# Le MapReduce

MapReduce est un patron d'architecture de développement informatique, inventé par Google, dans lequel sont effectués des calculs parallèles, et souvent distribués, de données potentiellement très volumineuses, typiquement supérieures en taille à 1 téraoctet.
Les termes « map » et « reduce » sont empruntés aux langages de programmation fonctionnelle utilisées pour leur construction.
MapReduce permet de manipuler de grandes quantités de données en les distribuant dans un cluster de machines pour être traitées. Ce modèle connaît un vif succès auprès des sociétés possédant d'importants centres de traitement de données telles Amazon ou Facebook. Il commence aussi à être utilisé au sein du Cloud computing. De nombreux frameworks ont vu le jour afin d'implémenter le MapReduce. Le plus conne reste Hadoop qui a été programmé par Apache Software Foundation. Mais ce framework possède des inconvénients qui réduisent ses performances.

MapReduce est un modèle de programmation massivement parallèle adapté au traitement de très grandes quantités de données. Les programmes adoptant ce modèle sont automatiquement parallélisés et exécutés sur des clusters d'ordinateurs.
Principe du MapReduce repose sur l'emprunt du langage fonctionnel, utiliser pour implémenter des opérations sur les données comme le tri, le filtrage, la projection , l'agrégation ou le regroupement.

Son fonctionnement :

MapReduce consiste en grande majorité en deux fonctions : map() et reduce()

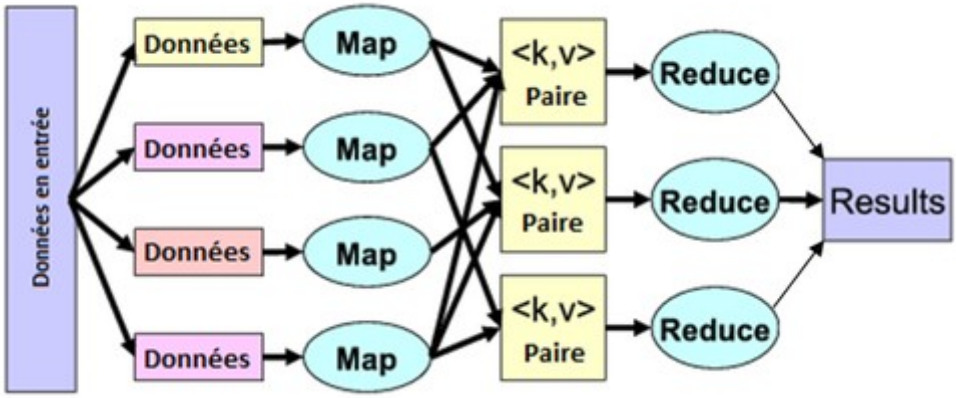
- Dans l'étape Map, le nœud analyse un problème, le découpe en sous-problème, et les délègues à d'autres nœuds, qui peuvent en faire de même récursivement. Les sous problèmes sont ensuite traités par les différents nœuds à l'aide de la fonction Reduce qui associe à un couple (clé,valeur) un ensemble de nouveaux couples(clé,valeur) : map(clé1,valeur1) → list(clé2,valeur2)

- Ensuite vient l'étape Reduce, où les nœuds les plus bas font remonter leurs résultats au nœud parent qui les avait sollicités. Celui-ci calcule un résultat partiel à l'aide de la fonction Reduce qui associe toutes les valeurs correspondantes à la même clé à une unique paire. Puis il remonte l'information à son tour. A la fin du processus, le nœud d'origine peut recomposer une réponse au problème qui lui avait été soumis : reduce(clé2,list(valeur2)) → list(valeur2)

Un cluster MapReduce utilise une architecture de type Maître-Esclave, c'est-à-dire qu'un nœud maître dirige tous les nœuds esclaves.

MapReduce possède quelques caractéristiques :

- Le modèle de programmation du MapReduce est simple mais très expressif. Bien qu'il ne possède que 2 fonctions, map() et reduce(), elles peuvent être utilisées pour de nombreux types de stockage et peut manipuler de nombreux types de variable.
- Le système découpe automatiquement les données en entrée en bloc de données de même taille. Puis, il planifie l'exécution des tâches sur les nœuds disponibles.
- Il fournit une tolérance aux fautes à grain fin grâce à laquelle il peut redémarrer les nœuds ayant rencontré une erreur ou affecter la tâche à un autre nœud.
- La parallélisation est invisible à l'utilisateur afin de lui permettre de se concentrer sur le traitement des données.




Une fois qu'un nœud a terminé une tâche, on lui affecte un nouveau bloc de données. Grâce à cela, un nœud rapide fera beaucoup plus de calculs qu'un nœud plus lent. Le nombre de tâches Map ne dépend pas du nombre de nœuds, mais du nombre de blocs de données en entrée. Chaque bloc se fait assigner une seule tâche Map. De plus, toutes les tâches Map n'ont pas besoin d'être exécutées en même temps en parallèle, les tâches Reduce suivent la même logique. Par exemple, si des données en entrée sont divisées en 400 blocs et qu'il y a 40 nœuds dans le cluster, le nombre de tâches Map sera de 400. Il faudra alors 10 vagues de Map pour réaliser le mapping des données4,5.

Le MapReduce est apparu en 2004. La technologie est encore jeune. Elle souffre de quelques points faibles :

- Elle ne supporte pas les langages haut niveau comme le SQL
- Elle ne gère pas les index. Une tâche MapReduce peut travailler après que les données en entrée sont stockées dans sa mémoire. Cependant, MapReduce a besoin d'analyser chaque donnée en entrée afin de la transformer en objet pour la traiter, ce qui provoque des baisses de performance.
- Elle utilise un seul flot de données. MapReduce est facile à utiliser avec une seule abstraction mais seulement avec un flot de donnée fixe. Par conséquent, certains algorithmes complexes sont difficiles à implémenter avec seulement les méthodes map() et reduce(). De plus, les algorithmes qui requièrent de multiples éléments en entrée ne sont pas bien supportés car le flot de données du MapReduce est prévu pour lire un seul élément en entrée et génère une seule donnée en sortie.
- Quelques points peuvent réduire les performances de MapReduce. Avec sa tolérance aux pannes et ses bonnes performances en passage à l'échelle, les opérations de MapReduce ne sont pas toujours optimisées pour les entrées/sorties. De plus, les méthodes map() et reduce() sont bloquantes. Cela signifie que pour passer à l'étape suivante, il faut attendre que toutes les tâches de l'étape courante soient terminées. MapReduce n'a pas de plan spécifique d'exécution et n'optimise pas le transfert de données entre ces nœuds.

Déroulement :
le travail du MapReduce est con trollé par par un programme appelé le JobTracker. Ce programme est contenu dans le NameNode ou le nœud maître. Le client lance un travail de MapReduce sur le JobTracker, et celui-ci va assigner les tâches de Maps et de Reduce aux autres nœuds sur le cluster. Chacun de ces nœuds lance un programme à leur tour appelé TaskTraker. Ce programme est responsable d'instancier les tâches de Maps ou de Reduce, et de rapporte la progression des tâches

au JobTracker