Chapitre VII Big Data

M1 - IA / GL

Ilyas Bambrik

2024

Table des matières

Introduction

La Big Data fait référence aux données collectées arrivant dans des volumes croissants, à une vitesse élevée et de sources variées. C'est ce que l'on appelle les trois « V ».

En d'autres termes, le Big Data est composé de jeux de données complexes, provenant de sources diversifiées. Ces ensembles de données sont si volumineux qu'un logiciel de traitement de données traditionnel ne peut tout simplement pas les gérer.

Définition du Big Data

I

Les trois V du Big Data :

Volume : La quantité de données a son importance. Avec le Big Data, vous devrez traiter de gros volumes de données non structurées. Il peut s'agir de données de valeur inconnue, comme des flux de données Twitter, des flux de cliques sur une page Internet ou une application mobile ou d'un appareil équipée d'un capteur. Pour certaines entreprises, cela peut correspondre à des dizaines de téraoctets de données. Pour d'autres, il peut s'agir de centaines de pétaoctets.

Vitesse: La vitesse à laquelle les données sont reçues et éventuellement traitées. Normalement, les données haute vitesse sont transmises directement à la mémoire, plutôt que d'être écrites sur le disque. Certains produits intelligents accessibles via Internet opèrent en temps réel ou quasi réel et nécessitent une évaluation et une action en temps réel.

Variété: La variété fait allusion aux nombreux types de données disponibles. Les types de données traditionnels ont été structurés et trouvent naturellement leur place dans une base de données relationnelle. Avec l'émergence du Big Data, les données ne sont pas nécessairement structurées. Les types de données non structurés et semi-structurés, tels que le texte, l'audio et la vidéo, nécessitent un prétraitement supplémentaire pour en déduire le sens et prendre en charge les métadonnées.

Cluster Computing

II

Le Cluster Computing définit plusieurs ordinateurs reliés en réseau et implémentés comme une entité individuelle. Chaque ordinateur lié au réseau est appelé nœud. L'informatique en cluster fournit des solutions pour résoudre des problèmes difficiles en offrant une vitesse de calcul plus rapide et une intégrité des données améliorée. Les ordinateurs connectés mettent en œuvre des opérations tous ensemble, générant ainsi l'impression d'un système centralisé. L'une des applications de cette architecture est le Big Data.

1. Types de calcul en cluster

Les types de Cluster Computing sont les suivants -

- *Haute disponibilité (HA)*

Ces modèles de cluster génèrent la disponibilité des services et des ressources de manière ininterrompue en utilisant la redondance implicite du système. Le terme de base du cluster est que si un nœud tombe en panne, les applications et les services peuvent être mis à la disposition à partir de d'autres nœuds. Ces méthodes de clusters servent d'élément pour les missions critiques, les courriers, les documents et les serveurs d'applications.

- *Clusters d'équilibrage de charge*

Ce cluster alloue tout le trafic/demandes de ressources entrantes provenant de nœuds qui exécutent les mêmes programmes et services. Dans ce modèle de cluster, certains nœuds sont responsables du suivi de l'exécution des requêtes, et ainsi les demandes sont donc réparties entre tous les nœuds disponibles. Une telle solution est généralement utilisée sur les fermes de serveurs web.

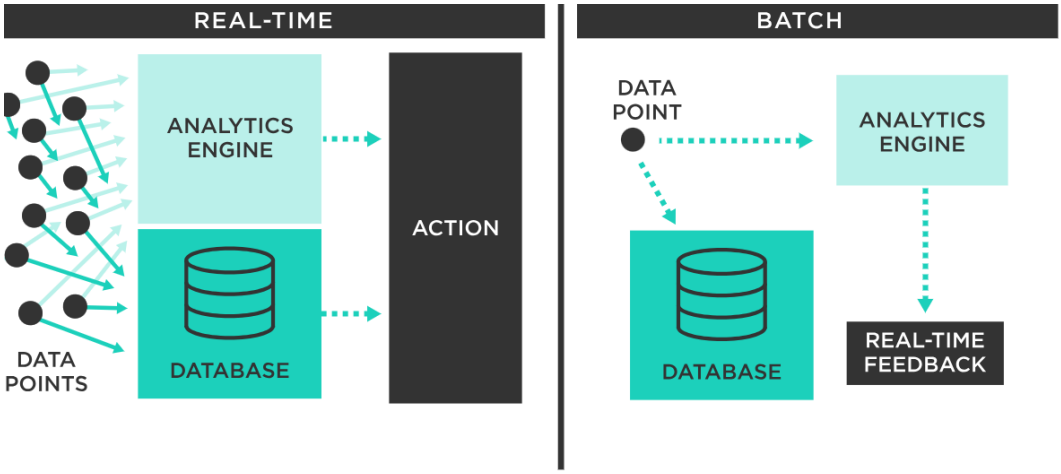
- *Clusters haute disponibilité et équilibrage de charge*

Ce modèle de cluster associe les deux fonctionnalités du cluster, ce qui améliore la disponibilité et l'évolutivité des services et des ressources. Ce type de cluster est généralement utilisé pour les serveurs de messagerie.

- *Clusters de traitement distribués et parallèles*

Ce modèle de cluster améliore la disponibilité et la mise en œuvre des applications comportant d'énormes tâches de calcul. Une grande tâche de calcul est divisée en tâches plus petites et réparties entre les stations. De tels clusters sont généralement utilisés pour le calcul numérique ou l'analyse financière nécessitant une puissance de traitement élevée.

Types de traitements sur le Big Data et frameworks



Il existe de nombreuses façons de traiter de grandes quantités de données et plusieurs éléments entrent en considération dans la classification des applications utilisant ces données. Le traitement du BigData peut être classifier en deux catégories possibles généralement : en lot (*Batch Processing*) ou en temps réel (*Reat-time Data Processing*).

Le traitement par lots de données, également appelé « traitement de données par lots », est un moyen de traiter de grandes quantités de données à la fois. Cela implique simplement de rassembler les données et de les traiter après. Les données sont prêtes lorsque le traitement est près à commencer, ce qui les rend idéales pour toute application qui n'a pas besoin de traiter les observations reçues individuellement et en temps réel.

Le traitement par lots de données présente de nombreux avantages par rapport au traitement des données en temps réel. Lorsque vous n'avez pas besoin du résultat du traitement immédiatement, le traitement par lots de données vous permet de sauvegarder vos données et de tout faire en une seule fois. À partir de là, vous pouvez travailler sur les résultats de ces données tout en en collectant davantage.

Un bon exemple de traitement par lots est la façon dont les sociétés de cartes de crédit effectuent leur facturation. Lorsque les clients reçoivent leurs factures de carte de crédit, il ne s'agit pas d'une facture distincte pour chaque transaction ; il y a plutôt une seule facture pour le mois entier. Cette facture est créée à l'aide d'un traitement par lots. Toutes les informations sont collectées au cours du mois, mais elles sont traitées à une certaine date, en une seule fois.

Permettre le traitement des données à un point où le système est peu sollicité permet une utilisation maximale du système existant. Étant donné que le traitement par lots peut être déclenché ou automatisé pour s'exécuter lorsque le système atteint un certain pourcentage d'utilisation, il est moins nécessaire d'acheter de nouveaux systèmes et les ressources existantes sont utilisées de manière plus intelligente.

Contrairement au traitement par lot, le traitement de données en temps réel est un processus d'analyse des données presque instantané. Lorsque les données brutes sont reçues, elles sont immédiatement traitées pour permettre une prise de décision quasi instantanée. Au lieu d'être stockées, elles sont mises à disposition pour consommer les informations le plus rapidement possible.

Deux frameworks populaires pour traitement parallèle / distribué du Big Data existent actuellement : a) Hadoop, b) Spark

1. Hadoop/ MapReduce

Hadoop est un framework logiciel dédié au stockage et au traitement de larges volumes de données. Il s'agit d'un framework open source regroupant des instructions pour le stockage et le traitement de données distribuées. Celui-ci support le traitement de donnée en lot.

Apache Hadoop repose sur quatre principaux modules :

- Tout d'abord, le *Hadoop Distributed File System (HDFS)* est utilisé pour le stockage de données. Il est comparable à un système de fichier local sur un ordinateur classique. Toutefois, ses performances sont nettement supérieures. Le HDFS délivre par ailleurs une excellente élasticité. Il est possible de passer d'une machine unique à plusieurs milliers d'entre elles très facilement.
- Le second composant est le *YARN* (Yet Another Resource Negotiator). Comme son nom l'indique, il s'ag d'un négociateur de ressources. Il permet de planifier des tâches, de gérer les ressources et de surveiller les noeuds de clusters et les autres ressources.
- De son côté, le module *Hadoop MapReduce* aide les programmes à effectuer des calculs parallèles. La tâche *Map* convertit les données en paires clés-valeurs. La tâche *Reduce* consomme les données d'entrée, les agrège et produit le résultat.
- Le dernier module est *Hadoop Common* qui est une collection de bibliothèques et d'utilitaires courantes qui fonctionnent avec différents modules Hadoop.

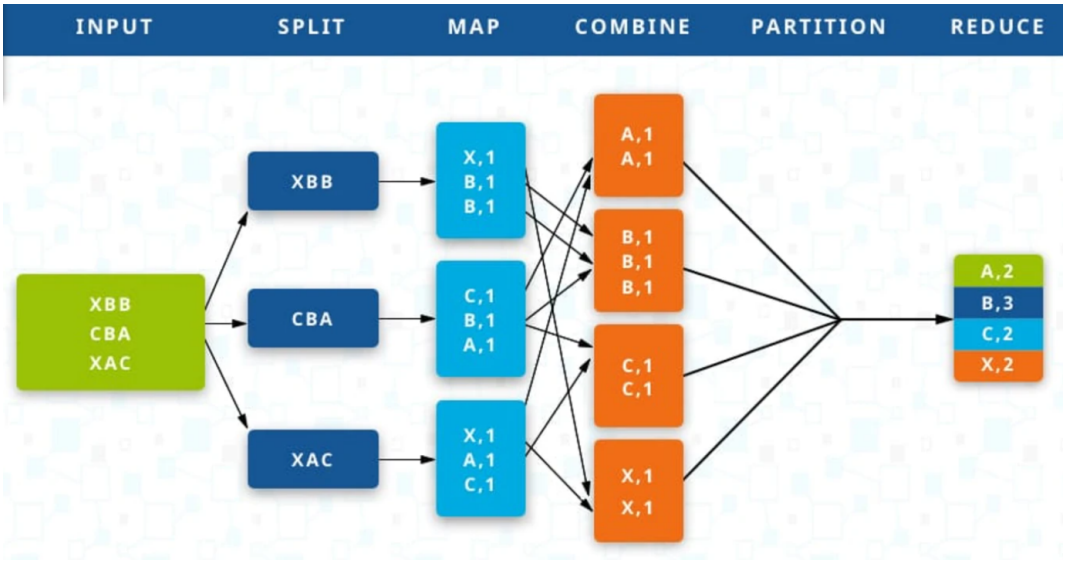
MapReduce est un modèle et un cadre de programmation au sein de l'écosystème Hadoop qui permet un traitement efficace du Big Data en distribuant et en parallélisant automatiquement le calcul. Il se compose de deux tâches fondamentales : Mapper et Réduire.

Dans la phase Map, les données d'entrée sont divisées en morceaux plus petits et traitées indépendamment en parallèle sur plusieurs nœuds dans un environnement distribué. Chaque morceau est transformé ou « mappé » en paires clé-valeur en appliquant une fonction définie par l'utilisateur. Le résultat de la phase Map est un ensemble de paires clé-valeur intermédiaires.

La phase Reduce suit la phase Map. Celle-ci rassemble les paires clé-valeur intermédiaires générées par les tâches Map, regrouper les paires avec la même clé, puis applique une fonction de réduction définie par l'utilisateur pour agréger et traiter les données. La sortie de la phase Reduce est le résultat final du calcul.

MapReduce permet un traitement efficace d'ensembles de données à grande échelle en tirant parti du parallélisme et en répartissant la charge de travail sur un cluster de machines. Ce paradigme simplifie le développement d'applications de traitement de données distribuées en éliminant les complexités de la parallélisation, de la distribution des données et de la tolérance aux pannes (les tâches qui échoues sont redémarrées automatiquement par le framework), ce qui en fait un outil essentiel pour le traitement du Big Data dans l'écosystème Hadoop.

Au cœur de MapReduce se trouvent deux fonctions : Map et Reduce, mais d'autres fonctions optionnels peuvent être appliquée (comme *Shuffle, Sort, Compbine*).



Exemple : Exemple MapReduce

Prenons l'exemple d'un système de commerce électronique qui reçoit chaque jour un million de demandes pour traiter les paiements. Plusieurs exceptions peuvent être levées lors de ces demandes, telles que « paiement refusé par une passerelle de paiement », « rupture de stock » et « adresse invalide ». Un développeur souhaite analyser les journaux des quatre derniers jours pour comprendre quelle exception est levée et combien de fois. L'objectif est d'isoler les cas d'utilisation les plus susceptibles aux erreurs et de prendre les mesures appropriées. Par exemple, si la même passerelle de paiement lève fréquemment une exception, est-ce à cause d'un service peu fiable ou d'une interface mal écrite ? Si l'exception « hors stock » est souvent levée, cela signifie-t-il que le service de calcul des stocks doit être amélioré, ou les stocks doivent-ils être augmentés pour certains produits ?

Le développeur peut poser des questions pertinentes et déterminer la bonne démarche à suivre. Pour effectuer cette analyse sur des fichiers de journaux volumineux, contenant des millions d'enregistrements, MapReduce est un modèle de programmation approprié. Plusieurs mappeurs peuvent traiter ces journaux simultanément : un mappeur peut traiter le journal d'une journée ou un sous-ensemble de celui-ci en fonction de la taille du journal et du bloc de mémoire disponible pour le traitement dans le serveur du mappeur.

Map

Pour simplifier, supposons que le framework Hadoop n'exécute que quatre mappeurs. Mappeur 1, Mappeur 2, Mappeur 3 et Mappeur 4.

La valeur entrée dans le mappeur est une partie du fichier journal. La clé peut être une chaîne de texte telle que « nom de fichier + numéro de ligne ». Le mappeur traite ensuite chaque enregistrement du fichier journal pour produire des paires clé-valeur. Le résultat des mappeurs ressemble à ceci :

Mappeur 1 - Exception A, 1 , Exception B, 1 , Exception A, 1 , Exception C, 1 , Exception A, 1

Mappeur 2 - Exception B, 1 , Exception B, 1 , Exception A, 1 , Exception A, 1

Mappeur 3 - Exception A, 1 , Exception C, 1 , Exception A, 1 , Exception B, 1 , Exception A, 1

Mappeur 4 - Exception B, 1 , Exception C, 1 , Exception C, 1 , Exception A, 1

En supposant qu'il existe un combinateur exécuté sur chaque mappeur (Combiner 1... Combiner 4) qui calcule le nombre de chaque exception (qui est la même fonction que le réducteur), l'entrée du Combiner 1 sera :

Exception A, 1 , Exception B, 1 , Exception A, 1 , Exception C, 1 , Exception A, 1

Combiner

- Le résultat du Combiner 1 sera :

Exception A, 3 , Exception B, 1 , Exception C, 1

La sortie des autres combinateurs sera :

Combinateur 2 : Exception A, 2 Exception B, 2

Combinateur 3 : Exception A, 3 Exception B, 1 Exception C, 1

Combinateur 4 : Exception A, 1 Exception B, 1 Exception C, 2

- Après cela, le partitionnement alloue les données des combinateurs aux réducteurs. Les données sont également triées pour le réducteur.

L'entrée dans les réducteurs sera la suivante :

Réducteur 1 : Exception A {3,2,3,1}

Réducteur 2 : Exception B {1,2,1,1}

Réducteur 3 : Exception C {1,1,2}

- S'il n'y avait aucun combinateur impliqué, l'entrée dans les réducteurs sera la suivante :

Réducteur 1 : Exception A {1,1,1,1,1,1,1,1,1}

Réducteur 2 : Exception B {1,1,1,1,1}

Réducteur 3 : Exception C {1,1,1,1}

- Désormais, chaque réducteur calcule simplement le nombre total d'exceptions comme suit :

Réducteur 1 : Exception A, 9

Réducteur 2 : Exception B, 5

Réducteur 3 : Exception C, 4

Les données montrent que l'exception A est levée plus souvent que les autres et nécessite plus d'attention. Lorsqu'il y a plus de quelques semaines ou mois de données à traiter ensemble, le potentiel du programme MapReduce peut être véritablement exploité.

2. Apache Spark

Apache Spark est un système de traitement open source distribué, utilisé pour les charges de travail de Big Data. Ce framework utilise la mise en cache en mémoire, ainsi que l'exécution de requête optimisée pour les requêtes d'analyse sur des données de toutes tailles. Il fournit des API de développement en Java, Scala, Python et R, et prend en charge : le traitement par lots, requêtes interactives, analytique en temps réel, machine learning et traitement de graphes. Celui-ci est utilisé par des organisations de tous les secteurs, notamment à la FINRA, chez Yelp, Zillow, DataXu, Urban Institute et CrowdStrike.

MapReduce lit les données du cluster, effectue des opérations et réécrit les résultats dans HDFS. Comme chaque étape nécessite une lecture et une écriture sur le disque, les tâches MapReduce sont plus lentes en raison de la latence des E/S du disque. Spark a été créé pour remédier aux limites de MapReduce, en effectuant un traitement en mémoire, en réduisant le nombre d'étapes d'une tâche et en réutilisant les données dans le cadre de plusieurs opérations parallèles. Avec Spark, une seule étape est nécessaire : les données sont lues en mémoire, les opérations sont effectuées et les résultats sont réécrits, ce qui se traduit par une exécution

beaucoup plus rapide. Spark réutilise également les données en utilisant un cache en mémoire pour accélérer considérablement les algorithmes de machine learning qui appellent à plusieurs reprises une fonction sur le même jeu de données. La réutilisation des données s'effectue par la création de DataFrames, une abstraction de Resilient Distributed Dataset (RDD), qui est un ensemble d'objets mis en cache en mémoire et réutilisés dans plusieurs opérations Spark. Cela réduit considérablement la latence, rendant Spark beaucoup plus rapide que MapReduce, en particulier lors du machine learning et de l'analytique interactive.

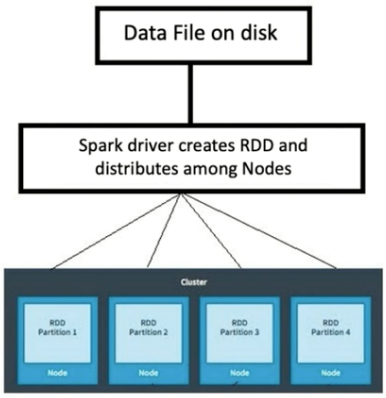
pySpark

IV

- pySpark offre des API similaires aux API *pandas* et *scikitlearn* et propose un invité de commande (*shell*) qui permet l'analyse interactive. Cela veux dire que spark permet d'écrire des programmes est d'accéder aux données similairement aux bibliothèques traditionnelles et en faisant abstraction de l'environnement distribué.
- Afin d'accéder aux fonctionnalités du cluster, il faudra commencer par définir un point d'entrée avec *SparkContext*. Dans le shell Spark, cette entité est représenté par la variable sc. Cette objet comporte plusieur information sur l'environnement distribué mais particulièrement, l'attribut *sc.master* contient l'URL du cluster.
- A travers l'objet *SparkContext*, il est possible de créer une collection parallèle dans le cluster avec la méthode *parallelize*. En outre, la méthode sc.textFile permet de lire un fichier depuis le système de fichier distribué.
- Le chargement de données dans le cluster crée un structure de base appelée RDD.

```
1 collectionRDD=sc.parallelize([50,247,45,56])# crée une collection distribuée rdd
```

RDD = Resilient Distributed Datasets



Le partitionnement des données (RDD) est fait selon les ressources disponibles dans le cluster par défaut. Cependant, il est possible de définir le nombre minimal de partitions lors de la création d'un RDD.

```
1 fichierRDD=sc.textFile("dataset.csv",minPartions=4)
```

Il est aussi possible d'obtenir le nombre de partition d'un RDD avec la méthode *getNumPartitions*()

Deux opérations principales sur les RDD sont proposées par pySpark : a) Transformations : opérations qui retournent un nouveau RDD, b) Actions : computations et calcules sur les RDD.

Par example, les méthodes *map(), foreach(), filter() et union()* classée comme une transformation et retourne un nouveau RDD :

```
collectionRDD=sc.parallelize([50,247,45,56])
newcollectionRDD=collectionRDD.map(lambda x:x/100)
```

Les actions sont des fonctions qui renvoient un résultat après que les computations sont terminées.

- *collect() :* retourne touts les éléments d'un RDD sous forme d'un tableau.
- *take(N) :* retourne les N premiers éléments d'un RDD sous forme d'un tableau.
- *first() :* retourne le premier élément.
- *count() :* retourne le nombre d'éléments dans le RDD.
- top(N)

Généralement, les enregistrements d'un DataSet sont de la forme <Key : identificateur,Value : donnée>. Pour ce type de donnée, pySpark offre un type de donnée Pair RDD.

```
parRDD= sc.parallelize([("key1",2),("key2" ,1),( "key1",1)])
```

Ce type de RDD offre les mêmes méthodes qu'un RDD ordinaire avec en plus des méthodes qui opères sur la structure <Key,Value> :

- *rdd.reduceByKey() :* combine les éléments ayant la même clé.
- *rdd.groupByKey()* : groupe les éléments ayant la même clé.
- *rdd.sortByKey() :* trie selon la clé.
- *rdd.join() :* jointure à base de la clé.
- *rdd1.subtractByKey(rdd2) :* retourne les pairs <Key,Value> de rdd1 qui ne sont pas dans rdd2.

Les opérations sur un RDD sont faites d'une manière distribué. Ainsi, il est nécessaire d'invoquer la méthode collect() sur le résultat pour récupérer le résultat finale après la fin du calcule.

```
rdd1=sc.parallelize([("A",1),("B",2),("C",12)])
rdd2=sc.parallelize([("C",12),("A",24),("B",10)])
rdd1.join(rdd2).collect()
# résultat : [("A",(1,24)), ("C",(12,12)), ("B",(2,10))]
```

Une autre fonction définie comme action est la fonction *reduce()* et *reduceByKey()*. Celle-ci effectue une agrégation selon une fonction fournie en entrée :

```
rdd.reduceByKey(lambda x,y : x+y).collect()
#[('a',9),('b',2)]
```