MIAGE Nice RÉSEAU DES MIAGE DE FRANCE

UNIVERSITÉ CÔTE D'AZUR

Projet de recherche sur les SGBD du marché

Dans le cadre du cours FSGBD

(M1 MIAGE UCA NICE)

2021 - 2022

L'unité d'Enseignement : Fonctionnement d'un SGBD

Enseignant : Monsieur Gabriel MOPOLO
Monsieur Gregory GALLI

Février 2021

Réalisé par MianMian SHAN, Rémi DUFEU, Sébastien CHOISY et Ewen HÄNNI

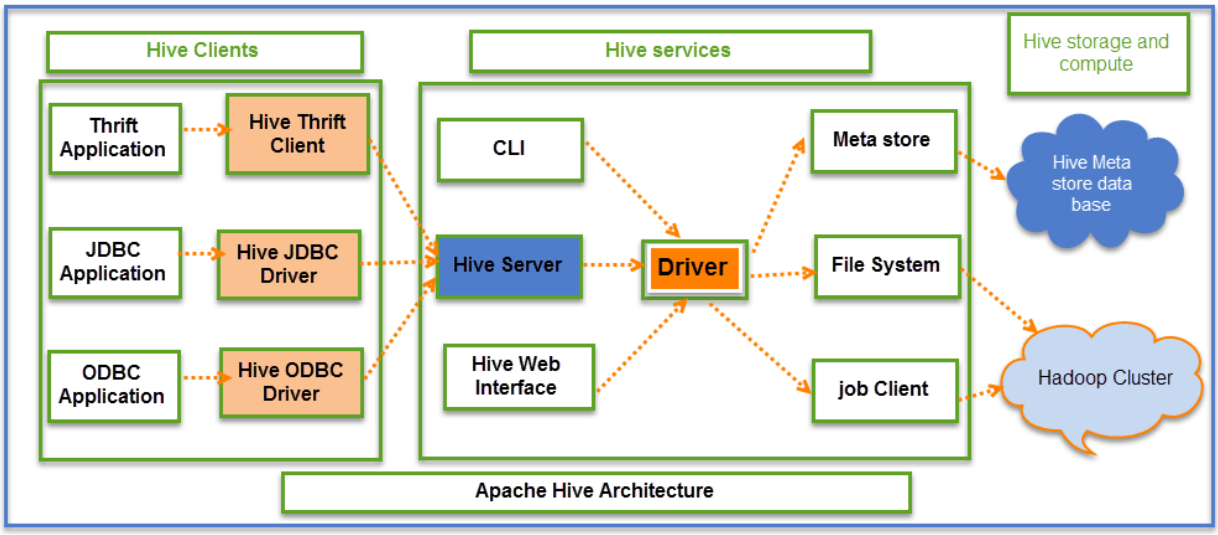
Partie 1 : Evaluation d'un SGBD relationnel du marché

1. Identification du SGBD

Nous avons choisi **APACHE HIVE**, paru le 9 novembre 2011 initialisé par Facebook et développé par une communauté. **Apache Hive** est un système d'entrepôt de données open source qui permet d'interroger et analyser de grands ensembles de données stockées sur Hadoop.
Hive utilise une base de données Derby pour stocker les métadonnées. Les autres bases de données supportées à part Derby, sont MySQL, MS SQL Server, Oracle et Postgres.

2. Architecture fonctionnelles du SGBD choisi

Architecture du Apache Hive



Les caches mémoires et leurs rôles

Apache Hive utilise deux caches différents : le "**query result cache**" ou "**cache des résultats de requêtes**" et un "**metadata cache**" ou "**cache des métadonnées**".

Le "**query result cache**" sauvegarde les requêtes et leurs résultats afin de ne pas exécuter plusieurs fois une même requête inutilement. Pour gagner du temps si la requête à déjà été exécutée son résultat sera chargé dans le cache. Pour l'activer ou le désactiver il faut changer le paramètre **hive.query.results.cache.enabled** qui est un Boolean. La taille du cache par défaut est de 2GB, modifiable avec l'argument **hive.query.results.cache.max.size..** Son chemin de sauvegarde est /tmp/hive/__resultcache__/.

Le "**metadata cache**" lui est consacré aux requêtes RDBMS. La mise en cache des métadonnées permet d'accélérer le temps de compilation d'environ 20% dans une installation MySQL.

Gestion des connexions

Apache Hive est un service de données Cloud. Vous pouvez utiliser le connecteur de données d'Apache Hive afin d'importer les données Hive de votre entreprise.
Pour créer la ressource de connexion à Apache Hive, nous avons besoin des informations de connexions suivantes :

- Nom de la base de données
- Nom d'hôte ou adresse IP
- Numéro de port
- Chemin HTTP (facultatif) : chemin du point de terminaison, tel que la passerelle, la valeur par défaut ou la ruche si le serveur est configuré pour le mode de transport HTTP.
- Nom d'utilisateur et mot de passe
- Certificat SSL (si requis par le serveur de base de données)

Les connexions Hive effectuées par le biais de la fenêtre Accès aux données utilisent un paramètre de chaîne de connexion EnableUniqueColumnName défini par défaut sur 0. Ce paramètre doit avoir la valeur 0 pour s'assurer que les bons noms de colonnes sont récupérés au moment de la connexion.

Si vous créez une connexion Hive à l'aide d'un DSN plutôt qu'à partir de la fenêtre Accès aux données, cette valeur est définie sur 1 par défaut. Pour que votre connexion fonctionne, vous devez remplacer cette valeur par 0 dans le registre Windows.

Processus d’exécution des requête

Hive fournit un langage de requête basé sur le SQL appelé *HiveQL* (Hive Query Language), qui est utilisé pour adresser des requêtes aux données stockées sur le HDFS. Le HiveQL permet également aux utilisateurs avancés/développeurs d'intégrer des fonctions Map et Reduce directement à leurs requêtes pour couvrir une plus large palette de problèmes de gestion de données.

Pour émettre une requête :

- Select...
- From...
- Where…

Qui va donc se diriger dans la table qui nous intéresse pour y chercher ce que l’on souhaite tout en respectant un ensemble de contraintes données.

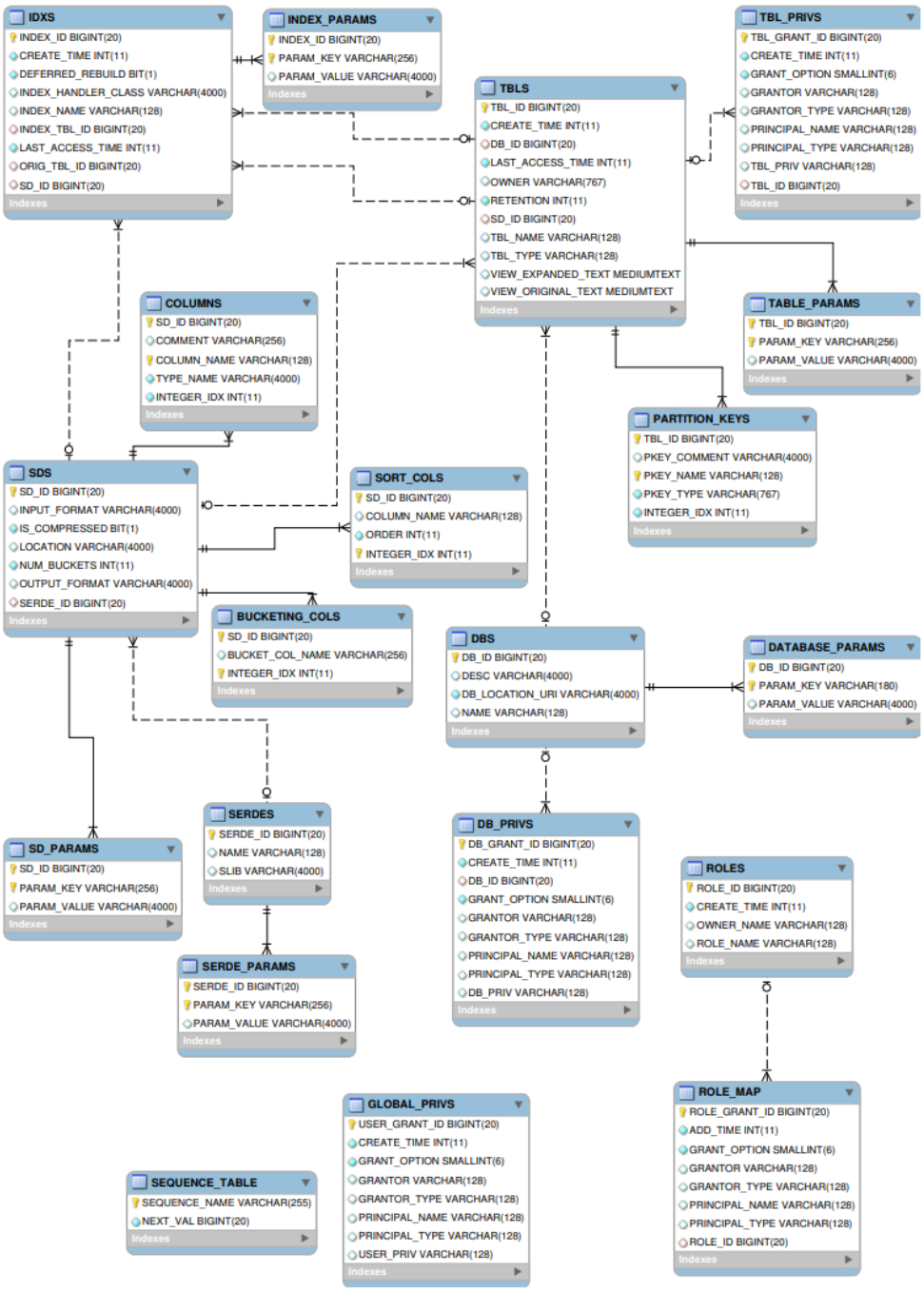
```
INSERT INTO
SELECT
FROM … JOIN …. ON
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT
```

Clauses SQL standards supportées par HiveQL

```
CREATE / ALTER / DROP TABLE / DATABASE
```

Exemple de commandes de définition de structure avec HiveQL

3. Le dictionnaire de données du moteur choisi

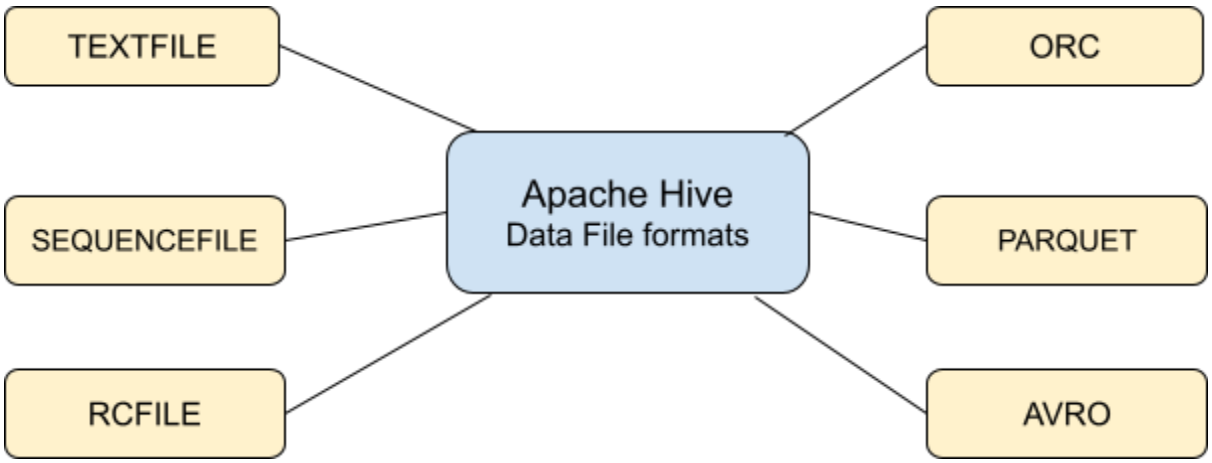


Metastore est un composant de Hive qui stocke toutes les métadatas des bases de données, des définitions de table et d'autres méta-informations. Par défaut, le metastore est stocké dans la base de données derby sur le système de fichiers local fourni avec la distribution Hive.

Un inconvénient de l'utilisation des bases de données derby c'est que vous ne pouvez pas avoir plus d'une instance de Hive en cours d'exécution. Mais généralement ce n'est pas le cas, vous donneriez l'accès à vos tables de Hive à plusieurs utilisateurs.
Donc, une façon de le faire est de partager le metastore. Cela peut être réalisé si vous créez votre metastore dans l'un des SGBDR et donnez l'accès à vos tables de Hive via le SGBDR comme l'image ci-dessus.

4. Organisation physique du SGBD

Voici les formats de fichiers de données Apache Hive qu'on peut utiliser pour traiter une variété de formats de données dans les écosystèmes Hive et Hadoop :



TEXTFILE : est le format de fichier par défaut, sauf si le paramètre de configuration hive.default.fileformat à un paramètre différent. Nous pouvons créer une table sur ruche en utilisant les noms de champs dans notre fichier texte délimité.

SEQUENCEFILE : permet de stocker les données compressées. Vous pouvez importer des fichiers texte compressés avec Gzip ou Bzip2 directement dans une table stockée sous TextFile. La compression sera détectée automatiquement et le fichier sera décompressé à la volée pendant l'exécution de la requête.

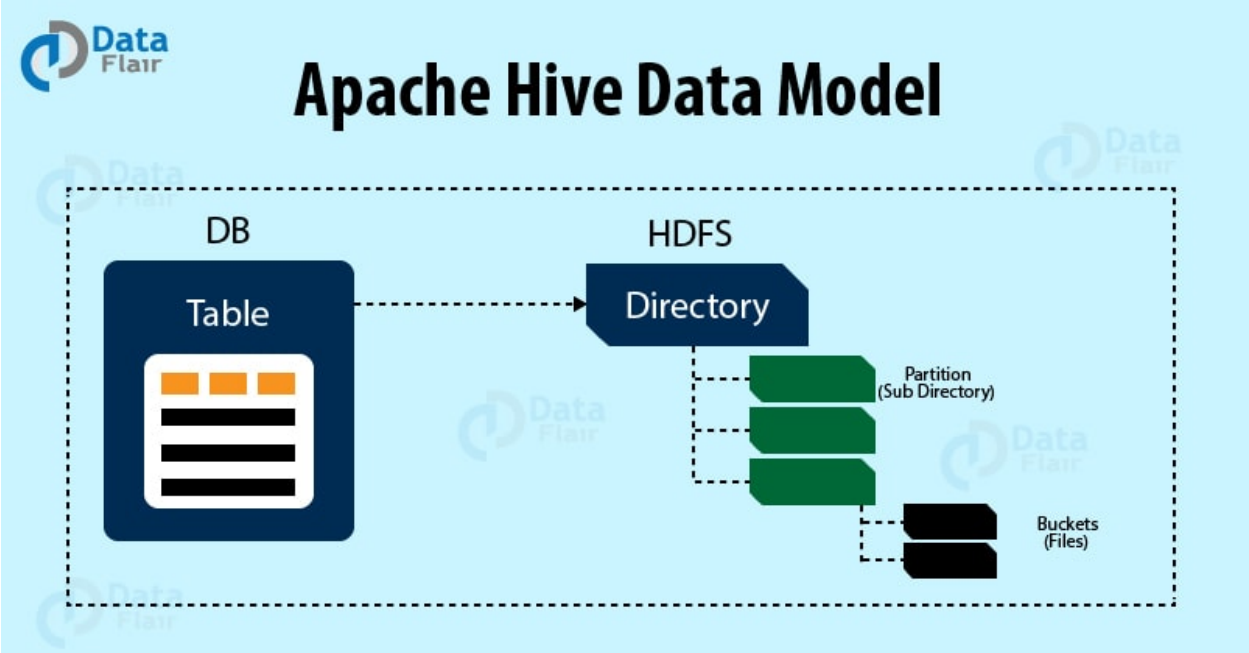
RCFILE : permet d'enregistrer le fichier en colonnes. C'est un format de ruche interne (binaire) efficace et supporté nativement par Hive. Utilisé lorsque l'organisation en colonnes est une bonne option de stockage pour certains types de données et d'applications. Si les données sont stockées par colonne au lieu de par ligne, seules les données des colonnes souhaitées doivent être lues, ce stagiaire améliore les performances.

ORC : Le format de fichier ORC (Optimized Row Columnar) offre un moyen très efficace de stocker des données Hive. Il a été conçu pour surmonter les limitations des autres formats de fichiers Hive. L'utilisation de fichiers ORC améliore les performances lorsque Hive lit, écrit et traite des données. Le fichier ORC peut contenir des index légers et des filtres de floraison.

PARQUET : Format de stockage en colonnes de parquet dans Hive 0.13.0 et versions ultérieures. Le parquet est conçu à partir des structures de données imbriquées complexes et utilise l'algorithme de destruction et d'assemblage d'enregistrements décrit dans l'article de Dremel. Nous pensons que cette approche est supérieure à la simple mise à plat des espaces de noms imbriqués.

AVRO : Les fichiers Avro sont pris en charge dans Hive 0.14.0 et versions ultérieures. Avro est une structure d'appel de procédure distante et de sérialisation de données développée dans le projet Hadoop d'Apache. Il utilise JSON pour définir les types de données et les protocoles et sérialise les données dans un format binaire compact. Son utilisation principale se situe dans Apache Hadoop, où il peut fournir à la fois un format de sérialisation pour les données persistantes et un format filaire pour la communication entre les nœuds Hadoop et entre les programmes clients et les services Hadoop.

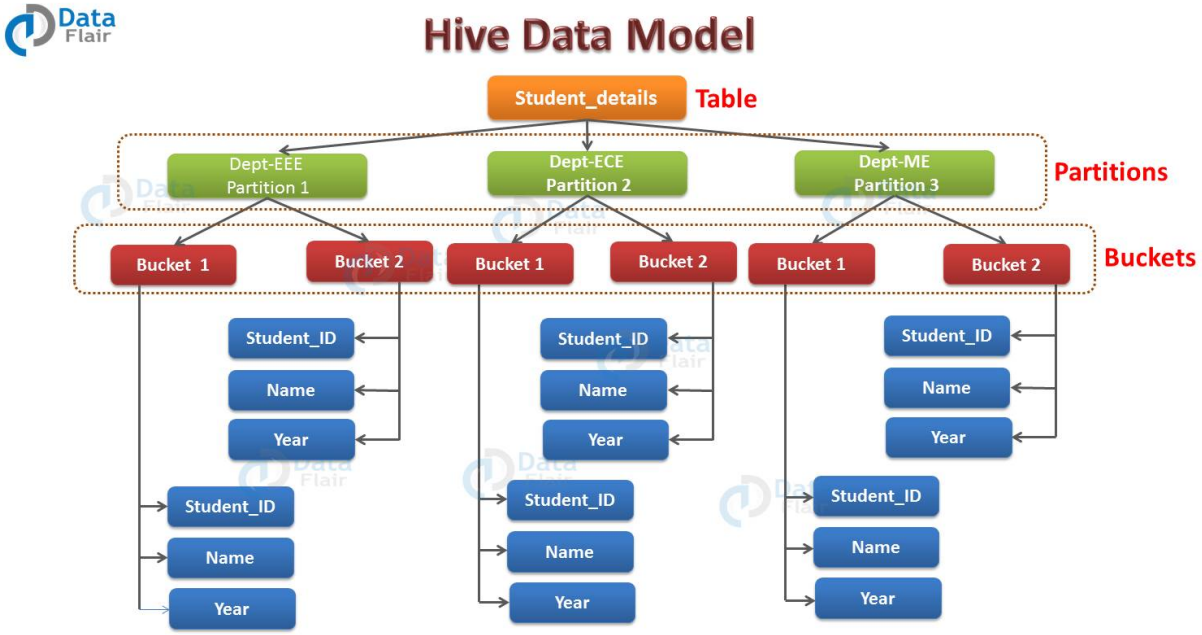
5. Organisation logique des données



Hive est un système d'entrepôt de données open source construit sur Hadoop Haused pour interroger et analyser de grands ensembles de données stockés dans des fichiers Hadoop. Il traite les données structurées et semi-structurées dans Hadoop. Les données d'Apache Hive peuvent être classées en :

- Tableau
- Cloison
- Seau

Les données qui y sont stockées peuvent être primitives (booléen, string etc) ou non (tableaux, cartes qui correspondent à des associations de tableaux ou objets sérialisés)



Apache Hive organise les tables en partitions pour regrouper le même type de données en fonction d'une colonne ou d'une clé de partition. Chaque table de Hive peut avoir une ou plusieurs clés de partition pour identifier une partition particulière. En utilisant la partition, nous pouvons également accélérer les requêtes sur des tranches de données… Il n'y a pas de format privilégié pour les fichiers de stockage de données.

Métadonnées :

À la manière des bases de données traditionnelles, le metastore gère les métadonnées de partitionnement. Il gère l'abstraction des données et en particulier leurs formats, et les extractions et chargement de données lors des requêtes, sans avoir à préciser ces informations : elles sont liées à la table et réutilisées lors de chaque requête.
Le metastore est composé de trois couches d'abstraction : la base, les tables et les partitions. La base stocke les tables.Les tables contiennent la liste des colonnes, leurs propriétaires, ainsi que des informations sur le stockage et la sérialisation/
désérialisation.

Mais également les clefs et valeurs des données. Les informations
sur le stockage incluent notamment l'emplacement des données sous-jacentes, les formats d'entrée et de sortie des fichiers et les informations de compartimentage.
Les métadonnées sérialisation / désérialisation incluent la classe d'implémentation du sérialiseur et du désérialiseur et toute information de support requise par l'implémentation. Toutes ces informations peuvent être fournies lors de la création du tableau.
Les partitions : chacune d'entre elles peut contenir ses propres colonnes et informations de stockage et de Serde (Sérialisation / désérialisation). Leur rôle est de faciliter les modifications de schéma de données sans affecter les autres partitions.

Architecture :

Le metastore stocke des objets et une base de données liée ou un système de sauvegarde par fichier. Le stockage avec redondance dans la base est implémenté avec un outil de mapping objet-relationnel qu'est DataNucleus. Et ce dans le but de satisfaire des exigences sur la capacité à interroger les métadonnées. L'utilisation d'un datastore dédié et donc séparé en lieu et place du Hadoop Distributed File System (HDPS) pose des problèmes de synchronicité et de scalabilité des données. De plus, implémenter un stockage d'objets au HDFS est techniquement complexe à cause du manque de mise à jour aléatoire sur ses fichiers.
Une "thrift " interface prévoit l'intégration de Apache Thrift qui crée les "thrift" types, des regroupement de types primitifs, ce qui permet une meilleure compatibilité, thrift étant présent dans de nombreux langages.

Hive query :

Hive n'utilise pas SQL mais un langage de requête de données similaire : HiveQL. Un des principaux avantages de HiveQl est de permettre une abstraction SQL pour les applications nécessitant MapReduce. HiveQL prévoit le passage d'un langage scripté à une suite de requêtes Mapreduce. Mapreduce est un concept qui correspond à une abstraction architecturale normée qui permet le traitement de données très volumineuse en parallèle et de manière distribuée. Sans entrer dans les détails du software ,il fonctionne par fractionnement et réduction – c'est-à-dire simplification – de requêtes.

Le fonctionnement du point de vue de l'architecture est le suivant :

- L'interface envoie les requêtes aux pilotes.
- Le pilote soumet la requête au compilateur qui crée un plan de requêtes et le convertit en un plan de tâches MapReduce.
- Le compilateur est mis en relation avec le Hive metastore qui contient les schémas de données. Le metastore charge également les pilotes (DLL).

- Le moteur d'exécution envoie les processus à Hadoop. Hadoop utilise son moteur Serde (Sérialisation / désérialisation) pour convertir les formats des données à l'origine et à destination des objets bruts contenus dans les colonnes de la base.
- Enfin le résultat parvient à l'interface.

Compilateur :

Le compilateur est divisé en plusieurs niveaux logiques :

- Le parser : Il transforme une requête String - donc constituée d'un ensemble de caractères - en arbre syntaxique.

- Le semantic analyser : Il transforme l'arbre syntaxique en une représentation de requête interne basée sur des blocs et non une arborescence d'opérateurs.
Lors de cette étape, les noms de colonne sont vérifiés et des expansions telles que * sont effectuées.
La vérification de type et les conversions de type implicites sont également effectuées. Si la table est une table partitionnée, ce qui est généralement le cas, toutes les expressions de cette table sont collectées afin qu'elles puissent être utilisées ultérieurement pour élaguer les partitions qui ne sont pas nécessaires.
Si la requête a spécifié un échantillonnage, celui-ci est également collecté pour être utilisé ultérieurement.

Logical plan generator : Il transforme la représentation de requête interne en un plan
Logique c'est-à-dire une arborescence d'opérateurs. Certains des opérateurs sont des opérateurs d'algèbre relationnelle comme 'filter', 'join' etc.

Mais certains des opérateurs sont spécifiques à Hive et sont utilisés plus tard pour convertir ce plan en une série de tâches de MapReduce. Ces opérateurs sont des opérateurs ReduceSink qui se trouvent aux frontières de MapReduce. Cette étape inclut également l'optimiseur pour transformer le plan afin d'améliorer les performances, il réalise des transformations telles que: la conversion d'une série de jointures en une seule jointure multidirectionnelle, l'exécution d'une agrégation partielle côté carte pour un group-by, l'exécution d'un group-by en 2 étapes pour éviter le scénario où un seul réducteur peut devenir un goulot d'étranglement en présence de données biaisées pour la clé de regroupement. Chaque opérateur comprend un descripteur qui est un objet sérialisable.

Query plan generator :

Il convertit le plan logique en une série de tâches de MapReduce. L'arborescence des opérateurs est parcourue de manière récursive, pour être divisée en une série de tâches sérialisables de MapReduce qui peuvent être soumises ultérieurement au framework MapReduce vers le système de fichiers distribué Hadoop.
L'opérateur reduceSink est situé à la limite de map-reduce, dont le descripteur contient les clés de réduction. Les clefs de réduction dans le descripteur reduceSink sont utilisées comme clés de réduction dans la limite map-reduce. Le plan comprend les échantillons/partitions requis si la requête le spécifie. Le plan est finalement sérialisé et écrit dans un fichier.

Optimizer :

L'optimiseur effectue d'autres transformations de plans encore telles que l'élagage de colonnes ou le refoulement de prédicats.

Fonctionnement de Hadoop MapReduce :

Pour exécuter ses fonctions, l'architecture contient deux éléments de traitement principaux conçus à la manière d'un contrôleur-opérateur. Ceux-ci sont:

Le Job Tracker : Le contrôleur. Il est responsable de la planification des travaux et de l'envoi de commandes d'exécution au composant opérateur qui réside dans chaque nœud. Il est également responsable de la réexécution des tâches ayant échoué.

Le Task tracker : l'opérateur. Il existe sur chaque nœud, exécute les instructions et relaie les commentaires au composant contrôleur.

Comme expliqué ci-dessus, le fonctionnement de MapReduce se déroule en deux phases : la phase de carte et la phase de réduction.

Phase de mappage (Map phase) :

Il s'agit de la première phase qui se produit lors de l'exécution d'une tâche de traitement dans le système Hadoop. Le framework comprend les données sous la forme de paires <key,value>, et par conséquent, les données d'entrée doivent être prétraitées pour correspondre au format attendu. L'ensemble de données d'entrée est décomposé en morceaux plus petits via une méthode appelée division logique.

Les blocs logiques sont ensuite affectés aux mappeurs disponibles, qui traitent chaque enregistrement d'entrée en paires <clé, valeur>. La sortie de cette phase est considérée comme une sortie intermédiaire.

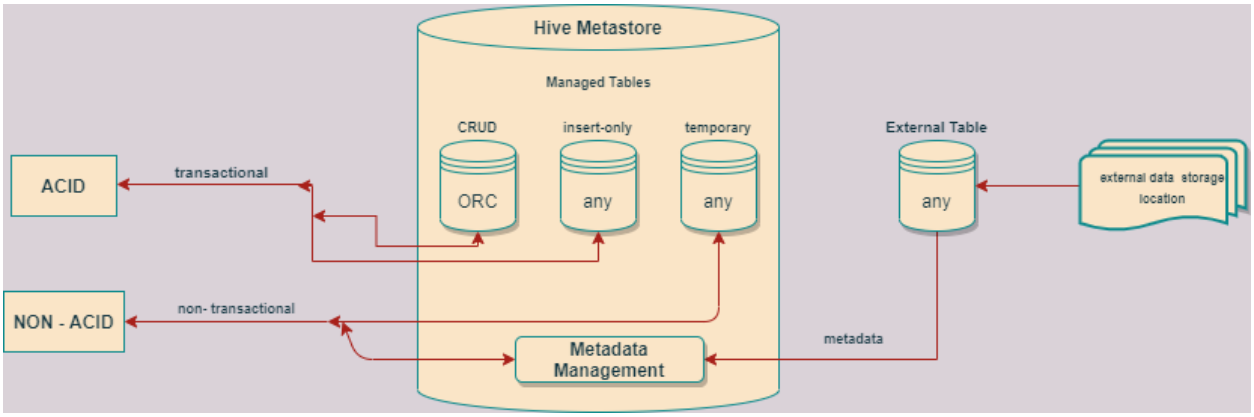
La sortie peut subir un processus intermédiaire où les données de sortie du mappeur sont traitées plus avant avant d'être introduites dans les réducteurs. Les processus comprennent la combinaison, le tri, le partitionnement et le brassage. Pour en savoir plus sur l'importance de ces processus, cliquez ici. Les données intermédiaires de ces processus sont stockées dans un système de fichiers local au sein du nœud de traitement respectif.

Phase de réduction
Le nombre de réducteurs pour chaque tâche est configurable et peut être défini dans le fichier de configuration mapred-site.xml.

La phase de traitement du réducteur prend la sortie de la phase de mappage et traite les données pour générer la sortie finale, qui est enregistrée dans un fichier de sortie au sein du système de fichiers distribués Hadoop (HDFS) par une fonction appelée enregistreur d'enregistrement.

6. Gestion de la concurrence d'accès

Notion de transaction



Hive prend en charge les instructions SQL d'insertion/mise à jour/suppression avec une sémantique transactionnelle et des opérations de lecture qui s'exécutent lors de l'isolement d'instantané.

Support des propriétés ACID

Les tables Hive ACID gèrent les données dans les fichiers de base et delta, ce qui augmente les performances de la tâche.

Atomicité : la transaction ou l'opération qui réussit ou échoue complètement, et ne laisse aucun état partiel.

Cohérence : Si une transaction est effectuée, les résultats de cette transaction sont visibles pour chaque requête/transaction.

Isolation : Si une transaction incomplète est effectuée, l'autre requête/transaction ne provoque pas d'effets secondaires inattendus pour les autres utilisateurs, ce qui signifie que toutes les transactions sont séparées les unes des autres.

Durabilité : Si la transaction est complète alors elle sera préservée même face à une panne de machine ou de système.
Lorsque nous écrivons des données continues dans la partition Hive, cela crée de nombreux petits fichiers toutes les quelques secondes qui dégradent les performances de Hive. Avec l'aide d'ACID, nous pouvons insérer/mettre à jour/supprimer sur la même partition Hive sans affecter les performances de la table.

Parfois, les données stockées dans le lac de données peuvent être incorrectes ou les entreprises peuvent souhaiter des modifications en fonction d'un changement de politique ou d'une exigence commerciale. Dans de telles situations, le défi consiste à corriger les données qui peuvent être obtenues par des opérations d'insertion/mise à jour/suppression.

Avec la fusion en bloc, nous pouvons fusionner de petits fichiers en un seul fichier sans affecter les performances de lecture.

Technique de gestion de la concurrence d'accès

La prise en charge de la concurrence est indispensable dans les bases de données et leurs cas d'utilisation sont bien compris. Au minimum, nous souhaitons prendre en charge les lecteurs et rédacteurs simultanés dans la mesure du possible. Il serait utile d'ajouter un mécanisme pour découvrir les verrous actuels qui ont été acquis. Il n'y a aucune exigence immédiate d'ajouter une API pour acquérir explicitement des verrous, de sorte que tous les verrous seraient acquis implicitement.

Les modes de verrouillage suivants seront définis dans la ruche (notez que le verrouillage d'intention n'est pas nécessaire).

- Partagé (S)
- Exclusif (X)

La sérialisation

SerDe est l'abréviation de Sérialiser/Désérialiser. Hive utilise l'interface SerDe pour IO. L'interface gère à la fois la sérialisation et la désérialisation et interprète également les résultats de la sérialisation en tant que champs individuels pour le traitement.

Un SerDe permet à Hive de lire les données d'une table et de les réécrire sur HDFS dans n'importe quel format personnalisé. N'importe qui peut écrire sa propre SerDe pour ses propres formats de données.

Le rôle de la sérialisation et de la désérialisation

1. La sérialisation est le processus de conversion d'objets en séquences d'octets.

2. La désérialisation est le processus de restauration du bytecode sur un objet.

La sérialisation a deux fonctions principales :

- Persistance orientée objet, autrement dit, enregistrez le fichier après avoir converti l'objet en bytecode
- Transmission de données d'objets

La fonction principale de la désérialisation :

Désérialiser <clé, valeur> dans la valeur de chaque colonne de la table Hive, Hive peut facilement charger des données dans la table sans convertir les données, ce qui peut faire gagner beaucoup de temps lors du traitement de données volumineuses.

7. Gestion des transactions distribuées

Une nouvelle entité logique appelée "gestionnaire de transactions" a été ajoutée, il est désormais responsable de la gestion des verrous de transactions. Le DummyTxnManager par défaut émule le comportement des anciennes versions de Hive : n'a pas de transactions et utilise la propriété hive.lock.manager pour créer un gestionnaire de verrouillage pour les tables, les partitions et les bases de données.
Le DbTxnManager nouvellement ajouté gère tous les verrous/transactions dans le metastore Hive avec DbLockManager (les transactions et les verrous sont durables en cas de défaillance du serveur). Cela signifie que le comportement précédent de verrouillage dans ZooKeeper n'est plus présent lorsque les transactions sont activées. Pour éviter que les clients meurent et laissent la transaction ou les verrous en suspens, un battement de cœur est envoyé régulièrement par les détenteurs de verrous et les initiateurs de transaction au metastore. Si une pulsation n'est pas reçue dans le délai configuré, le verrou ou la transaction sera abandonné.

8. Gestion de la reprise sur panne

9. Techniques d’indexation

L'indexation est supprimée depuis la version 3.0.

Il existe d'autres options qui pourraient fonctionner de la même manière que l'indexation :

- Les vues matérialisées avec réécriture automatique peuvent donner des résultats très similaires. Hive 2.3.0 ajoute la prise en charge des vues matérialisées.
- En utilisant des formats de fichiers en colonnes (Parquet , ORC) - ils peuvent effectuer une analyse sélective ; ils peuvent même ignorer des fichiers/blocs entiers.

Exemple : Créer un index

```
CREATE INDEX index_name
ON TABLE base_table_name (col_name, ...)
AS 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IDXPROPERTIES (property_name=property_value, ...)]
[IN TABLE index_table_name]
[PARTITIONED BY (col_name, ...)]
[
  [ ROW FORMAT ...] STORED AS ...
  | STORED BY ...
]
[LOCATION hdfs_path]
[TBLPROPERTIES (...)]
[COMMENT "index comment"]
```

10. Optimisation de requêtes

Dans Hive, l'optimiseur de requête ne cherche pas le plan optimal pour les jointures mais cherche à exprimer la requête sous forme d'un arbre dont les nœuds sont des MapReduce.
Le modèle de coût utilisé est complexe, et fait intervenir entre autres les coûts de communications réseau.

Partie 2 : Projet sur les thématiques liées à l'indexation et la recherche

Annexe :

Documentation :

https://drill.apache.org/docs/hive-metadata-caching/

https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.1.4/performance-tuning/content/hive_query_result_cache_ms_cache.html

https://help.talend.com/r/y1ddRZ4ppskQjK6OQmmqyw/WZ9FcVdQG486rFxpchSBNA

https://help.highbond.com/helpdocs/analytics/15/fr/Content/analytics/defining_importing_data/data_access_window/connecting_to_apache_hive.htm

https://data-flair.training/blogs/hiveql-select-statement/

https://big-data.developpez.com/faq/hadoop/?page=Hive#Quels-sont-les-differents-composants-d-une-architecture-Hive

https://juvenal-chokogoue.developpez.com/tutoriels/sql-dans-hadoop/#LII-A-1

https://learntutorials.net/fr/hive/topic/4513/formats-de-fichier-dans-hive

https://learntutorials.net/fr/hive/topic/4513/formats-de-fichier-dans-hive

https://cwiki.apache.org/confluence/display/Hive/AdminManual+Metastore+Administration

http://whatsbuzzingapachehive.blogspot.com/2016/02/using-apache-hive-to-convert-from-and.html

https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.1.4/using-hiveql/content/hive_3_internals.html

https://blog.clairvoyantsoft.com/hive-acid-transactions-part-i-f08da70b591b

https://cwiki.apache.org/confluence/display/hive/hive+transactions

https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.1.4/using-hiveql/content/hive_query_information_schema.html

https://docs.cloudera.com/runtime/7.2.10/using-hiveql/topics/hive_hive_3_tables.html

https://issues.apache.org/jira/browse/HIVE-1293

https://cwiki.apache.org/confluence/display/Hive/Locking

https://cwiki.apache.org/confluence/display/Hive/Data+Connectors+in+Hive

https://blog.clairvoyantsoft.com/hive-acid-transactions-part-i-f08da70b591b

https://cwiki.apache.org/confluence/display/HIVE

https://cwiki.apache.org/confluence/display/Hive/Design

https://www.pluralsight.com/guides/getting-started-with-hadoop-mapreduce

https://community.cloudera.com/t5/Community-Articles/A-Day-In-the-Life-of-a-Hive-Query/ta-p/287905