

Gestion de données massives

Emmanuel Coquery

Outline

- 1 Introduction
- 2 Distribution
- 3 Calcul fonctionnel
- 4 Partitionnement
- 5 Industrialisation
- 6 Hébergement des données
- 7 Conclusion

Données massives (BigData) ?

Wikipedia

" [...] ensembles de données devenus si volumineux qu'ils dépassent l'intuition et les capacités humaines d'analyse et même celles des outils informatiques classiques de gestion de base de données ou de l'information."

Mais encore . . .

Les 3+2 "V"

- Volume ← dans cette présentation
- Vitesse
- Hétérogénéité (Variety)
- Véracité
- Valeur

Mais encore . . .

Les 3+2 "V"

- Volume ← dans cette présentation
- Vitesse
- Hétérogénéité (Variety)
- Véracité
- Valeur

Volume

Grandes quantités de données, variable selon l'usage

- serveur PostgreSQL bien dimensionné -> plusieurs To de données
 - mais dépend de l'utilisation: on évitera de multiplier les jointures
- en datamining: peut être difficile pour quelques centaines de Mo

Dans certains cas les données peuvent dépasser 10 To (voir des Po)

Vitesse (Velocity)

Quantités de nouvelles données à traiter

- systèmes de *streaming*: requêtes continues, *complex event processing*, etc
- stockage
 - écriture brute
 - e.g. peut-on écrire 20 000 message/s ?)
 - écriture précédée d'un traitement
 - e.g. indexation, data cleaning

Hétérogénéité (Variety)

Variations dans la structure des données, e.g.:

- nombreux attributs NULL
- structure JSON *schema-less*
- nombreux schemas à intégrer dans une source unique

Ce dont on ne parlera pas

- HPC
- Streams
- Intégration
- Véracité
- Valeur

Passer à l'échelle - Verticalement

Augmenter la puissance de la machine qui fait les traitements, dans un serveur:

- plusieurs dizaines de To de disque
- plusieurs To de RAM
- plusieurs dizaines de cores

C'est beaucoup, mais

- c'est (très) cher
- souvent peu évolutif
- comment le mutualiser ?

Passer à l'échelle - Horizontalement

Distribuer données et calcul

- sans limite en pratique (*i.e.* $> 10\,000$ machines)
- évolution plus simple

Mais

- le traitement doit pouvoir être distribué
- multiplication des communications
- complexe à mettre en place

Distribuer

Données

- réplication: fiabilité
- répartition: passage à l'échelle
 - 1 lame: au max $24 \times 12 \text{ To} = 288 \text{ To}$ (sans raid ...)

Traitements

- puissance de calcul
- répartition sur les coeurs de calcul
- fiabilité: pouvoir refaire ailleurs en cas d'échec

Pousser les données vers le calcul ou le calcul vers les données ?

- HPC: intensif en calcul
 - On amène la donnée vers la capacité de calcul
 - Systèmes de fichiers distribués
 - Systèmes de caches
- BigData: intensif sur les accès aux données
 - Chaque emplacement de stockage de données dispose de puissance de calcul
 - Importance du placement des données en fonction du traitement
- Convergence sur le long terme, mais les technologies sont encore trop différentes

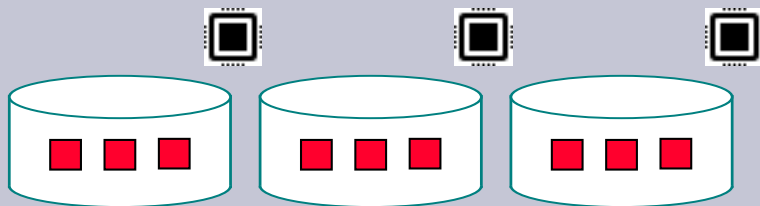
Programmation fonctionnelle

- Immutabilité des structures
 - Pas besoin de propager de mises à jour
- Paramétrisation naturelle des traitements
 - Facilite l'expression de transformation

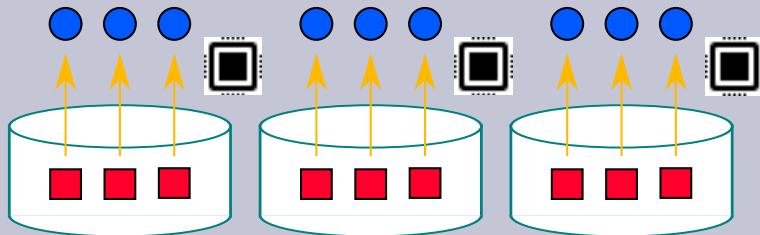
Transformation de collections: un classique de programmation fonctionnelle

- Données souvent organisées en collections
 - tuples, documents, triplets, *etc*
- APIs bien connues de transformation de collections:
 - `filter`, `map`, `groupBy`
 - `union`, `intersection`, `diff`, `zip`
 - `reduce / fold`

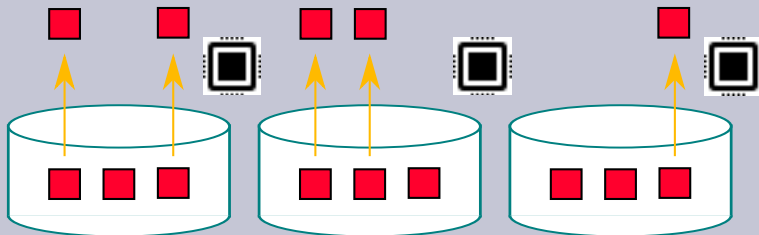
Distribution des données → du traitement



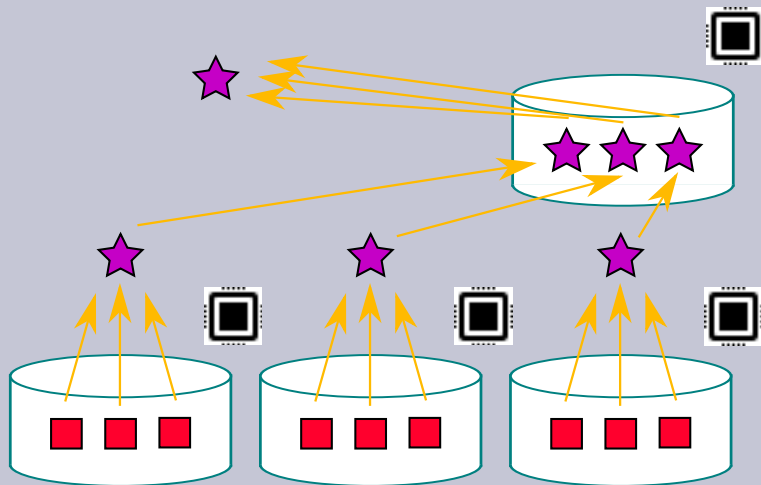
Distribution de map



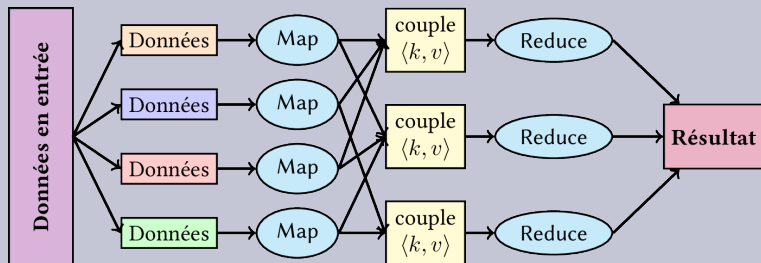
Distribution de filter



Distribution de reduce



Map/Reduce



<https://commons.wikimedia.org/wiki/File:Mapreduce.png>

Répartir les données

- Équilibre: éviter une majorité des données sur quelques noeuds
- Réfléchir au traitements et à leur parallélisation
 - Répartition du traitement après sélection
 - Colocalisation de certaines données (e.g. pour faciliter une jointure)

Clé de partitionnement = une des clés du succès

- Clé de partitionnement: valeur utilisée pour déterminer où placer une donnée
- Souvent déterminée à l'initialisation, au moment du chargement de données initial
- Souvent difficile, voir impossible à changer après coup

Charge réseau

Cas typique: phase de shuffling (entre map et reduce)

- Si les clés de reduce sont distribuées en termes de stockage
 - *i.e.* clé reduce $\neq$ clé partitionnement données
- et que l'on filtre peu la collection
- beaucoup d'échange au niveau réseau pour colocaliser les valeurs dans les reducer

Compromis

Coût (calcul de partitionnement, stockage)

- e.g. répartition d'observation astronomiques en zones du ciel
- e.g. dénormalisation en pré-calculant une jointure
 - ! mises à jour !

v.s. gain (temps de calcul)

Industrialiser, c'est quoi (ici) ?

- Automatiser le maximum de processus
 - Le cas échéant: préparation du contexte de calcul (cluster)
 - Import des données
 - Déploiement du code et lancement
 - Monitoring
- Anticiper les erreurs communes
 - e.g. rupture de connexion

Tentation: faire à la main

Pourquoi ?

- Processus compliqués
- On ne les fait pas souvent
 - Pourquoi passer en plus du temps à les automatiser ?

Mais ...

- Les cycles essais-erreurs sont coûteux
- On peut tomber dans le syndrome
"j'attends devant mon écran que ça se passe"

S'équiper

Outils pour faciliter l'automatisation

- Copie avec rsync
- Installation système avec ansible, chef, etc
- Déploiement de (cluster de) VM avec openstack
- Utilisation d'outils de *build & deploy*, e.g. maven, npm, etc.
- Éventuellement outils d'intégration continue (e.g. gitlab-ci)

Ne pas refaire: analogie avec Makefile

Makefile

- Production de fichier: prend du temps
- On ne refait un fichier que si ses dépendance ont été mises à jour

Gestion de données massives

Idem, mais:

- Fichiers distribués, non stockés localement
- Les dépendances parfois moins claires

Pouvoir se détacher: surveillance

Il faut prévoir de se "détacher" du travail:

- fermer sa connexion, son ordinateur, etc
- tout en restant au courant de ce qui se passe:
- logs & surveillance de ces logs
 - résumé de situation
 - alertes, e.g. mail

Data Management Plan

- la **description des données** collectées et/ou créées,
- les standards, **formats** et méthodologies appliqués sur le paquet de données,
- les questions d'ordre **éthiques**, de **propriété intellectuelle** et de restrictions (les éventuelles périodes d'embargo),
- les prévisions pour le partage et l'**ouverture des données**,
- et la stratégie de la **préservation à long-terme** (archivage).

https://fr.wikipedia.org/wiki/Plan_de_gestion_des_donn%C3%A9es

Données volumineuses:

- sur un disque dur ?
- externe ?
- plusieurs disques durs ?
- dans un *data center* ?

Politique de conservation:

- scratch (jetées après utilisation) ?
- à conserver ?
- quel niveau de sûreté ?
 - # rélicats ?

Mise à disposition

- quel niveau de sensibilité ?
- téléchargeables ?
 - ou copie sur disque ?
- chiffrement (des backups) ?

Conclusion: gérer des données massives, c'est

- Passer à l'échelle via la distribution
- Exprimer les traitements avec un style fonctionnel plus facilement parallélisable
- Répartir les données en fonction des traitements
- Industrialiser et surveiller les processus
- Penser à son *Data Management Plan*