare la media campionaria del tempo di risposta si utilizza il metodo delle **Batch Means**, che opera suddividendo l’esecuzione della simulazione in K batches di dimensione B:

* Si ricavano le statistiche tramite extract\_analysis() e si resettano questi valori tramite reset\_stats() per ogni singolo batch.
  + Si resettano soltanto gli integrali così da pulire i valori per l’esecuzione del batch successivo.
* In questo modo si genera un campione di K batches indipendenti, sul quale è possibile valutare la media campionaria.
  + Lo stato del sistema non viene alterato ma si mantengono tutti i job rimanenti dal precedente batch.

Le dimensioni di B e K impattano sulla qualità del campione, un valore di B grande permette di avere un campione con bassa autocorrelazione mentre un numero K più grande di batch fornisce un valore migliore in termini di intervallo di conﬁdenza:

* Per tale motivo sono stati utilizzati K = 200 batch, valore che risulta essere un buon compromesso come riportato in letteratura (a partire da K=128).
* Il numero di jobs scelto è pari a B = 500.000, questo per avere un tempo di simulazione di all’incirca 500 giorni.
* In totale, per una run di simulazione sono processati B\*K=110.000.000 jobs.

Gli obiettivi della simulazione ad orizzonte inﬁnito per il caso di studio in esame sono principalmente tre:

* Analisi dei tempi di risposta a steady state.
* Analisi della probabilità **Ploss**.
  + Valutata come il rapporto tra il numero di clienti che non riescono a procedere con la convalida del pagamento (scartati) ed il numero totale di arrivi nel nodo in questione
* Ricerca della conﬁgurazione ottima che permette di rispettare i due QoS e allo stesso tempo minimizzare i costi.

## Simulazione ad Orizzonte Finito

Nella simulazione ad orizzonte ﬁnito il sistema viene simulato per un tempo pari a quello della singola giornata (24 ore) facendo fede al tempo del caso di studio reale: in questo modo si producono le cosiddette statistiche transienti del sistema.

Per ricavare la media campionaria del tempo di risposta si utilizzano 200 repliche della singola run di simulazione, viene quindi eseguito un ensamble di dimensione pari a 200 repliche di simulazione.:

* Ogni replica viene utilizzata per misurare le stesse statistiche da cui verranno poi ricavati i valori finali come media campionaria.
* Il seed viene impostato tramite PlantSeeds() fuori dal ciclo di replicazione, nelle repliche successive alla prima invece viene usato come stato iniziale di ogni stream rng lo stato finale degli streams della replica precedente.
  + Mettere in bibliografia il riferimento alla linea guida: Hill, R. Discrete-Event Simulation: A First Course. J Simulation 1, 377 (2004).
  + Non mi pare noi lo facciamo così ma dovrebbe comunque rispettare le linee guida avendo gli stream che non si sovrappongono
  + Evitiamo sovrapposizioni degli eventi generati dalle singole repliche.
* Il termine della simulazione è legato alla fine della giornata, ovvero quando il clock di simulazione raggiunge 86400 secondi (24 ore).

Gli obiettivi della simulazione ad orizzonte finito per il caso di studio in esame sono principalmente due:

* Analizzare i costi reali sull’arco dell’intera giornata.
* Testare diverse conﬁgurazioni per valutare più nel dettaglio il comportamento del sistema nel transiente.

# Verifica

Verifichiamo che il modello computazionale è effettivamente conforme al modello delle specifiche, quindi che l’implementazione del modello computazionale è effettivamente corretta.

Per il caso di studio in esame, la verifica si basa su tre diverse condizioni:

1. Non mi ricordo che ci eravamo detti
2. Sicuro una era sui flussi in ingresso e uscita da ogni nodo verificando che matchano

# Per valutare queste condizioni sono state eseguite diverse run ad orizzonte sia finito che infinito, nel caso specifico riportiamo i risultati di una run nell’arco delle 24 ore della singola giornata.

# Direi di far vedere tipo tabellina i valori ottenuti dai diversi blocchi per poi mostrare la verifica facendo i conti opportuni.

# Validazione

# Analisi dei Risultati

# 