Comparaison de différents formats de fichier en Big Data

By NGOM Aida
23 juil. 2020

Dans l'univers du traitement des données, il existe différents types de formats de fichiers pour stocker vos jeu de données. Chaque format a ses propres avantages et inconvénients selon les cas d'utilisation et existe pour servir un ou plusieurs objectifs. Il est important de connaître et d'exploiter leurs spécificités lors du choix d'un type de format spécifique.

Certains formats sont plus adaptés à certains usages ou traitements, comme en Business Intelligence, en communication réseau, en applications web, en traitements batch ou streaming. Par exemple, le format CSV est très compréhensible et, alors que tout le monde critique son manque de formalisme, il est encore largement utilisé. En cas de communication Web, le JSON est privilégié et couvre ainsi le gros des communications mondiales même si, dans certains cas, XML peut faire le même travail. Le dernier format bénéficie d'un large écosystème d'outils et est largement utilisé dans le monde de l'entreprise. En termes de traitement en streaming, AVRO et Protocol Buffers sont des formats privilégiés. De plus, Protocol Buffers est le protocole sur lequel est fondé gRPC (RPC pour *Remote procedure call*) et sur lequel s'appuie l'écosystème Kubernetes. Enfin, le stockage sur colonnes offert par ORC et Parquet offre une amélioration significative des performances dans l'analyse des données.

Caractéristiques à prendre en compte pour le choix d'un format de fichier

Nous présenterons ci-dessous une sélection des formats de fichiers les plus populaires et les plus représentatifs. Sont décrits les différentes considérations à garder à l'esprit lors du choix d'un format sur un autre. Mais d'abord, passons en revue leurs principales caractéristiques en ce qui concerne le stockage, les types de requêtes, les utilisations par batch ou en streaming, ...

Text ou binary

Les formats de fichiers texte sont plus faciles à utiliser. Ils peuvent être lus par l'utilisateur final qui peut également modifier le contenu du fichier avec un éditeur de texte. Ces formats sont généralement compressés pour réduire leur empreinte de stockage. Les formats de fichiers binaires nécessitent l'utilisation d'un outil ou d'une bibliothèque pour être créés et consommés, mais ils offrent de meilleures performances en optimisant la sérialisation des données.

Typage des données

Certains formats ne permettent pas de déclarer des types de données. Les types sont classés en 2 catégories. Les types scalaires contiennent une valeur "unique" (par exemple un entier, un booléen, un string, null, ...). Les types complexes sont un composé de types scalaires (par exemple un tableau, un objet, ...). La connaissance du type associé à une valeur apporte plusieurs avantages. Le consommateur peut distinguer par exemple un nombre d'un string ou une valeur nulle du string "null". Une fois encodé sous forme binaire, le gain de stockage est important. Par exemple, la chaîne `"1234"` utilise 4 octets de stockage alors que le booléen `1234` ne nécessite que 2 octets.

Déclaration d'un schéma

Le schéma peut être associé aux données ou laissé au consommateur qui est supposé connaître et comprendre comment interpréter les données. Il peut être fourni avec les données ou séparément. L'utilisation d'un schéma garantit que l'ensemble de données est valide et fournit des informations supplémentaires telles que le type et le format d'une valeur. Les schémas sont également utilisés dans des formats de fichiers binaires pour crypter et décrypter le contenu.

Prise en charge de l'évolution du schéma

Le schéma stocke la définition de chaque attribut et ses types. À moins que vos données ne soient garanties de ne jamais changer (par exemple de l'archivage) ou soient par nature immutable (par exemple des logs), vous devrez penser à l'évolution du schéma afin de déterminer si votre schéma de données change au fil du temps. En d'autres termes, l'évolution du schéma permet de mettre à jour le schéma utilisé pour écrire de nouvelles données tout en conservant une compatibilité descendante avec le schéma de vos anciennes données.

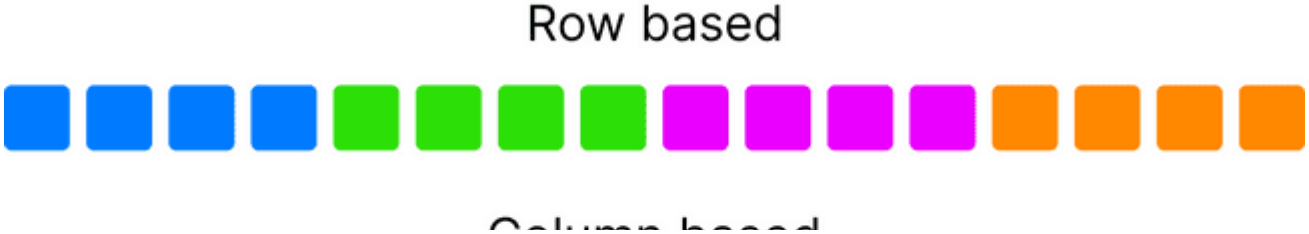
Stockage en lignes et en colonnes, OLTP contre OLAP

Dans le stockage basé sur les lignes, les données sont stockées comme leur nom l'indique, ligne par ligne, de sorte que la première colonne d'une ligne sera à côté de la dernière colonne de la ligne précédente. Cette architecture est appropriée pour ajouter des enregistrements facilement et rapidement. Il convient également dans le cas où la ligne entière de données doit être accédée ou traitée simultanément. Ceci est couramment utilisé pour le traitement transactionnel en ligne (OLTP). Les systèmes OLTP traitent généralement les requêtes CRUD (lecture, insertion, mise à jour et suppression) au niveau de l'enregistrement. Les transactions OLTP sont généralement très spécifiques dans la tâche qu'elles effectuent et impliquent généralement un seul enregistrement ou une petite sélection d'enregistrements. Dans ces systèmes, l'accent principal est mis sur le maintien de l'intégrité des données dans les environnements à accès multiple et sur l'efficacité mesurée par le nombre de transactions par seconde.

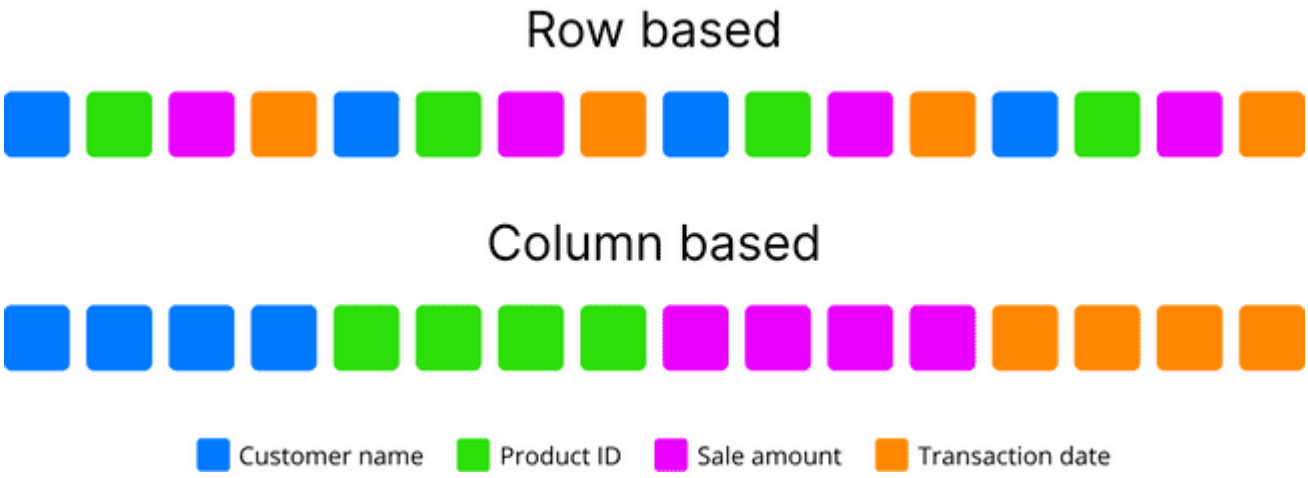
Inversement, dans le stockage basé sur les colonnes, les données sont stockées de sorte que chaque ligne d'une colonne se trouve à côté d'autres lignes de cette même colonne. Cela est particulièrement utile lors de l'exécution de requêtes analytiques qui ne nécessitent qu'un sous-ensemble de colonnes examinées sur de très grands ensembles de données. Cette façon de traiter les données est généralement appelée une requête OLAP (OnLine Analytical Processing). OLAP est une approche conçue pour répondre rapidement aux requêtes d'analyse impliquant plusieurs dimensions. Cette approche a joué un rôle essentiel dans l'analyse de la Business Intelligence, en particulier en ce qui concerne le Big Data. Le stockage des données en colonne évite les temps de traitement excessifs de la lecture d'informations non pertinentes dans un ensemble de données en ignorant toutes les données qui ne s'appliquent pas à une requête particulière. Il fournit également un taux de compression plus élevé en alignant les données d'un même type et en optimisant les valeurs «nulles» des colonnes éparses.

Here is an example if you wanted to know the average population density in cities with more than a million people. With a row-oriented storage and, your query would access each record in the table (meaning all of its fields) to get the necessary information from the three relevant columns `city name`, `density` and `population`. That would involve a lot of unnecessary disk seeks and disk reads, which impact performance. Columnar databases performs better by reducing the amount of data that needs to be read from disk.

Exemple regroupant des enregistrements par couleur



Exemple regroupant des propriétés par couleur



Divisibilité

Dans un système de fichiers distribué comme Hadoop HDFS, il est important d'avoir un fichier qui peut être divisé en plusieurs morceaux. Dans Hadoop, le système de fichiers HDFS stocke les données par morceaux et le traitement des données est initialement distribué en fonction de ces morceaux. Il est utile de pouvoir commencer à lire des données à tout moment dans un fichier afin de profiter pleinement des avantages du traitement distribué de Hadoop. Lorsque le format de données n'est pas fractionnable, il est de la responsabilité de l'utilisateur de partitionner ses données en plusieurs fichiers pour permettre le parallélisme.

Compression (Bzip2, LZO, Sappy,...)

Un système est aussi lent que ses composants les plus lents et, la plupart du temps, les composants les plus lents sont les disques. L'utilisation de la compression réduit la taille du jeu de données stocké et réduit ainsi la quantité d'IO (Input Output) lues à effectuer. La compression accélère également les transferts de fichiers sur le réseau. Le stockage en colonne apporte une ration de compression plus élevée et de meilleures performances par rapport au stockage sur ligne, car des données similaires sont stockées ensemble. Comme mentionné précédemment, il est particulièrement efficace sur les colonnes éparses qui contiennent plusieurs valeurs `null`.

En outre, la compression s'applique principalement aux formats de fichiers texte, sauf si le format de fichier binaire inclut la compression dans sa définition. Tenter de compresser un fichier binaire conduit souvent à un fichier généré plus lourd que son original. De plus, lors de la comparaison du taux de compression, nous devons également mettre en perspective la taille du fichier d'origine. Il ne serait pas judicieux de comparer le taux de compression du JSON par rapport à de l'XML si nous ne tenons pas compte du fait que l'XML est à l'origine plus volumineux que son homologue le JSON.

Il existe de nombreux types d'algorithmes de compression qui diffèrent selon le rapport de compression, la vitesse, les performances, leur capacité à être divisibles, ...

Batch et Streaming

Le traitement par batch (aussi appelé par lot) permet de lire, d'analyser et de transformer plusieurs enregistrements à la fois. À l'opposé, le traitement en streaming (aussi appelé en flux) fonctionne en temps réel et traite les enregistrements dès leur arrivée. Parfois, un même ensemble de données peut être traité à la fois en streaming et en batch. Examinons les transactions financières reçues à l'intérieur d'un système. Les données arrivent en streaming et pourraient être traitées instantanément pour détecter la détection de fraude. En complément, une fois les données stockées, un travail peut être exécuté périodiquement en batch pour préparer certains rapports. Dans ce cas, nous pourrions compter sur un ou deux formats de fichier entre le format des données arrivant sur le réseau et le format des données stockées dans notre entrepôt de données ou Data Lake.

Écosystème

Il est souvent considéré comme une bonne pratique de respecter l'usage et la culture d'un écosystème lorsqu'il s'agit de choisir entre plusieurs alternatives. Par exemple, lors du choix entre Parquet ou ORC avec Hive, il est recommandé d'utiliser le premier sur les plateformes Cloudera et le dernier sur les plateformes Hortonworks. Les chances sont que l'intégration sera plus fluide et que la documentation et le soutien de la communauté seront très utiles.

Formats de fichier populaires

CSV

CSV est un format de texte qui utilise une nouvelle ligne comme délimiteur d'enregistrement avec un en-tête facultatif. Il est facile d'éditer manuellement et de façon

très compréhensible par la perspective humaine. CSV ne prend pas en charge l'évolution du schéma même si parfois l'en-tête est facultativement considéré comme le schéma des données. Dans sa forme la plus complexe, il n'est pas fractionnable car il ne contient pas de caractère spécifique sur lequel vous pourriez vous baser pour fractionner le texte tout en restant pertinent.

Généralement dans le Big Data, le CSV est utilisé comme format en entré mais à proprement parler, c'est un format plus proche du TSV. Les différences entre les deux formats sont subtiles.

Le format CSV n'est pas un format entièrement standardisé. Ses délimiteurs de champ peuvent être, par exemple, deux points ou des points-virgules. Il devient difficile d'analyser lorsque le champ lui-même peut également contenir ce type de charactère ou même des sauts de ligne intégrés. De plus, les données de champ peuvent être entourées de guillemets de quotation. Ces quotation complexifie l'identification du début ou de la fin des phrases ou même des valeurs nulles. Contrairement à TSV, CSV incorpore un caractère d'échappement (\ ) dans ses données. Ainsi, toutes ces diverses considérations rendent complexe l'analyse. Alors que TSV étant généralement délimité par une tabulation ou une virgule, il ne prend pas en charge un caractère d'échappement. Cela permet de distinguer facilement chaque enregistrement les uns des autres ce qui rend le fichier divisible.

- *Avantage*
  - Excellent rendement de compression
  - Facile à lire
  - Prise en charge de traitements par lots et en streaming
- *Inconvénients*
  - Pas de support des valeurs nulles, identiques aux valeurs vides
  - Pas de garanti d'être divisible et non prise en charge de l'évolution du schéma
  - Format non standardisé, chacun ayant ses propres interprétations
- *Écosystèmes*
  - Supporté par un large éventail d'applications (Hadoop, Spark, kafka, ...) et de languages (par exemple le parseur CSV pour Node.js)
  - Un des formats utilisés les plus populaires, mais généralement non recommandé

Exemple de format CSV

```
Player Name;Position;Nicknames;Years Active
Skippy Peterson;First Base;"""Blue Dog"", ""The Magician""";1908-1913
Bud Grimsby;Center Field;"""The Reaper"", ""Longneck""";1910-1917
Vic Crumb;Shortstop;"""Fat Vic"", ""Icy Hot""";1911-1912
```

JSON (JavaScript object notation)

Le JSON est un format texte contenant un enregistrement qui peut être sous n'importe plusieurs forme (chaîne, entier, booléen, tableau, objet, clé-valeur, données imbriquées ...). Étant lisible par l'homme et la machine, JSON est un format relativement léger et privilégié dans les applications Web. Il est largement pris en charge par de nombreux logiciels et relativement simple à mettre en œuvre dans plusieurs languages. JSON prend en charge nativement le processus de sérialisation et de désérialisation.

Son inconvénient majeur est lié à son incapacité à être divisible. Le format alternatif utilisé dans ce cas est généralement du JSON Lines. Celui-ci contient plusieurs lignes où chaque ligne individuelle est un objet JSON valide, séparé par un caractère de nouvelle ligne `\n` . Étant donné que le JSON a garanti l'absence de caractères de nouvelle ligne dans les données sérialisées, nous pouvons utiliser la présence de ces caractères pour diviser un jeu de données en plusieurs blocs.

- *Avantage*
  - JSON est une syntaxe simple. Il peut être ouvert par n'importe quel éditeur de texte
  - Il existe de nombreux outils disponibles pour valider le schéma
  - JSON est compressible et utilisé comme format d'échange de données
  - Il prend en charge le traitement par lots et en streaming
  - Il stocke les métadonnées avec des données et prend en charge l'évolution du schéma
  - JSON est un format texte léger par rapport à XML
- *Inconvénients*
  - JSON n'est pas séparable et manque d'indexation autant de formats de texte
- *Écosystèmes*
  - C'est un format privilégié pour les applications web

- JSON est un format de fichier largement utilisé pour les bases de données NoSQL telles que MongoDB, Couchbase et Azure Cosmos DB. Par exemple, dans [MongoDB] (https://docs.mongodb.com/), BSON est utilisé à la place du format standard. Cette architecture de MongoDB est plus facile à modifier qu'une base de données structurée comme [Postgres] (https://ww.postgresql.org/docs/12/index.html) où vous auriez besoin d'extraire tout le document pour le changer. Ideed, BSON est le codage binaire de documents de type JSON. Il est indexé et permet d'être analysé beaucoup plus rapidement que le JSON standard. Cependant, cela ne signifie pas que vous ne pouvez pas considérer MongoDB comme une base de données JSON.
- JSON est également un langage utilisé dans [GraphQL] (https://graphql.org/) sur lequel s'appuient des données entièrement typées. GraphQL est un langage de requête pour une API. Au lieu de travailler avec des points de terminaison rigides définis par le serveur, vous pouvez envoyer des requêtes pour obtenir exactement les données que vous recherchez en une seule demande.

Exemple montrant respectivement un format JSON et un format JSON Lines

```
{
  "fruits": [{
    "kiwis": 3,
    "mangues": 4,
    "pommes": null
  },
  {
    "panier": true
  }],
  "legumes": {
    "patates": "amandine",
    "poireaux": false
  },
  "viandes": [
    "poisson","poulet","boeuf"
  ]
}
```

```
{"id":1,"father":"Mark","mother":"Charlotte","children":["Tom"]}
{"id":2,"father":"John","mother":"Ann","children":["Jessika","Antony"
{"id":3,"father":"Bob","mother":"Monika","children":["Jerry","Karol"]
```

XML

L'XML un langage basé sur des tags créé par le W3C pour définir une syntaxe de documents lisibles à la fois par l'homme et par la machine. L'XML permet de définir des langages plus ou moins complexes. Il permet également d'établir un format de fichier standard pour l'échange entre les applications.

L'XML est également utilisé pour sérialiser et encapsuler les données. Sa syntaxe est verbeuse, en particulier pour les lecteurs humains, par rapport aux autres formats textes alternatifs comme le CSV et le JSON.

- *Avantages*
  - Supports des traitements batch et streaming.
  - Stockage des meta data with data et évolution du schema.
  - Étant un format verbeux, il offre un bon rapport de compression par rapport au fichier JSON ou à d'autre format textes.
- *Inconvénients*
  - Non fractionnable car l'XML a une balise d'ouverture au début et une balise de fermeture à la fin. Vous ne pouvez pas commencer le traitement des données au milieu sans connaître le contexte parent.
  - La nature redondante de la syntaxe entraîne des coûts de stockage et de transport plus élevés lorsque le volume de données est important.
  - Les namespaces XML sont problématiques à utiliser. La prise en charge des namespace peut être difficile à implémenter correctement dans un analyseur XML.
  - Difficultés d'analyse nécessitant la sélection et l'utilisation d'un analyseur DOM ou SAX approprié.
- *Écosystèmes*
  - Le language bénéficie d'une large précence historique en entreprise qui à tendance à décroitre au fil du temps.

Exemple d'XML

```
<?xml version="1.0" encoding="UTF-8"?>
<scientists>
  <!-- Some important physicists -->
  <scientist id="phys0">
    <name firtname="Galileo" lastname="Galilei" />
    <dates birth="1564-02-15" death="1642-01-08" />
    <contributions>Relativity of movement &amp;amp; heliocentric system</
  </scientist>
</scientists>
```

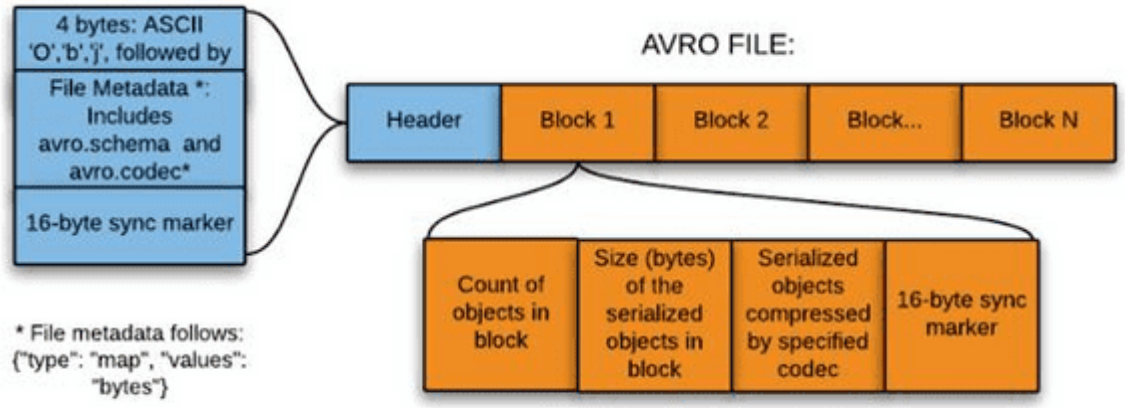
AVRO

Avro est un framework développé dans le cadre du projet Apache Hadoop. Il s'agit d'un format de stockage basé sur des lignes qui est largement utilisé comme processus de sérialisation. AVRO stocke son schéma au format JSON, ce qui le rend facile à lire et à interpréter par n'importe quel programme. Les données elles-mêmes sont stockées au format binaire en les rendant compactes et efficaces. Une caractéristique clé d'AVRO est liée au fait qu'il gère facilement l'évolution des schémas. Il joint des métadonnées à leurs données dans chaque enregistrement.

- *Avantages*
  - Avro est un système de sérialisation de données.
  - Il est divisible (AVRO a un marqueur de synchronisation pour séparer le bloc) et compressible.
  - Avro est un bon format de fichier pour l'échange de données. Il dispose d'un stockage de données qui est très compact, rapide et efficace pour l'analyse.
  - Il prend fortement en charge l'évolution du schéma (à différents moments et indépendamment).
  - Il supporte le batch et et particulière approprié au streaming.
  - Les schémas sont définis en JSON, facile à lire et à interpréter.
  - Les données sont toujours accompagnées d'un schéma qui permet un traitement complet des données.
- *Inconvénients*

- En étant binaire, les données ne sont pas lisible par un humain.
- Non intégré à tous les languages.

- *Écosystèmes*
  - Largement utilisé dans de nombreuses applications (Kafka, Spark, ...).
  - Utilisable dans un context RPC (Remote Procedure Call).



Protocol Buffers

Protocol buffers a été développés par Google et fut mis open source en 2008. Il est indépendant d'un langage. C'est un moyen extensible de sérialiser des données structurées. Il est utilisé dans les protocoles de communication, le stockage de données, etc. Il est facile à lire et compréhensible par l'homme. Contrairement à Avro, le schéma est requis pour intepréter les données.

- *Avantages*
  - Les données sont entièrement typées.
  - Protobuf prend en charge l'évolution du schéma et le traitement par lots / streaming.
  - Principalement utilisé pour sérialiser des données, comme AVRO.
  - Large choix de types de données, les messages sérialisés sur Protobuf peuvent être automatiquement validés par le code chargé de les échanger.
- *Inconvénients*
  - Non séparable et non compressible.

  - Pas de prise en charge de Map Reduce.
  - Nécessite un fichier de référence avec le schéma.
  - Non conçu pour gérer des messages volumineux. Comme il ne prend pas en charge l'accès aléatoire, vous devrez lire l'intégralité du fichier, même si vous ne souhaitez accéder qu'à un élément spécifique.

- *Écosystèmes*
  - Les outils sont disponibles sur la plupart des langages de programmation, comme JavaScript, Java, PHP, C #, Ruby, Objective C, Python, C ++ et Go.
  - Fondation de gRPC, largement utilisé dans l'écosystème kubernetes.
  - ProfaneDB est une base de données pour les objets Protocol Buffers.

```
syntax = "proto3";

message dog {
  required string name = 1;
  int32 weight = 2;
  repeated string toys = 4;
}
```

Parquet

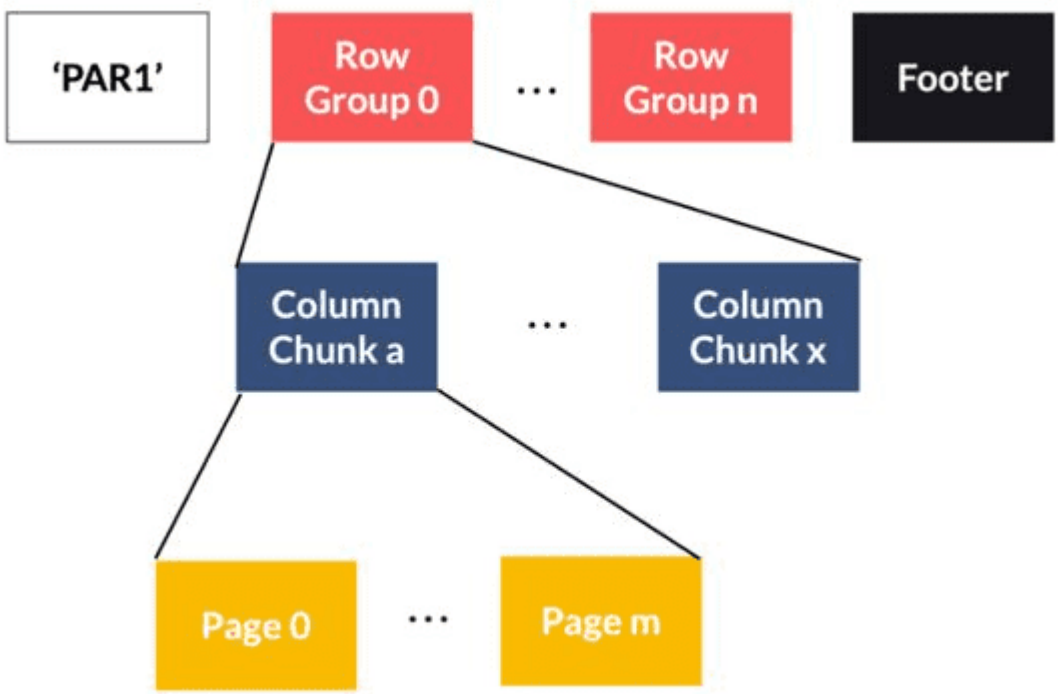
Parquet est un format de fichier open source créer dans l'écosystème Hadoop. Sa conception est un effort combiné entre Twitter et Cloudera pour un stockage efficace des données d'analyse. Parquet stocke les données par colonne comme le format ORC. Il fournit des schémas de compression et d'encodage de données efficaces avec des performances améliorées pour gérer des données complexes en masse. Un fichier Parquet est un fichier binaire contenant des métadonnées sur leur contenu. Les métadonnées de colonne sont stockées à la fin du fichier, ce qui permet une écriture rapide en une seule passe. Le parquet est optimisé pour le paradigme "write once read many" (WORM).

- *Avantages*
  - Fichiers divisibles.
  - Organisé par colonne, permettant une meilleure compression, car les données sont plus homogènes.

- Haute efficacité pour les workloads OLAP.
- Prise en charge de l'évolution du schéma.
- Limité au traitement par batch.

- *Inconvénients*
  - Non lisible par l'homme.
  - Difficile d'appliquer les mises à jour, sauf si vous supprimez et recréez à nouveau le fichier.
- *Écosystèmes*
  - Analyse efficace pour BI (Business Intelligence).
  - Rapide à lire par les moteurs de traitement tels que Spark.
  - Couramment utilisé avec Spark, Impala, Arrow et Drill.

Structure d'un fichier Parquet



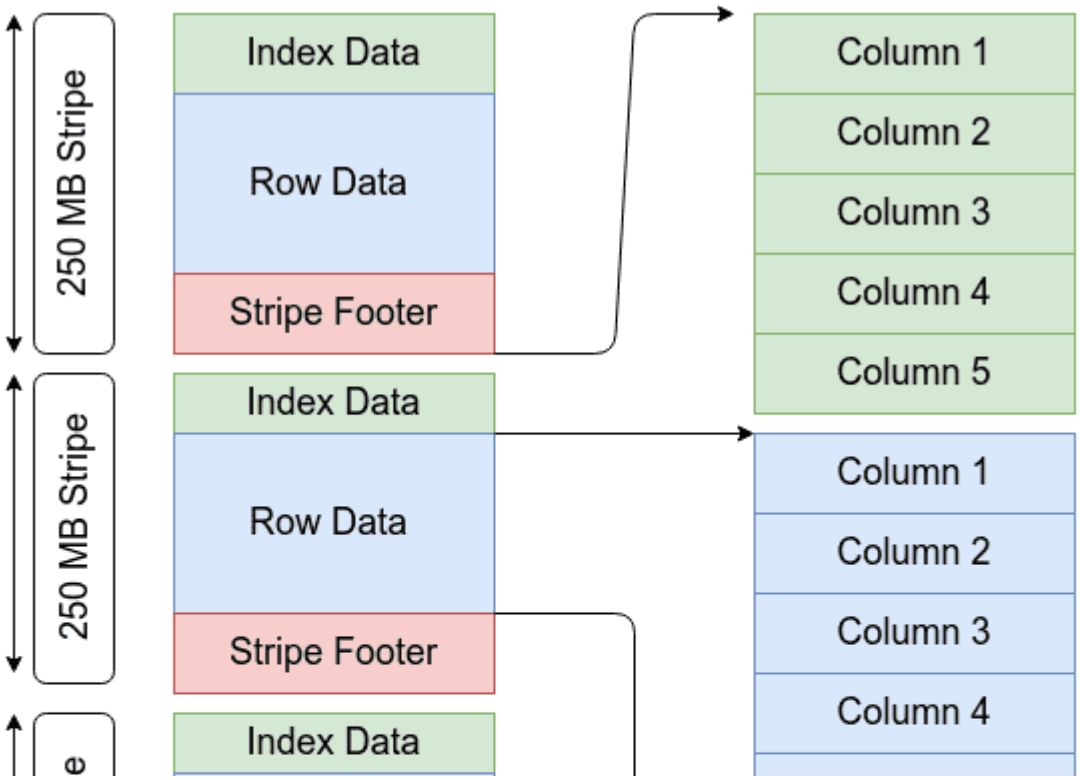
ORC (Optimized Row Columnar)

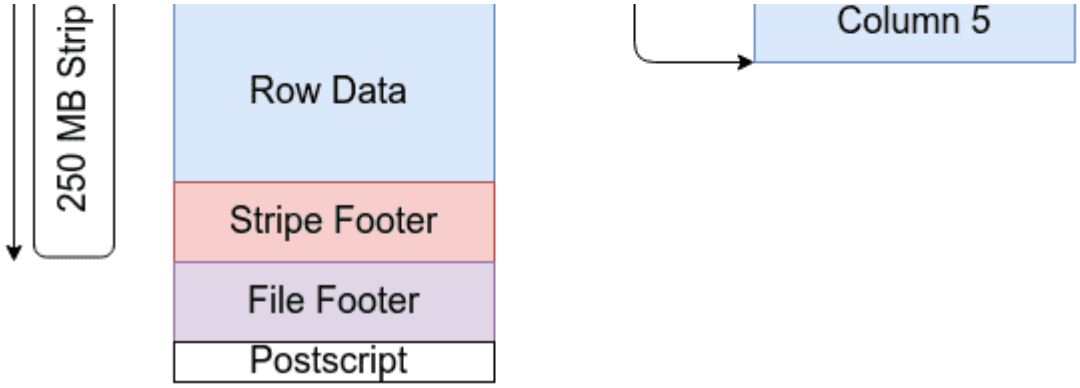
Le format ORC a été créé et open source par Hortonworks en collaboration avec Facebook. Il s'agit d'un format de stockage de données orienté colonne similaire à Parquet. Les fichiers ORC contiennent des groupes de données de ligne appelées

bandes, ainsi que des informations auxiliaires dans un pied de page de fichier. À la fin du fichier, un post-script contient les paramètres de compression et la taille du pied de page compressé. La taille de bande par défaut est de 250 Mo. Les grandes tailles de bande permettent de grandes lectures efficaces à partir de HDFS.

- *Avantages*
  - Très compressible, réduisant la taille des données originales jusqu'à 75%.
  - Plus efficace pour OLAP que pour les requêtes OLTP.
  - Limité au traitement par batch.
- *Inconvénients*
  - Impossibilité d'ajouter des données sans avoir a recréer le fichier.
  - Ne prend pas en charge l'évolution du schéma.
  - [Impala] (https://docs.cloudera.com/documentation/enterprise/5-5-x/topics/impala.html) ne supporte actuellement pas le format ORC.
- *Écosystèmes*
- Efficace pour les traitements de Business Intelligence.
- ORC améliore les performances de lecture, d'écriture et de traitement dans Hive.

Structure d'un fichier ORC





Conclusion

Le choix d'un type de stockage est liée au type de requête (cas d'utilisation) et aux propriétés de chaque format de fichier. Habituellement, les bases de données orientées lignes sont bien adaptées aux charges de travail de type OLTP tandis que les systèmes orientés colonnes sont bien adaptés aux requêtes OLAP. Étant donné que son type de stockage de colonne contient un type uniforme dans chaque colonne, le format en colonnes sera meilleur en compression par rapport à un stockage basé sur des lignes.

En termes de comparaison, si vous souhaitez travailler avec un format texte, le JSON lines et éventuellement le CSV lorsque vous contrôlez les types de données (pas de texte avec des caractères spéciaux) sont de bons choix. À moins que la source de données ne l'impose, évitez de sélectionner l'XML. Il n'est pratiquement pas utilisé dans le traitement des données en raison de sa grande complexité nécessitant souvent un parseur personnalisé. Dans les traitements en streaming et l'échange de données, si vous souhaitez un format flexible et puissant, sélectionnez Avro. Il fournit également un excellent support pour l'évolution de schéma. Bien que Protocol Buffers soit efficace en streaming, ils n'est ni divisible ni compressible, ce qui le rend moins attrayant pour le stockage de données au repos. Enfin, les formats ORC et parquet sont optimisés pour les analyses en Business Intelligence.

En résumé, la principale considération à garder à l'esprit est que la sélection d'un format de fichier dépend des cas d'utilisation.

Overview

Types	CSV	JSON	XML	AVRO	Protocol Buffers	
				text stored in		

Types	CSV	JSON	XML	AVRO	Protocol Buffers	
Text ou binary	text	text	text	JSON and data is binary format	text	b
Typage des données	non	oui	non	oui	oui	o
Déclaration d'un schéma	non (minimale avec header)	externe pour validation	externe pour validation	oui	oui	o
Schema evolution	no	oui	oui	oui	no	o
Storage type	row	row	row	row	row	o
OLAP/OLTP	OLTP	OLTP	OLTP	OLTP	OLTP	O
Splittable	oui en version simplifiée	oui avec JSON lines	no	oui	no	o
Compression	oui	oui	oui	oui	oui	o
Batch	oui	oui	oui	oui	oui	o
Stream	oui	oui	oui	oui	oui	n
Ecosystems	populaire grâce à sa simplicité	API et web	enterprise	Big Data et Streaming	RPC et Kubernetes	E D E

Canada - Morocco - France

Nous sommes une équipe passionnées par l'Open Source, le Big Data et les technologies associées telles que le Cloud, le Data Engineering, la Data Science le DevOps...

Nous fournissons à nos clients un savoir faire reconnu sur la manière d'utiliser les technologies pour convertir leurs cas d'usage en projets exploités en production, sur la façon de réduire les coûts et d'accélérer les livraisons de nouvelles fonctionnalités.

Si vous appréciez la qualité de nos publications, nous vous invitons à nous contacter en vue de coopérer ensemble.