A novel cloud workflow scheduling algorithm based on stable matching game theory

Lorenzo Cappetti

July 4, 2025

1 Riassunto

La pianificazione dei workflow (workflow scheduling) è uno dei problemi più popolari e complessi nel cloud computing. Tuttavia, tra gli studi esistenti su questo tema, pochissimi prendono in considerazione l'equità tra i diversi compiti del workflow, un aspetto che può ritardare significativamente l'esecuzione dei workflow e, di conseguenza, ridurre la soddisfazione degli utenti. In questo articolo, proponiamo un algoritmo di pianificazione dei workflow basato sulla teoria dei giochi di abbinamento stabile (stable matching game theory), con l'obiettivo di minimizzare il makespan (tempo totale di completamento del workflow) e garantire equità tra i compiti. Per migliorare le prestazioni dell'algoritmo, sono stati sviluppati metodi di ottimizzazione locale basati sul percorso critico e sulla duplicazione dei task. Inoltre, viene proposto un nuovo indicatore per valutare l'equità tra i task del workflow. Esperimenti approfonditi sono stati condotti per confrontare le prestazioni dell'algoritmo proposto con altri quattro algoritmi rappresentativi. I risultati dimostrano che il nostro algoritmo supera gli altri in tutti e tre i parametri di valutazione delle prestazioni, in differenti scenari applicativi.

2 Introduzione

Il cloud computing [1] è un servizio che fornisce in modo efficiente un grande numero di risorse computazionali e una gestione automatizzata delle risorse attraverso il software. Gli utenti possono accedere a risorse teoricamente illimitate tramite la rete, secondo un modello di pagamento "pay-as-you-go" [2], senza vincoli di tempo o luogo. Inoltre, le risorse cloud presentano numerose caratteristiche vantaggiose, come elasticità, provisioning su richiesta e virtualizzazione. Esistono tre principali tipologie di servizi cloud: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) e Software as a Service (SaaS) [3].

I workflow rappresentano applicazioni complesse del mondo reale, come quelle in fisica, bioinformatica, previsioni meteorologiche e astronomia [4, 5]. La maggior parte dei problemi di pianificazione dei workflow in ambiente cloud mira

alla minimizzazione del makespan (tempo totale di esecuzione), un problema noto come NP-completo nel contesto cloud [6]. Una pianificazione adeguata può ridurre il consumo energetico nei data center, aumentare l'utilizzo delle risorse, ridurre il makespan e migliorare la soddisfazione dell'utente. Pertanto, la questione di come pianificare in modo efficiente i workflow in ambiente cloud è di grande rilevanza.

Gli algoritmi di pianificazione si suddividono generalmente in euristici e metaeuristici. I primi forniscono soluzioni approssimate con complessità polinomiale, mentre i secondi, basati su ricerche iterative, migliorano la qualità delle soluzioni al costo di maggiore tempo computazionale. Gli algoritmi euristici si dividono ulteriormente in algoritmi basati su liste, duplicazione e raggruppamento (clustering). Tra gli algoritmi metaeuristici si annoverano i genetic algorithm (GA), il particle swarm optimization (PSO), gli algoritmi memetici (MA) e altri ancora.

Tuttavia, la maggior parte degli algoritmi di pianificazione dei workflow in cloud si focalizza su obiettivi comuni come la minimizzazione del costo totale o del makespan complessivo del workflow. Nella pratica, però, esistono molti workflow — ad esempio nella videosorveglianza, nel tracciamento di oggetti [7] o nel riconoscimento facciale [8] — in cui specifici task hanno obiettivi individuali, come il tempo di risposta minimo o la massima velocità di elaborazione. Se si utilizza un algoritmo di pianificazione che assegna sempre le risorse migliori (es. con massima larghezza di banda e velocità) ai task in base alla loro priorità, alcuni task potrebbero non riuscire a soddisfare i requisiti dell'utente, il che risulta ingiusto. Questa distribuzione non equa delle risorse può ridurre significativamente la soddisfazione di alcuni task, influenzando negativamente la soddisfazione generale degli utenti. Pertanto, è necessario considerare anche l'equità tra i task, oltre agli obiettivi globali del workflow [9–11].

Poiché la teoria dei giochi di abbinamento stabile (Stable Matching Game Theory, SMGT) è stata largamente adottata per bilanciare equità e competizione tra task, in questo articolo proponiamo un algoritmo di pianificazione per workflow in ambienti cloud eterogenei basato sulla SMGT [12, 13], con l'obiettivo di minimizzare il makespan e garantire equità tra i task. A nostra conoscenza, questa è la prima applicazione della SMGT ai problemi di pianificazione dei workflow, e i risultati ottenuti sono promettenti. In particolare, per migliorare l'efficienza, sono state integrate due tecniche di ottimizzazione locale basate sul percorso critico [14] e sulla duplicazione dei task [15–17].

L'obiettivo del nostro algoritmo è distribuire più workflow cloud di diverse dimensioni su macchine virtuali (VM) con differenti capacità di elaborazione, in modo da minimizzare il makespan e massimizzare l'equità tra i task.

I principali contributi di questo articolo sono i seguenti:

- Viene considerata l'equità tra i task dei workflow e si propone un nuovo algoritmo di pianificazione basato su SMGT, capace di bilanciare gli obiettivi globali dei workflow con l'equità dei singoli task.
- Vengono progettati metodi di ottimizzazione locale basati sul percorso critico e sulla duplicazione dei task per migliorare l'efficienza dell'algoritmo.

- Si propone un nuovo indicatore per misurare l'equità tra i task del workflow.
- L'algoritmo proposto viene confrontato con altri quattro algoritmi rappresentativi, tramite esperimenti approfonditi su quattro strutture tipiche di workflow.

Struttura del resto dell'articolo: La Sezione 2 fornisce una rassegna dei recenti lavori sulla pianificazione dei workflow in ambiente cloud. La Sezione 3 definisce il modello di workflow, il modello cloud e la formulazione del problema. La Sezione 4 descrive in dettaglio l'algoritmo proposto, includendo anche un esempio. I risultati sperimentali e la relativa analisi sono presentati nella Sezione 5. Infine, la Sezione 6 conclude l'articolo.

3 Rassegna della letteratura

Negli ultimi anni sono stati proposti numerosi studi sulla pianificazione dei workflow in ambiente cloud. In generale, gli algoritmi di pianificazione dei workflow cloud possono essere suddivisi in due categorie principali: euristici e metaeuristici (si veda la Tabella 1). Uno degli algoritmi euristici più popolari è l'HEFT (Heterogeneous Earliest Finish Time) [14], che mira a minimizzare il tempo di completamento del workflow assegnando a ciascun task un valore di priorità crescente (upward rank) e utilizzando un approccio basato sull'inserimento dei task.

Un'altra strategia, denominata CPOP (Critical-Path-on-a-Processor), assegna la priorità ai task sulla base della somma tra i rank crescenti e decrescenti, al fine di selezionare i task critici da pianificare per primi durante la fase di scelta del processore. I risultati sperimentali hanno dimostrato la superiorità di entrambi i metodi sia in termini di qualità delle soluzioni sia in termini di velocità di ricerca. Sulla base di HEFT sono stati proposti molti algoritmi migliorati. Ad esempio, Samadi et al. [18] hanno proposto E-HEFT, una versione migliorata dell'algoritmo HEFT che considera i vincoli economici degli utenti e il bilanciamento del carico tra le VM, per ridurre il makespan del workflow.

Cao et al. [19] hanno invece presentato una strategia chiamata K-HEFT, basata sulla selezione dei Top-k task più programmabili, in base alla loro priorità, per ciascuna assegnazione, in modo da ridurre il tempo di completamento minimo del workflow.

TAB1

Considerando i vincoli di scadenza, la variabilità delle prestazioni delle VM e i ritardi nell'acquisizione delle istanze, Sahni et al. [20] hanno proposto un algoritmo euristico per minimizzare sia il costo sia il makespan. Zheng et al. [21] hanno introdotto tre algoritmi distinti:

1. **CFMax (Change From Maximum):** utilizza HEFT per generare una pianificazione iniziale, quindi riassegna i task per ridurre i costi;

2. **CRR (Cost–Runtime Ratio):** mira alla riduzione del makespan partendo dai risultati di CFMax;

3. **CBT-MD (basato sulla distanza di Manhattan) e CBT-ED (basato sulla distanza Euclidea):** valutano globalmente la pianificazione preesistente tramite due diverse metriche

Tutti questi algoritmi riescono a ridurre i costi di esecuzione dei workflow senza violare i vincoli di scadenza. In particolare, CBT-MD e CBT-ED si sono dimostrati i più efficaci, poiché la riassegnazione globale dei task consente risultati migliori.

Wu et al. [22] hanno proposto un approccio di pianificazione dei task consapevole degli errori soft per workflow in data center cloud dotati di meccanismi di scalabilità dinamica di tensione e frequenza (DVFS). Questo metodo consente di generare pianificazioni energeticamente efficienti rispettando i vincoli di affidabilità e tempo di completamento.

Ijaz e Munir [23] hanno proposto un algoritmo euristico basato su liste con duplicazione ottimizzata per ridurre il makespan, mantenendo la stessa complessità temporale degli algoritmi esistenti.

Zhang et al. [24] hanno descritto il problema della pianificazione dei workflow come un problema di programmazione intera, proponendo uno schema di prioritizzazione dei job che mantiene le priorità per i task con vincoli di precedenza, e assegna priorità a quelli senza precedenze basandosi sulla loro importanza. Inoltre, hanno ideato una strategia uniforme di suddivisione del budget extra che migliora la gestione della domanda e riduce il makespan.

Djigal et al. [25] hanno proposto un algoritmo migliorato di pianificazione basato sulla predizione delle priorità, articolato in due fasi: assegnazione delle priorità e selezione dei processori. Questo algoritmo riduce significativamente il makespan anticipando le decisioni in entrambe le fasi.

Geng et al. [26] hanno proposto un nuovo algoritmo basato sulla duplicazione e il raggruppamento dei task per minimizzare il tempo totale di esecuzione del workflow.

Kumar et al. [27] hanno presentato un algoritmo innovativo che sfrutta la granularità dei task per minimizzare il makespan e massimizzare l'utilizzo medio delle VM.

Gupta et al. [28] hanno calcolato soglie dinamiche per i task tramite normalizzazione min–max, per garantire sia il rispetto del makespan sia l'uso efficiente delle risorse cloud.

Infine, l'algoritmo Min–min [29], considerato uno dei benchmark più comuni per la pianificazione dei workflow, fornisce una base semplice ma efficace per valutare le prestazioni di altri algoritmi. Attualmente, la maggior parte degli algoritmi euristici si concentra sull'ottimizzazione di un singolo obiettivo, con alcuni vincoli, il che rappresenta una situazione ideale [30]. Tuttavia, per affrontare problemi di pianificazione dei workflow con obiettivi multipli — più vicini alla realtà — gli algoritmi metaeuristici hanno dimostrato la loro superiorità.

Tenendo conto sia dei costi di trasmissione dati che di elaborazione, Wu et al. [31] hanno proposto una versione rivisitata del Particle Swarm Optimization (PSO) discreto per la pianificazione dei workflow. Per minimizzare contemporaneamente tre obiettivi — flowtime, makespan e costo d'uso delle risorse — Kaur e Kadam [32] hanno esteso l'algoritmo originale di foraggiamento batterico (bacteria foraging) incorporando un nuovo metodo di assegnazione della fitness e una strategia di selezione batterica per l'ottimizzazione multi-obiettivo. Tuttavia, il loro metodo considera compiti indipendenti, e ciò lo rende inadatto ai problemi di pianificazione di workflow.

Sulla base del PSO, Hu et al. [33] hanno proposto un algoritmo di pianificazione multi-obiettivo che mira a minimizzare sia il makespan che i costi, considerando anche il vincolo di affidabilità. Inoltre, nel metodo di codifica, vengono considerate sia la posizione di esecuzione dei task sia l'ordine di trasmissione dei dati.

Huang et al. [34] hanno integrato nel PSO un algoritmo euristico chiamato simplified swarm optimization, con parametri dinamici progettati per regolare le capacità di ricerca locale e globale dell'algoritmo. Ding et al. [35] hanno presentato una strategia di pianificazione a basso costo basata su PSO per multi-workflow, sotto vincoli di scadenza nel contesto del Fog computing. Alsmady et al. [36] hanno proposto un algoritmo memetico (MA) in cui viene adottata una ricerca locale hill climbing come operatore aggiuntivo per migliorare le soluzioni durante la fase di ricerca globale del Genetic Algorithm. I risultati sperimentali hanno mostrato che il MA è in grado di ridurre simultaneamente il makespan e il costo del workflow.

Rispetto agli algoritmi euristici, i metaeuristici presentano vantaggi nella ricerca di soluzioni migliori ai problemi di pianificazione dei workflow. Tuttavia, il tempo di calcolo dei metaeuristici è in genere molto più elevato. Di conseguenza, quando è richiesto un tempo di risposta rapido o la dimensione del problema è elevata, i metaeuristici risultano meno adatti. Per questo motivo, gli algoritmi euristici sono spesso preferibili per affrontare problemi con un solo obiettivo.

La teoria dei giochi (Game Theory, GT) si concentra principalmente sull'interazione strategica tra decisori razionali, ed è largamente applicata in vari campi della logica e delle scienze dei sistemi. Considerando l'affidabilità nella distribuzione equilibrata dei compiti, Yang et al. [37] hanno presentato un algoritmo di pianificazione dei task basato su un modello cooperativo di gioco, capace di ridurre la complessità dell'algoritmo mantenendone l'efficienza.

Per affrontare la pianificazione dei task in ambienti grid computing, Gao et al. [38] hanno trattato il problema del bilanciamento del carico come un gioco non cooperativo, proponendo un algoritmo basato su GT per minimizzare i costi della griglia. I risultati sperimentali dimostrano che l'algoritmo basato sul gioco ha buone capacità di risoluzione del problema di pianificazione.

Wang et al. [39] hanno proposto un algoritmo di pianificazione multi-obiettivo basato su un modello dinamico di teoria dei giochi, con l'obiettivo di minimizzare makespan e costo totale e, al contempo, massimizzare l'equità nella distribuzione del carico tra VM eterogenee.

Sujana et al. [40] hanno formulato la pianificazione multi-obiettivo dei workflow come un nuovo gioco cooperativo sequenziale, con due obiettivi: minimizzare il tempo di esecuzione e il costo economico, soggetti a due vincoli.

Sebbene la teoria dei giochi abbia dimostrato vantaggi nella pianificazione dei workflow, sono ancora pochissimi gli studi che prendono in considerazione la questione dell'equità tra i task. Per questo motivo, in questo lavoro, applichiamo l'algoritmo di abbinamento stabile (stable matching) — uno dei modelli della teoria dei giochi — per affrontare la pianificazione dei workflow cloud, con l'obiettivo di minimizzare il makespan e massimizzare l'equità tra i task del workflow, simultaneamente.

4 Descrizione del problema

Per descrivere chiaramente il problema affrontato, in questa sezione vengono presentati il modello di workflow, il modello cloud e la formulazione matematica del problema. I principali simboli utilizzati sono riportati nella Tabella 2.

4.1 Modello di workflow

Un modello di workflow viene solitamente rappresentato tramite un grafo aciclico diretto (DAG) [41–44]. Un esempio del modello di workflow studiato è illustrato nella Figura 1. In tale figura, il DAG è definito come DAG = (T, E), dove:

- $T = \{t_0, t_1, \ldots, t_{n1}\}$ è l'insieme dei n task (compiti) del workflow;
- E rappresenta le dipendenze tra i task.

Per ogni task t_i, si definiscono:

- pre(t_i): l'insieme dei predecessori, ovvero i task che devono essere completati prima di t_i;
- succ(t_i): l'insieme dei successori, ovvero i task che dipendono da t_i.

Un task che non ha predecessori viene definito task di ingresso (t_{entry}), mentre un task che non ha successori viene definito task di uscita (t_{exit}). È importante notare che un DAG può avere più di un task di ingresso o di uscita.

Un task può iniziare la propria esecuzione solo quando tutti i suoi predecessori sono completati e i dati in uscita da questi ultimi sono stati ricevuti.

La dimensione della trasmissione dei dati tra i task è rappresentata da una matrice chiamata TT, definita come segue:

MATRICE, TABELLA 2, FIG.1

dove:

- $TT_{i,j} = 0$ se non esiste una dipendenza tra i task t_i e t_j, oppure se i = j;
- l'elemento $TT_{i,j}$, con $0 \leq i \leq n-1 e 0 \leq j \leq n-1$, rappresenta la quantità di dati trasmessi da t_i a t_j.

4.2 Modello Cloud

Si suppone che l'infrastruttura sia un sistema cloud di tipo *Infrastructure as a Service (IaaS)*, composto da un insieme di m macchine virtuali (VM), indicato come:

$$V = \{VM_0, VM_1, \ldots, VM_{m1}\}$$

Tutte le VM sono indipendenti tra loro e possiedono diverse capacità di elaborazione.

Sia B una matrice che rappresenta la larghezza di banda tra le diverse VM. È importante notare che questa matrice non è necessariamente simmetrica, ovvero:

$$B(VM_k, VM_l) \neq B(VM_l, VM_k)$$

Inoltre, per ogni VM:

$$B(VM_k, VM_k) = 0$$

poiché non esiste trasmissione tra VM identiche.

Il tempo di trasmissione tra due task t_i e t_j, indicato con $T_{trans}(t_i, t_j)$, quando t_i è eseguito su VM_k e t_j su VM_l, dipende dalla quantità di dati da trasmettere ($TT_{i,j}$) e dalla larghezza di banda tra VM_k e VM_l.

Se i due task sono eseguiti sulla stessa VM, allora:

$$T_{trans}(t_i, t_j) = 0$$

In particolare, il tempo di trasmissione si calcola con la seguente formula:

$$T_{trans}(t_i, t_j) = \frac{TT_{i,j}}{B(VM_k, VM_l)}2 \tag{1}$$

Il tempo di inizio dell'esecuzione del task t_i sulla VM VM_k, indicato con $ST(t_i, VM_k)$, è calcolato come segue:

$$ST(t_i, VM_k) = \{\, 0 \; set_i = t_{entry} \max\{FT_j + T_{trans}(t_i, t_j)\} set_i \neq t_{entry},\; t_j \in pre(t_i)3 \tag{2}$$

Il tempo di esecuzione del task t_i sulla VM VM_k, indicato con $ET(t_i, VM_k)$, è definito come il rapporto tra la dimensione del task (s_i) e la capacità di elaborazione della macchina p_k:

$$ET(t_i, VM_k) = \frac{s_i}{p_k}4 \tag{3}$$

Il tempo di completamento del task t_i sulla VM VM_k, indicato con $FT(t_i, VM_k)$, è dato dalla somma tra il tempo di inizio e il tempo di esecuzione:

$$FT(t_i, VM_k) = ST(t_i, VM_k) + ET(t_i, VM_k)5 \tag{4}$$

4.3 Formulazione del problema

Il *makespan* di un workflow, utilizzato per valutare le prestazioni dell'algoritmo proposto, corrisponde al **tempo massimo di completamento** tra tutti i task nel workflow [**?**, **?**, **?**, **?**]. Indichiamo con $MS(VM_k)$ il makespan della macchina virtuale VM_k, per $0 \leq k \leq m-1$. Allora, il makespan del workflow è definito come:

$$makespan = \max(MS(VM_k)), \quad 0 \leq k \leq m-1$$

(5)

Per valutare in modo più intuitivo le prestazioni degli algoritmi, si definiscono le seguenti metriche derivate dal makespan:

1. Rapporto di lunghezza della schedulazione (SLR – Scheduling Length Ratio) Per evitare differenze eccessive nel makespan dovute a parametri differenti, è necessario normalizzare il makespan rispetto a un limite inferiore. Questa metrica è il rapporto tra il makespan del workflow e la somma dei tempi minimi di esecuzione dei task nel percorso critico (CP – Critical Path):

$$SLR = \frac{makespan}{\sum_{i \in CP} \min_{0 \leq k \leq m-1} ET(t_i, VM_k)} \tag{6}$$

2. Utilizzo medio delle VM (AVU – Average VM Utilization) È definito come il rapporto tra il tempo totale di esecuzione dei task su una VM e il makespan del workflow. L'utilizzo della VM VM_k è dato da:

$$VU(VM_k) = \frac{\sum_{i \in VM_k.waiting} ET(t_i, VM_k)}{makespan} \tag{7}$$

   L'utilizzo medio di tutte le VM è calcolato come:

$$AVU = \frac{1}{m} \sum_{k=0}^{m-1} VU(VM_k) \tag{8}$$

3. Varianza dell'equità (VF – Variance of Fairness) Questa è una nuova metrica introdotta per misurare l'equità tra i task. Si basa sul concetto di *soddisfazione* di ciascun task, calcolata come il rapporto tra il tempo di esecuzione effettivo (AET) e il tempo di esecuzione atteso (EET), cioè quello del task sulla VM più veloce.

$$S_i = \frac{AET_i}{EET_i} \tag{9}$$

Dove EET_i è il tempo di esecuzione di t_i sulla VM più veloce. Un valore S_i più alto indica una maggiore soddisfazione del task.

La varianza dell'equità (VF) misura la deviazione della soddisfazione di ciascun task rispetto alla media M, ed è definita come:

$$VF = \frac{1}{n} \sum_{i=0}^{n-1} (M - S_i)^2 \tag{10}$$

Dove M è la media di tutte le soddisfazioni S_i. Più basso è il valore di VF, più equa è la schedulazione tra i task.

5 Algoritmo proposto

L'algoritmo proposto, denominato SM-CPTD, si basa sulla teoria dei giochi di abbinamento stabile (SMGT – Stable Matching Game Theory). SM-CPTD è progettato per coordinare in modo efficace sia gli obiettivi dei singoli task che quelli globali del sistema, bilanciando il carico tra le VM e ottimizzando l'utilizzo delle risorse.

Inoltre, vengono presentati i metodi per la selezione e la generazione delle liste di preferenza. Per migliorare la qualità della soluzione, l'algoritmo integra anche due tecniche di ottimizzazione locale, basate rispettivamente sul percorso critico e sulla duplicazione dei task.

L'algoritmo si articola in tre fasi principali, illustrate nell'Algoritmo 1:

- Determinazione del percorso critico (DCP – Determine Critical Path): serve a identificare il percorso critico del grafo aciclico diretto (DAG) del workflow.
- Applicazione della SMGT: genera un piano di pre-schedulazione dei task.
- Ottimizzazione locale tramite duplicazione dei task (LOTD – Local Optimization based on Task Duplication): migliora il piano generato nella fase precedente.

ALGORITMO1

5.1 Ottimizzazione locale basata sul percorso critico

Il metodo utilizzato in HEFT [**?**] viene impiegato per calcolare il *percorso critico* di un DAG. In questo approccio, basato sul tempo medio di calcolo e di comunicazione, a ciascun task viene assegnato un *rank* durante l'attraversamento completo del DAG, che avviene dai task di uscita verso quelli di ingresso.

Il **rank** di un task t_i, che rappresenta la lunghezza del percorso più lungo da t_i al task di uscita, è definito ricorsivamente come segue:

$$rank(t_i) = W_i + \max_{t_j \in succ(t_i)} (c_{i,j} + rank(t_j))12 \tag{11}$$

dove:

- W_i indica il tempo medio di esecuzione (computazione) del task t_i;
- $c_{i,j}$ rappresenta il tempo medio di comunicazione dell'arco (t_i, t_j);
- $succ(t_i)$ è l'insieme dei successori di t_i.

Per i task di uscita si ha:

$$rank(t_{exit}) = W_{exit}$$

In ciascun livello del DAG, il task con il rank più alto viene selezionato per costruire il **percorso critico**. Successivamente, i task appartenenti al percorso critico vengono pianificati sulla VM che consente il completamento più precoce possibile. Infatti, ridurre il tempo totale di esecuzione dei job nel percorso critico permette di ridurre efficacemente il *makespan* di un DAG [**?**, **?**].

Il procedimento per determinare il percorso critico è indicato come **DCP (Determine Critical Path)** ed è descritto come segue.

ALGORITMO2

5.2 Teoria dei giochi di abbinamento stabile (SMGT)

La *Stable Matching Game Theory* (SMGT) è una delle teorie più recenti dell'economia moderna, capace di integrare in modo efficace **equità ed efficienza** nell'allocazione delle risorse pubbliche. Inoltre, è stata applicata con successo a numerosi problemi, come il problema matrimoniale, la scelta scolastica degli studenti e altri ancora. La SMGT ha mostrato vantaggi significativi nell'aiutare i partecipanti a generare un ordine di preferenza rigido a basso costo.

Nel contesto della pianificazione dei workflow, un modello di abbinamento stabile è composto da due insiemi di partecipanti:

- L'insieme T dei task,
- L'insieme V delle macchine virtuali (VM).

Ogni task $t_i \in T$ possiede una **matrice di preferenza** rispetto a tutte le VM, e ogni VM $VM_k \in V$ ha a sua volta una matrice di preferenza sui task. La matrice di preferenza di un task viene generata ordinando le VM in base al **tempo di completamento crescente** del task su ciascuna di esse. Analogamente, la matrice di preferenza di una VM viene costruita ordinando i task in base al loro tempo di completamento su quella VM.

L'obiettivo è trovare un **abbinamento stabile** che consenta a ciascun elemento di ottenere il risultato più desiderato possibile.

A causa delle peculiarità del problema di pianificazione dei workflow, in questo lavoro viene adottato un **metodo di abbinamento gerarchico**:

1. I task appartenenti al livello l vengono assegnati alla VM più preferita secondo la loro matrice di preferenza.

2. Per evitare che un numero eccessivo di task venga assegnato alla VM più veloce, si imposta una **soglia** per ciascuna VM.

   In particolare, un task può essere assegnato alla VM VM_k solo se:

   $$threshold(k, l) > 0$$

   Altrimenti, il task con il valore di preferenza più alto nella lista d'attesa della VM_k viene eliminato e riassegnato.

3. Il task rimosso seleziona un'altra VM secondo la propria matrice di preferenza.

4. I passi precedenti vengono ripetuti finché tutti i task del livello l non sono assegnati.

La soglia $threshold(k, l)$ è calcolata in base al numero di task nel livello l del DAG (n_l) e alla capacità di elaborazione di ciascuna VM. In questo modo, una VM più potente potrà gestire un numero maggiore di task rispetto a una VM meno performante.

La soglia per la VM VM_k nel livello l è definita come:

$$threshold(k, l) = \frac{\sum_{v=0}^{l-1} n_v}{\sum_{i=0}^{m-1} p_i} \times p_k \tag{12}$$

dove n_l indica il numero di task nel livello l, e p_k è la capacità di elaborazione della VM k.

Poiché l'ordine di assegnazione dei task ha un impatto trascurabile sull'esito dell'abbinamento stabile, l'algoritmo proposto **non considera la priorità dei task**.

Inoltre, una volta che un task t_i è stato assegnato, si aggiornano:

- $ACT(t_i)$: il tempo di attivazione,
- $AFT(t_i)$: il tempo di completamento,

e vengono riassegnati i valori di preferenza per i task restanti.

L'algoritmo SMGT è descritto come segue.

ALGORITMO3

5.3 Ottimizzazione locale basata sulla duplicazione dei task

L'essenza della duplicazione dei task consiste nell'utilizzare il **tempo di inattività** delle VM per ridurre il *makespan*. In pratica, si sfruttano task ridondanti per ridurre il tempo di comunicazione tra task.

Per evitare un eccessivo spreco di risorse, si considera la duplicazione **solo per i task appartenenti al primo livello** del DAG.

In particolare, un task t_i può essere duplicato e assegnato a un'altra VM solo se sono soddisfatte entrambe le seguenti condizioni:

1. t_i può essere duplicato solo su **VM che ospitano i suoi successori**, cioè su quelle VM a cui sono già stati assegnati i task successivi a t_i;
2. Il **tempo di completamento** degli altri task presenti nella lista d'attesa della VM selezionata **non deve aumentare** a seguito della duplicazione di t_i.

L'algoritmo per l'Ottimizzazione Locale basata sulla Duplicazione dei Task è denominato **LOTD** (Local Optimization based on Task Duplication) ed è descritto nell'Algoritmo 4.

ALGORITMO4

5.4 Caso di studio

Per illustrare in modo più chiaro il funzionamento dell'algoritmo proposto, viene presentato un esempio basato sul workflow *CyberShake*, come mostrato nella Figura 1.

Nella Figura 1, il workflow è composto da 30 task, indicati con:

$$T = \{t_0, t_1, t_2, \ldots, t_{29}\}$$

distribuiti su quattro livelli, con un totale di 52 archi che rappresentano le dipendenze tra i task.

Le dimensioni (size) dei task sono riportate nella Tabella 3, dove il **rank** di ciascun task è calcolato secondo l'Equazione (8). I valori di rank evidenziati in grassetto nella Tabella 3 rappresentano i task con il rank massimo per ciascun livello.

La matrice TT, che rappresenta la quantità di dati trasmessi tra i task, è calcolata come segue:

$$TT_{i,j} = s_{t_i} \times CCR$$

dove $CCR = 0.4$, e rappresenta il rapporto tra il costo medio di comunicazione e il costo medio di computazione.

Le capacità di elaborazione delle cinque VM sono, rispettivamente:

$$5,\ 8,\ 7,\ 9,\ 6$$

Le larghezze di banda tra le VM sono riportate nella Tabella 4.

Secondo quanto riportato nella Tabella 3, il **percorso critico** risulta essere:

$$\{t_2,\ t_5,\ t_6,\ t_0\}$$

Due casi specifici vengono analizzati, relativi al **livello 0** e al **livello 1**, e sono illustrati nelle Tabelle 5 e 6. Il task che necessita di essere riassegnato è indicato in grassetto nella Tabella 6.

Poiché nei task del livello 0 non esiste trasmissione di dati da predecessori, l'unico criterio di ordinamento per la matrice di preferenza è il **tempo di completamento**, pertanto l'array di preferenza dei task risulta identico per tutte le VM.

Il **diagramma di Gantt** della soluzione ottenuta tramite l'algoritmo proposto SM-CPTD è riportato nella Tabella 7.

Nella Tabella 7:

- Il simbolo * rappresenta uno **slot di inattività** (idle time) della VM;
- Il simbolo $\sim$ rappresenta la **durata di un task assegnato** nella soluzione;
- I **task duplicati** sono evidenziati in **grassetto**.

TABELLA3

5.5 Analisi della complessità

In questa sezione viene presentata l'analisi della complessità computazionale dell'algoritmo **SM-CPTD**. L'algoritmo è composto da tre procedure principali:

- determinazione del percorso critico,
- abbinamento stabile basato sulla teoria dei giochi,
- duplicazione dei task.

Siano:

- n: il numero totale di task,
- l: il numero totale di livelli nel workflow,
- m: il numero di VM attive.

Algoritmo 2 – Determinazione del percorso critico: La complessità temporale per calcolare il valore di `rank` di ciascun task e determinare il percorso critico è:

$$\mathcal{O}(n^2)$$

Algoritmo 3 – Stable Matching Game Theory: Il ciclo che va dal passo 2 al passo 35 viene eseguito l volte. Il passo 3 richiede tempo $\mathcal{O}(n)$. Nel caso ideale, in cui ogni task venga assegnato direttamente alla VM migliore, il ciclo dal passo 9 al passo 34 ha complessità:

$$\mathcal{O}(nm)$$

Tuttavia, nel **caso peggiore**, ogni task abbinato può essere successivamente rifiutato, provocando fino a n nuovi abbinamenti per task. In tal caso, la complessità complessiva può raggiungere:

$$\mathcal{O}(n^2)$$

Algoritmo 4 – Duplicazione dei task (LOTD): Ha complessità:

$$\mathcal{O}(nm)$$

Complessità complessiva: Poiché l'intero processo viene ripetuto per ogni livello l, la complessità totale dell'algoritmo SM-CPTD è:

$$\mathcal{O}(n^2 l) \tag{13}$$

6 Simulazione e analisi

In questa sezione vengono condotti esperimenti approfonditi per valutare le prestazioni dell'algoritmo **SM-CPTD** rispetto ad altri quattro algoritmi di riferimento, e per verificare l'efficacia delle strategie proposte.

UN MONTE DI TABELLE

6.1 Impostazioni sperimentali

L'algoritmo proposto è stato testato su quattro workflow di riferimento (benchmark), mostrati nella Figura 2, ovvero:

- **Montage**,
- **CyberShake**,
- **LIGO**,
- **Epigenomics** [**?**].

I parametri utilizzati in questo articolo e i relativi intervalli di valori sono riportati nella Tabella 8, come indicato anche in [**?**]. Le dimensioni dei task, le capacità di elaborazione delle VM e le larghezze di banda tra VM sono generate in modo casuale tramite una **distribuzione uniforme**.

L'intervallo del parametro CCR riflette la transizione dei workflow da *computation-intensive* a *data-intensive*, al crescere del valore di CCR.

Tutti gli algoritmi sono stati implementati e testati utilizzando:

- **Ambiente di sviluppo**: Eclipse R2019, versione 4.13.0;
- **Processore**: Intel Core i7-9750H CPU @ 2.60GHz;
- **RAM**: 8 GB;
- **Sistema operativo**: Microsoft Windows 10 Professional, 64-bit.

FIG2 TAB8

6.2 Risultati sperimentali

I risultati sperimentali vengono discussi sotto due punti di vista principali:

1. **Valutare le prestazioni dell'algoritmo SM-CPTD**,
2. **Verificare l'efficacia delle strategie proposte**.

I risultati dettagliati degli esperimenti sono riportati nelle Figure **??**–**??**. Per ridurre l'impatto di eventuali incertezze sperimentali, ogni algoritmo è stato eseguito **10 volte** su ciascuna istanza, e i risultati presentati corrispondono alla **media** delle esecuzioni.

UN MONTE DI GRAFICI

6.2.1 Verifica dell'algoritmo SM-CPTD

Per valutare in modo completo le prestazioni dell'algoritmo **SM-CPTD**, sono stati selezionati quattro algoritmi comparativi: **TDA** [**?**], **GSS** [**?**], **NMMWS** [**?**] e **min–min** [**?**]. Le rispettive complessità temporali sono:

- TDA: $\mathcal{O}(n^3)$,
- GSS: $\mathcal{O}(n^2 m)$,
- NMMWS: $\mathcal{O}(n^3 ml)$,
- min–min: $\mathcal{O}(n^2 m)$.

Chiaramente, l'algoritmo proposto SM-CPTD, con complessità $\mathcal{O}(n^2 l)$, ha una complessità simile a quella di *min–min*, che è generalmente considerato uno degli algoritmi di scheduling più efficienti come riferimento.

Per analizzare l'impatto del parametro CCR e del numero di VM m sulle prestazioni degli algoritmi, vengono adottate le seguenti impostazioni:

- Per workflow di dimensioni piccole, medie e grandi, il numero di VM è impostato rispettivamente a 5, 10 e 50 quando si considera l'influenza di CCR.
- Quando si analizza l'impatto di m, il valore di CCR è fisso a 1.
- Per studiare la metrica VF nei diversi workflow, si imposta $m = 50$ e $CCR = 1$.

I risultati della valutazione sono riportati nelle Figure **??**–**??**.

Dalle Figure **??**, **??** e **??**, si osserva chiaramente che i valori di **SLR** ottenuti da tutti gli algoritmi aumentano all'aumentare del valore di CCR da 0,4 a 2, in quanto cresce il tempo di trasmissione dei dati tra i task. Di conseguenza, aumenta anche il tempo di completamento dei workflow.

Tuttavia, **SM-CPTD** ottiene in media:

- un miglioramento del **20,93%** rispetto a TDA,
- **2,66%** rispetto a GSS,
- **5,25%** rispetto a NMMWS,
- **11,55%** rispetto a min–min.

Ciò significa che SM-CPTD ottiene i **valori minimi di SLR**, corrispondenti al **makespan minimo**, su istanze di workflow di diverse dimensioni rispetto a tutti gli algoritmi confrontati.

Inoltre, le prestazioni di GSS risultano comparabili con quelle di NMMWS, e sia questi ultimi che min–min superano TDA nella maggior parte dei casi.

In TDA, un grande numero di task viene replicato per ridurre il tempo di comunicazione, il che può causare ridondanza nei dati e un aumento del tempo di esecuzione. Per questo motivo, TDA si comporta meglio nei workflow *data-intensive* rispetto a quelli *computation-intensive*. Le prestazioni di NMMWS dipendono dalla dimensione dei task e dalla capacità computazionale delle VM; in particolare, NMMWS ha prestazioni peggiori nel caso di task di piccole dimensioni, in quanto fatica a ottenere risultati di batching ottimali.

Dalle Figure **??**, **??** e **??**, si nota che i valori di **AVU** di tutti gli algoritmi diminuiscono con l'aumento di CCR, a causa del maggiore tempo necessario per la trasmissione dei dati. Tuttavia, diversamente dai risultati relativi a SLR, TDA ottiene i **migliori valori di AVU**, poiché i numerosi task replicati possono essere inseriti negli slot inattivi delle VM, migliorandone così l'utilizzo.

Anche se SM-CPTD ha valori di AVU inferiori del **9,42%** rispetto a TDA, mostra comunque miglioramenti pari a:

- **4,07%** rispetto a GSS,
- **6,56%** rispetto a NMMWS,
- **12,89%** rispetto a min–min.

Dalle Figure **??** e **??**, si evidenzia che, per workflow su larga scala, **SM-CPTD ottiene il valore minimo di SLR** indipendentemente dal numero di VM. Con l'aumento di m, aumenta la capacità di elaborazione parallela, che riduce il tempo di completamento, ma diminuisce l'utilizzo delle VM.

Anche se SM-CPTD ha un'AVU approssimativamente **5,26%** inferiore rispetto a TDA, ottiene comunque:

- **2,98%** in più rispetto a GSS,
- **6,12%** in più rispetto a NMMWS,
- **11,69%** in più rispetto a min–min.

Infine, il confronto tra i valori di **VF** nei diversi workflow su larga scala, mostrato nella Figura **??**, evidenzia che l'algoritmo SM-CPTD presenta valori di **equitŏ0e0 migliori** in tutti i workflow rispetto agli altri algoritmi confrontati.
UN MONTE DI GRAFICI

6.2.2 Verifica dei metodi proposti

Per verificare l'efficacia delle due tecniche di ottimizzazione locale presentate in questo articolo, sono stati adottati tre algoritmi comparativi:

- **SM**: algoritmo di abbinamento stabile *senza* ottimizzazione locale;
- **SM-CP**: algoritmo SM con ottimizzazione locale basata sul *percorso critico*;
- **SM-TD**: algoritmo SM con ottimizzazione locale basata sulla *duplicazione dei task*.

Senza perdita di generalità, gli algoritmi sono stati testati su workflow di grande scala, impostando $m = 50$ e $CCR = 1$.
I risultati della valutazione sono mostrati nella Figura **??**.

Dalla Figura **??**a, si osserva che **entrambe le tecniche di ottimizzazione locale riducono il makespan** del workflow, con **SM-TD** che risulta più efficace di **SM-CP**. Inoltre, applicando entrambe le tecniche, le prestazioni dell'algoritmo migliorano dell'**11,28%** rispetto alla versione base.
Dalla Figura **??**b si evince che, poiché l'**AVU** è legato al makespan, al diminuire del makespan l'AVU tende ad aumentare. Pertanto, rispetto a SM, l'algoritmo **SM-CP** ottiene risultati migliori in termini di AVU. Inoltre, **SM-TD** sfrutta meglio i periodi di inattività delle VM, ottenendo valori di AVU superiori rispetto a SM-CP. Applicando entrambe le tecniche, **SM-CPTD** raggiunge il miglior tasso di utilizzo con un incremento di circa **7,69%**.
Dalla Figura **??**c si nota che **SM** ottiene il valore minimo di **VF** tra tutti i workflow analizzati. Ciò indica che l'algoritmo originale di abbinamento stabile è efficace nel bilanciare l'equità tra i task. Tuttavia, le due ottimizzazioni locali influenzano la soddisfazione di alcuni task, con conseguente aumento del valore di VF negli altri tre algoritmi rispetto a SM.

In conclusione, l'algoritmo proposto **SM-CPTD** mostra prestazioni superiori rispetto a tutti gli altri algoritmi considerati.
Le due tecniche di ottimizzazione locale, **DCP** e **LOTD**, risultano efficaci nel ridurre il makespan del workflow.
Inoltre, quando il valore di CCR è impostato a 1, i tempi medi di esecuzione dell'algoritmo SM-CPTD per workflow di diversa scala e struttura sono i seguenti:

- **Small-scale**: 10–20 ms,
- **Medium-scale**: 30–40 ms,

- **Large-scale**: 1200–1400 ms.

Pertanto, **SM-CPTD è altamente efficiente** e adatto ad essere applicato in scenari di scheduling di workflow **online**.

7 Conclusioni

Considerando l'importanza dell'equità tra i task nelle applicazioni reali di workflow, in questo lavoro è stato proposto un nuovo algoritmo di schedulazione basato sulla **teoria dei giochi di abbinamento stabile**, denominato **SM-CPTD**, con l'obiettivo di **minimizzare il makespan** del workflow e **massimizzare l'equità tra i task**.

Inoltre, sono state introdotte due tecniche di ottimizzazione locale:

- una basata sul **percorso critico**,
- l'altra sulla **duplicazione dei task**,

entrambe incorporate in SM-CPTD per ridurre il tempo di completamento dei task critici e sfruttare appieno le VM disponibili.

I risultati sperimentali, valutati tramite le metriche **SLR**, **AVU** e **VF**, hanno dimostrato che l'algoritmo proposto è in grado di:

- ridurre efficacemente il makespan del workflow,
- migliorare l'utilizzo delle risorse,
- garantire equità nell'assegnazione dei task.

Per sviluppi futuri, sarà interessante:

- Estendere l'algoritmo alla gestione di **obiettivi multipli**, come il costo e il consumo energetico;
- Studiare la schedulazione dei workflow in ambienti di **mobile edge computing**, dove i task devono essere offloadati verso server edge o cloud prima dell'elaborazione.