

Requêtes relationnelles dans les systèmes Map-Reduce

Master DAC – Bases de Données Large Echelle
Mohamed-Amine Baazizi
baazizi@ia.lip6.fr
Octobre 2017

Introduction

Manipulations des données en distribué

– Algèbre Map-Reduce étendue

- Spécifier des UDF (User defined functions) pour préparation, nettoyage de données brutes
- Traitement ‘bas niveau’, optimisation compliquée




– Algèbre relationnelle (Dataframe, SQL)


- Manipulation de données structurées
- Langages déclaratifs, techniques d’optimisations connues pour SGBD parallèles

2

Implantations de SQL en distribué

- Native MapReduce
 - Hive 
 - Facebook Presto 
 - MapR Drill, LinkedIn Tajo

- Systèmes dédiés
 - Spark SQL 
 - FlinkSQL 
 - IBM BigSQL 








- Hadoop comme data store
 - Impala 
 - Hadapt 
 - Pivotal HAWQ 

Remarques :

- Requêtes analytiques seulement (pas de transactions)
- sous-langages du standard SQL

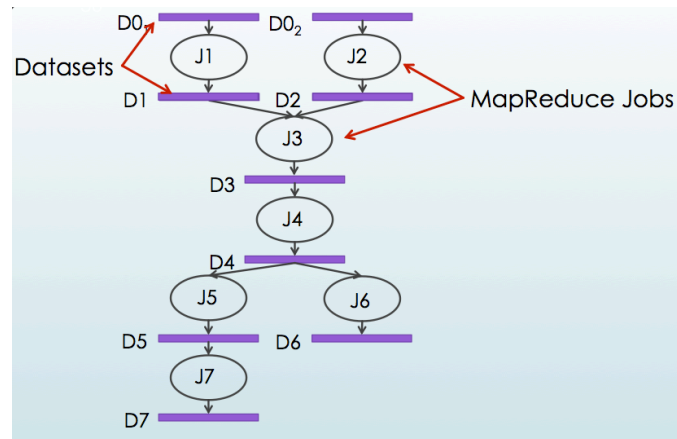
3

Implantations de SQL en distribué : principes

- Native MapReduce 
 - Génération d’un DAG Map-Reduce
- Hadoop comme data store   
 - Utilisation de SGBD relationnels (Postgres, mysql)
- Systèmes dédiés   
 - Génération d’un workflow pour algèbre propre

4

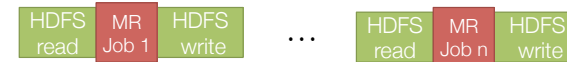
Implantation Native MapReduce



5

Implantation Native MapReduce

- Avantages (ceux de MapReduce)
 - Passage à l'échelle, reprise sur panne
- Inconvénients (ceux de MapReduce)
 - Coût de la matérialisation entre chaque job



- Pistes pour l'optimisation
 - Réduire le nombre de jobs, gestion efficace des données intermédiaires, jointures optimisées

6

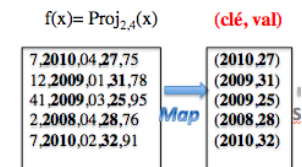
Traduction SQL en MapReduce

- Opérateurs considérés
 - Unaires : projection, sélection, agrégations
 - Binaires : produit cartésien, jointure, union, intersection, différence
 - Fonctions d'agrégations de SQL : min, max, ...
- Approche naïve
 - Associer à chaque opérateur un job correspondant
 - Enchaîner les job selon leur dépendance

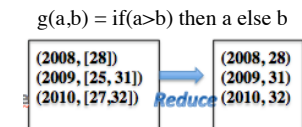
7

Traduction opérateurs unaires

- Projection, sélection
 - Triviale : UDF du *map*



- Agrégation
 - Si données partitionnées, appliquer fonction d'agg. durant *reduce*

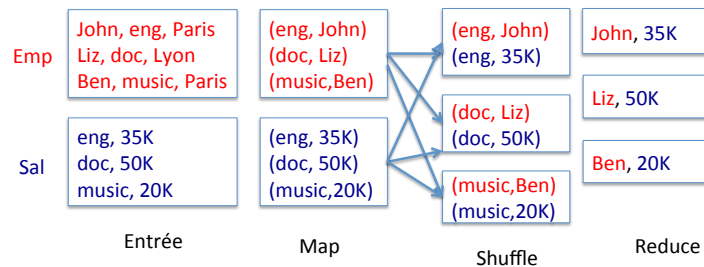


8

Traduction de la jointure

• Jointure par partitionnement

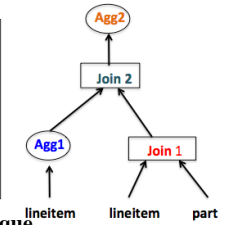
- *Map* : étiqueter chaque relation, exposer attribut(s) de jointure comme clé
- *Reduce* : fusionner les valeurs de la même clé



9

Traduction requête complexe

```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM (SELECT l_partkey, 0.2* avg(l_quantity) AS t1
      FROM lineitem GROUP BY l_partkey) AS inner,
      (SELECT l_partkey, l_quantity, l_extendedprice
      FROM lineitem, part
      WHERE p_partkey = l_partkey) AS outer
WHERE outer.l_partkey = inner.l_partkey; AND outer.l_quantity <
inner.t1;
```



La requête R et son arbre algébrique

```
Agg1
Map : lineitem → (l_partkey, l_quantity)
Reduce : v = 0.2* avg(l_quantity)

Join1
Map : lineitem → (l_partkey, (l_quantity, l_extendedprice))
      part → (l_partkey, null)
Reduce : jointure

Join2
Map : inner → (l_partkey, v)
      outer → (l_partkey, (l_quantity, l_extendedprice))
Reduce : jointure

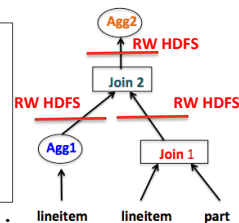
Agg2 ...
```

Traduction de R en MapReduce

10

Traduction requête complexe

```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM (SELECT l_partkey, 0.2* avg(l_quantity) AS t1
      FROM lineitem GROUP BY l_partkey) AS inner,
      (SELECT l_partkey, l_quantity, l_extendedprice
      FROM lineitem, part
      WHERE p_partkey = l_partkey) AS outer
WHERE outer.l_partkey = inner.l_partkey; AND outer.l_quantity <
inner.t1;
```



La requête R et son arbre algébrique

```
Agg1
Map : lineitem → (l_partkey, l_quantity)
Reduce : v = 0.2* avg(l_quantity)

Join1
Map : lineitem → (l_partkey, (l_quantity, l_extendedprice))
      part → (l_partkey, null)
Reduce : jointure

Join2
Map : inner → (l_partkey, v)
      outer → (l_partkey, (l_quantity, l_extendedprice))
Reduce : jointure

Agg2 ...
```

Traduction de R en MapReduce

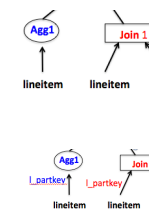
Coût communication !!

11

Constat

• Redondance

- Lectures
- partitionnement initial
- partitionnement final



```
Agg1
Map : lineitem → (l_partkey, l_quantity)
Reduce : 0.2* avg = v

Join1
Map : lineitem → (l_partkey, (l_quantity, l_extendedprice))
      part → (l_partkey, null)
Reduce : jointure

Join2
Map : inner → (l_partkey, v)
      outer → (l_partkey, (l_quantity, l_extendedprice))
Reduce : jointure

Agg2 ...
```

La traduction de R

12

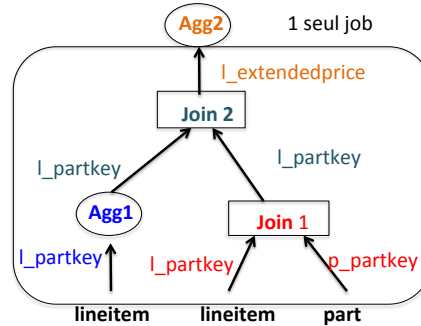
Traduction optimisée

Join

Map :
 lineitem → (p_partkey, (l_quantity, l_extendedprice))
 part → (l_partkey, null)
 Reduce :
 agg1 sur l_quantity
 join1 sur l_partkey = p_partkey
 join2 sur l_partkey = l_partkey

Agg 2

...

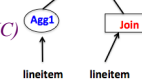


13

Redondance

- Lectures

→ *Input correlation (IC)*



- partitionnement initial

→ *transit correlation (TC)*



- partitionnement final :

→ *Jobflow correlation (JFC)*



Agg1

Map : lineitem → (l_partkey, l_quantity)
 Reduce : 0.2* avg = v

Join1

Map : lineitem → (l_partkey, (l_quantity, l_extendedprice))
 part → (l_partkey, null)
 Reduce : jointure

Join2

Map : inner → (l_partkey, v)
 outer → (l_partkey, (l_quantity, l_extendedprice))
 Reduce : jointure

Agg2 ...

La traduction de R

14

Traduction optimisée : gain

SQL → MR^{std} → MR^{optim}

| | | |
|----------------------------|----------------------------------|--|
| <i>Input correlation</i> | Partager les <u>Map</u> | <ul style="list-style-type: none"> Gain local (accès disque) communication <u>si map distant</u> |
| <i>Transit correlation</i> | Partager Map, mutualiser reduces | <ul style="list-style-type: none"> Gain local (accès disque) communication |
| <i>Jobflow correlation</i> | Mutualiser reduce | <ul style="list-style-type: none"> Gain local (accès disque) communication |

Gain

15

Traduction optimisée : étapes

A. Aplatisir l'arbre MR^{std} : parcours post-order

→ *séquence de tâches (PS-A-J-S)*

B. Réduire la séquence de tâches en appliquant 3+1 règles

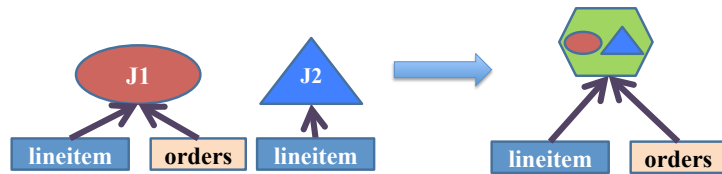
Phase 1 : appliquer règle 1 jusqu'à plus de tâches IC et TC

Phase 2 : appliquer les règles 2 à 3

16

Règle 1

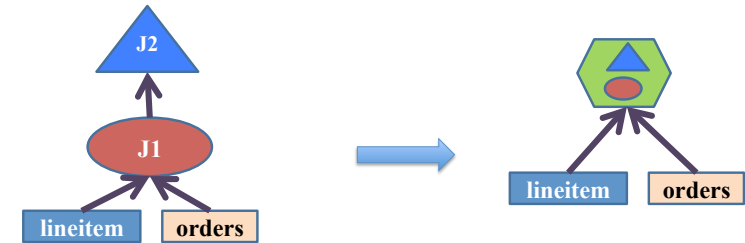
si IC (J1, J2) et TC (J1, J2) alors Fusionner (J1, J2)



17

Règle 2

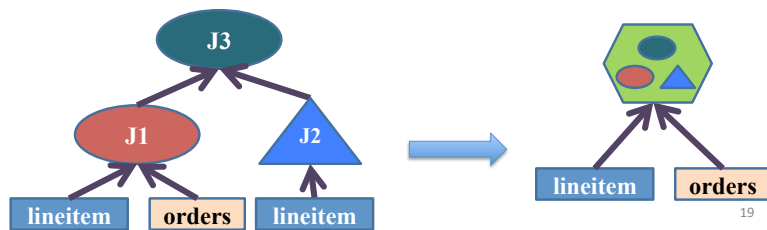
si J2 : agrégation, JFC(J1, J2) et J2 parent de J1 alors Fusionner (J1, J2)



18

Règle 3

si JFC(J3, J1), JFC(J3, J2), TC(J1, J2) et $J1 < J2 < J3$
alors exécuter J3 pendant *reduce* de J1, J2



19

Exemple

R(A,B) S(B,C) et T(B,D) trois schémas de relations

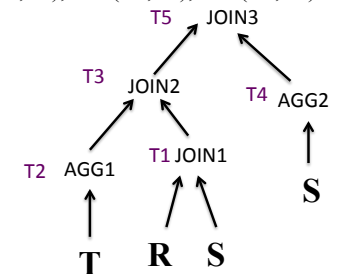
$Q = R \bowtie S \bowtie \text{agg}(T) \bowtie \text{agg}(S)$

Les corrélations : TC(T1, T4), JFC(T3, T1), JFC(T5, T3), JFC(T5, T4)

R1 si IC (J1, J2) et TC (J1, J2) alors Fusionner (J1, J2)

R2 Si J2 : agrégation, JFC(J1, J2) et J2 parent J1 alors Fusionner (J1, J2)

R3 si JFC(J3, J1), JFC(J3, J2), TC(J1, J2) et $J1 < J2 < J3$ alors exécuter J3 pendant *reduce* de J1, J2



Résultat : T2, T1435
(2 tâches)

20

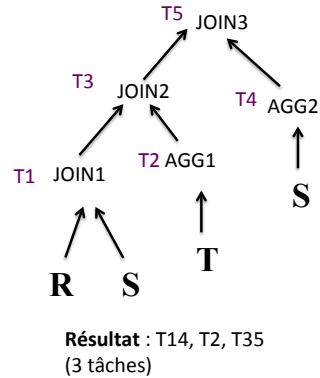
Exemple

R(A,B) S(B,C) et T(B,D) trois schémas de relations

$Q = R \bowtie S \bowtie \text{agg}(T) \bowtie \text{agg}(S)$

Les corrélations : TC(T1,T4), JFC(T3,T1), JFC(T5,T3), JFC(T5,T4)

- R1 si IC (J1, J2) et TC (J1, J2) alors Fusionner (J1, J2)
- R2 Si J2 : agrégation, JFC(J1 J2) et J2 parent J1 alors Fusionner (J1, J2)
- R3 si JFC(J3,J1), JFC(J3,J2), TC(J1,J2) et $J1 < J2 < J3$ alors exécuter J3 pendant *reduce* de J1,J2
- R4 si JFC(J3,J1), ~~JFC(J3,J2)~~ et $J2 < J1$ alors fusionner (J1, J3)



Autres techniques d'optimisation

- Logiques
 - Jointures n-aires en une seule tâche
 - Pousser sélection et projections
 - Réécriture de requêtes avec corrélation
- Basées sur le coût
 - Développements en cours