Big Data : enjeux, stockage et extraction

Module R6.01 - TP 2 : MapReduce

2024

L'algorithme Map / Reduce vu d'avion

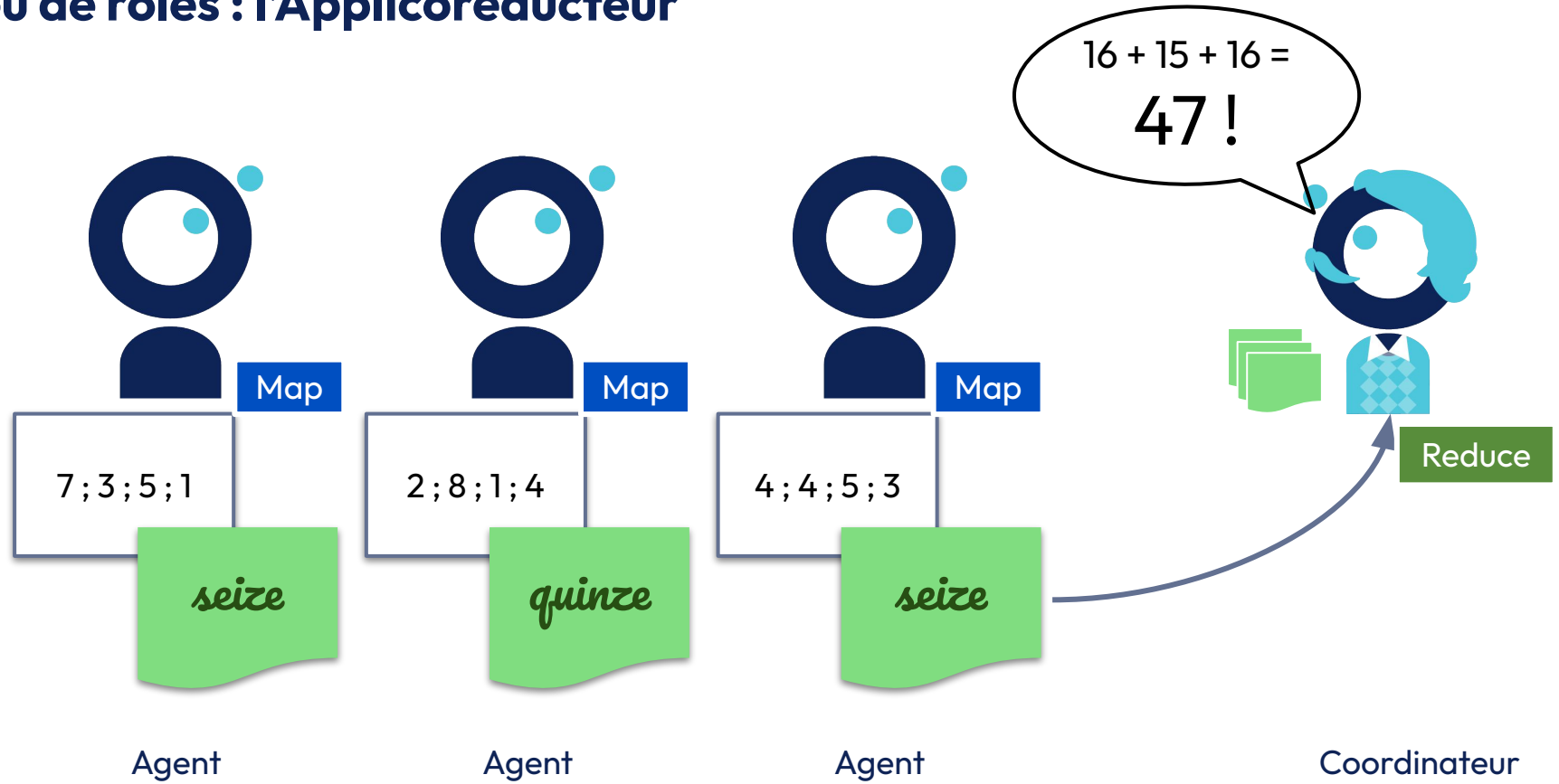
Map / Reduce est un algorithme de traitement parallèle sur des données volumineuses

- Adapter la logique de son traitement au cadre Map / Reduce permet de faciliter la parallélisation par la plateforme

Fonctionnement :

- Les données à traiter sont découpées en morceaux appelés **splits**. Les splits sont traités en parallèle lors d'une phase de **Map**. A la fin, une phase de **Reduce** combine les résultats intermédiaires

Jeu de rôles : l'Applicoréducteur



Que s'est-il passé ?

Où avons-nous perdu du temps ?

Dans la distribution initiale des fiches, avant le démarrage du "traitement"

- Un traitement distribué a un coût incompressible lié lié au démarrage agents d'exécution avec le coordinateur

Dans la transmission des Post-It vers le coordinateur

- L'échange des informations entre agents a aussi un coût, proportionnel à son volume
- En pratique, il s'agit principalement des entrées/sorties : réseau, disque, ...

Dans l'écriture et la lecture des nombres en toutes lettres sur les Post-It

- La transmission d'informations nécessite une mise : c'est la sérialisation/désérialisation
- Elle a un coût proportionnel au volume d'informations, et à la complexité de la représentation

Que se serait-il passé si...

... nous étions 2 fois plus nombreux, avec 2 nombres par fiche au lieu de 4 ?
(pour un même nombre total de nombres à additionner)

Plus de cerveaux n'auraient pas suffi à compenser le temps de démarrage supérieur et les échanges plus nombreux

- Au-delà d'un certain nombre, ajouter plus de puissance de calcul devient contre-productif
- Cela arrive quand le coût de démarrage et de synchronisation est supérieur à celui de l'exécution réelle du traitement

Que se serait-il passé si...

... il y avait 400 nombres par fiche à additionner ?

Cette fois c'est le calcul qui aurait été le goulet d'étranglement, à cause du volume à traiter par chacun(e)

- Le facteur limitant n'est plus les échanges, mais les capacités de chaque agent : puissance de calcul, mémoire nécessaire pour mémoriser le résultat transitoire. On passe d'un problème ***I/O bound*** à ***CPU bound***

Le temps de calcul total devient à peu près proportionnel au volume de données à traiter

- Mais à partir d'un certain volume ou d'une certaine complexité, un nouveau problème peut apparaître : le besoin de stocker et relire des calculs intermédiaires qui ne "tiennent" plus en mémoire. On revient en *I/O bound*

Que se serait-il passé si...

... l'un(e) d'entre vous avait reçu un appel urgent pendant son calcul ?

Sa partie du traitement aurait été mise en pause le temps que l'appel se termine

- Le résultat final aurait mis plus de temps à arriver
- Le coordinateur surveille l'état des agents, et veille à équilibrer la charge de travail
- Si un agent est indisponible pendant trop longtemps, il peut décider de donner sa tâche à un autre qui a de la disponibilité. On perd alors le temps de la première tentative, celui de décision du coordinateur, et celui du deuxième agent qui doit refaire une partie du calcul du premier

La probabilité d'un tel événement augmente avec le nombre d'agents

- Le système distribué doit être conçu pour faire aboutir le traitement quoi qu'il arrive, quitte à perdre un peu de temps

Que se serait-il passé si...

... il avait fallu additionner séparément nombres pairs et multiples de 3 ?

A volume de données égal, on aurait perdu du temps à traiter 2 fois certains nombres : les multiples de 6

- La présentation des données en entrée n'est pas très adaptée au nouveau besoin, c'est un sujet de modélisation
- Il aurait été alors plus efficace de classer d'abord les nombres : multiples de 2 seul, de 3 seul, de 6, et d'aucun de ces facteurs ⇒ modélisation par l'usage

Quand des besoins contradictoires coexistent, on peut être amené à faire des compromis

- Décider qu'une modélisation est plus importante qu'une autre
- Trouver une modélisation intermédiaire qui satisfasse partiellement chaque besoin
- Avoir plusieurs versions des données, modélisées différemment

Que se serait-il passé si...

... il avait fallu traiter des symboles, qu'une autre personne nous aurait traduits en chiffres (ex. 🍀 = 4) ?

Notre traitement aurait été dépendant d'un service tiers : la traduction

- Notre performance dépendrait de celle du service, qui pourrait être surchargé
- Si le service est indisponible, tous les agents sont bloqués en attendant, ou dans le pire des cas le traitement ne peut pas se faire
- On parle de point unique de faiblesse (*SPOF* ou *Single-Point Of Failure*), il est important de bien les identifier et d'avoir un "plan B" (cache, ...)

Installation du TP

Récupérer le répertoire TP2 du dépôt https://github.com/tvial/BUT-R6.01

- soit en téléchargeant TP2.ipynb et data.zip depuis le navigateur, puis dézipper data.zip
- soit en clonant le dépôt (`git clone git@github.com:tvial/BUT-R6.01.git`)

Lancer ensuite TP2.ipynb dans Jupyter

Enjoy :-)

Flot de données du TP