UNIVERSITÀ
DEGLI STUDI
FIRENZE

Università degli Studi di Firenze
Dipartimento di Ingegneria dell'Informazione (DINFO)
Corso di Laurea Magistrale in Ingegneria Informatica

Quantitative Evaluation Of Stochastic Models

Autore
Francesco Orlandi

Relatore
Prof. Enrico Vicario
Prof. Benedetta Picano

Anno Accademico 2024-2025

Indice

Capitolo 1

Introduzione

I sistemi informatici di nuova generazione, dovranno servire un'ampia classe di nuovi servizi che avranno requisiti eterogenei in termini di qualità del servizio (QoS). I dati grezzi vengono tipicamente pre-elaborati su nodi di elaborazione intermedi, definiti IPN, situati tra nodi sorgente e il server. Gli IPN sono nodi ad alte prestazioni, ma limitati a livello computazionale, con diversa disponibilità di elaborazione (architetture serverless o multitenant), al fine di pre-elaborare i carichi di lavoro, ad alta intensità di dati, prima di inviarli al server. In particolare, il tempo trascorso tra la generazione della richiesta e il suo arrivo al server di destinazione è indicato come *tempo di completamento*. Garantire un tempo di completamento conforme alla scadenza associata a un dato flusso, è solitamente un punto critico nei sistemi distribuiti, nel rispetto della QoS. La differenza tra il tempo di completamento effettivo e la scadenza è definita *ritardo*. Bisogna specificare, che il sistema soffre anche di una *latenza di rete* innata, che influisce sulla lentezza di comunicazione all'interno del sistema. Queste operazioni, fanno parte del concetto di *offloading computazionale*, ovvero quel processo di trasferire l'elaborazione di compiti, o dati, da un dispositivo locale (utente) ad un dispositivo più potente, ma remoto (server), per migliorare le prestazioni e ridurre il consumo di risorse locali. Vogliamo così massimizzare l'utilità del nostro sistema informatico riducendo il più possibile le latenze in gioco. Con utilità si intende la misura del beneficio derivato dall'uso del sistema in relazione agli obiettivi dell'utente. Si ha così un problema di *instardamento*.

Per la gestione di questi sistemi è possibile applicare la *Matching Theory* (MT), un approccio nato nell'area economica, per stabilire relazioni tra gli

elementi appartenti a due insiemi opposti. L'aspetto saliente è la capacità di eseguire connessioni reciprocamente vantaggiose tra gli elementi coinvolti nel gioco del matching. Ci si basa su liste di preferenza, associata ad ogni giocatore, nel quale si esprime la propria soddisfazione nell'essere accoppiati con ciascun giocatore dell'insieme opposto. L'idea è quella di convergere verso un abbinamento dove non esiste alcuna coppia, i cui elementi, si preferiscano reciprocamente al loro partner attuale. L' esempio più noto di MT è l'algoritmo *Gale-Shapley*, il quale però, è troppo rigido per il sistema informatico delineato, perchè le liste di preferenza sono indipendenti, comportando la non cattura delle corrette dinamiche del sistema. A questo scopo si introduce la MT basata sulle *externalities*, per cui, oltre ai loro partner, i partecipanti al gioco si preoccupano anche degli abbinamenti degli altri giocatori. Ora, le liste di preferenza, cambiano durante l'esecuzione dell'algoritmo, sulla base delle scelte effettuate in precedenza. Lo studio della stabilità diventa un problema NP-hard, non banale da risolvere.
Il sistema è così caratterizzato:

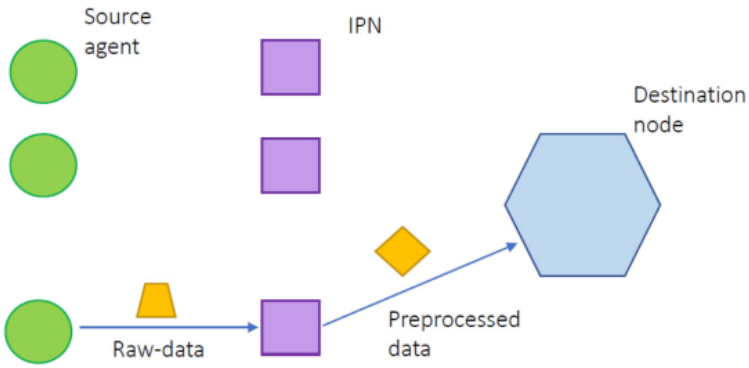

Figura 1.1: Sistema Informatico

Capitolo 2

Modello implementato

Il modello analizzato è caratterizzato da due attori principali, *Utente - IPN*, dove lo scopo del modello consiste nell'operazione di pre-processazione dei *tasks* (o flussi) grezzi dell'utente nei Nodi IPN, per agevolare il carico di lavoro ad un ipotetico database (nodo di destinazione). Il "paper" preso in considerazione è il seguente: *"An Efficient Flows Dispatching Scheme for Tardiness Minimization of Data-Intensive Applications in Heterogeneous Systems"*, il quale delinea un algoritmo basato sulla Matching Theory, come alternativa più promettente alla forma vanilla Gale-Shapley. L'obiettivo di questo algoritmo è la definizione di uno schema di distribuzione dei flussi per massimizzare l'utilità del sistema in una rete di IPN con capacità limitata ed eterogenea, cercando di tenere sotto controllo i ritardi. l'algoritmo tiene conto delle *externalities* e si introduce una stabilità definita *two-sided exchange stability*, la quale essendo NP-hard, avrà una soluzione sub-ottimale.

2.1 Parametri

Ho implementato tutti i parametri eccetto la latenza di rete dai nodi IPN verso il database ($t_{z,d}$), perchè ho implementato il modello *Utente - IPN*. Ho assunto che l'assegnazione del flusso "i" ($i \in \mathcal{F}$) per una pre-elaborazione in un dato IPN "z" ($z \in \mathcal{I}$), comporti l'allocazione di risorse R_i tra quelle disponibili L_z nel nodo IPN. La latenza sperimentata dal flusso generico i è definita $t_{i,z}$, ovvero il tempo necessario per inviare dati dal nodo sorgente all'IPN. La latenza di comunicazione della rete end-to-end è definita come: $r_{i,d,z} = t_{i,z} + t_{z,d}$, dove nel mio modello $t_{z,d} = 0$. Il tempo im-

piegato dal flusso i nell'IPN per completare la sua computazione è definito *tempo di pre-elaborazione*: $P_{i,d,z} = q_{z,i} + p_{z,i}$, dove $q_{z,i}$ rappresenta il *tempo di attesa in coda* subito dal flusso i all'IPN a causa della presenza di altri flussi in coda al nodo precedentemente assegnati, mentre $p_{z,i}$ è il *tempo di elaborazione* effettivo, cioè di servizio, del flusso i. Il parametro $q_{z,i}$ è di fondamentale importanza, perchè rappresenta l'*externalities* dell'algoritmo di matching, influenzando le liste di preferenza sia lato IPN che lato flusso, creando dipendenze e interrelazioni tra loro. I flussi, in una stessa coda IPN, vengono selezionati per accedere al servizio in base all'ordinamento dell'allocazione fornito dall'algoritmo MT che ho utilizzato, ovvero quello proposto dal paper. E' importante sottolineare che il servizio di una nuova richiesta non può iniziare in nessun IPN finchè la trasmissione dati della richiesta di pre-elaborazione precedente non è stata completata, per cui consegue che il *tempo di completamento* del flusso i nell'IPN z risulta: $C_{i,d,z} = P_{i,d,z} + r_{i,d,z}$. La scadenza associata al flusso i è definita β_i, consegue che il servizio del flusso è effettuato nel rispetto della QoS solo se: $C_{i,d,z} \leq \beta_i$, per cui parliamo di *ritardo* definito come $T_i = \max\{0, C_{i,d,z} - \beta_i\}$, se computo in tempo, il mio ritardo risulterà sempre nullo. La casistica di ritardo è causata da due situazioni principali, una richiesta viene eseguita con un tempo di completamento maggiore della scadenza corrispondente, oppure una richiesta non viene accettata dall'IPN a causa di una coda piena o per insufficienza di capacità L_z. In termini formali possiamo esprimere questo fenomeno come: $D = \{i \in \mathcal{F} \mid T_i > 0 \vee C_{i,d,z} \to \infty\}$. Introduciamo ora la metrica dell'utilità del nostro sistema, difatti è correlata alla probabilità di QoS negata e al tempo medio di completamento sperimentato da ciascun flusso nella rete. Per cui è definita come:

$$\mathcal{U} = \left(\pi \frac{\sum_{i \in \mathcal{F}} \sum_{z \in \mathcal{I}} C_{i,z,d} \alpha_{i,z}}{\mathcal{A}} \right)^{-1}$$

dove:

$$\pi = \frac{|D|}{|\mathcal{F}|}$$

che denota il numero di flussi che vengono pre-elaborati al nodo IPN, ma con ritardo, su tutti i flussi pre-elaborati in quell'iterazione. L'elemento $\alpha_{i,z}$ è l'elemento generico della matrice di allocazione $\mathcal{A}$, ovvero quella matrice che contiene tutti i flussi del nostro sistema i quali assumono valore "1" se sono allocati ai nodi IPN in gioco in quell'iterazione e zero altrimenti.

Ricapitolando, i parametri utilizzati nel nostro modello sono:

Parameter	Notation
$\mathcal{F}$	flows set
$\mathcal{I}$	IPNs set
$t_{i,z}$	network latency for link (i, z)
$t_{z,d}$	network latency for link (z, d)
$r_{i,d,z}$	communication time
$q_{z,i}$	queuing time
$p_{z,i}$	service time
$P_{i,d,z}$	pre-processing time
$C_{i,d,z}$	completion time
β_i	deadline
T_i	tardiness

Figura 2.1: Parametri

2.2 Struttura del codice

E' costituito da 6 classi, il *class diagram* è il seguente:

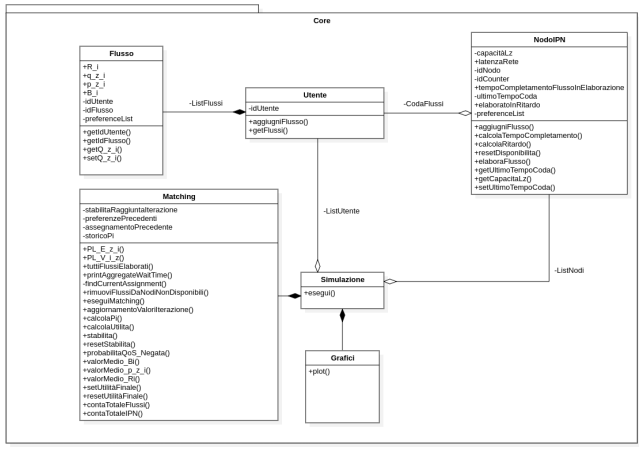

Figura 2.2: Codice

Come possiamo vedere in Figura 2.2, il codice è stato strutturato per delineare una classe Utente la quale ha dei flussi, che esistono, se esiste almeno un utente. I nodi IPN attingono ai flussi dagli utenti. In questo modo è possibile la creazione delle liste di preferenza da ambo i lati dei

giocatori per il matching. La classe Simulazione si occupa di ricevere le liste di nodi e di utenti, inoltre, richiamando i metodi della classe Matching potrà eseguire l'algoritmo e collezionare i vari dati che serviranno per creare i grafici di test mediante i vari metodi di plot utilizzati nella classe Grafici.

2.3 Algoritmo

Ho implementato l'algoritmo di MT basato sulle *externalities* per la distribuzione ed instardamento dei flussi appartenenti al sistema. Si crea una relazione molti-a-uno tra flussi ed IPN in grado di fornire efficacemente la massimizzazione dell'utilità. A tal fine per ogni flusso $i \in \mathcal{F}$ e ogni IPN $z \in \mathcal{I}$ indichiamo con $V_i(z)$ ed $E_z(i)$ le *funzioni di utilità* degli elementi appartenenti rispettivamente ad $\mathcal{F}$ ed $\mathcal{I}$, Sulla base delle quali devono essere costruite le corrispondenti liste di preferenza:

- *Lista di preferenza dei flussi verso gli IPN*: il singolo flusso ha una lista di preferenza degli IPN ordinata in modo crescente su base del parametro $V_i(z) = C_{i,d,z}$ (il primo IPN avrà $V_i(z)$ più basso in quella specifica iterazione). Si va così a preferire il nodo IPN con meno flussi in coda, dove il singolo flusso ha un tempo di completamento minore. Questa particolare lista di preferenza deve essere aggiornata ad ogni iterazione, modificando le assegnazioni dei flussi ai nodi IPN, dato che il parametro $C_{i,d,z}$ dipende dall'externality $q_{z,i}$.

- *Lista di preferenza degli IPN verso i flussi*: il singolo nodo IPN ha una lista di preferenza di flussi ordinata in modo decrescente su base del parametro $E_z(i) = 1/\beta_i$. L'IPN preferisce $E_z(i)$ bassi, ovvero con β_i del flusso larga. Questa particolare lista di preferenza si crea all'inizio dell'algoritmo, quindi all'inizio della simulazione del nostro sistema e non cambierà durante l'esecuzione delle varie iterazioni, dato che il valore della scadenza associata al flusso è statico.

L'algoritmo proposto dal paper che ho implementato è il seguente:

```
1: for each flow i in F do
2:   Build the preference list according to (14);
3: for each IPN z do
4:   Build the preference list according to (15);
5: for each flow i do
6:   Send proposal to its favorite IPN z*
7:     on its preference list;
8: for each IPN z receiving at least one proposal do
9:   accept the favorite request i*
10:    among those received
11:    in accordance with (15), reject the other proposals;
```

Figura 2.3: Algoritmo MT

Per rendere chiara l'idea di come funziona, fornisco uno pseudocodice più dettagliato:

1. **Lista di Preferenze Statica dei nodi IPN (PL $E_{z,i}$)**:
   - All'inizio, viene creata una lista di preferenze statica per **tutti** i nodi basata su $\frac{1}{B_i}$.
   - La lista è ordinata in modo decrescente di $\frac{1}{B_i}$, il che significa che l'**ultimo** elemento ha il B_i più grande (scadenza più permissiva).
   - Questa lista è la stessa per **tutti** i nodi IPN.
2. **Lista di Preferenze Dinamica dei flussi (PL $V_{i,z}$)**:
   - Per ogni flusso, viene calcolata una lista di preferenze basata sui tempi di completamento.
   - I nodi sono ordinati in maniera crescente in base al tempo di completamento associato a quel flusso.
   - Il **primo** nodo in questa lista è quello dove il flusso ha il tempo di completamento più basso.
3. **Prima Iterazione (Matching Iniziale)**:
   - Nella prima iterazione, quando tutte le code sono vuote ($q_{z,i} = 0$), i flussi si collegano ai nodi.
   - I flussi vengono assegnati ai nodi che:
     - **(a)** Hanno il flusso nella loro lista di preferenze $E_{z,i}$;
     - **(b)** Hanno meno di 3 flussi in coda;

  - **(c)** Hanno capacità sufficiente L_z.

4. **Elaborazione e Iterazioni Successive**:

   - Per ogni nodo con almeno un flusso in coda ed al massimo tre:
     - Il flusso che viene elaborato è quello che ha la scadenza più grande, secondo $E_{z,i}$;
     - Viene calcolato il tempo di completamento di quel flusso;
     - I tempi di attesa in coda al nodo per gli altri flussi vengono aggiornati sulla base del tempo di completamento del flusso elaborato;
     - Viene calcolato il ritardo rispetto il flusso elaborato;
   - Il flusso viene elaborato e quindi rimosso dalla lista $E_{z,i}$ (limite di elaborazione di un solo flusso ad iterazione per nodo IPN);
   - Si decrementa la capacità dell'IPN del peso R_i del flusso elaborato, nel caso in cui la capacità del nodo vada a zero, svuoto la sua coda dai flussi rimanenti per poterli assegnare a nuovi nodi disponibili.

5. **Aggiornamento Liste di Preferenze Dinamica**:

   - Dopo l'elaborazione, le liste di preferenza per ogni flusso su ogni nodo IPN (PL $V_{i,z}$) vengono ricalcolate per i flussi rimanenti in gioco in base ai nuovi tempi di attesa in coda aggiornati di ogni nodo IPN disponibile. I flussi vengono così riassegnati ai nodi IPN preferiti.

La classe Matching è la classe che implementa questo algoritmo di MT, il *sequence diagram* è il seguente:

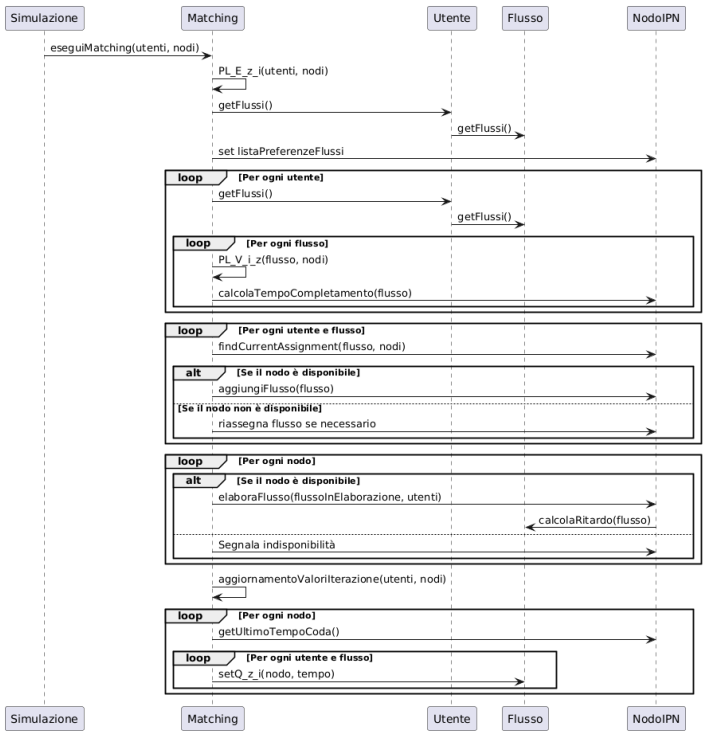

Figura 2.4: sequence diagram

2.4 Stabilità

La teoria afferma che nel matching-game con *externalities*, non esiste alcun algoritmo di matching che converga in un abbinamento stabile a causa della presenza delle dipendenze tra le preferenze dei giocatori. Il comportamento dinamico delle liste di preferenza invalida i risultati di stabilità convenzionali secondo cui l'algoritmo Gale-Shapley converge a un matching stabile, ovvero la configurazione in cui i giocatori non hanno incentivi a scambiare reciprocamente il partner. Quindi la stabilità canonica nel mio caso non può essere applicata. Parliamo così di *two-sided exchange stability*, in pratica il matching è definito stabile in senso strettamente bilaterale se non esiste alcuna coppia di agenti, non abbinati tra loro nell'allocazione corrente, che possono scambiarsi risorse o modificare il loro abbinamento in modo che en-

trambi migliorino la loro situazione. Non dovrebbe esserci nessuna coppia di soggetti che, pur non essendo attualmente associati, possano concordare uno scambio che li porterebbe a ottenere un risultato migliore rispetto a quello che hanno. Questo requisito garantisce che nessuna deviazione strettamente bilaterale (cioè un cambiamento concordato tra due agenti) possa rendere entrambi i partecipanti più soddisfatti, mantenendo così il matching stabile. L'idea che ho implementato è il metodo "stabilità", la cui funzione è riassunta in:

- Se siamo alla prima iterazione, salva lo stato corrente e restituisce `false`.
- Nelle iterazioni successive confronta lo stato corrente con quello salvato.
- Se le preferenze o l'assegnamento di un flusso sono cambiati, la stabilità non è ancora raggiunta.
- Se non ci sono cambiamenti, la stabilità è raggiunta e viene stampato un messaggio con l'iterazione in cui ciò è avvenuto.

Se nessun flusso (e quindi nessuna coppia flusso-nodo) ha interesse a deviare, allora l'allocazione è stabile sul piano bilaterale. Questo significa che non esiste alcuna coppia che possa scambiare o modificare l'assegnamento per migliorare entrambi i lati.

2.5 Conclusioni

Ho effettuato tre simulazioni sequenziali, la prima simulazione è caratterizzata da 6 nodi IPN e 3 utenti, dove ogni utente ha 6 flussi. La seconda simulazione è caratterizzata da 5 nodi IPN e 3 utenti, dove ogni utente ha 8 flussi. La terza simulazione è caratterizzata da 4 nodi IPN e 3 utenti, dove ogni utente ha 13 flussi. le deadline β_i sono più larghe nella prima simulazione e vanno a essere più stringenti alle simulazioni successive, mentre avviene il contrario per il tempo di servizio $p_{z,i}$ del flusso. I pesi R_i dei flussi sono prevalentemente equilibrati tra le tre simulazioni. I dati delle tre simulazioni sono stati salvati (avrò un grafico costituito da tre punti) in modo tale da creare vari grafici che rappresentano le funzionalità del mio sistema informatico. L'utilità è l'unico parametro che ho espresso in percentuale. Ho riscontrato un andamento simile e coerente con quelli forniti dal paper, come segue:

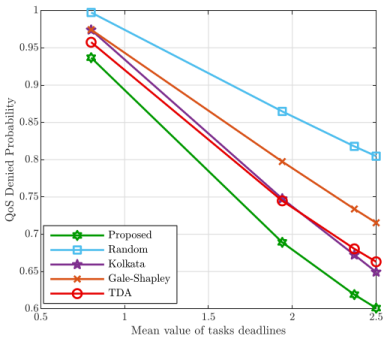

Figura 2.5: Grafico 1 paper

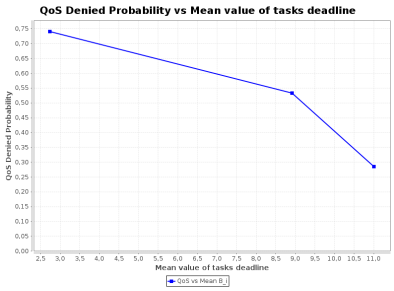

Figura 2.6: Grafico 1 riscontrato

Per valori medi delle deadline bassi, possiamo notare che essendo stringenti, il valore del disservizio aumenta verso 1, per valori alti delle deadline, notiamo l'effetto opposto, ovvero che il disservizio diminuisce nel rispetto della QoS per l'utente.

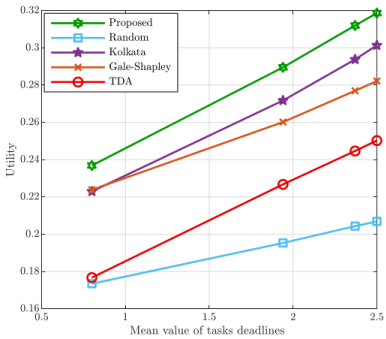

Figura 2.7: Grafico 2 paper

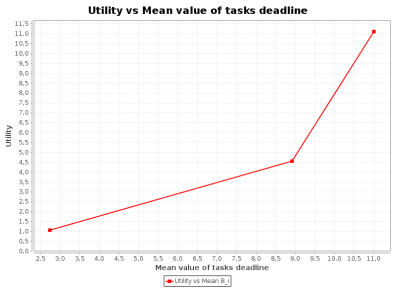

Figura 2.8: Grafico 2 riscontrato

Con deadline medie più alte, l'utilità del sistema aumenta, poichè con scadenze più flessibili, più flussi possono essere computati entro la deadline.

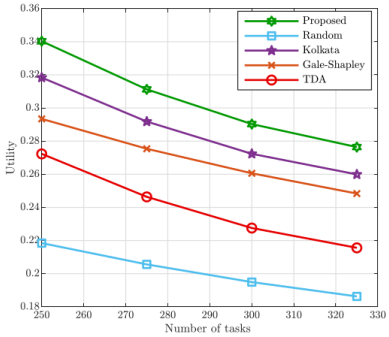

Figura 2.9: Grafico 3 paper

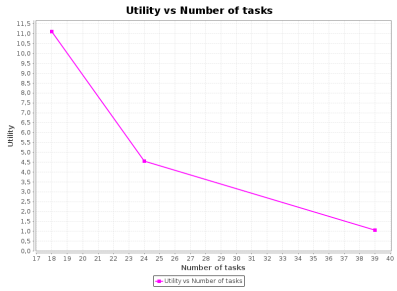

Figura 2.10: Grafico 3 riscontrato

All'aumentare del numero medio dei flussi, l'utilità del sistema tende a diminuire, perchè richiede più risorse di rete rendendo più difficile il rispetto delle deadline.

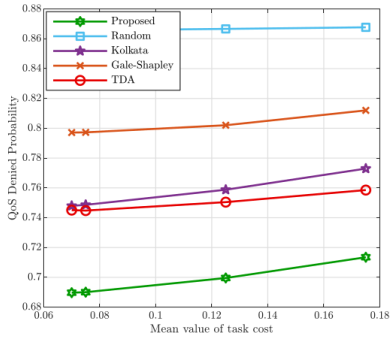

Figura 2.11: Grafico 4 paper

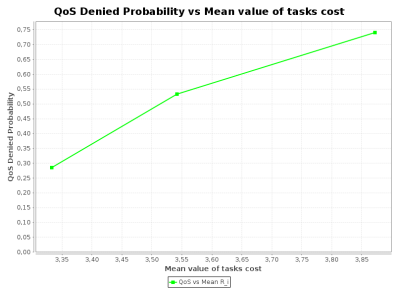

Figura 2.12: Grafico 4 riscontrato

All'aumentare del costo medio dei flussi, il disservizio aumenta, perchè i flussi più costosi richiedono più risorse di elaborazione dei nodi IPN rendendo difficile il rispetto delle deadline.

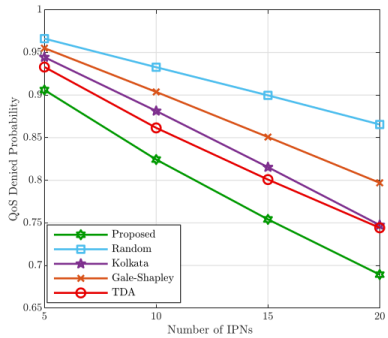

Figura 2.13: Grafico 5 paper

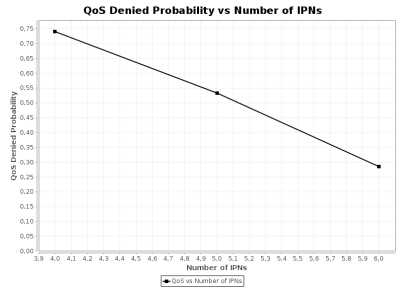

Figura 2.14: Grafico 5 riscontrato

Aumentando il numero di nodi IPN, il disservizio diminuisce, poichè più risorse di elaborazione diventano disponibili, riducendo il carico di lavoro di ciascun IPN, migliorando la capacità del sistema di completare i flussi.

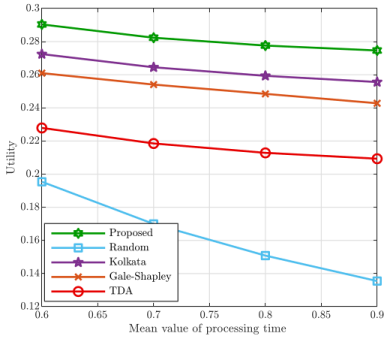

Figura 2.15: Grafico 6 paper

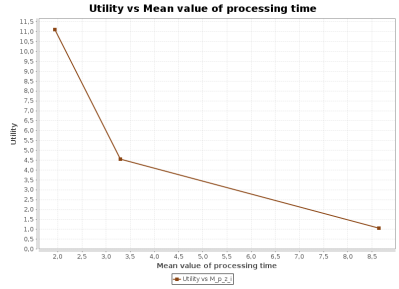

Figura 2.16: Grafico 6 riscontrato

All'aumentare del valor medio del tempo di servizio dei flussi, l'utilità del sistema decrementa perchè diventa più difficile rispettare le deadline.

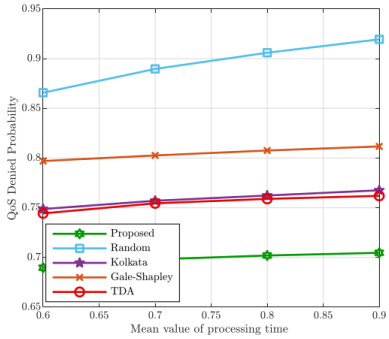

Figura 2.17: Grafico 7 paper

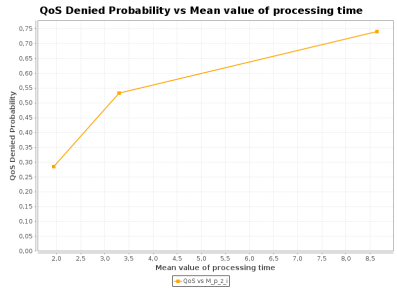

Figura 2.18: Grafico 7 riscontrato

All'aumentare del valor medio del tempo di servizio dei flussi, il disservizio aumenta perchè diventa più difficile rispettare le deadline.

Bibliografia

[1] B. Picano, E. Vicario, and R. Fantacci, "An efficient flows dispatching scheme for tardiness minimization of data-intensive applications in heterogeneous systems," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 6, pp. 3232–3241, 2023.