MapReduce - Produit Matriciel

Quentin Fournier
quentin.fournier@polymtl.ca

Les exemples et les figures proviennent de livre *Mining of Massive Datasets (Second Edition)* de J. Leskovec, A. Rajaraman, et J. Ullman.

28 septembre 2021

Le produit matrice-vecteur

- L'implémentation originale de MapReduce par Google avait pour but de calculer le produit entre des matrices et des vecteurs massifs.
- Le produit entre une matrice $\boldsymbol{M} \in \mathbb{R}^{k \times n}$ et un vecteur $\boldsymbol{v} \in \mathbb{R}^{n}$ est un vecteur $\boldsymbol{x} \in \mathbb{R}^{k}$ tel que $x_i = \sum_{j=1}^{n} m_{ij} \times v_j$

$$\underbrace{\begin{bmatrix} 5 & 6 & 2 & 1 \\ 0 & 1 & 7 & 6 \\ 4 & 8 & 9 & 0 \end{bmatrix}}_{\text{Matrice } \boldsymbol{M}} \times \underbrace{\begin{bmatrix} 3 \\ 6 \\ 7 \\ 1 \end{bmatrix}}_{\text{Vecteur } \boldsymbol{v}} = \underbrace{\begin{bmatrix} 66 \\ 61 \\ 123 \end{bmatrix}}_{\text{Produit } \boldsymbol{x}}$$

Cas simple

- Le vecteur $\boldsymbol{v}$ est stocké entièrement dans la mémoire de chaque *worker*.
- Supposons que les données de $\boldsymbol{M}$ et $\boldsymbol{v}$ soient stockées sous la forme (i, j, m_{ij}) et (j, v_j) respectivement.
- `Map` retourne $(i, m_{ij} \times v_j)$.
- `Reduce` retourne la somme de toutes les valeurs associées à la clé i.

Cas simple

Algorithm 1: Map(Tuple<int> (i, j, m_{ij}))

```
// Suppose que v soit stocké en mémoire
```
emit($i, m_{ij} \times v_j$)

Algorithm 2: Reduce(Int i, List<int> *products*)

$x_i \leftarrow$ sum(*products*)
emit(i, x_i)

Cas réaliste

- Le vecteur $\boldsymbol{v}$ ne peut pas être stocké entièrement dans la mémoire de chaque *worker*.
- Il suffit lire le vecteur $\boldsymbol{v}$ depuis le disque chaque fois que nécessaire...
- mais cela entraîne un très grand nombre de communications et une baisse significative des performances !

Cas réaliste

- La solution est de découper $\boldsymbol{M}$ en *stripes* verticales de taille égale, et $\boldsymbol{v}$ en un nombre identique de *stripes* horizontales.
- La i-ème *stripe* de $\boldsymbol{M}$ ne multiplie que la i-ème *stripe* $\boldsymbol{v}$.
- Chaque `Map` reçoit un *stripe* de $\boldsymbol{M}$ et de $\boldsymbol{v}$.
- `Map` et `Reduce` fonctionne de la même manière que dans le cas simple.

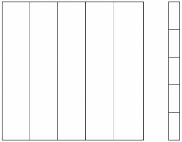

Figure 2.4: Division of a matrix and vector into five stripes

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

Le produit matriciel

- Le produit entre deux matrices $\boldsymbol{M} \in \mathbb{R}^{p \times q}$ et $\boldsymbol{N} \in \mathbb{R}^{q \times r}$ est une matrice $\boldsymbol{P} \in \mathbb{R}^{p \times r}$ tel que $p_{ik} = \sum_{j=1}^{q} m_{ij} \times n_{jk}$

$$\underbrace{\begin{bmatrix} 5 & 6 & 2 & 1 \\ 0 & 1 & 7 & 6 \\ 4 & 8 & 9 & 0 \end{bmatrix}}_{\text{Matrice } \boldsymbol{M}} \times \underbrace{\begin{bmatrix} 4 & 2 \\ 6 & 9 \\ 1 & 6 \\ 2 & 4 \end{bmatrix}}_{\text{Matrice } \boldsymbol{N}} = \underbrace{\begin{bmatrix} 60 & 80 \\ 25 & 75 \\ 73 & 134 \end{bmatrix}}_{\text{Produit } \boldsymbol{P}}$$

- Supposons que les données de $\boldsymbol{M}$ et $\boldsymbol{N}$ soient stockées sous la forme (i, j, m_{ij}) et (j, k, n_{jk}) respectivement.

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

Solution en deux étapes

1) Faire la jointure naturelle de $\boldsymbol{M}$ et $\boldsymbol{N}$, c'est-à-dire selon l'attribut j. La jointure retourne $(i, j, k, m_{ij}, n_{jk})$.

2) Grouper selon les attributs i et k, puis agréger en faisant la somme des produits $m_{ij} \times n_{jk}$

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

1) Jointure naturelle

Pour la matrice $\boldsymbol{M}$:

Algorithm 3: Map(Tuple<int> (i, j, m_{ij}))

emit(j, (“M”, i, m_{ij}))

Pour la matrice $\boldsymbol{N}$:

Algorithm 4: Map(Tuple<int> (j, k, n_{jk}))

emit(j, (“N”, k, n_{jk}))

Algorithm 5: Reduce(Int j, List<tuple> *values*)

```
// Pour chaque combinaison de (i, k)
for each ("M", i, m_ij) and ("N", k, n_jk) in values do
 |  emit((i, k), m_ij × n_jk)
end
```

2) Groupe et agrège

Algorithm 6: Map(Tuple<int> (i, k), Int p_{ik})

```
// Fonction identité
```
emit((i, k), p_{ik})

Algorithm 7: Reduce(Tuple<int> (i, k), List<int> *produits*)

$p_{ik} \leftarrow \text{sum}(\textit{produits})$

emit((i, k), p_{ik})

Solution en une étape

- `Map` retourne l'ensemble des données nécessaires pour calculer chaque élément de $\boldsymbol{P}$. Notez que chaque élément de $\boldsymbol{M}$ et $\boldsymbol{N}$ contribue à plusieurs éléments de $\boldsymbol{P}$ et doit donc être retourné plusieurs fois.
- Chaque clé (i, k) est associée à une liste contenant les tuples $(\text{“M”}, j, m_{ij})$ et $(\text{“N”}, j, n_{jk})$ pour toutes les valeurs de j possible.
- `Reduce` doit connecter les deux valeurs m_{ij} et n_{jk} qui ont le même j, pour chaque j. Ces valeurs sont ensuite multipliées, et l'ensemble des produits est sommé pour obtenir p_{ik}.

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

Solution en une étape

Pour la matrice $\boldsymbol{M}$:

Algorithm 8: Map(Tuple<int> (i, j, m_{ij}))

```
// Pour chaque colonne de N
for k = 0, 1, ..., r do
    emit((i, k), ("M", j, m_ij))
end
```

Pour la matrice $\boldsymbol{N}$:

Algorithm 9: Map(Tuple<int> (j, k, n_{jk}))

```
// Pour chaque ligne de M
for i = 0, 1, ..., p do
    emit((i, k), ("N", j, n_jk))
end
```


POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

Solution en une étape

Algorithm 10: Reduce(Tuple<int> (i, k), List<tuple> *values*)

```
m ← dict()
n ← dict()
for matrix, j, value do
    if matrix is equal to "M" then
        m[j] ← value
    end
    if matrix is equal to "N" then
        n[j] ← value
    end
end
p_ik ← 0
for all j do
    p_ik ← p_ik + m[j] × n[j]
end
emit((i, k), p_ik)
```


POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE