Video Streaming

Marcello Esposito

7 giugno 2025

1 Introduzione

In questo paper verrà mostrato come è stato affrontato il problema Video Streaming del torneo Google Hashing Problem del 2017. Partendo dalla formulazione matematica, saranno poi descritte le euristiche utilizzate quando le istanze diventano troppo grandi.

2 Descrizione del problema

Il problema cerca di affrontare la distribuzione dei video (YouTube) di un datacenter in diversi possibili cache server in maniera ottima rispettando la capacità massima di quest'ultimi.
Per allocazione ottima si intende l'allocazione dei video nei cache server che permette di massimizzare il savings in termini latenza, questo perché ogni endpoint può effettuare un numero variabile di richieste per video differenti ed essere collegato solo a specifici cache server, dove ognuno di questi collegamenti ha una latenza differente.
Quindi oltre alla distribuzione dei video sarà necessario anche selezionare da quale server un endpoint dovrà richiedere i video desiderati.

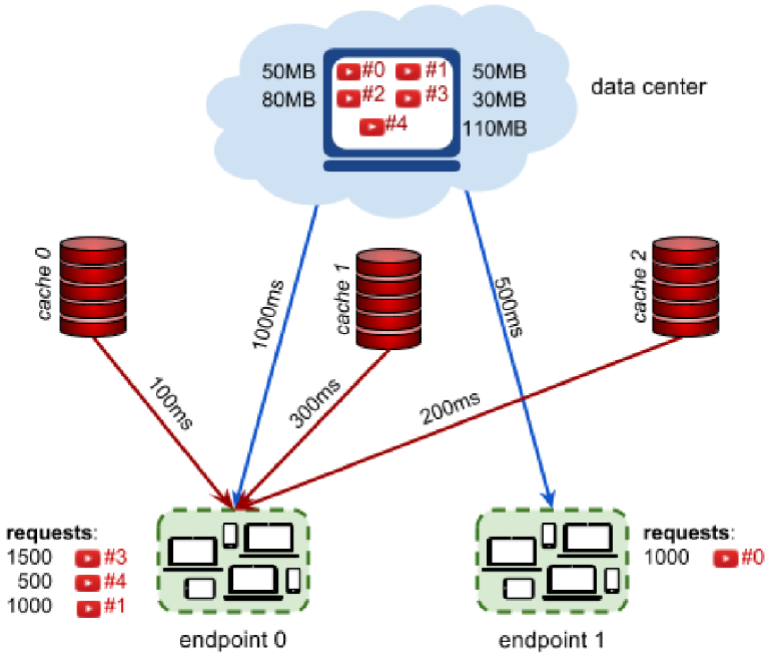

Figura 1: rappresentazione a grafo

2.1 dataset

Per il problema sono stati forniti diversi dataset di piccole e grandi dimensioni, mentre alcuni sono stati creati manualmente per controllare la correttezza delle soluzioni manualmente, così che in alcuni

```
5 2 4 3 100
50 50 80 30 110
1000 3
0 100
2 200
1 300
500 0
3 0 1500
0 1 1000
4 0 500
1 0 1000
```

```
5 videos, 2 endpoints, 4 request descriptions, 3 caches 100MB each.
Videos 0, 1, 2, 3, 4 have sizes 50MB, 50MB, 80MB, 30MB, 110MB.
Endpoint 0 has 1000ms datacenter latency and is connected to 3 caches:
The latency (of endpoint 0) to cache 0 is 100ms.
The latency (of endpoint 0) to cache 2 is 200ms.
The latency (of endpoint 0) to cache 1 is 300ms.
Endpoint 1 has 500ms datacenter latency and is not connected to a cache.
1500 requests for video 3 coming from endpoint 0.
1000 requests for video 0 coming from endpoint 1.
500 requests for video 4 coming from endpoint 0.
1000 requests for video 1 coming from endpoint 0.
```

Figura 2: esempio struttura dataset

casi è stato anche possibile risolvere il problema con Gurobi.
La struttura dei dataset è lo stesso del dataset di esempio in figura 2.
Il caricamento dei dati è stato gestito nel seguente modo:

- il numero dei video, endpoint, descrizioni di richiest, di server cache e la capacità di questi ultimi sono semplici variabili
  (il numero dei server è: numero cache + 1 per modellare il datacenter come un generico server)
- la dimensione dei video è stata inserita in una lista *video_size[index del video] = size*
- le latenze da ogni endpoint ad ogni server sono state gestite con una matrice bidimensionale *latency[endpoint][cache/datacenter] = latency*
- le richieste di ogni endpoint per ogni video sono state gestite con una matrice bidimensionale *reqs[endpoint][video] = num reqs*

3 Formulazione Matematica

Dal punto di vista concettuale, volendo trovare un riscontro con uno dei problemi affrontati durante il corso, il problema è 'assimilabile' ad un problema di simple plant location, dove i centri di distribuzione da posizionare sono i video, le possibili posizioni sono i cache server in cui possono essere posizionati più centri di distribuzione entro un certo limite (il datacenter ha sempre tutti i video), mentre i collegamenti tra server e endpoint per richiedere un video sono gestibili tramite variabili binarie non essendoci limitazioni sulla capacità dei collegamenti.

3.1 variabili decisionali

$\boldsymbol{x_{esv}} \in \boldsymbol{\{0,1\}}$indica se l'endpoint e richiede al server s il video v
$\boldsymbol{y_{sv}} \in \boldsymbol{\{0,1\}}$ indica se sul server s è stato posizionato il video v

3.2 modello

Una nota molto importante da fare è che il problema di default richiedeva di massimizzare il delay risparmiato in totale dagli endpoint richiedendo i video ai cache server invece che al datacenter quando possibile, in questo caso invece è stata utilizzata una funzione obiettivo analoga più semplice, ovvero **minimizzare la latenza totale delle richieste**.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e=1}^{E}\sum_{s=1}^{S}\sum_{v=1}^{V} reqs_{ev} \cdot lat_{es} \cdot x_{esv} \\
\text{subject to} \quad & x_{esv}, y_{sv} \in 0,1 && \forall e,s,v, \\
& \sum_{e=1}^{E} x_{esv} \leq E \cdot y_{s,v} && \forall s,v \mid \text{video must be placed on cache to satisfy request}, \\
& \sum_{v=1}^{V} x_{esv} \leq V \cdot lat_{s,v} && \forall e,s \mid \text{endpoint can use only existing connection}, \\
& \sum_{s=1}^{S} x_{esv} = 1 && \forall e,v \mid reqs_{ev} = 1 \mid \text{every request must be satisfied}, \\
& \sum_{s=1}^{S} x_{esv} = 0 && \forall e,v \mid reqs_{ev} = 0 \mid \text{analogue (could be removed)}, \\
& \sum_{v=1}^{V} d_v \cdot y_{sv} \leq C && \forall s - datacenter \mid \text{cache server capacity}, \\
& y_{datacenter,v} = 1 && \forall v \mid \text{datacenter keep videos}
\end{aligned}
\tag{1}
$$

I risultati mostrati sono stati ottenuti sul dataset *me_at_the_zoo.in*, che ha una dimensione tale da essere risolvibile con il presolutore di Gurobi ma non troppo piccolo da essere banale, in modo da poter valutare più accuratamente il GAP delle soluzioni euristiche.
In particolare la **soluzione ottima** ottenuta da Gurobi è di un **delay totale** di **4292938** ms

4 Euristiche

Come metodi euristici sono stati utilizzati inizialmente degli algoritmi greedy per ottenere delle soluzioni di partenza da utilizzare successivamente per algoritmi euristici più avanzati come tabù search e genetico.
Anche in questo caso per modellare il problema sono state utilizzate le stesse variabili decisioni del modello matematico, implementandole utilizzando la libreria numpy data la sua efficienza nel gestire matrici e di semplificazione delle operazioni su quest'ultime.

4.1 Greedy

In particolare sono state realizzati 5 algoritmi greedy principalmente basati sul numero totale di richieste dei video, **la differenza sarà invece nel modo in cui questi ultimi vengono posizionati nei server cache**.

1. Itera in ogni endpoint e per ognuno di essi posiziona i video da esso richiesti, ordinati per numero di richieste, nel miglior cache server disponibile
2. Itera in ogni endpoint con uno schema round robin (cambiando endpoint ad ogni video posizionato) e posiziona il suo video più richiesto rimasto nel miglior cache server disponibile
3. Itera in ogni video in base al numero di richieste e posizionalo nel server migliore disponibile per l'endpoint che lo richiede maggiormente
4. Itera ogni video in base al numero di richieste e posizionale nel server con il maggior numero di endpoint che richiedono quest'ultimo
5. Itera ogni server e posiziona in esso i video più richiesti dagli endpoint a lui connessi

4.1.1 risultati

Come si può osservare dalla tabella 1, gli algoritmi che hanno funzionato meglio sono quelli che hanno un **focus maggiore sulla ricerca dell'ottimo locale** (i primi 3) **senza effettuare considerazioni più 'globali'**, come ad esempio avviene nell'algoritmo 4 in cui posizioniamo i video in base al numero di endpoint connessi ad un determinato server che li richiedono.

Algoritmo	Delay	GAP
Gurobi	4292938	0
1	6347627	47.9%
2	7091338	65.2%
3	6776093	57.4%
4	9816241	128.7%
5	9519517	121.7%

Tabella 1: Risultati algoritmi greedy

4.2 Tabù Search

Partendo dalle soluzioni generate dagli algoritmi greedy, sono state implementate due varianti di tabù search in cui **la differenza è nella mossa effettuata**, così da diversificare gli intorni esplorabili, in particolare le mosse sono:

1. toggle di un singolo video in un singolo server
2. aggiunta di un singolo video in un singolo server

In entrambi i casi sono stati utilizzati il **criterio di aspirazione e di intensificazione randomizzati**, così da cercare di esplorare meglio gli intorni delle soluzioni migliori.

4.2.1 risultati

Come si può osservare dalla tabella 2, la semplicità delle mosse utilizzate ha fatto sì che per diverse soluzioni di partenza non si sia riusciti ad ottenere dei miglioramenti significativi, eccezione fatta per il risultato con soluzione di partenza dell'algoritmo 4.

Soluzione di partenza	Tabu Toggle	Tabu Add
1 - 6347627	n.d.	n.d.
2 - 7091338	n.d.	n.d.
3 - 6776093	n.d.	n.d.
4 - 9816241 - 128.7%	7161592 - 66.8%	7275848 - 69.5%
5 - 9519517 - 121.7%	9013457 - 109.9%	n.d.

Tabella 2: Risultati algoritmi tabù search

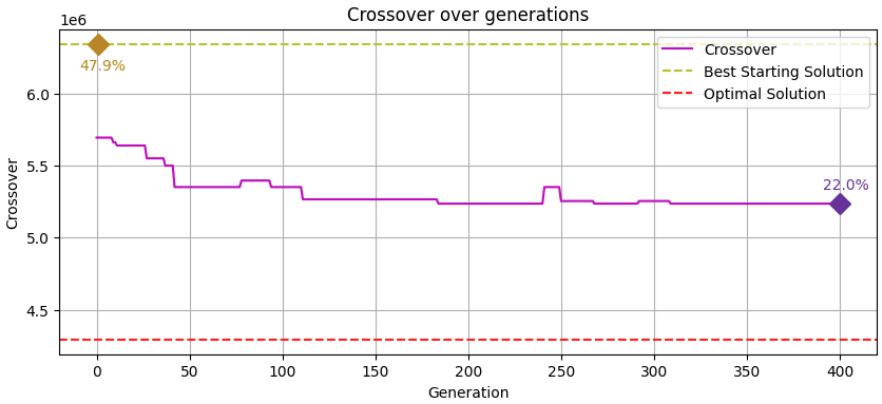

Figura 3: performance algoritmo genetico

4.3 Genetic

Una volta ottenuto un insieme di **soluzioni di partenza dagli algoritmi in precedenza**, sono tutte state utilizzate come **popolazione di partenza** a un algoritmo genetico con le seguenti caratteristiche:

- Fitness: 1 / delay
- Corredo genetico: configurazione dei server cache
- Popolazione iniziale: soluzioni euristiche precedenti
- Randomizzazione: simulazione di Montecarlo
- Codifica del problema: mosse eseguite direttamente sulla matrice y
- Selezione: sono selezionati 2 genitori randomicamente
- Crossover: viene presa la configurazione dei server cache da 0 al punto di split dal genitore 1, i restanti server cache dal genitore 2 ed infine si ricalcola l'assegnamento ottimale di x
- Mutazione: flip su presenza o meno di un video in un server (bit flip su y come in tabù search)
- Sostituzione: ad ogni nuova generazione, viene eliminato il 10% della popolazione randomicamente

4.3.1 risultati

Come mostrato dal grafico in figura 3, questo algoritmo permette di migliorare notevolmente il fitness già nelle sue prime generazioni ed anche come la sostituzione randomica possa portare anche ad eliminare le soluzioni migliori, ma ciò a cui si dovrebbe fare maggior attenzione è in realtà la tendenza a **convergere** un determinato valore di fitness **senza più ottenere miglioramenti** dopo un certo numero di generazioni.
A discapito di quest'ultima osservazione però, è possibile osservare come la riduzione del GAP sia netta, passando da un miglior **GAP iniziale di circa il 50%** ad un GAP che si attesta tra il [**20%, 25%**], osservando i risultati di più reiterazioni dell'algoritmo.

Credits

Esposito Marcello M63001768
GitHub Repository