Università degli studi di
Milano-Bicocca

Decision Models

Final Project

Optimization in Fog Computing

Authors:
Davide Lagano - 838358 - d.lagano@campus.unimib.it
Anastasia Marzi - 847848 - a.marzi3@campus.unimib.it
Davide Sangalli - 848013 - d.sangalli5@campus.unimib.it

15 luglio 2019

Sommario

Nel mondo moderno dell'*Internet of Things* di fianco al conosciuto paradigma del *Cloud Computing* ne sta nascendo uno nuovo: il *Fog Computing*.
In questo scenario, una serie di nodi sono collegati logicamente uno con l'altro e lavorano assieme per la gestione delle richieste che arrivano loro in ogni momento. Lo scopo di questo progetto è minimizzare lo scambio di richieste che avvengono all'interno del fog, cercando di rigettarne e, quindi perderne, il meno possibile. Per questo, sono state implementate due diverse euristiche che affrontano il problema da due diversi punti di vista. Esse sono poi state applicate anche ad un caso reale, cioè ad un sistema di gestione dei taxi nei 18 distretti di Pechino (Cina).
Infine, è stato analizzato uno scenario leggermente diverso, con un'architettura modificata, in cui è stato inserito un nodo centrale, come se fosse un Cloud con capacità infinita, per gestire le richieste in eccesso, affinchè non venissero tutte rigettate.

1 Introduzione

Negli ultimi anni è sempre più crescente l'attenzione verso il mondo dell'*Internet of Things*: il numero di dispositivi connessi sta crescendo vertiginosamente e, di conseguenza, anche il numero di dati prodotti.
Quando si parla di immagazzinamento di dati e di risorse computazionali, il paradigma del *Cloud Computing* è probabilmente quello più utilizzato e conosciuto all'interno dell'IoT. Tuttavia, nonostante questo modello garantisca scalabilità e risorse potenzialmente illimitate alle applicazioni, esistono alcuni domini applicativi per cui un'architettura di questo tipo non è sufficiente a soddisfarne i requisiti. Proprio per questo motivo, negli ultimi tempi si sta sviluppando un altro paradigma che rappresenta un'estensione del primo: il *Fog Computing*. Questo si propone di sopperire ad alcune lacune del cloud quali basse latenze di comunicazione, la geo-distribuzione delle risorse e la cosidetta 'location awareness' delle applicazioni andando a distribuire risorse e servizi di calcolo, lungo l'infrastruttura che connette i dispositivi al Cloud.
Entrando nel particolare, il Fog Computing è uno scenario comune dell'IoT (Internet of Things) dove un numero variabile di nodi, geograficamente distribuiti, cooperano fra loro e con il network per performare il salvataggio e

il processamento di task senza l'intervento di terze parti.

Nel nostro caso specifico, l'infrastruttura è descritta come un insieme di Host distribuiti H_i, con $i \in I := \{1, 2, ..., I\}$, i quali rappresentano antenne 5G con una certa quantità di potenza computazionale. Tutti quanti sono logicamente collegati attraverso un network privato. Gli utenti, basandosi sulla loro posizione geografica, possono accedere ad una singola antenna in ogni momento. Avremo quindi un $\bar{\delta}_{jk}$ che rappresenta la latenza media. Ogni utente richiede che una classe di servizi F_i sia performata all'interno del fog. Ogni host è caratterizzato da una limitata quantità di risorse computazionali sintetizzate dal parametro di capacità totale c_i e, quando il numero di utenti connessi all'host eccedono la capacità dell'host stesso, una percentuale delle richieste in entrata viene dirottata verso uno o più nodi del network, preferibilmente selezionando i più vicini e i meno occupati.

1.1 Modello Base

In questa sezione, viene formalizzato matematicamente il problema. Per prima cosa, viene presentata la funzione obiettivo che si vuole minimizzare, la quale rappresenta lo scambio di dati/richieste fra i diversi nodi, e i relativi vincoli alla quale è sottoposta.
Il parametro $x_{iik}\ \forall i \in I, \forall k \in K$ identifica il carico di domanda per ogni funzione che l'host gestisce al proprio interno, mentre $x_{ijk} > 0\ \forall j \neq i \in I, \forall k \in K$ corrisponde alla domanda che viene inviata agli host H_j attraverso l'attivazione di uno o più archi di collegamento. La variabile $x_{ik}\ \forall k \in K$ rappresenta la domanda che può essere rifiutata dal sistema nel caso estremo in cui le richieste siano troppe o non sia possibile rifiutare tutti i constraints.
Funzione obiettivo:

$$\min \sum_{i \in I} \sum_{\forall j \neq i \in I} \sum_{k \in K} x_{ijk} \tag{1}$$

$$s.t. \quad x_{iik} + \sum_{\forall j \neq i \in I} x_{ijk} + \sum_{\forall k \in K} x_{ik} = f_{ik} \qquad \forall i \in I, \forall k \in K \tag{2}$$

$$\sum_{k \in K} x_{iik} * d_k + \sum_{\forall j \neq i \in I} \sum_{k \in K} x_{jik} * d_k \leq c_i \qquad \forall i \in I \tag{3}$$

$$\bar{\delta}_k \leq \Delta_k \qquad \forall j \in I, \forall k \in K \tag{4}$$

Nelle equazioni sopra descritte, oltre alle variabili già descritte precedentemente, sono presenti anche:

- $f_{ik} \quad \forall i \in I, \forall k \in K$, che corrisponde alla domanda totale di ogni classe di applicazione per ogni singolo host, la cui sommatoria risulta essere maggiore di 0;
- d_k che rappresenta il consumo per l'elaborazione della funzione;
- Δ_k che identifica il ritardo end-to-end massimo tollerabile tra la funzione e l'utente che la richiede.

In questa modellazione stiamo considerando la coesistenza di richieste eterogenee, infatti le diverse richieste che vengono mandate ad ogni singolo nodo possono appartenere a classi di funzioni differenti (F_k dove $k \in K := \{1, ..., K\}$). Dato che lo scopo è quello di minimizzare gli scambi, alla fine del processo non è necessario che il sistema risulti anche bilanciato, quindi non per forza tutti i nodi del fog devono processare lo stesso numero di richieste.
Il primo vincolo (2) assicura che tutta la domanda richiesta dagli utenti entri nel sistema per essere processata o rigettata.
Il secondo (3) rappresenta un vincolo di capacità che mette in relazione i consumi delle funzioni accettate al proprio interno e delle funzioni provenienti da altri host con la capacità dello stesso.
Infine, l'ultimo (4) setta il ritardo massimo per ogni funzione.

1.2 Variazione dell'architettura del Fog - Presenza di un Cloud centrale con capacità illimitata

Il problema presentato nel caso base non tiene in considerazione la possibile presenza di un Cloud centrale che possieda capacità infinita e che possa servire da supporto ai nodi del Fog in modo tale che le richieste in eccesso rispetto alla capacità totale del Fog non vengano rigettate, ma, piuttosto, vengano direzionate verso questo Cloud.
Al fine dell'implementazione è stato introdotto un sistema di costi, sia di rigetto per ogni singolo nodo che di invio dei dati al Cloud. Si è deciso di mantenere il costo di invio al Cloud costante per ogni singolo nodo. I costi associati ai nodi non vengono considerati all'interno del modello matematico in quanto non ritenuti influenti al fine della minimizzazione degli scambi.

La funzione obiettivo e i relativi vincoli sono stati modificati come segue:

$$\min \sum_{i \in I} \sum_{\forall j \neq i \in I} \sum_{k \in K} x_{ijk} + \rho * \sum_{n \in N} \sum_{k \in K} x_{nk} \qquad \rho \in [0, +\infty) \qquad (5)$$

$$s.t. \quad x_{iik} + \sum_{\forall j \neq i \in I} x_{ijk} + x_{nk} = f_{ik} \qquad \forall i \in I, \forall k \in K \qquad (6)$$

$$\sum_{k \in K} x_{iik} * d_k + \sum_{\forall j \neq i \in I} \sum_{k \in K} x_{jik} * d_k \leq c_i \qquad \forall i \in I \qquad (7)$$

$$\bar{\delta_k} \leq \Delta_k \qquad \forall j \in I, \forall k \in K \qquad (8)$$

In queste equazioni è stata aggiunta la variabile x_{nk} che definisce i dati inviati da ogni singolo Host ai nodi del cloud centrale.

2 Dataset

Per l'applicazione delle euristiche implementate si è deciso di proseguire in due direzioni diverse.
In primis, le euristiche sono state applicate a dati generati randomicamente. Nel sistema che è stato analizzato inizialmente erano presenti solamente 6 nodi, tutti interconnessi fra loro. Dopo aver definito randomicamente una capacità per ogni singolo nodo (definendo in maniera casuale sia CPU che RAM), la quale si è assunto rimanesse costante nelle diverse prove e iterazioni, anche le richieste sono state generate in maniera casuale, prendendone la quantità in modo randomico da una distribuzione poissoniana con media la media delle capacità dei diversi nodi. Questo procedimento è stato effettuato iterativamente per creare una simulazione. Inoltre, per analizzare come le prestazioni dei modelli variassero a seconda del numero dei nodi, si sono svolte altre simulazioni aumentando il numero dei nodi in esame, fino ad un totale di 34 nodi.

In seguito, le euristiche sono state applicate ad una situazione reale. Si è studiato un dataset che contiene le richieste di taxi a Pechino, in particolar modo in 18 distretti diversi della città. I dati riguardano un periodo temporale di 7 giorni, dal 2 all'8 di Febbraio 2008. Oltre ad essere suddivisi per i distretti di Pechino, le richieste sono anche state raggruppate per un intervallo temporale di mezz'ora.

Il dataset utilizzato ha la seguente forma:

	datetime	district	taxi_id	period	time_to_eat	new_date	new_time	On_Off
0	2008-02-02 13:30:00	Changping	338	Afternoon	1	2008-02-02	13:30:00	0
1	2008-02-02 13:30:00	Chaoyang	17537	Afternoon	1	2008-02-02	13:30:00	0
2	2008-02-02 13:30:00	Chongwen	8366	Afternoon	1	2008-02-02	13:30:00	0

Figura 1: Prime righe del dataset sui taxi

Nella prima colonna *datetime* si possono distinguere la data e la fascia di mezz'ora espressa in datetime. In seguito sono presenti le informazioni relative al distretto di Pechino in cui avvengono le richieste (*district*) e al numero di taxi richiesti in quel lasso temporale, identificati nella colonna *taxi_id*. Le colonne successive sono: *period*, la quale indica il periodo della giornata in cui avvengono le richieste, *time_to_eat*, la quale indica se era ora di pranzo/cena (1) oppure no (0), *new_date* e *new_time*, che sono i dati relativi alla data e all'ora e, infine, *On_Off*, la quale indica se in quel momento le luci erano accese(1) oppure spente(0). Queste 5 colonne non sono state considerate in quanto non utili alla nostra analisi.
In totale sono presenti 5346 osservazioni all'interno di questo dataset.

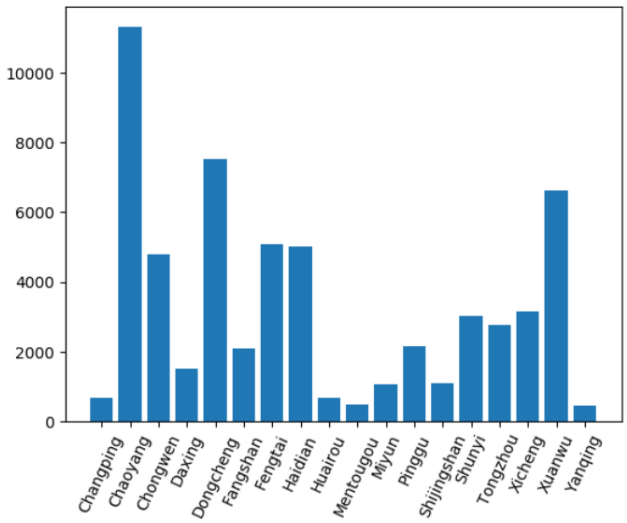

Figura 2: Quantità media di richieste per ogni distretto di Pechino

Nel grafico 2 sono state rappresentate, per mezzo di un barchart, le quantità medie di richieste per ogni distretto di Pechino. La media è stata fatta considerando tutti i dati aggregati a nostra disposizione.

3 Approcci Metodologici

L'obiettivo è quello di minimizzare gli scambi fra i diversi nodi e per fare questo sono state implementate due tecniche euristiche. Queste sono, in particolare, delle euristiche costruttive in quanto seguendo un determinato criterio, costruiscono la soluzione ottimale.
Per la loro implementazione sono state fatte alcune assunzioni:

- La latenza e il tempo d'inter-arrivo fra un richiesta e l'altra non vengono considerati nell'implementazione del modello. Infatti, ci mettiamo nell'ottica in cui al tempo t_0 le richieste vengano eseguite tutte assieme in maniera aggregata e non in sequenza, una dopo l'altra, seguendo una coda.

- Ogni singolo nodo del sistema Fog ha una capacità totale espressa in termini di due componenti: RAM e CPU. Allo stesso modo, anche le richieste che arrivano ai nodi hanno una demand espressa negli stessi termini.

- Per semplificare il problema, è stata inoltre tenuta in considerazione l'esistenza di un'unica classe di funzioni. In questo modo, tutte le richieste entranti nei nodi possedevano la stessa demand di RAM e di CPU.

- Come già detto, i nodi sono tutti logicamente interconnessi fra loro e è stato ipotizzato che questi fossero posizionati tutti alla stessa distanza uno dall'altro. In questo modo, lo scambio di dati non viene influenzato e penalizzato da nessun tipo di variabile 'geografica'.

3.1 Euristiche per il modello Base

Nella prima euristica si è considerata una situazione di tipo centralizzato in cui si è a conoscenza di tutte le informazioni sul sistema. Come punto di partenza, è stato analizzato un sistema formato da 6 nodi, ognuno con la propria capacità. L'idea alla base di questa euristica è la seguente: dopo che

ogni nodo ha processato tutte le richieste che gli erano possibili, i nodi con richieste in eccesso le mandano agli altri nodi in modo tale che possano essere processate. In particolare, il procedimento parte dal nodo con più richieste in eccesso che vengono inviate al nodo che, dopo aver processato le sue richieste in entrata, ha più spazio libero e quindi è il più vuoto. Questo processo va avanti finchè tutte le richieste non saranno state processate. Ricordiamo che lo scopo non è che il sistema sia bilanciato, ma che tutte le richieste vengano eseguite e che sia minimizzato lo scambio fra i nodi e l'attivazione degli archi. In questo caso, è stato necessario inserire un controllo iniziale. Infatti, bisognava verificare che il numero totale di richieste in entrata all'interno del fog non eccedesse la capacità massima totale di tutti i nodi. In caso questa situazione si fosse presentata, si è deciso di effettuare un 'ribilanciamento' del sistema. Le richieste in eccesso rispetto alla capacità totale venivano rigettate dai diversi nodi, togliendone sequenzialmente uno alla volta da ogni nodo, fino al raggiungimento del numero giusto di richieste. Ovviamente, in questi casi, alla fine del processo tutti i nodi del sistema risultavano essere completi.

Per la seconda euristica il problema è stato analizzato sotto una diversa luce. Si è ipotizzato che i nodi siano indipendenti uno dall'altro e che non siano a conoscenza della situazione presente negli altri nodi. In questo modo, i nodi a cui arrivano troppe richieste e che hanno bisogno di trasferirle, devono inizialmente fare una domanda ad un altro nodo per sapere se gli sia rimasto spazio per eseguire altre richieste oppure no. Si è deciso che il primo nodo a cui chiedere informazioni fosse scelto casualmente fra gli altri presenti nel sistema. Se il nodo a cui viene posta la domanda non ha lo spazio necessario a contenere la totalità delle richieste che il nodo di partenza doveva trasferire, allora il primo nodo cambia obiettivo e chiede la stessa cosa ad un secondo nodo finchè non ne incontra uno che può soddisfare la sua richiesta. Se dopo aver interrogato tutti gli altri nodi presenti nel sistema, nessuno di essi può contenere la totalità delle sue richieste in eccesso allora inizia un procedimento di split.
Per prima cosa, il nodo attivo, avendo conoscenza ora di tutte le informazioni del sistema, controlla che quest'ultimo non sia in *overload* e che, quindi, ci sia la necessità di rigettare delle richieste. Nel caso in cui ce ne fosse bisogno, lui rigetta tutte le sue richieste in eccesso e si passa ad analizzare il nodo successivo. Ovviamente questo non sta a significare che il sistema non sia più in overload, perchè potrebbe capitare che le richieste in eccesso del sistema siano di più rispetto a quelle in eccesso nel singolo nodo. Le richieste

mancanti da rigettare verranno individuate da un altro nodo.
Nel caso in cui, invece, il sistema non abbia problemi di un eccessivo numero di richieste, la fase di split consiste nel trovare il numero minimo di nodi necessari a distribuire le richieste in eccesso. Quando il primo nodo preso in esame ha completato questo procedimento, si passa al successivo con richieste in eccesso che ricomincia il processo dall'inizio, senza alcuna conoscenza degli altri nodi.

Per l'implementazione di entrambe le euristiche è stato utilizzato il linguaggio di programmazione Python.
Sono stati effettuati diversi test applicando entrambe le euristiche. Come già detto nella sezione precedente, il modello base è stato applicato per primo a dati generati randomicamente. In questo modo è stato possibile fare diverse simulazioni in cui sono stati modificati i parametri di partenza del sistema e anche il numero dei nodi.
In seguito entrambe le euristiche sono state applicate, e in questo modo confrontate, anche al problema della gestione dei taxi a Pechino. In questo scenario, sono presenti 18 distretti e, di conseguenza, 18 nodi.
Per svolgere le analisi nel caso reale, è stata fatta l'ipotesi che la capacità massima di ogni singolo nodo fosse data dalla media di tutte le richieste di tutti i giorni a cui veniva sommato 850. Questa quantità è stata scelta dopo aver effettuato vari test. Se fosse stata utilizzata solamente la media, il sistema sarebbe stato sempre in overload e, quindi, sarebbe stata sempre presente una percentuale di richieste rifiutate cosa che, in un caso reale, non risulta essere molto plausibile. La cifra sopra indicata è stata quella che meglio stabiliva un equilibrio fra richieste rigettate e non.

3.2 Euristiche per la variazione di scenario - Architettura con Cloud centrale

In questo secondo caso, in cui aggiungiamo un nodo centrale con capacità infinita all'interno del sistema, non è stato possibile utilizzare il dataset a disposizione in quanto non adatto per questo scenario e per questo motivo è stato applicato esclusivamente a dati randomici.
Le euristiche applicate in questo caso sono le stesse implementate precedentemente, ma sono state effettuate delle modifiche al fine di poterle adattare

a questo problema. Come già anticipato, è stato introdotto un sistema di costi che permettesse di prendere delle decisioni al momento del rigetto delle richieste in eccesso. Infatti, si suppone che inviare le richieste al nodo centrale abbia un determinato costo e di conseguenza sia necessario effettuare una valutazione sul fatto che sia meglio rigettare le richieste o inviarle al cloud. Il costo dell'invio è stato settato arbitrariamente a 10 unità ed è stata fatta l'assunzione che i nodi del fog avessero dei costi differenti di rigetto. Questi costi sono stati generati randomicamente da una poissoniana di media pari a 10 (cioè al costo di invio al Cloud).

Queste assunzioni e questo sistema di costi hanno avuto due impatti diversi all'interno delle due euristiche. Per quanto riguarda la prima, poichè si hanno a disposizione tutte le informazioni, si è fatto in modo che, in caso ci fosse bisogno di rigettare alcune richieste, si scegliesse il nodo con il costo di rigetto maggiore e più grande del costo di invio. Da questo, sono state tolte tutte le richieste in eccesso del sistema, sia nel caso in cui questo nodo avesse delle richieste in eccesso che nel caso non le avesse. In questo modo, si è anche cercato di minimizzare la perdita di informazioni e dati causata dal rigetto delle richieste.
D'altra parte, se si considera il sistema di nodi indipendenti, decisioni di questo tipo non possono essere prese. Infatti, in questo caso, nel momento in cui un singolo nodo con richieste in eccesso si rende conto che il sistema è in *overload* dovrà valutare singolarmente, rispetto al suo costo di rigetto, se sia meglio inviare le richieste al cloud oppure rigettarle. Ovviamente, in questo secondo caso, la probabilità che le richieste vengano rifiutate è decisamente più alta rispetto al primo caso, in cui poteva accadere solamente se tutti i nodi avevano dei costi di rigetto inferiori al costo di invio al cloud.

4 Risultati e valutazioni

4.1 Modello Base

Grazie all'utilizzo di dati generati casualmente, è stato possibile valutarne la quantità di scambi nelle due euristiche, all'aumentare del numero di nodi.

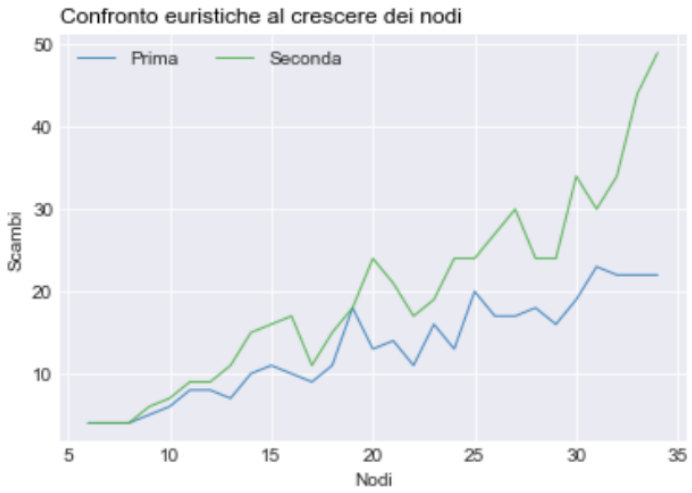

Figura 3: Confronto delle euristiche implementate all'aumentare del numero dei nodi

I risultati presentati nel grafico 3, raffigurano due linechart, il blu è riferito alla prima euristica mentre il verde è associato alla seconda. Si può osservare come, all'aumentare dei nodi, il numero di scambi ottenuti con la seconda euristica sia sempre più elevato rispetto a quello della prima. A 34 nodi, limite a cui si è deciso di fermarsi, il numero di scambi per la seconda euristica risulta essere circa il doppio rispetto alla prima.

Viene poi riportato un grafico riguardante i risultati delle analisi svolte sul dataset dei taxi. Per mostrare l'andamento degli scambi nelle due euristiche, secondo le diverse fasce orarie, è stato deciso di plottare i risultati riguardanti un solo giorno, il secondo (3 Febbraio 2008).
Si può osservare come nelle prime ore della giornata, il numero di scambi calcolato dalle due euristiche risulti essere sempre lo stesso. Invece, nel corso del pomeriggio si nota un picco degli scambi fra nodi, in particolar modo per quanto riguarda la seconda euristica. Si è osservato che in questo stesso periodo si verificano anche delle situazioni in cui è stato necessario rigettare delle richieste.

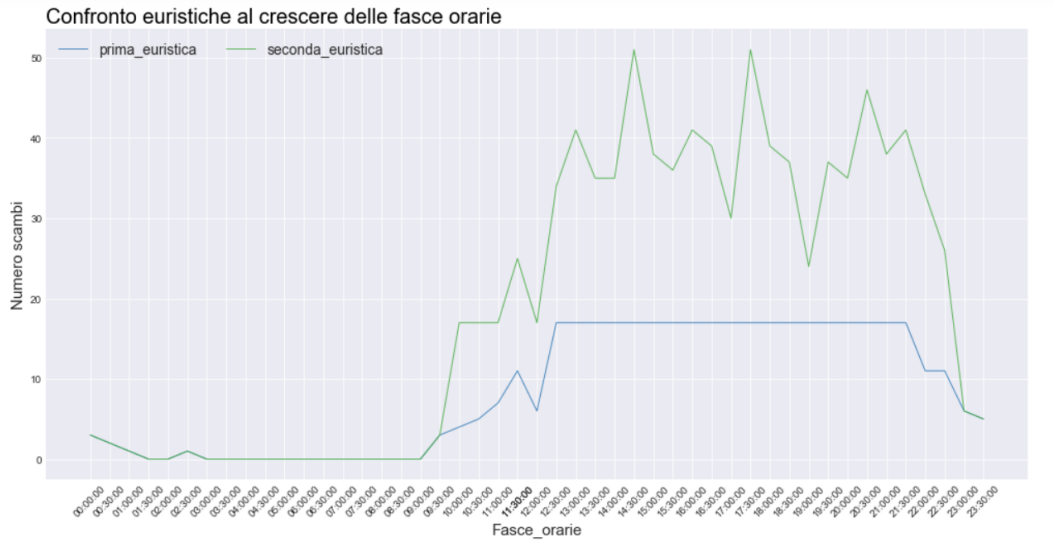

Figura 4: Confronto delle euristiche implementate al variare delle fasce orarie

4.2 Variazione dell'architettura

Per quanto riguarda l'analisi dei risultati nella variazione di scenario, vengono riportati alcuni casi nella tabella 1.

Tabella 1: Variazione dello scenario con aggiunta del nodo centrale

Iterazioni	Prima Euristica			Seconda Euristica		
	n_scambi	req_cloud	req_rig	n_scambi	req_cloud	req_rig
1	6	9	0	6	9	0
2	5	0	10	7	0	10
3	5	20	0	3	0	20
4	7	19	0	5	12	7
5	7	21	0	4	14	7

Si può notare come la prima euristica, per come è stata costruita, gestisce in blocco le richieste, rigettandole o inviandole al cloud. Al contrario, nella seconda euristica possono verificarsi delle situazioni in cui le richieste vengono in parte rifiutate e in parte inviate al cloud. Si dimostra interessante il fatto che nel caso in cui alcune richieste vengano rifiutate, il numero di scambi

tenda ad abbassarsi leggermente.
Sono stati eseguiti svariati test per verificare se fosse presente una soglia sotto la quale, all'interno della prima euristica, si possa verificare il caso di rigetto delle richieste. Per le assunzioni fatte, questa situazione può avvenire solamente se i costi di rigetto di tutti i nodi risultano essere minori rispetto al costo di invio al cloud. Si è verificato che questa situazione si presentava ponendo la media della distribuzione poissoniana dalla quale venivano estratti questi costi uguale a 10 (costo del cloud) - 4.65.

5 Discussione

Osservando i grafici plottati nella sezione precedente si può notare che, nel sistema iniziale di 6 nodi, le due euristiche hanno una prestazione quasi alla pari nella minimizzazione degli scambi. Tuttavia, al crescere dei nodi, è evidente come la prima euristica sia molto più efficace rispetto alla seconda. Questo è dato dal fatto che nella seconda, quando gli altri nodi vengono indagati, non è detto che possano essere individuati immediatamente quelli con più spazio disponibile, ciò pora ad un aumento degli scambi.
Tuttavia, la seconda euristica può rivelarsi efficace nel caso in cui secondo una determinata architettura non ci dovesse essere comunicazione fra nodi, oppure ci dovesse essere un problema di comunicazione all'interno del sistema. Infatti, ogni singolo nodo, avendo la possibilità di agire indipendentemente dagli altri, troverebbe una soluzione e gestirebbe tutte le sue richieste, cosa che sarebbe molto difficile nel caso venisse applicata la prima euristica.

Per quanto riguarda lo scenario in cui è presente un nodo centrale, il numero di scambi segue la stessa logica. Tuttavia, come già spiegato, in questo caso la differenza si presenta nel numero di richieste che vengono rigettate. Per come è stata costruita la prima euristica, il caso in cui le richieste vengano rigettate piuttosto che inviate al nodo centrale è davvero improbabile. Al contrario, nella seconda, se i costi di rigetto risultano in media relativamente bassi rispetto al costo di invio, la probabilità di rigetto e di perdita di dati risulta decisamente più elevata.

Il paradigma del Fog Computing è una soluzione architetturale nuova, di conseguenza, in letteratura, si trovano pochi dati relativi a questo problema.

Per questo motivo non è stato possibile confrontare i risultati con quelli già esistenti. Il nostro approccio si è basato totalmente sull'implementazione di tecniche euristiche e innovative per la risoluzione del problema. Per valutare la validità e l'efficienza di queste tecniche si potrebbe utilizzare un solver di programmazione lineare per trovare la soluzione ottimale della funzione obiettivo rispetto ai suoi constraints e si potrebbero comparare i risultati.
Inoltre, per rendere più complesso il problema si potrebbe inserire un sistema di distanze fra i nodi oppure si potrebbe introdurre nel sistema altri parametri, come la latenza e il tempo di inter-arrivo delle richieste, che nella nostra analisi non sono stati tenuti in considerazione.
Per valutare il lavoro, si potrebbe inoltre cercare di applicare le metodologie a scenari più reali, cosa che è stata possibile solo in minima parte attraverso il dataset sui taxi, a causa della forte novità rappresentata dal paradigma del *Fog Computing*.

6 Conclusioni

Lo scopo del nostro lavoro era quello di presentare delle metodologie euristiche che potessero portare all'ottimizzazione di un sistema di *Fog Computing*, attraverso la minimizzazione degli scambi di dati fra i diversi nodi. É stato evidenziato come la prima euristica sia quella che funzioni meglio per questo scopo, soprattutto se si considera un sistema che comprende un numero elevato di nodi interconnessi. Tuttavia, anche la seconda euristica può considerarsi rilevante nel momento in cui ci siano dei problemi all'interno del fog.
Per quanto riguarda lo scenario in cui è presente un Cloud centrale, in cui l'architettura del fog risulti differente, grazie alla prima euristica è anche possibile minimizzare il rigetto delle richieste e la conseguente perdita di dati.
La scarsa presenza in letteratura di informazioni relative a questo problema non ha permesso il confronto dei risultati con altri tipi di analisi.

Riferimenti bibliografici

[1] Tesi di Laurea Magistrale, Luigi Maio, *Scenari applicativi nell'ambito del paradigma Fog Computing in presenza di reti partizionate e non affidabili*

[2] Lavoro di Tesi di Laurea Magistrale, Stefano Fiorini