UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

MACROAREA DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Tesi di laurea magistrale

TITOLO

2023-2024

RELATORE:
Prof.essa Valeria Cardellini

RELATORE:
Dr. Gabriele Russo Russo

CANDIDATO:
Simone Nicosanti

MATRICOLA:
0334319

Indice

Capitolo 2

Problema di Ottimizzazione

Manca secondo me una breve descrizione informale del problema. L'obiettivo è minimizzare X e Y considerando che .... Altrimenti bisogna arrivare a fine capitolo per scoprirlo

SEZ: Modello del sistema

Modelliamo

Sia dato G_M una DNN. Consideriamo questa DNN come un DAG logico, rappresentabile mediante una coppia $G_M = (V_M, E_M)$, dove:

- V_M rappresenta l'insieme dei layer presenti all'interno del modello;

terminologia grafo o terminologia DNN

- E_M rappresenta l'insieme degli archi orientati presenti all'interno del modello tra i vari layer $\{(u, v) : u, v \in V_M\}$.

Indichiamo con:

- $V_I \subset V_M$ i nodi di input del modello: questi nodi sono tali per cui non esistono archi di ingresso a questi nodi; in simboli $\forall i \in V_I \quad \nexists u \in V_M \ tc \ \exists (u, i) \in E_M$;
- $V_O \subset V_M$ i nodi di output del modello: questi nodi sono tali per cui non esistono archi di uscita da questi nodi; in simboli $\forall i \in V_O \quad \nexists u \in V_M \ tc \ \exists (i, u) \in E_M$

G_N e G_M molto simili

Sia invece G_N il grafo della rete. G_N può essere rappresentato come un grafo diretto mediante una coppia $G_N = (V_N, E_N)$, dove:

- V_N rappresenta l'insieme dei dispositivi server all'interno della rete;

- E_N rappresenta l'insieme dei collegamenti di rete presenti tra i nodi di rete, ovvero $\{(u, v) : u, v \in V_N\}$.

Si notino i seguenti due aspetti:

- Non trattandosi di un DAG, all'interno di E_N è ammessa la presenza dei così detti *cappi*: in simboli abbiamo che $\forall u \in V_N \ \exists (u, u) \in E_N$; questo permette di modellare il caso in cui un nodo passa dati a se stesso. Questa modellazione richiederà un post-processing aggiuntivo che sarà aggetto delle parti successive della trattazione.

- ~~Trattandosi di un grafo diretto~~, non è necessario che tutte le coppie di archi siano presenti in E_N: questo modella la possibilità che non vi sia connessione tra due dispositivi all'interno della rete; in un contesto di edge-cloud continuum, questo può essere realistico, in quanto è possibile che un dispositivo edge abbia connessione con dispositivi a livello fog, ma non a livello del cloud.

Vogliamo dunque costruire un mapping del grafo logico (i.e. la DNN) sul grafo fisico (i.e. la rete di server). Ci riconduciamo, quindi, ad un algoritmo di graph partitioning simile a quello definito in [21, 22].

Definiamo in prima istanza le seguenti variabili:

- $x_{ik} \in \{0, 1\} \ \forall i \in V_M, \forall k \in V_N$; in particolare $x_{ik} = 1$ se e solo se il layer $i \in V_M$ viene mappato su server $k \in V_N$;

- $y_{mn} \in \{0, 1\} \ \forall m \in E_M, \forall n \in E_N$; in particolare $y_{mn} = 1$ se e solo se l'arco tra layer $m \in E_M$ viene mappato su arco tra server $n \in E_N$;

- $q_i \in \{0, 1\} \ \forall i \in V_M$; in particolare $q_i = 1$ se e solo se il layer $i \in V_M$ viene quantizzato. Notiamo che nel nostro caso stiamo considerando i soli due

casi *quantizzato* e *non quantizzato*, quindi la variabile binaria è sufficiente a rappresentare la nostra situazione.

Specifichiamo le seguenti notazioni: questa parte la sposterei subito dopo la definizione del grafo, dato che caratterizza i nodi

- Dato un $i \in V_M$ indichiamo con:
  - $i.quantizable$ un valore che indica se il livello è quantizzabile o meno.
- Dato un $k \in V_N$ indichiamo con:
  - $k.computePower$ un valore che indica la potenza consumata dal dispositivo in fase di calcolo.
  - $k.baseComputeEnergy$ un valore che indica l'energia consumata di base in fase di calcolo.
  - $k.selfTxPower$ un valore che indica la potenza consumata dal dispositivo in fase di trasmissione a se stesso.
  - $k.baseSelfTxEnergy$ un valore che indica l'energia base consumata dal dispositivo in fase di trasmissione a se stesso.
  - $k.otherTxPower$ un valore che indica la potenza consumata dal dispositivo in fase di trasmissione ad altri dispositivi.
  - $k.baseSelfTxEnergy$ un valore che indica l'energia base consumata dal dispositivo in fase di trasmissione ad altri dispositivi.
- Dato un $m \in E_M$ indichiamo con:
  - $m.dataSize$ un valore che indica la dimensione dei dati passati tra i layer del modello;

- Dato un $n \in E_N$ indichiamo con:
  - $n.bandWidth$ un valore che indica la banda disponibile sul collegamento fisico;
  - $n.latency$ un valore che indica la latenza sul collegamento fisico.

2.1 Modello di Tempo

2.1.1 Tempo di Calcolo

Indichiamo con $f_k(i, q_i)$ quella funzione che ci dice per il server $k \in V_N$, il tempo di calcolo per il layer $i \in V_M$, quando il suo stato di quantizzazione è q_i (i.e. quantizzato vs non-quantizzato). Partendo da questo, possiamo definire il seguente:

partirei scrivendolo in maniera più chiara come q*...+ (1-q)*...

$$\begin{aligned} T^c_{ik} &= f_k(i,0) \cdot x_{ik} - (f_k(i,0) - f_k(i,1)) \cdot x_{ik} \cdot q_i \\ &= f_k(i,0) \cdot x_{ik} - (f_k(i,0) - f_k(i,1)) \cdot x^q_{ik} \end{aligned} \tag{2.1.1}$$

Notiamo che in Equazione 2.1.1 è stata aggiunta la definizione della variabile x^q_{ik}.

introdurla brevemente insieme alle altre variabili prima?

Questa variabile è una variabile binaria (i.e. $x^q_{ik} \in \{0, 1\}$) che modella il prodotto tra le due variabili x_{ik} e q_i ed è tale per cui valgono i seguenti vincoli:

$$\begin{cases} x^q_{ik} \leq x_{ik} \\ x^q_{ik} \leq q_i \\ x^q_{ik} \geq x_{ik} + q_i - 1 \end{cases} \tag{2.1.2}$$

Da un punto di vista semantico quindi, questa variabile è tale che $x^q_{ik} = 1$ se e solo se il layer i è assegnato al server k ed il livello è quantizzato.

Analizziamo in dettaglio Equazione 2.1.1. Possiamo notare come il primo termine rappresenti il tempo di calcolo del livello i sul server k: questo tempo c'è sempre, a prescindere dallo stato di quantizzazione del livello; al contrario, il termine tra

parentesi lo possiamo vedere come un *guadagno di quantizzazione*, che viene sottratto al tempo del livello quando si sceglie di attivare la quantizzazione.

A questo punto, quindi, abbiamo che il tempo di calcolo per il server k è dato da:

$$T_k^c = \sum_{i \in V_M} T_{ik}^c \tag{2.1.3}$$

Per concludere, il tempo di calcolo complessivo del modello all'interno del sistema è dato da:

$$T^c = \sum_{k \in V_N} T_k^c \tag{2.1.4}$$

2.1.2 Tempo di Trasmissione

Indichiamo con $g(m, n)$ la funzione che stima il tempo di trasmissione dell'arco logico $m \in E_M$ sull'arco fisico $n \in V_N$. Nel nostro caso possiamo definirla come

$$g(m, n) = \frac{m.dataSize}{n.bandWidth}$$

Da ciò, indicato con $m[0] = i \in V_M$ il nodo sorgente dell'arco $m \in E_M$, possiamo definire il tempo di trasmissione dei dati dell'arco logico $m \in E_M$ attraverso l'arco di rete $n \in E_N$ come:

$$\begin{aligned} T_{ik}^x &= g(m,n) \cdot y_{mn} - \left(g(m,n) - \frac{g(m,n)}{4} \right) \cdot y_{mn} \cdot q_i + n.latency \cdot y_{mn} \\ &= g(m,n) \cdot y_{mn} - \left(g(m,n) - \frac{g(m,n)}{4} \right) \cdot y_{mn}^q + n.latency \cdot y_{mn} \end{aligned} \tag{2.1.5}$$

? userei un parametro a cui, in base alla quant.,
assegni poi il valore corretto

Notiamo che in Equazione 2.1.5 è stata definita la variabile y_{mn}^q. Questa variabile è una variabile binaria (i.e. $y_{mn}^q \in \{0, 1\}$) che modella il prodotto tra y_{mn} e q_i ed è tale per cui valgono i seguenti vincoli:

$$\begin{cases} y_{mn}^q \leq y_{mn} \\ y_{mn}^q \leq q_i \\ y_{mn}^q \geq y_{mn} + q_i - 1 \end{cases} \tag{2.1.6}$$

Da un punto di vista semantico quindi, questa variabile è tale che $y^q_{mn} = 1$ se e solo se l'arco logico m è assegnato all'arco di rete n e il livello sorgente i dell'arco logico è quantizzato.

Analizziamo in dettaglio Equazione 2.1.5. Possiamo notare come il primo termine rappresenti il tempo di trasmissione dei dati dell'arco logico sull'arco fisico: questo tempo c'è sempre, a prescindere dallo stato di quantizzazione del livello; al contrario, il termine tra parentesi lo possiamo vedere come un *guadagno di quantizzazione*, che viene sottratto al tempo di trasmissione base quando si sceglie di quantizzare il livello sorgente dei dati. Si noti che stiamo assumendo una quantizzazione dei dati di tipo *INT8*, quindi in caso di quantizzazione la dimensione finale dei dati è un quarto di quella originale (assunta *FLOAT32*). Per concludere, notiamo come il termine relativo alla latenza sia sempre presente: questo è infatti un costo che viene sempre pagato e che non può essere scontato in nessun modo in quanto dipendente dalla distanza tra i due dispositivi che stanno comunicando lungo quel collegamento.

Partendo da Equazione 2.1.5 possiamo definire il tempo complessivo di trasmissione di un server nel seguente modo:

$$\begin{aligned} T^x_k &= \sum_{m=(i,j)\in E_M} \left(\sum_{n\in E_N \wedge k=n[0]=n[1]} T^x_{ik} \quad + \sum_{n\in E_N \wedge k=n[0] \wedge k\neq n[1]} T^x_{ik} \right) \\ &= \sum_{m=(i,j)\in E_M} \left(T^{x-self}_{ik} + T^{x-other}_{ik} \right) \end{aligned}$$

Dove T^{x-self}_{ik} rappresenta il tempo che il server k impiega a trasmettere i dati dell'arco logico $(i,j) \in E_M$ attraverso l'arco di rete $(k,k) \in E_N$, ovvero a "trasmettere" i dati a se stesso; viceversa, $T^{x-other}_{ik}$ rappresenta il tempo che il server k impiega a trasmettere i dati dell'arco logico $(i,j) \in E_M$ attraverso l'arco di rete $(k,h) \in E_N$ con $h \neq k$.

In conclusione, il tempo di trasmissione complessivo per il modello è dato da:

$$T^x = \sum_{k \in V_N} T_k^x \tag{2.1.7}$$

2.1.3 Tempo Complessivo

Il tempo complessivo lo possiamo definire sommando Equazione 2.1.4 e Equazione 2.1.7, trovando:

$$T = T^c + T^x \tag{2.1.8}$$

2.2 Modello di Energia

2.2.1 Energia di Calcolo

Indichiamo con $h_k(t)$ la funzione che stima l'energia usata per il calcolo di durata t dal dispositivo k. Nel nostro caso assumiamo una funzione lineare del tipo:

$$h_k(t) = k.computePower \cdot t + k.baseComputeEnergy \tag{2.2.1}$$

L'energia consumata per il calcolo sul server k sarà data da:

$$E_k^c = h_k(T_k^c)$$

Il consumo energetico complessivo dato dal calcolo del modello è dato da:

$$E^c = \sum_{k \in V_N} E_k^c$$

2.2.2 Energia di Trasmissione

Siano $l_k^{self}(t)$ e $l_k^{other}(t)$ rispettivamente le funzioni che stimano l'energia usata per la trasmissione di durata t dal dispositivo k a se stesso e dal dispositivo k ad un

dispositivo h dove $k \neq h$. Assumiamo anche in questo caso delle funzioni lineari di tipo:

$$\begin{aligned} l_k^{self}(t) &= k.selfTxPower \cdot t + k.baseSelfTxEnergy \\ l_k^{other}(t) &= k.otherTxPower \cdot t + k.baseSelfTxEnergy \end{aligned} \tag{2.2.2}$$

L'energia consumata sul server k per la trasmissione è definita come:

$$E_k^x = l_k^{self}(T_k^{x-self}) + l_k^{other}(T_k^{x-other})$$

In questo caso, quindi, come nel caso del tempo di trasmissione, il contributo al consumo energetico è dato dal contributo di due termini, di cui il primo rappresenta il consumo energetico per trasmettere verso se stessi, che dipenderà tendenzialmente dal consumo dell'interfaccia di loopback, e il secondo rappresenta il consumo energetico per trasmettere verso altri nodi della rete.

Il consumo energetico complessivo per la trasmissione è dato da:

$$E^x = \sum_{k \in V_N} E_k^x$$

2.2.3 Energia del Server k-esimo

Il consumo energetico sul server k lo possiamo definire attraverso la somma seguente:

$$E_k = E_k^c + E_k^x \tag{2.2.3}$$

2.2.4 Consumo Energetico Complessivo

Il consumo energetico complessivo sarà dato dalla seguente relazione:

$$E = E^c + E^x \tag{2.2.4}$$

2.3 Modello di Memoria

TODO

2.4 Modello di Rumore di Quantizzazione

Come discusso nella sezione (TODO: INDICARE LA SEZIONE IN CUI SI DESCRIVE IL PROBLEMA DELLA QUANTIZZAZIONE), consideriamo per la quantizzazione soltanto un sottoinsieme di livelli; sia quindi $V_Q = \{i \in V_M : i.quantizable = True\} \subseteq V_M$. Indichiamo poi con $\mathbf{q} = \{q_i\}_{i \in V_Q}$ il vettore delle variabili q_i di quantizzazione dei livelli quantizzabili: questo vettore definisce uno schema di quantizzazione del modello, ovvero una certa combinazione di quantizzazioni/non-quantizzazioni per i livelli potenzialmente quantizzabili.

Supponiamo di avere una regressione polinomiale $\eta(\mathbf{q})$ di grado d che, dato in input il vettore $\mathbf{q}$ sopra definito, restituisce come risultato il valore del rumore di quantizzazione quando viene applicato lo schema di quantizzazione definito da $\mathbf{q}$.

Trattandosi di una regressione polinomiale di grado d, al suo interno potremo avere prodotti di al massimo d variabili $\{q_i\}_{i \in V_Q}$. Sia quindi $\hat{q}_k \in \{0, 1\}$ una variabile che rappresenta il prodotto logico delle variabili q_i per $i \in V_{Q_k}$ e per $V_{Q_k} \subset V_Q$ e $|V_{Q_k}| \leq d$. Ognuna di queste variabili è soggetta ai seguenti vincoli:

$$\begin{cases} \hat{q}_k \leq q_i & \forall i \in V_{Q_k} \\ \hat{q}_k \geq \sum_{i \in V_{Q_k}} q_i - (|V_{Q_k}| - 1) \end{cases} \tag{2.4.1}$$

Definiamo una variabile di esistenza $p \in \{0, 1\}$ tale che $p = 1$ se e solo se $\exists i \in V_Q$ tale che $q_i = 1$. Questa variabile ci dice se esiste almeno un livello quantizzato e sarà dunque soggetta ai vincoli seguenti:

$$\begin{cases} p \geq q_i & \forall i \in V_Q \\ p \leq \sum_{i \in V_Q} q_i \end{cases} \tag{2.4.2}$$

Possiamo quindi ridefinire il regressore come $\hat{\eta}(\hat{\mathbf{q}}, p)$, dove $\hat{\mathbf{q}} = \{\hat{q}_k\}_{V_{Q_k} \subset V_Q \ , |V_{Q_k}| \leq d}$, ottenendo in definitiva, indicati con $\mathbf{w}^T$ i pesi del regressore e con c la sua intercetta:

$$\hat{\eta}(\hat{\mathbf{q}}, p) = \mathbf{w}^T \hat{\mathbf{q}} + c \cdot p \tag{2.4.3}$$

Che è rappresentabile attraverso i vincoli lineari definiti fino a questo punto.

Si noti che la variabile binaria p è stata definita per garantire che in assenza di livelli quantizzati il risultato del regressore sia sempre nullo (i.e. $\hat{\eta}(\mathbf{0}, 0) = 0$). Potrebbe essere possibile, infatti, che in fase di addestramento del regressore l'intercetta sia calcolata come non nulla per avere un fitting migliore; tuttavia, considerando come baseline per il rumore il modello originale, un valore del rumore non nullo in assenza di quantizzazione sarebbe paradossale, in quanto l'assenza di livelli quantizzati implica l'uguaglianza con il modello originale.

2.5 I Vincoli

2.5.1 Vincoli di Quantizzazione

Ricordando che soltanto alcuni livelli possono essere quantizzati, possiamo imporre il vincolo:

$$q_i = 0 \qquad \forall i \notin V_Q = \{i \in V_M : i.quantizable = True\} \tag{2.5.1}$$

Questo assicura che i livelli non quantizzabili non saranno scelti per la quantizzazione in fase di ottimizzazione.

2.5.2 Vincoli di Assegnazione

I vincoli di assegnazione che dobbiamo definire per il modello sono i seguenti:

$$\begin{cases} \sum_{k \in V_N} x_{ik} = 1 & \forall i \in V_M \\ \sum_{n \in E_N} y_{mn} = 1 & \forall m \in E_M \\ x^a_{i0} = 1 & \forall i \in V_I \\ x^a_{j0} = 1 & \forall j \in V_O \end{cases} \tag{2.5.2}$$

Dove:

- Il primo e il secondo vincolo garantiscono assegnazione unica rispettivamente di nodi e archi logici a server e archi di rete.

- Il terzo e il quarto vincolo garantiscono che, detto $k = 0$ il server da cui viene fatta partire l'inferenza, i nodi di input e di output saranno mappati su quel server.

2.5.3 Vincoli di Flusso

I vincoli di flusso che dobbiamo definire per il modello sono i seguenti:

$$\begin{cases} x_{ik} = \sum_{h \in V_N} y_{(i,j)(k,h)} & \forall (i,j) \in E_M, \forall k \in V_N \\ x_{jh} = \sum_{k \in V_N} y_{(i,j)(k,h)} & \forall (i,j) \in E_M, \forall h \in V_N \end{cases} \tag{2.5.3}$$

Dove:

- Il primo vincolo garantisce il rispetto del flusso in uscita: se un livello i è assegnato ad un server k, allora i dati prodotti da i sono inviati ai server che hanno i nodi successori di i nel grafo logico.

- Il secondo vincolo garantisce il rispetto del flusso in ingresso: se un livello j è assegnato ad un server h, allora i dati ricevuti da j sono inviati dai server che hanno i nodi predecessori di j nel grafo logico.

2.5.4 Spazio delle Variabili

Gli spazi delle variabili sono definiti nella seguente relazione:

$$\begin{cases} x_{ik} \in \{0,1\} & \forall i \in V_M, \forall k \in V_N \\ y_{mn} \in \{0,1\} & \forall m \in E_M, \forall n \in E_N \\ q_i \in \{0,1\} & \forall i \in V_M \\ x^q_{ik} \in \{0,1\} & \forall i \in V_M, \forall k \in V_N \\ y^q_{mn} \in \{0,1\} & \forall m \in E_M, \forall n \in E_N \\ \hat{q}_k \in \{0,1\} & \forall V_{Q_k} \subset V_Q \ , |V_{Q_k}| \leq d \\ p \in \{0,1\} & \end{cases} \tag{2.5.4}$$

2.6 Il Problema

Formulazione del Problema

Siano:

- ω_t il peso associato al tempo di inferenza;
- ω_e il peso associato all'energia di inferenza;
- J_0 l'energia massima consumabile sul device $k = 0 \in V_N$, cioè il server che fa partire l'inferenza;
- η_{max} il rumore di quantizzazione massimo accettato.

Supponiamo che i pesi siano normalizzati: $\omega_t + \omega_e = 1$.

Definiamo quindi il problema di minimizzazione nel seguente modo:

$$\begin{aligned} min \quad & o(\omega_t \cdot T, \omega_e \cdot E) \\ subject\ to \quad & E_0 \leq J_0 \\ & \hat{\eta}(\hat{\mathbf{q}}, p) \leq \eta_{max} \end{aligned} \tag{2.6.1}$$

Oltre a questi vincoli, abbiamo anche: (2.1.2), (2.1.6), (2.4.1), (2.4.2), (2.5.1), (2.5.2), (2.5.3) e (2.5.4).

Ottimizzazione gerarchica

2.7 Rilassamento del Problema

Notiamo che il problema così definito è un problema di ottimizzazione multi-obiettivo, che in generale risulta complesso da risolvere.

Una delle prime complicazioni da affrontare riguarda la scala dei valori: essendo gli obiettivi misurati in unità di misura diverse, cioè il tempo in secondi e l'energia in Joule, abbiamo bisogno di un metodo di normalizzazione. La cosa migliore sarebbe scalare le due funzioni del tempo e dell'energia nell'intervallo $[0, 1]$ per poi ottimizzare una somma pesata delle funzioni normalizzate; al variare dei pesi otterremmo i vari punti della frontiera di Pareto. Per riportare nell'intervallo $[0, 1]$ l'approccio standard adottato è dividere ciascun obiettivo per il suo valore massimo, ottenuto dalla risoluzione di un problema di ottimizzazione al massimo con obiettivo singolo: ciò richiede la risoluzione di tre problemi di ottimizzazione in totale. Tuttavia, da alcune prove fatte, anche in casi molto semplici, la risoluzione dei problemi a obiettivo singolo al massimo risulta molto lenta. In conclusione, quindi, questo approccio di risoluzione è stato scartato.

Per affrontare il problema seguiamo l'approccio della minimizzazione gerarchica simile a come descritto in [23]. Procediamo in questo modo:

1. A seconda del valore dei pesi, scegliamo come obiettivo del problema quello associato al peso maggiore; a parità di pesi si sceglie sempre il tempo. Otteniamo quindi la funzione obiettivo o_1;

2. Si risolve il problema di ottimizzazione avente:

   (a) Vincoli come in Equazione 2.6.1;

   (b) Obiettivo $min\ o_1$, ottenendo il valore o_1^{min};

3. Partendo dai pesi, si definisce il valore $\epsilon = 1 - max(\omega_t, \omega_e) \in [0.5, 1]$; questo valore rappresenta la variazione percentuale che è ammessa per il valore del primo obiettivo;

4. Si risolve il problema di ottimizzazione avente:

   (a) Vincoli come in Equazione 2.6.1;

   (b) Vincolo aggiuntivo $o_1 \leq o_1^{min} \cdot (1 + \epsilon)$;

   (c) Obiettivo $min\ \ o_2$, trovando o_2^{min}.

Notiamo che i valori estremi di ϵ comprendono i due casi estremi di ottimizzazione:

- $\epsilon = 0$ significa che per migliorare il secondo obiettivo non è ammesso peggiorare il valore del primo: questo rappresenta quindi il caso di ottimizzazione più sbilanciato possibile.

- $\epsilon = 0.5$ rappresenta il caso più bilanciato di ottimizzazione , in cui il valore del primo obiettivo si può aumentare fino ad un massimo del 50%.

In conclusione, stabilire il valore di ϵ equivale a definire uno spazio sulla frontiera di Pareto in cui l'ottimizzatore può muoversi cambiando il valore del primo obiettivo al fine di migliorare il secondo obiettivo.

2.8 Post Elaborazione della Soluzione

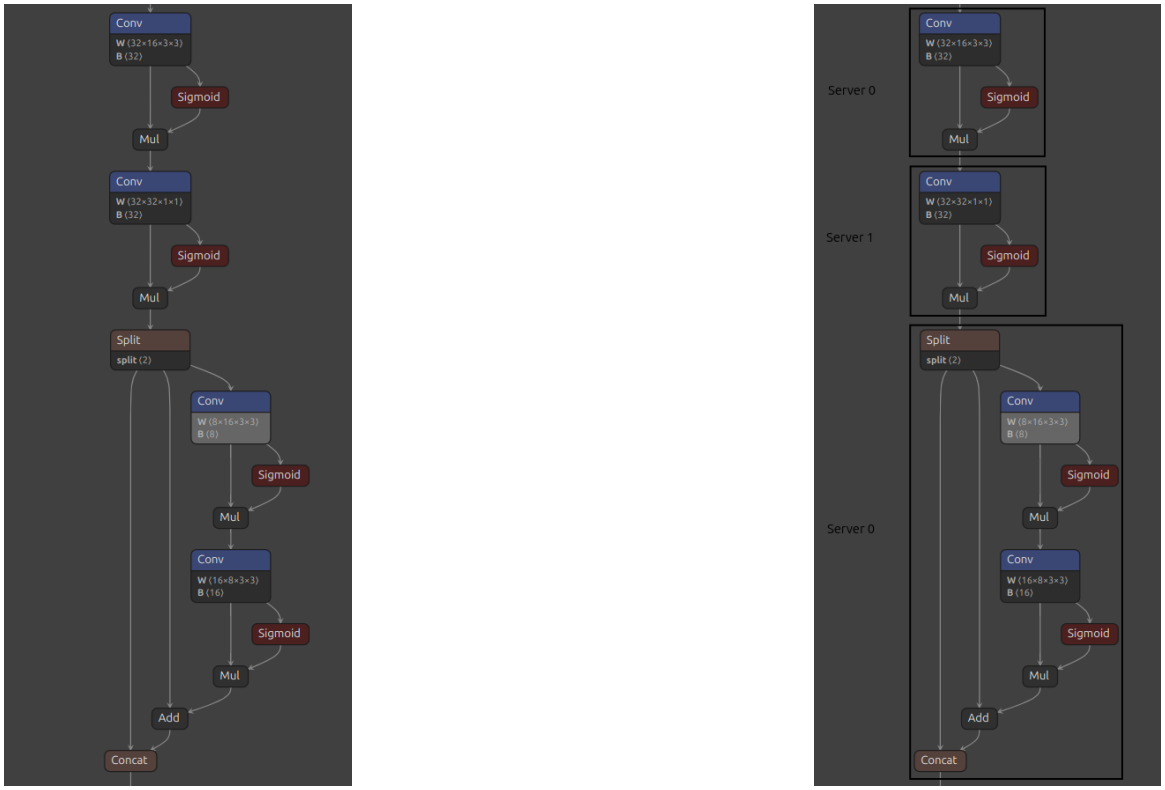
Risolto il problema di ottimizzazione ~~con un solver~~, possiamo raccogliere il risultato. Le informazioni fondamentali da ricavare dalla soluzione sono:

- Valori delle variabili $x_{ik} = 1$, per vedere, dato un livello, a quale server è stato assegnato;

- Valori delle variabili $q_i = 1$, per vedere, dato un livello (quantizzabile), se la quantizzazione per quel livello è stata attivata o meno.

Definiamo *componente* come un insieme di livelli del modello originale che sono interconnessi tra di loro e che vengono assegnati allo stesso server. Supposto che il modello originale sia quello rappresentato in Figura 2.1a una possibile divisione in componenti del modello è quella rappresentata in figura Figura 2.1b. Da questo esempio si può vedere come non sia scontato che ad un server corrisponda una singola componente: è infatti possibile che l'insieme di livelli assegnati ad un server sia tale per cui non vi siano dei collegamenti tra questi livelli. Dato un server, quindi, ci sarà un certo numero di componenti ad esso assegnate e questo numero dipenderà da come è stata fatta la suddivisione in fase di ottimizzazione. Stabilita la divisione in componenti, è possibile definire un grafo avente per nodi queste componenti e per archi i collegamenti tra livelli adiacenti assegnati a server diversi; mantenendoci sull'esempio in figura Figura 2.1b avremo un totale di tre nodi e due archi, di cui il primo è rappresentato dall'arco che collega la prima `Mul` al secondo `Conv` e il secondo dall'arco che collega la seconda `Mul` allo `Split`. Questo grafo di componenti non sarà altro che un'astrazione costruita sul grafo originale del modello sulla base delle assegnazioni dei livelli ai server e sulla base delle connessioni tra i livelli assegnati allo stesso server.

Tuttavia, la costruzione delle componenti sulla base delle sole assegnazioni e connessioni tra i livelli del modello originale modo potrebbe non essere suffiente ad evitare problemi. Essendo il grafo originale un DAG è necessario che il grafo delle componenti sia a sua volta un DAG, pena la possibilità di incorrere in deadlock in fasi successive. L'aciclicità del grafo delle componenti non è garantita per due motivi: in primo luogo stiamo assegnando nodi di un DAG ai nodi di un grafo (i.e. il grafo di rete) che non

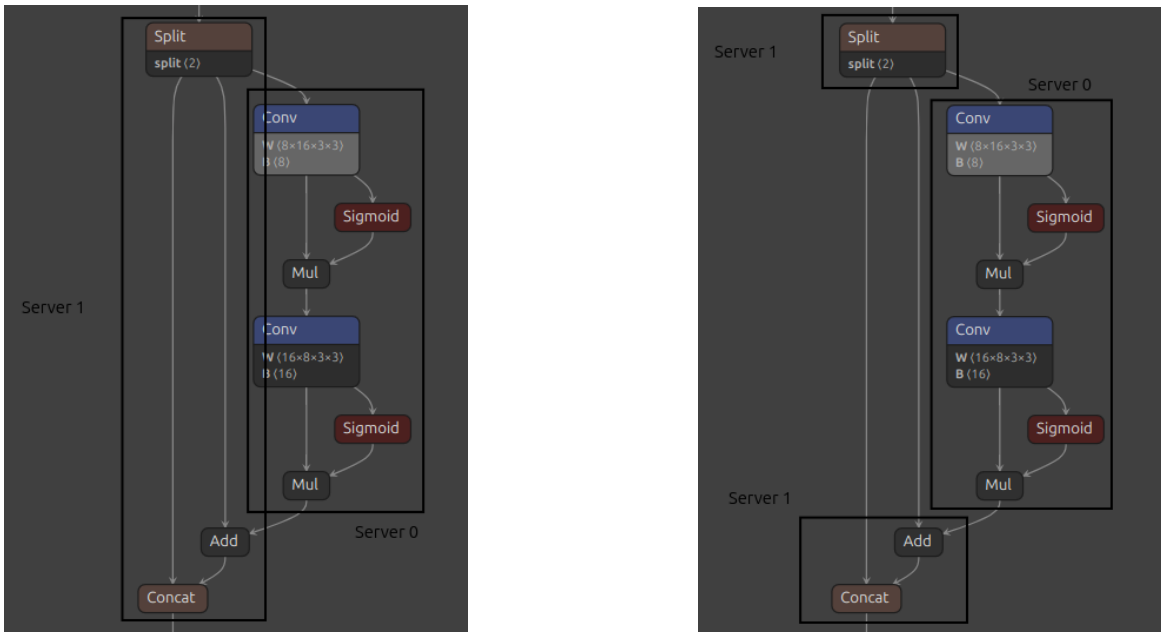


(a) Esempio di Modello (b) Esempio Assegnazione

Figura 2.1: Esempio di Costruzione del Grafo delle Componenti

è aciclico; in secondo luogo, all'interno del problema non ci sono vincoli che garantiscano l'assegnazione aciclica. Una possibile soluzione potrebbe essere l'introduzione di un vincolo di aciclicità all'interno del problema, come le soluzioni descritte in [24]: ciò però porterebbe ad un'ulteriore complicazione del problema, oltre all'impossibilità di avere dei ritorni indietro nel flusso dei dati tra i server.

Si è quindi preferito adottare un approccio di post-elaborazione della soluzione, in cui eventuali cicli tra le componenti vengano risolti. Un esempio di assegnamento ciclico è presentato in Figura 2.2a: in questo caso la componente assegnata al server 0 riceve il suo input da quella assegnata al server 1 e, allo stesso tempo, invia il suo output a questa medesima componente. Questo assegnamento ciclico si può risolvere, ad esempio, come mostrato in Figura 2.2b, dividendo la componente assegnata al server 1 in due componenti separate. In Algoritmo 1 e Algoritmo 2 viene riportato lo pseudo codice dell'algoritmo usato per la costruzione delle componenti.



(a) Assegnamento Ciclico (b) Risoluzione dei cicli

Figura 2.2: Esempio di assegnazione ciclica e risoluzione

Algoritmo 1 Assegnazione Nodi a Componenti

```
1: Function assignNodesToComponents(modelGraph, assignmentMap)
2: nodeComponentAssignment : dict[NodeId, ComponentId] ← dict()
3: nodeDepDict : dict[NodeId, set[ComponentId]] ← dict()
4: nodePosDict : dict[NodeId, set[ComponentId]] ← dict()
5: compDepDict : dict[ComponentId, set[ComponentId]] ← dict()
6: for nodeId in modelGraph.getNodes() do
7:    nodeDepDict[nodeId] ← set()
8:    nodePosDict[nodeId] ← set()
9: end for
10: for nodeId in modelGraph.topologicalSort() do
11:    dependencySet ← nodeDepDict[nodeId]
12:    possibleSet ← nodePosDict[nodeId]
13:    excludeSet ← set()
14:    for depCompId in dependencySet do
15:       for possCompId in possibleSet do
16:          if possCompId in compDepDict[depCompId] then
17:             excludeSet.add(possCompId)
18:          end if
19:       end for
20:    end for
21:    differenceSet ← possibleSet - excludeSet
22:    if diferenceSet.empty() or modelGraph.isGeneratorOrReceiverNode(nodeId)
       then
23:       nodeCompId ← generateNewComponentId(assignmentMap[nodeId])
24:    else
25:       nodeCompId ← differenceSet.getRandom()
26:    end if
27:    nodeComponentAssignment[nodeId] ← nodeCompId
28:    updateSets(modelGraph, assignmentMap, nodeId, nodeCompId, nodeDepDict,
       nodePosDict, compDepDict)
29: end for
30: return nodeComponentAssignemt
    =0
```

Analizziamo alcuni aspetti significativi di Algoritmo 1:

- L'attraversamento del grafo viene fatto seguendo un ordinamento topologico del grafo stesso: questo permette una migliore gestione e analisi delle dipendenze.
- L'insieme `excludeSet` rappresenta l'insieme di componenti che dobbiamo escludere dalle possibili perché l'aggiunta del livello in una di queste componenti introdurrebbe una dipendenza circolare: nello specifico, queste componenti da escludere sono quelle possibili p tali per cui esiste una componente d da cui il livello dipende e per cui c'è dipendenza tra p e d; inserire il livello nella componente p creerebbe un ciclo perché, indicata con $\rightarrow$ la dipendenza tra componenti, $(p \rightarrow d)$ di base, ma inserire il livello in p introdurrebbe la dipendenza $(d \rightarrow p)$ perché il livello dipende proprio da d.
- Si noti che, per motivi di semplicità di gestione, i nodi di input (i.e. generatori) e i nodi di output (i.e. receiver) sono gestiti in componenti separate.

Analizziamo alcuni aspetti significativi di Algoritmo 2:

- Il primo ciclo `for` aggiorna le dipendenze dei livelli dalle componenti: una volta che un nodo è stato assegnato ad una componente, tutti i nodi discendenti, cioè tutti i nodi che si trovano dopo quel livello nel grafo e che quindi dipendono direttamente o indirettamente dal suo output, saranno dipendenti da questa componente.
- Il secondo ciclo `for` aggiorna le componenti possibili dei nodi successori: se un nodo successore è assegnato allo stesso server del nodo corrente, allora la componente assegnata al nodo corrente può essere anche la componente del

Algoritmo 2 Aggiornamento degli Insiemi

```
1: Function updateSets(modelGraph, assignmentMap, nodeId, nodeCompId,
   nodeDepDict, nodePosDict, compDepDict)
2: for descNodeId in modelGraph.descendants(nodeId) do
3:    nodeDepDict[descNodeId].add(nodeCompId)
4: end for
5: if not modelGraph.isGenerator(nodeId) then
6:    for nextNodeId in modelGraph.successors(nodeId) do
7:       if assignmentMap[nodeId] = assignmentMap[nextNodeId] then
8:          nodePosDict[nextNodeId].add(nodeCompId)
9:       end if
10:   end for
11:   for parNodeId in modelGraph.parallels(nodeId) do
12:      if assignmentMap[nodeId] = assignmentMap[parNodeId] then
13:         nodePosDict[parNodeId].add(nodeCompId)
14:      end if
15:   end for
16: end if
   compDepDict[nodeCompId].expand(nodeDepDict[nodeId] - nodeCompId)
   =0
```

successore. Questo permette di accorpare livelli successivi assegnati allo stesso server nella stessa componente.

- Il terzo ciclo `for` aggiorna le componenti possibili dei nodi paralleli, dove con nodi paralleli si intendono livelli che non sono né discendenti né antenati del nodo corrente (si trovano appunto su un ramo del grafo parallelo a quello del nodo corrente): se un nodo parallelo è assegnato allo stesso server del nodo corrente, allora la componente assegnata al nodo corrente può essere anche la componente del parallelo. Questa soluzione permette di ridurre il numero totale di componenti generate dall'algoritmo e di avere uno sfruttamento migliore delle ottimizzazioni in fase di inferenza: avremo infatti componenti, quindi sotto modelli, più grandi su cui sarà più facile applicare ottimizzazioni.

Bibliografia

[1] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, p. 1–30, Dec. 2022. [Online]. Available: http://dx.doi.org/10.1145/3527155

[2] M. Yu, Z. Jiang, H. C. Ng, W. Wang, R. Chen, and B. Li, "Gillis: Serving large neural networks in serverless functions with automatic model partitioning," Jul. 2021. [Online]. Available: http://dx.doi.org/10.1109/ICDCS51616.2021.00022

[3] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, p. 595–608, Apr. 2021. [Online]. Available: http://dx.doi.org/10.1109/TNET.2020.3042320

[4] Open Neural Network Exchange. [Online]. Available: https://onnx.ai/

[5] sklearn-onnx: Convert your scikit-learn model into ONNX. [Online]. Available: https://onnx.ai/sklearn-onnx/

[6] tf2onnx - Convert TensorFlow, Keras, Tensorflow.js and Tflite models to ONNX. [Online]. Available: https://github.com/onnx/tensorflow-onnx

[7] torch.onnx. [Online]. Available: https://docs.pytorch.org/docs/main/onnx.html

[8] onnx.shape_inference. [Online]. Available: https://onnx.ai/onnx/api/shape_inference.html

[9] onnx.utils. [Online]. Available: https://onnx.ai/onnx/api/utils.html

[10] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[11] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[12] YoloV8 Pre e Post Processing. [Online]. Available: https://github.com/levipereira/ultralytics/blob/main/examples/YOLOv8-Segmentation-ONNXRuntime-Python/main.py

[13] "Roboflow supervision." [Online]. Available: https://supervision.roboflow.com/latest/

[14] O. R. developers, "Onnx runtime," https://onnxruntime.ai/, 2021, version: x.y.z.

[15] onnxruntime. [Online]. Available: https://github.com/microsoft/onnxruntime

[16] Graph Optimizations in ONNX Runtime . [Online]. Available: https://onnxruntime.ai/docs/performance/model-optimizations/graph-optimizations.html

[17] Quantize ONNX Models. [Online]. Available: https://onnxruntime.ai/docs/performance/model-optimizations/quantization.html

[18] gRPC. [Online]. Available: https://grpc.io/

[19] A. Hagberg, P. J. Swart, and D. A. Schult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[20] S. Mitchell, M. OSullivan, and I. Dunning, "Pulp: a linear programming toolkit for python," *The University of Auckland, Auckland, New Zealand*, vol. 65, p. 25, 2011.

[21] G. Russo Russo, V. Cardellini, F. L. Presti, and M. Nardelli, "Towards a security-aware deployment of data streaming applications in fog computing," *Fog/Edge Computing For Security, Privacy, and Applications*, pp. 355–385, 2021.

[22] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey, "The node capacitated graph partitioning problem: a computational study," *Mathematical programming*, vol. 81, pp. 229–256, 1998.

[23] O. Grodzevich and O. Romanko, "Normalization and other topics in multi-objective optimization," 2006.

[24] M. Y. Özkaya and Ü. V. Çatalyürek, "A simple and elegant mathematical formulation for the acyclic dag partitioning problem," *arXiv preprint arXiv:2207.13638*, 2022.