Master d'Informatique

BDR - 4I803 - Cours 8

Bases de données parallèles

1

Plan

Introduction

Architectures

Placement des données

Parallélisme dans les requêtes

Optimisation de requêtes

2

Introduction

- Les machines parallèles deviennent courantes et abordables
  - Les prix des microprocesseurs, de la mémoire et des disques ont fortement diminué
  - Les réseaux permettent d'accéder à des systèmes distants
- Les BD sont de plus en plus volumineuses (ex. entrepôts de données, multimedia, index google…)
- Principe des BD parallèles :
  - utiliser les machines parallèles (systèmes distribués) pour exploiter le parallélisme dans la gestion de données.

3

Parallélisme dans les BD

- Les données peuvent être partitionnées sur plusieurs disques pour faire des entrées/sorties en parallèle.
- Les opérations relationnelles (tri, jointure, agrégations) peuvent être exécutées en parallèle
  - Les données peuvent être partitionnées, et chaque processeur peut travailler indépendamment sur sa partition
- Les requêtes sont exprimées dans un langage de haut niveau (SQL), et traduites en opérateurs algébriques.
  - Facilite le parallélisme
- Les requêtes peuvent être exécutées en parallèle

4

Différents types de parallélisme

- Inter-requête : chaque proc. exécute une requête différente
- Intra-requête : plusieurs proc. exécutent (des morceaux de) la même requête
  - Inter-opérateur : chaque proc. exécute un sous arbre du plan d'exécution.
  - Intra-opérateur : plusieurs proc. exécute le même opérateur mais sur un sous-ensemble des données

5

Objectifs des BD parallèles

Contrairement à BD réparties, on ne s'intéresse pas à la localité par rapport à l'émetteur de la requête

- Améliorer les performances (temps de réponse)
- Améliorer la disponibilité (réplication)
- Approche cluster : réduire les coûts (utiliser plusieurs petites machines est moins cher qu'un gros ordinateur).
- Approche clouds : datacenters = n clusters « pay on-demand »

- Les SGBD parallèles sont utilisés pour :
  - Stocker de très grandes quantités de données
  - Effectuer des requêtes d'aide à la décision coûteuses
  - Permettre le traitement de transactions à haut débit

6

Architectures de BD parallèles

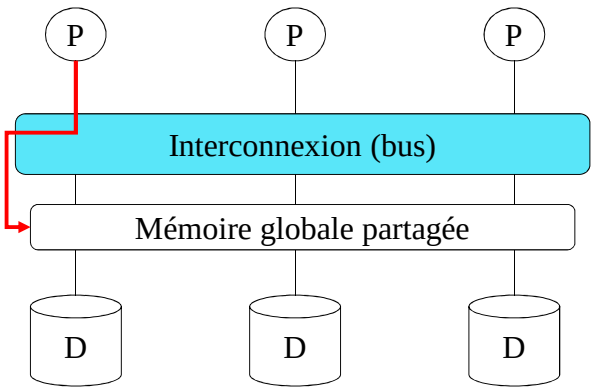
Moins on partage, plus on est extensible..

- Se classent selon ce qui est partagé :
  - Shared Memory Systems (partage de la mémoire)
  - Shared Disk Systems (partage du disque)
  - Shared Nothing Systems (pas de partage –sauf bus rapide, e.g. PC cluster)
  - Hybrid Systems (systèmes hybrides)

7

Mémoire partagée

- Tous les processeurs partagent le disque et la mémoire



8

Mémoire partagée

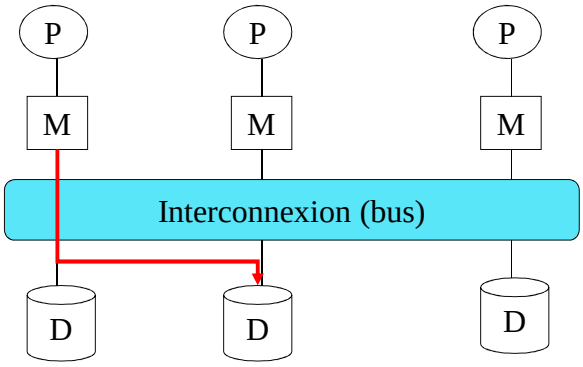
+ simplicité (comme si une seule mémoire)
+ équilibrage de charge (on peut choisir n'importe quel proc.)
+ parallélisme inter-requête (ajout de processeur) direct, intra-requête possible (assez simple)
- Peu extensible : limité à des dizaines de processeurs (à cause des accès concurrents à la mémoire)
- Cher : chaque processeur doit être lié à chaque module de mémoire
- Conflits d'accès à la mémoire (dégrade les performances)
- Pb de disponibilité des données en cas de défaut de mémoire (affecte plusieurs processeurs)

Utilisé dans plusieurs systèmes commerciaux : Oracle, DB2, Ingres

9

Disques partagés

- Ensemble de noeuds (processeurs et mémoire) ayant accès via un réseau d'interconnexion à tous les disques (les mémoires ne sont pas partagées).



10

Disques partagés

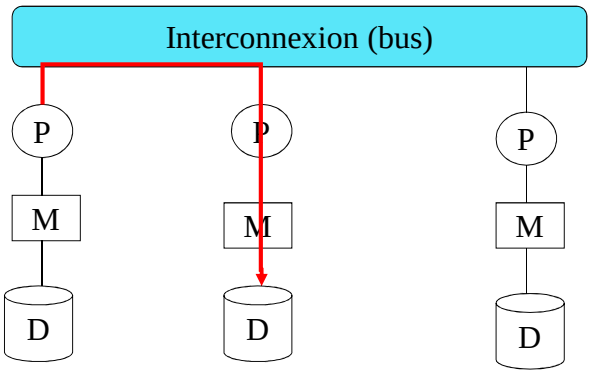
+ moins cher que systèmes mémoire partagée (interconnexion moins complexe).
+ extensibilité (centaines de processeurs) : les processeurs ont suffisamment de mémoire cache pour minimiser les interférences sur le disque.
+ bonne disponibilité (répli. et défauts de mémoire locaux à un processeur)
+ migration facile de systèmes existants (pas de réorganisation des données sur le disque)
- Assez complexe
- Pbs de performance :
  - Verrous distribués pour éviter les accès conflictuels aux mêmes pages,
  - maintien de la cohérence des copies,
  - l'accès aux disques partagés peut devenir un goulot d'étranglement

- Utilisé dans IMS/VS(IBM), VAX (DEC)

11

Aucun partage

- Les différents noeuds sont reliés par un réseau d'interconnexion. Chaque processeur a un accès exclusif à la mémoire et au disque (idem BD réparties).



12

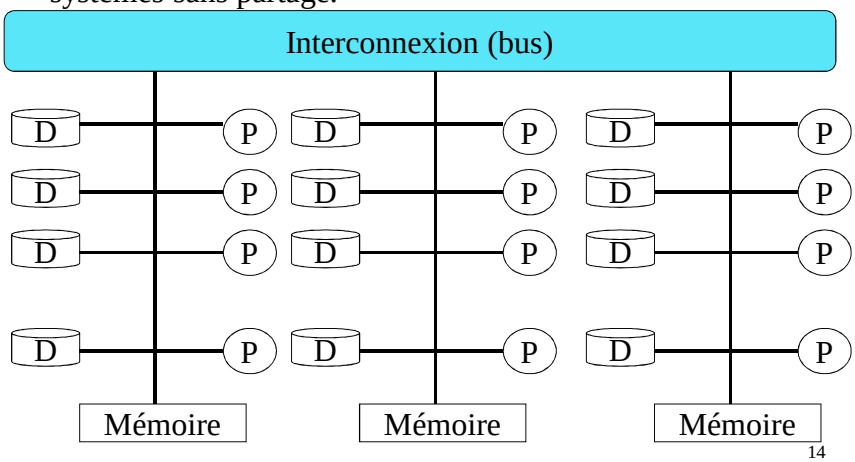
Aucun partage

+ Coût semblable à celui des disques partagés
+ Bonne extensibilité (milliers de noeuds)
+ Disponibilité par réplication des données
+ réutilisation des technique de BD réparties
+ Bien adapté au parallélisme intra-requête
- Plus complexe que les systèmes à mémoire partagée (mêmes fct. qu'en BD réparties)
- Répartition de charge difficile à mettre en oeuvre, à cause de la répartition statique des données (le proc. n'accède qu'à SON disk)
- Ajouter des noeuds entraîne une réorganisation des données (pour permettre la répartition de charge), sauf si réplication totale
- Utilisé dans ICL Goldrush, Teradata DBC, EDS, solutions intergicielles pour clusters (*Leg@net/Refresco*…)

13

Systèmes hybrides

- Combinaison de systèmes à mémoire partagée et de systèmes sans partage.



14

Systèmes hybrides

- Chaque noeud individuel est un système à mémoire partagée.

+ Combine la souplesse et les performances des systèmes à mémoire partagée avec les capacités d'extensibilité des systèmes sans partage.

+ La communication au sein de chaque noeud est efficace (mémoire partagée).

+ Bon compromis entre répartition de charge et passage à l'échelle.

15

Parallélisme inter-requêtes

- Forme la plus simple du parallélisme
- Les requêtes (et les transactions) s'exécutent en parallèle
- Augmente le débit de transactions (utilisé pour augmenter le nombre de transactions par seconde)
- Plus difficile à implémenter sur les architectures disque partagé et sans partage
  - La coordination des verrouillages et des journaux s'effectue par envois de messages entre processeurs
  - Les données d'un buffer local peuvent avoir été mises à jour sur un autre processeur
  - Nécessité de maintenir la cohérence des caches (les lectures et écritures dans le buffer doivent concerner la version la plus récente.)

16

Cohérence du cache

- Protocole pour les architectures à disques partagés :
  - Avant de lire/écrire une page, la verrouiller en mode partagé/exclusif
  - Une page verrouillée doit être lue du disque (version la plus récente)
  - Avant de déverrouiller une page, elle doit être écrite sur le disque (si elle a été modifiée)
- Des protocoles plus complexes existent, limitant les lectures/écritures sur disque.
- Des protocoles semblables existent pour les architectures sans partage.

17

Parallélisme intra-requêtes

- Exécuter une seule requête en parallèle sur plusieurs processeurs (important pour les requêtes « longues »)
- Deux formes de parallélisme :
  - **Inter-opérateurs** : exécuter plusieurs opérations en parallèle (longue= complexe)
  - **Intra-opérateurs** : paralléliser l'exécution de chaque opération dans la requête (longue= sur bcp de données)

18

Parallélisme inter-opérateurs

- Deux formes :
  - **Pipeline** : plusieurs opérateurs successifs sont exécutés en parallèle, le résultat intermédiaire n'est pas matérialisé. Gain de place mémoire et minimise les accès disque. Risque d'attente des résultats de l'opé. précédent.
  - **Parallélisme indépendant** : les opérateurs sont indépendants, pas d'interférence entre les processeurs. Le résultat est matérialisé = coût

19

Parallélisme intra-opérateurs

- Décomposer un opérateur en un ensemble de sous-opérateurs, chacun s'exécutant sur une partition de la relation.

  Exemple pour la sélection :

$$\sigma_S (R) \equiv \sigma_{S1} (R_1) \cup \sigma_{S2}(R_2) \cup \ldots \cup \sigma_{Sn}(R_n)$$

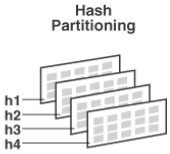
si S porte sur l'attribut de partitionnement, on peut éliminer certains Si

- Pour les jointures, on utilise les propriétés du partitionnement : par ex. si R et S sont partitionnées par hachage sur l'attribut de jointure, et s'il s'agit d'une équi-jointure, on peut partitionner la jointure en jointures indépendantes.
- => dépend de la manière dont on partitionne les données

20

Partitionnement des données

- Répartir les données sur les disques permet de réduire le temps d'accès aux données par parallélisme (pas par localité, car utilisateur pas affecté à nœud)
- Partitionnement horizontal : les n-uplets d'une relation sont répartis sur les différents disques (pas de partitionnement de n-uplet).
- Différentes techniques de partitionnement (nb de disques = n) :
  - **Round-robin** : le $i^{ème}$ n-uplet inséré dans la relation est stocké sur le disque *i mod n*
  - **Partitionnement par hachage (hash partitioning) :**
    - Choisir un ou plusieurs attributs comme attributs de partitionnement
    - Appliquer une fonction de hachage (renvoyant un entier de 0 à n-1) à ces attributs
    - Stocker le n-uplet sur le disque correspondant au résultat de la fonction de hachage



21

Partitionnement des données (suite)

- **Partitionnement par intervalles (range partitionning)** : répartit les n-uplets en fonction des intervalles de valeurs d'un attribut
  - Choisir un attribut pour le partitionnement

    Choisir un vecteur de partitionnement $[v_0, v_1, \ldots, v_{n-2}]$

    Soit v la valeur de l'attribut de partitionnement.
    - Les n-uplets tq $v<v_0$ vont sur le disque 0
    - Les n-uplets tq $v \geq v_{n-2}$ vont sur le disque n-1
    - Les n-uplets tq $v_i < v \leq v_{i+1}$ vont sur le disque i+1

22

Comparaison

- Comparer sur les opérations :
  - Parcours de la relation (scan)
  - Sélection sur égalité (ex: R.A=15)
  - Sélection sur intervalles (ex: $10 \leq R.A < 25$)
- **Round-robin** :
  - Convient bien au parcours séquentiel de relations
  - Bonne répartition des n-uplets sur les disques (bonne répartition de charge)
  - Mal adapté aux requêtes avec sélection (pas de regroupement, les données sont dispersées)

23

Comparaison (suite)

Partitionnement par hachage

- Efficace pour les accès séquentiels
  - Si on a une bonne fonction de hachage, et si le(s) attribut(s) de partitionnement est (sont) clé, il y a bonne répartition des n-uplets sur les disques, et une bonne répartition de la charge pour rechercher les données
- Efficace pour les sélections par égalité sur l'attribut de partitionnement
  - La recherche peut se limiter à un seul disque
  - Possibilité d'avoir un index local sur l'attribut de partitionnement
- Mal adapté aux sélections sur intervalles, car il n'y a pas de regroupement.

24

Comparaison (suite)

Partitionnement par intervalle

- Les données sont regroupées par valeurs de l'attribut de partitionnement
- Efficace pour les accès séquentiels
- Efficace pour les sélections par égalité sur l'attribut de partitionnement
- Bien adapté aux sélections sur intervalles (seul un nombre limité de disques est concerné)
  - Efficace si les n-uplets du résultat se trouvent sur peu de disques
- Risque de répartition inégale sur les disques (*data skew*)

25

Partitionnement d'une relation

Compromis :

- Ne pas partitionner une relation qui tient sur un seul disque (minimiser les transfert)
- Utiliser l'ensemble des disques pour un partitionnement (maximiser le parallélisme)
- Ne pas partitionner plus que nécessaire : si une relation tient sur *m* blocs et qu'il y a *n* disques, partitionner sur *min(n,m)* disques

26

Gestion des répartitions non uniformes

- Certains algorithmes de répartition risquent de donner lieu à une répartition inégale des données (bcp sur un disque, peu sur un autre), ce qui peut dégrader les performances:
  - Mauvais choix de vecteur de partition
  - Mauvais choix de fonction de hachage
  - Répartition inégale des valeurs d'un attribut de partitionnement
- Solutions (partitionnement par intervalles) :
  - Créer un vecteur de partitionnement équilibré (trier la relation sur l'attribut de partitionnement et diviser en sous-ensembles de même taille)
  - Utiliser des histogrammes
  - Coûteux mais peut valoir le coup

27

Traitement parallèle des opérateurs relationnels

- Suppositions :
  - Requêtes en lecture seule
  - Architecture sans partage
  - N processeurs, et N disques (D_1 est associé à P_1)
- Les architectures sans partage peuvent être simulées efficacement sur des architectures à mémoire partagées ou à disques partagés. Les algorithmes peuvent être appliqués, avec différentes optimisations, sur ces architectures.

28

Tri parallèle

Range-partitioning sort :

- Choisir les processeurs P_0,… P_m, avec m<n, pour faire le tri.
- Créer des vecteurs de partition par intervalle avec m entrées, sur l'attribut de jointure
- Redistribuer la relation selon cette partition :
  - tous les n-uplets du $i^{ème}$ intervalle sont envoyés au processeur P_i
  - P_i stocke temporairement les n-uplets sur son disque D_i
- Chaque processeur trie sa partition de la relation localement (en parallèle, sans interaction avec les autres processeurs)
- On fusionne les différents résultats (les partitions sont déjà ordonnées donc simple concaténation).

29

Tri parallèle

Trifusion parallèle externe :

- La relation est partitionnée sur les n disques (RR).
- Chaque processeur trie localement ses données.
- La fusion est parallélisée :
  - Les partitions triées sur chaque processeur sont partitionnées par intervalle sur les différents processeurs
  - Chaque processeur fusionne les données qu'il reçoit à la volée.
  - Les différents résultats sont concaténés

30

Jointure parallèle

- Principes :
  - La jointure consiste à tester les n-uplets deux par deux et à comparer la valeur des attributs de jointure.
  - Le parallélisme consiste à répartir ces tests sur des processeurs différents, chacun calculant une partie de la jointure localement.
  - Les résultats sont ensuite rassemblés.

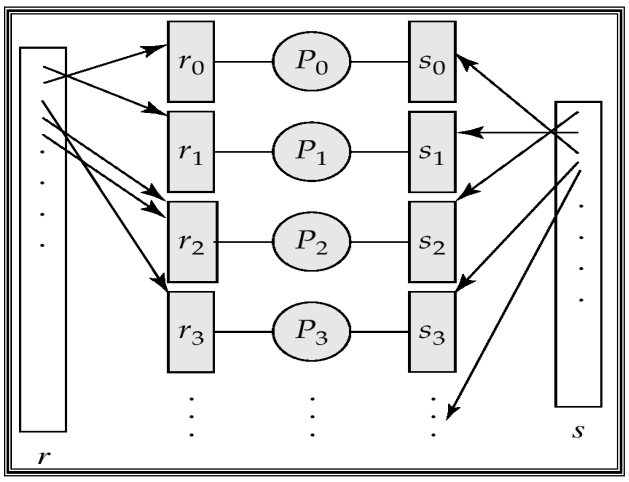
31

Jointure par partitionnement

- Partitionner (attention, coûteux) les deux relations (par hachage ou par intervalle) en n partitions, sur l'attribut de jointure, en utilisant la même fonction de hachage ou le même vecteur de partitionnement.
- Les partitions r_i et s_i sont envoyées au processeur P_i.
- Chaque processeur calcule la jointure entre r_i et s_i (avec n'importe quel algorithme)
- Les résultats sont concaténés.
- Attention : difficile de contrôler la taille de r_i et s_i. Equilibrage de charge difficile

32

Jointure par partionnement



33

Boucles imbriquées en parallèle

- Les relations R et S sont fragmentées en m et n fragments respectivement.
- Envoyer (en parallèle) les m fragments de R sur les n nœuds où se trouvent des fragments de S.
- Faire la jointure sur ces nœuds.(en parallèle)
- Concaténer les résultats.

34

Jointure parallèle par hachage

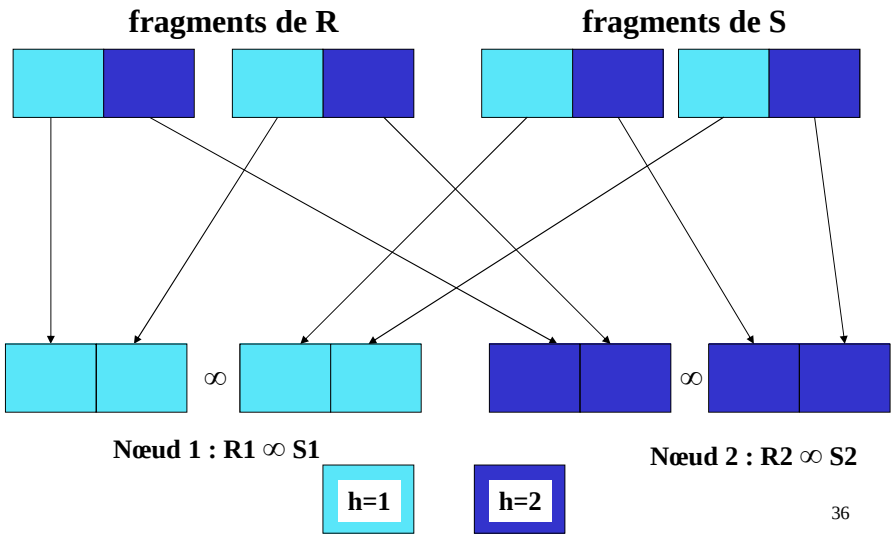
- Partitionner (en parallèle) les relations R et S en k partitions à l'aide d'une fonction de hachage h appliquée à l'attribut de jointure tq

$$R \infty S = U_{(i=1..k)} (Ri \infty Si)$$

- Les partitions de R et de S ayant même valeur de hachage sont envoyées sur le même processeur.
- Faire les jointures sur chaque processeur en parallèle (k jointures en parallèle)
- Concaténer les résultats

35

Jointure parallèle par hachage



36

Optimisation de requêtes

- Semblable à l'optimisation de requêtes dans les BD réparties.
- Espace de recherche
  - Les arbres algébriques sont annotés pour permettre de déterminer les opérations pouvant être exécutées à la volée (pipeline). Génération d'arbres linéaires gauche, droite, en zig-zag, touffus.
- Modèle de coût
  - Dépend en partie de l'architecture
- Stratégie de recherche
  - Les mêmes qu'en environnement centralisé (les stratégies aléatoires sont mieux adaptées en raison de la taille de l'espace de recherche, plus grand car on considère plus de possiblité de paralléliser)

37

Map Reduce

Hadoop et Map Reduce

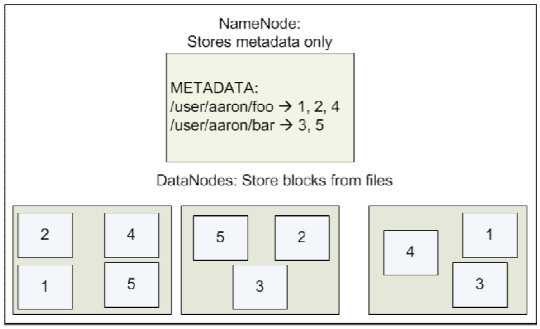
(source : Arnault Jeanson)

- projet Open Source écrit en java, distribué Apache. Adapté au stockage et traitement par lots de très grandes quantités de données
- Utilisé par Yahoo! ou Facebook.
- Système de fichiers HDFS :
  - permet de distribuer le stockage des données
  - analyses très performantes sur ces données grâce au modèle MapReduce
  - distribue une opération sur plusieurs nœuds dans le but de paralléliser leur exécution.

39

Stockage HDFS

- Système de fichiers distribués:
  - blocs d'informations sont répartis et répliqués (configurable) sur les différents nœuds du cluster.



  - un ensemble de commandes est mis à disposition pour interagir avec ce système

40

Exemple n°1 : Log de serveur web

- Données : un fichier de log. Une ligne contient :
  (User, URL, timestamp, complement-info)
- Chargement des données dans un SGBD
  - Nettoyer les données
  - Extraction
  - Vérification
  - Spécifier le schéma
- Chargement dans un système Hadoop
- Directement sans faire les opérations préliminaires ci-dessus.
  - Contrepartie : les opérations de nettoyage, extraction, vérification, spécification de schéma devront être faites **pendant l'exécution**, mais seulement sur la **portion** des données qu'on manipule, pas sur toutes les données.

Exemple de requêtes

- Requête 1
  - Trouver les enregistrements selon un critère : User ou URL ou timestamp
  - Traitement facilement parallélisable
  - Bien adapté aux solutions NoSQL
- Requête 2
  - Trouver les paires d'utilisateurs qui accèdent la même URL
    - Auto jointure
    - Moins évident à paralléliser
- Requête 3
  - Age Moyen des utilisateurs accédant la même url
  - Besoin de cohérence assez faible.

Exemple 2 : Graphe social

- Données: schéma composé de 2 tables
  - Profile(u, nom, age, genre)
  - Lien(u, v) u et v sont amis
- Requêtes
  - Trouver les amis des amis de l'utilisateur u1
  - Plus généralement : traversé d'un graphe
    - pas exprimable en SQL
      - Rmq: SQL se limite aux traversées d'arbre
        - » Cf la clause *connect by* d'oracle
    - Etendre SQL avec la récursion

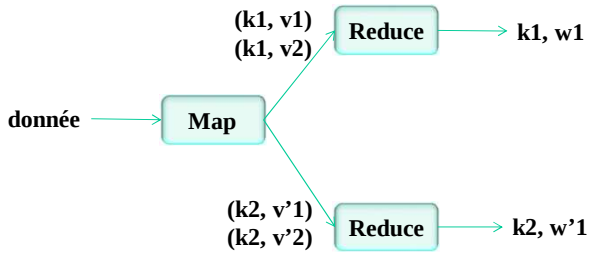
Système Map Reduce

- Un système Map Reduce apporte
  - un moteur pour répartir les traitements sur des données réparties
  - Répartition des traitements transparente pour l'utilisateur
  - la tolérance aux pannes est transparente pour l'utilisateur
    - Reprise partielle des sous tâches défectueuses
- Pas de modèle de données
  - les données sont dans des fichiers
  - Stockage réparti: dans un système de fichiers tq HDFS
- L'utilisateur défini plusieurs fonctions
  - map()
  - reduce()
  - reader() pour lire un fichier et construire des enregistrements
  - writer() pour écrire les enregistrements dans un fichier
  - combine()

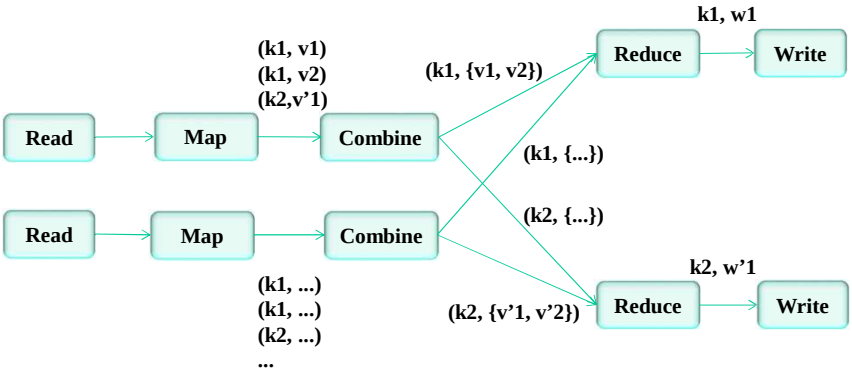
Principe de Map Reduce

- Traite des données quelconques
  - Fichier: liste d'éléments (un élément par ligne)
- Map : diviser le problème en sous problèmes
  - Map(élément) → 0 ou n couples (clé, valeur)
- Reduce : est évalué pour chaque sous problème
  - Reduce (clé, liste de valeurs) → 0 ou n élément

Architecture fonctionnelle de Map Reduce

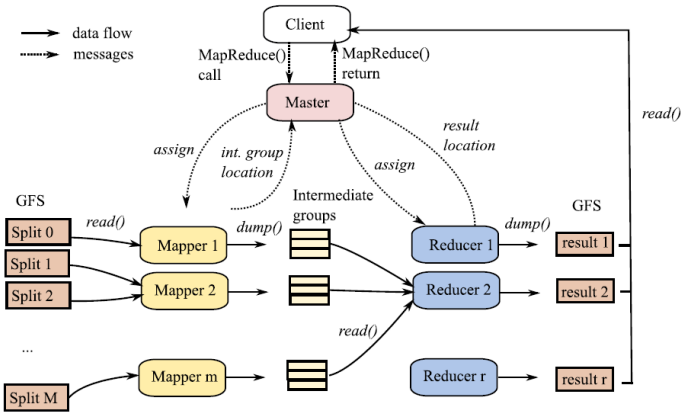


Architecture Générale



Combine: prétraitement (regroupement et tri) avant d'invoquer la fonction reduce
Combine: local, Reduce :global
Map : travaille sur un bloc de fichier d'entrée. Combine agrège sur les blocs de la machine

MR : Modèle d'exécution



Execution MR dans Hadoop

- Etape 1 : Reader, Map, Combine
  - Lire les fichiers d'entrée, pour chaque élément lu
    - Produire une liste de couples
  - Combine (avec tri local)
    - Regrouper les couples par clé
- Etape 2 : Shuffle : Transmission des couples
  - répartir les couples sur plusieurs machines
    - Par hachage ou trié
- Etape 3 : Merge, Combine, Reduce, Write
  - Fusionner les données pour avoir une seule liste de valeurs par clé
    - Fusion en plusieurs passes si nécessaire (selon ram dispo)
    - Invoquer Combine à chaque passe
  - Invoquer reduce pour chaque valeur de clé puis écrire le résultat dans un fichier

Exemple 1

- Données
  D (clé, champ)
- Requête (grep)
  - Select * from D where champ like '%xyz%'
- MR
  - Map(ligne)
    - (clé, champ) ← split(ligne)
    - If (champ contient 'xyz')
    - then ajouter (clé, champ) en sortie
  - Pas besoin de reduce()

Exemple : word count

- Afficher le nombre d'occurrences de chaque mot d'un fichier
  - Fichier très grand
- Données
  - un fichier texte
- Map
  - Argument d'entrée : une ligne du fichier
  - Retourne: une liste de couples (mot, null)
  - Corps:
    - Extraire les mots de la ligne
    - Construire (mot_1, null), (mot_2, null), ….
- Reduce
  - Argument d'entrée : (mot, liste)
    - La liste est {null, null, …}
  - Retourne le couple (mot, longueur de la liste)

Exemple 2

- Données
  - **Document** (URL, contents)
  - **Ranking** (pageURL, pageRank, avgDuration)
  - **UserVisit** (sourceIP, destURL, visitDate, adRevenue, userAgent, countryCode, languageCode, search Word, duration)

Requetes d’analyse : sélection

- Sélection
  - select pageURL, pageRank
  - from Rankings
  - where pageRank > x
- MR
  - Map(ligne de Ranking)
    - (pageURL, pageRank, avgDuration) ←Split(ligne)
    - If pageRang > X
    - Then ajouter (pageURL, pageRank) en sortie
  - Pas de reduce

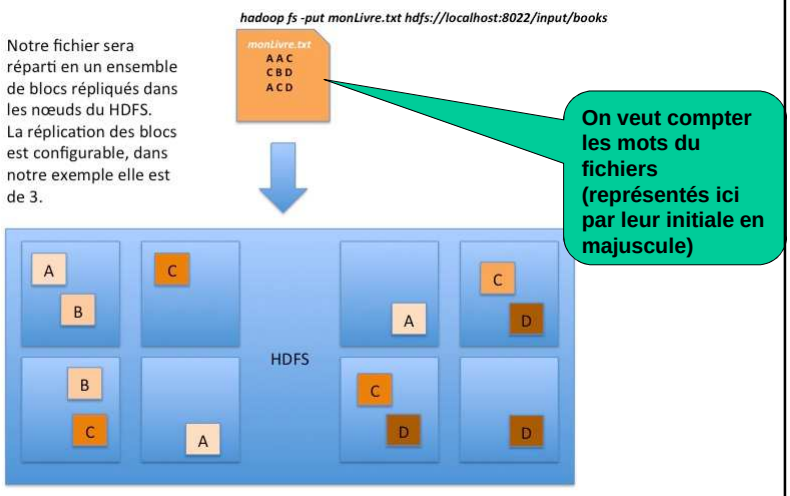
Requête d’analyse : agrégation

- Agrégation
  - select sourceIP, sum(adRevenue)
  - from UserVisit
  - group by sourceIP
- MR
  - Map
    - Ajouter (sourceIP, adRevenue)
  - Combine
    - Regrouper par sourceIP et somme des adRevenue
  - Reduce :
    - Somme des adRevenue

Requête d'analyse : jointure

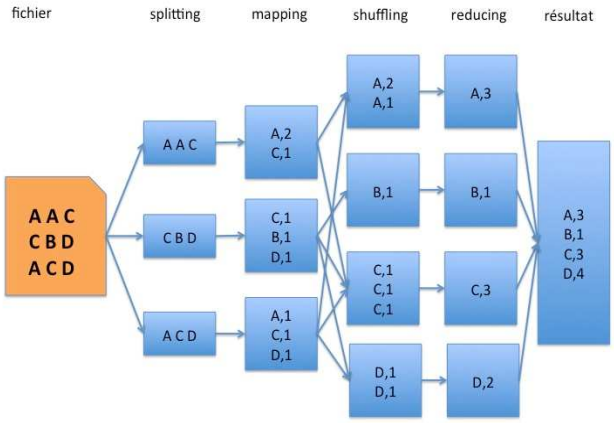
- Jointure
  - Select sourceIP, avg(pageRank), sum(adRevenue)
  - From Ranking r, UserVisit v
  - Where r.pageURL = v.destURL
  - And visitDate between jan-2011 and jan-2012
  - Group by sourceIP

Map Reduce : exemple



56

1. le fichier est découpé (splitting) en différents blocs répartis sur les noeuds
2. le mapping consiste à envoyer le traitement de comptage sur chaque bloc
3. le shuffling consiste à répartir ces informations en fonction de la clé
4. le reducing consiste à agréger les informations récupérées



57

Conclusion

- Les BD parallèles améliorent
  - Les performances,
  - La disponibilité
  - L'extensibilité

Tout en réduisant le ratio prix/performance.

Plusieurs systèmes commercialisés.

Hadoop et map reduce très à la mode

Pbs :

choisir la meilleure architecture

améliorer les techniques de placement de données (éviter les répartitions inégales)

58