

0.1 Analisi di complessità di un job map-reduce

0.1.1 Es: moltiplicazione di matrici

$$A_{m \times n} \times B_{n \times o}(i, j, a_{ij}) \mapsto M_A((i, j), (A, k, a_{ik})) \forall j = 1, \dots, o(k, j, b_{kj}) \mapsto M_B((i, j), (B, k, b_{ik})) \forall i = 1, \dots, m(i, j) [(A, 1, a_{i1}), \dots, (A, m, a_{im})]$$

$$R(A, B) \bowtie S(B, C) \bowtie T(C, D)(a, b) \mapsto_{M_R} (b, (R, a))(b, c) \mapsto_{M_S} (b, (S, a))$$

Quale è il costo di questo algoritmo. Indichiamo con r, s, t i rispettivi numeri di tuple delle tabelle R, S, T . Il primo processo M_R riceve tutte e sole le tuple di R , quindi ha costo r . Il secondo, similmente, ha costo s .

Il risultato del costo di complessità sarà quindi un $O(r + s)$. Ma questo è tra due relazioni. Se volessi farlo da 3 relazioni (**join in cascata**) cosa andrei ad ottenere?

$$(R \bowtie S) \bowtie T$$

Ottengo il costo $O(r + s + t + r s p)$, con p rappresentante la probabilità che due valori di R e S hanno un attributo uguale.

0.1.2 Join multi-way

Date due funzioni di hash, una h per l'attributo B ed una g per l'attributo C , con b **bucket** e c **bucket**, avendo che $bc = k$.

Nel caso di una tupla $(u, v) \in R$ viene inviata ad un'unica colonna verticale, riducendo i nodi (**c reducer**).

Nel caso di una tupla $(w, z) \in T$ viene inviata ad un'unica colonna orizzontale, riducendo i nodi (**b reducer**).

Nel caso di una tupla $(v, w) \in S$ viene inviata ad un'unica cella, riducendo i nodi ad uno soltanto (**1 reducer**).

Il costo quindi risulta essere:

$$O(r + 2s + t + cr + bt)$$

0.1.3 Rilassamento lagrangiano

N.B. Il parametro λ non può essere negativo.

$$L(b, c) = cr + br - \lambda(bc - k)$$

$$\frac{dL(b, c)}{db} = 0$$

$$\frac{dL(b, c)}{dc} = 0$$

Ottengo quindi un sistema:

$$\begin{cases} t - \lambda c = 0 \\ r - \lambda b = 0 \end{cases} \implies \begin{cases} t = \lambda c \\ r = \lambda b \end{cases}$$

$$\lambda = \sqrt{\frac{rt}{k}}$$

$$c = \sqrt{\frac{kt}{r}}$$

$$b = \sqrt{\frac{kr}{t}}$$

Il **costo ottimizzato** della **join multiway** risulta quindi essere:

$$O(r + 2s + t + 2\sqrt{kr t})$$

0.1.4 Es: Join sui nodi di facebook

Prendiamo ad esempio il grafo dei nodi facebook, dotato di 10^9 nodi.

$R(U_1, U_2), |R| = r = 3 \times 10^1$ (dati arbitrari)

$$R \bowtie R \bowtie R$$

Approccio Multi-way: $r + 2r + r + 2r\sqrt{k} = 4r + 2r\sqrt{k} = 1 \times 2 \times 10^1 2 + 6 \times 10^1 1\sqrt{k}$

Approccio cascata (nell'ipotesi che $absR \bowtie R = 30r$): $r + r + r + r^2 \times p = \dots = 2r + 60r = 1 \times 2 \times 10^1 2 + 1 \times 86 \times 10^1 3$

Otengo quindi che: $6 \times 10^1 1\sqrt{k} \leq 1 \times 86 \times 10^3 3 \rightarrow k \leq 961$, e risulta quindi migliore utilizzare l'approccio multi way quando si hanno meno di 961 nodi da allocare a dei reducer.

0.1.5 Es: Google pagerank

Come funziona **pagerank**:

	A	B	C	D
A	0	$\frac{1}{2}$	1	0
B	$\frac{1}{3}$	0	0	$\frac{1}{2}$
C	$\frac{1}{3}$	0	0	$\frac{1}{2}$
D	$\frac{1}{3}$	$\frac{1}{2}$	0	0

$$v_j(t+1) = P(\text{Trovarsi in } j \text{ al tempo } t+1) = \sum_i P(\text{trovarsi in } i \text{ al tempo } t) \cdot P(\text{spostarsi da } i \text{ a } j \mid \text{trovarsi in } i \text{ al punto } t)$$

$$\sum_i v_i(t) m_{ji} = \sum_i m_{ji} v_i(t) = (Mv(t))_j$$

con $M_{ij} = P(\text{spostarsi da } j \text{ a } i)$.

$$\vec{V}(t+1) = M\vec{v}(t)$$

Prova di convergenza

Definizione 0.1.1 (Determinante)

$$\det A = \sum_i a_{ij} c_{ij} = \sum_j a_{ij} c_{ij}$$

Definizione 0.1.2 (Determinante di matrice trasposta) Una matrice e la sua trasposta possiedono lo stesso determinante.

$$\det A^T = \sum_i a_{ij}^T c_{ij}^T = \sum_i a_{ji} c_{ji} = \sum_j a_{ij} c_{ij}$$

Definizione 0.1.3 (Autovalori) Si ottengono trovando gli zeri dell'equazione caratteristica.

$$\det(A - \lambda I) = 0 \leftrightarrow \det(A - \lambda I)^T = 0 \leftrightarrow (A^T - \lambda I) = 0$$

Definizione 0.1.4 (Matrice stocastica per righe) Una qualsiasi matrice stocastica per righe ammette 1 come autovettore.

$$\forall i \sum_j a_{ij} = 1$$

Definizione 0.1.5 (Matrice stocastica per colonne) Una qualsiasi matrice stocastica per colonne ammette 1 come autovettore poiché si tratta della trasposta di una matrice stocastica per righe.

Theorem 0.1.6 (La potenza di una matrice stocastica è sempre stocastica) Il risultato dell'elevazione a potenza di una matrice stocastica risulta sempre essere stocastico. Dimostriamo per induzione:

$$\text{Base: } k = 1 \quad A^k = A$$

$$\text{Passo: } A^k \text{ stocastica} \rightarrow A^{k+1} \text{ stocastica}$$

Dimostriamo che ad un generico passo k , otteniamo sempre 1 quando andiamo a sommare i termini.

$$a_{ij}^{k+1} = \sum_s a_{ij}^k a_{sj} = \sum_j \sum_s a_{is}^k a_{sj}$$

Inverto le sommatorie ed ottengo:

$$\sum_s a_{is}^k \sum_j a_{sj} = \sum_s a_{is}^k = 1$$

Theorem 0.1.7 (Autovalori di una stocastica) Se A è stocastica per colonne, il suo autovalore massimo è 1.

Procediamo a dimostrare per assurdo.

Sappiamo che 1 è un autovalore di A , poiché A è stocastica. Affermiamo che, per assurdo, esista un $\lambda > 1$ autovalore di A (che è anche l'autovalore di A^T) e v un autovettore per λ .

$$A^T v \Rightarrow \lambda v$$

$$A^{2T} v = A^T \times A^T v = A(\lambda v) = \lambda A^T v = \lambda^2 v \Rightarrow A^{Tk} v = \lambda^k v$$

Maggioriamo / minoriamo la sommatoria:

$$\sum_j a_{ij}^{Tk} v_{max} \geq \sum_j a_{ij}^{Tk} v_j = \lambda^k v_i > G$$

$$v_{max} \sum_j a_{ij}^{Tk} > G$$

Ma siccome la sommatoria è dei termini di una matrice stocastica per righe (essendo la trasposta di A), deve essere pari a 1, per cui:

$$v_{max} \sum_j a_{ij}^{Tk} = v_{max} > G$$

$$1 > \frac{G}{v_{max}} \Rightarrow \text{assurdo!}$$

Theorem 0.1.8 (Autovalori di potenza di matrice)

$$v_0 \rightarrow v_1 = Av_0 \rightarrow v_2 = Av_1 = A^2 v_0 \rightarrow \dots \rightarrow v_k = A^k v_0$$

Dimostrazione

$$Av_0 = A(\alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_n x_n) = \alpha_1 Ax_1 + \alpha_2 Ax_2 + \dots + \alpha_n Ax_n = \alpha_1 \lambda_1 x_1 + \alpha_2 \lambda_2 x_2 + \dots + \alpha_n \lambda_n x_n$$

$$v_k = A^k v_0 = \alpha_1 \lambda_1^k x_1 + \alpha_2 x_2 \frac{\lambda_2^k \lambda_1^k}{\lambda_1^k} + \dots + \alpha_k x_k \frac{\lambda_k^k \lambda_1^k}{\lambda_1^k}$$

$$v_k = A^k v_0 = \lambda_1^k (\alpha_1 x_1 + \alpha_2 x_2 \frac{\lambda_2^k}{\lambda_1^k} + \dots + \alpha_k x_k \frac{\lambda_k^k}{\lambda_1^k})$$

per k "grande" $v_k = A^k v_0 \approx \lambda_1^k \alpha_1 x_1$, nel nostro caso $\lambda_1 = 1$, cioè $\lim_{k \rightarrow \infty} \alpha_1 \lambda_1^k x_1 = 1$.

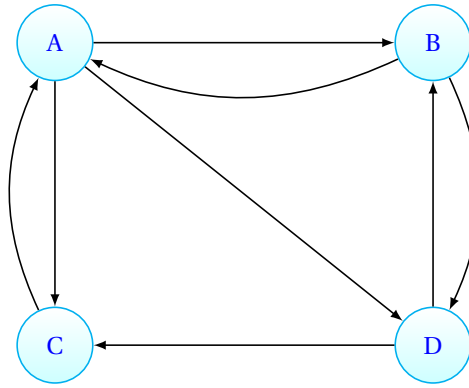


Figure 1: In questo grafo, dopo 10 esecuzioni, pagerank assegna ai nodi A, B, C, D rispettivamente $\frac{3}{9}, \frac{2}{9}, \frac{2}{9}, \frac{2}{9}e^2/9$.

0.1.6 Trappole per ragni

Nodi di grafi (figura 2) con loop, da cui gli spider non possono uscire.

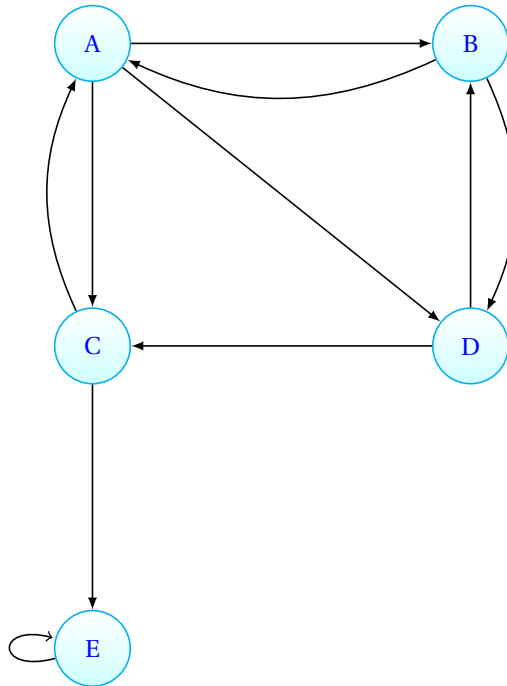


Figure 2: Dopo un certo numero di iterazioni, tutta la "massa" distribuita inizialmente sul grafo sarà finita su E.

0.1.7 Risolvere le trappole per ragni col teletrasporto

Per evitare le trappole per ragni, andiamo a inserire una certa probabilità β di teletrasportarci da un nodo ad un altro al posto di continuare a navigare tramite link.

$$v_{t+1} = \beta M v_t + (1 - \beta) \frac{1}{n} \mathbf{1}$$

$P(\text{mi trovo in } i \text{ al tempo } t + 1) = P(\text{spostarsi tramite link in } i \mid \text{ scelgo i link}) P(\text{scelgo i link}) + P(\text{teletrasportarsi in } i \mid \text{ mi teletrasporto}) P(\text{scegliere teletrasporto})$