Livre blanc

Accélérer la BI sur Hadoop : analyse complète, Cubes ou index ?

jethro

Aperçu

Les organisations stockent de plus en plus de données dans Hadoop et les lacs de données basés sur le cloud. Naturellement, les utilisateurs BI commencent à pointer leur Applications BI sur cette source de données en pleine croissance. Traditionnellement, beaucoup Les applications BI fonctionnent en extrayant des données d'une source de données, puis en le chargeant dans la mémoire de l'outil BI pour un traitement rapide. Cependant, à mesure que la taille des ensembles de données augmente en BB de lignes, cette approche n'est pas plus pratique car les données sont tout simplement trop volumineuses pour être traitées en mémoire. Dans ce cas, la plupart des outils BI offrent une alternative : au lieu d'importer les données dans l'outil BI, poussez les requêtes en direct à la source de données. C'est ce qu'on appelle Live Connect (dans Tableau), Découverte directe (Qlik), DirectQuery (PowerBI), Accès direct (MicroStrategy) et ainsi de suite. Lors du passage à l'accès à la base de données en direct, les limitations de taille ne s'appliquent plus. Cependant, la vitesse d'interaction BI dépend désormais des performances du moteur de la source de données.

Le problème

Il existe plusieurs moteurs SQL qui peuvent être utilisés comme source de données sur Hadoop. Il s'agit notamment de Hive, Impala, Presto, SparkSQL, Drill et autres. Tous ces outils partagent la même architecture de base, appelée Analyse complète MPP. Bien qu'ils soient parfaitement adaptés à l'ETL et aux données les charges de travail scientifiques telles que la modélisation prédictive et l'apprentissage automatique, ces outils sont grandement mis à rude épreuve lorsqu'ils sont utilisés pour des applications BI. La raison car la charge de travail BI a des caractéristiques uniques :

- Concurrence : un tableau de bord BI typique émettra de nombreuses requêtes pour afficher les résultats. Multipliez cela par des centaines d'utilisateurs et le système est inondé de requêtes.
- Performance : les utilisateurs BI s'attendent à ce que leurs tableaux de bord répondent pas plus de 5 à 10 secondes. De plus, il n'est pas acceptable que certains graphiques sont rendus rapidement – l'ensemble du tableau de bord doit être complété.
- Sélectivité et variance : un utilisateur est généralement intéressé par un sous-ensemble relativement petit de données lors d'une interaction donnée et utiliserait plusieurs filtres pour l'identifier. Cependant, chaque utilisateur est susceptible d'être intéressé par un sous-ensemble différent.
- Ad-Hoc et Agile : grâce aux tableaux de bord et applications BI en libre-service, en constante évolution avec de nouvelles requêtes créées fréquemment.
- Manipulation de données complexes : l'analyse multidimensionnelle nécessite la jointure de tables, le tri des données, de grandes agrégations, et d'autres opérations coûteuses.

Les moteurs SQL-on-Hadoop ne sont pas adaptés au type et au volume des requêtes BI car leur architecture d'analyse complète nécessite énormément quantité de travail d'analyse redondant. Ces moteurs devront lire la colonne entière (c'est-à-dire toutes les lignes) de chaque colonne de filtre, de chaque requête, chacun temps. Quelques optimisations, comme cloisons/projections/etc.', ne sont pas non plus adaptés à un monde d'auto-service BI, dans lequel à tout moment, les requêtes peuvent utiliser un colonne pour filtrer ou trier, même si cette colonne n'est pas celle utilisée comme colonne pour le partitionnement. Le résultat n'est pas seulement lent, il ajoute également à la charge globale du système et entraîne un plafond pour le nombre de requêtes simultanées pouvant être servies.

Solutions de contournement

Afin d'accélérer l'accès en direct des applications BI à de grands ensembles de données dans Hadoop, les entreprises s'engagent dans une gamme de données manuelles et complexes projets d'ingénierie pour combler le fossé. Voici quelques exemples courants :

- Dénormalisation : comme la jointure des données de plusieurs tables a tendance à être lent, les données peuvent être fusionnées dans une seule grande table et les jointures peuvent être évité. Le commerce de est que nous avons un encore plus grand ensemble de données à conserver. Nous ne sommes plus en mesure d'ajouter progressivement données – au lieu de cela, nous devons reconstruire fréquemment l'ensemble de données complet. Ce Le modèle sacrifie la cohérence des applications en termes de flexibilité pour surmonter l'incapacité d'une source de données à fonctionner.

• Pré-agrégation : créez un grand nombre d'agrégations pour répondre à toutes les requêtes utilisées par l'application. Cela se traduit par maintenance continue, en essayant de suivre les modifications de l'application. Dans De plus, les requêtes granulaires devront toujours analyser complètement la table et souffriront de performances lentes.

• Multi-partitionnement des données : création de plusieurs répliques du données avec différentes clés de partition afin de fournir des requêtes plus rapides sur plusieurs colonnes de filtre.

Tous ces éléments s'avèrent exigeants en main d'œuvre, coûteux à entretenir et finalement inefficace.

Cubes OLAP sur Hadoop

Une solution émergente consiste à utiliser un logiciel commercial (AtScale, Kyvos, Outils Dremio, Kyligence) ou open source (Kylin) pour créer des cubes OLAP au-dessus des ensembles de données Hadoop. Ceux-ci peuvent ensuite être utilisés pour accélérer requêtes pouvant correspondre à l'un des cubes. Comme nous le savons par beaucoup années de travail avec des cubes, cette approche présente plusieurs limites :

• Manuel : les cubes et les agrégations doivent être définis manuellement pour répondre aux besoins des applications. Ce processus nécessite des experts avec connaissance approfondie du modèle de données et des requêtes des applications. Comme l'application et les changements de données, une équipe de ces experts est nécessaire de modifier constamment les cubes pour suivre changements en cours.

• Incomplet : les cubes fonctionnent bien pour les requêtes hautement agrégées. Cependant, pour les requêtes très granulaires, les cubes sont simplement pas pratique. Par exemple, pensons à un ensemble de données qui comprend un colonne à cardinalité élevée telle que « customer_id » avec les MM de des valeurs uniques. Certaines requêtes peuvent nécessiter d'examiner un customer_id individuel. Construire un cube qui contient ceci La colonne donnera un cube si gros qu'il ne sera pas pratique. Applications où les utilisateurs ont tendance à filtrer selon plusieurs dimensions de la même manière, ils nécessitent des cubes volumineux et donc peu pratiques.

• Coûteux : dans la mesure où nous traitons de Big Data, les cubes peuvent devenir très volumineux, et globalement, encore plus volumineux que les données originales elles-mêmes. Tel les gros cubes sont difficiles à entretenir et sont particulièrement difficiles lorsque les données sont mises à jour quotidiennement, toutes les heures ou même toutes les quelques minutes. Le rafraîchissement et la reconstruction des cubes sont un processus qui demande beaucoup de main-d'œuvre.

Lorsque les requêtes ne peuvent pas utiliser un cube, elles sont obligées de s'exécuter avec l'un des moteurs SQL-on-Hadoop natifs et effectuez une analyse complète processus. Malheureusement, une solution partielle qui pose quelques requêtes fonctionner rapidement mais d'autres ne le font pas, est souvent inacceptable pour les utilisateurs de BI.

Index sur Hadoop

Une autre solution émergente consiste à indexer entièrement les ensembles de données dans Hadoop à l'aide de outils commerciaux (Jethro) ou open source (Druid). Dans ce cas, un DB- comme l'index est construit pour chaque colonne. Lorsque des requêtes sont envoyées depuis la BI app, les index sont utilisés pour affiner les lignes nécessaires à l'application. requête, au lieu d'effectuer une lente analyse complète. Avec l'indexation complète, nous

peut être préparé pour toute requête existante ou future, quel que soit le
sur quelles colonnes l'utilisateur choisira de filtrer, il y aura un
index pour l'accélérer.

Et comme pour toute autre technologie, les indices ont leurs propres compromis :

- Coûteux : les index nécessitent un stockage supplémentaire et la maintenance des index peut ralentir l'ingestion de nouvelles données.
- Incomplet : toutes les requêtes ne peuvent pas utiliser un index. Certaines requêtes n'utilisent pas de filtre du tout ou utilisent un filtre avec une cardinalité si faible que la plupart des lignes doivent encore être accessibles.

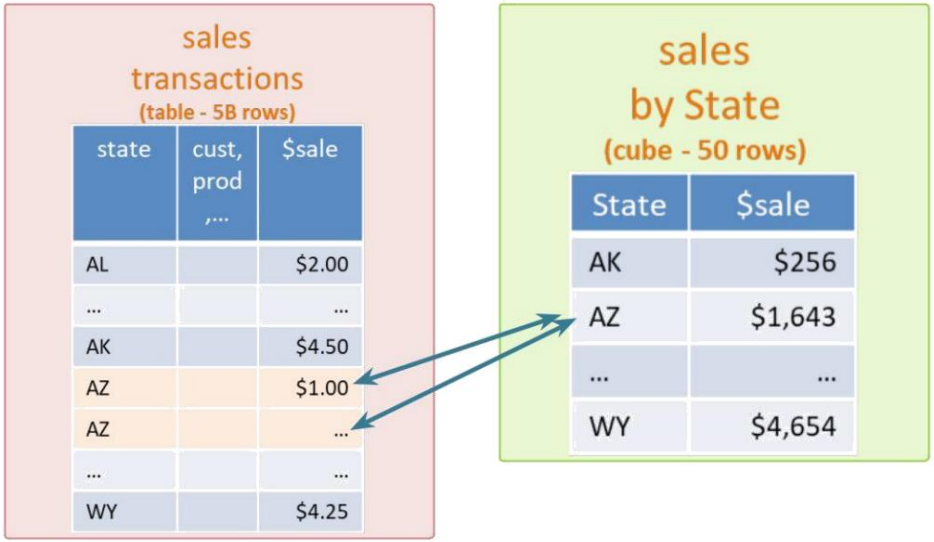
Lorsqu'une requête ne peut pas utiliser un index, elle est obligée d'effectuer une analyse complète
ce qui est généralement lent et consommateur de ressources.

Qu'en est-il des cubes et des index ?

Il s'avère que les cubes et les index sont parfaitement complémentaires
approches avec très peu de chevauchement. Les scénarios où les cubes seront
inefficaces sont les scénarios exacts dans lesquels les indices brilleraient, et
vice versa. Les cubes fonctionnent très bien lorsque les requêtes sont hautement agrégées,
et les cubes sont de petite taille. C'est généralement le cas lorsque peu de
les dimensions de cardinalité sont utilisées. Par exemple, un cube comprenant
des dimensions telles que le sexe, le groupe d'âge, l'état ou la catégorie de marché seront
être petit et très efficace. En même temps, utiliser des index pour filtrer
sur de telles colonnes de dimension ne sera pas très efficace en tant que facteur significatif
Une partie des données devra encore être numérisée. A l'inverse, les index
fonctionne très bien lorsque les requêtes sont très granulaires et recherchent un petit
sous-ensemble des données. Par exemple, une requête qui filtre par item_id
(MM de lignes de valeurs) et par code postal du client (milliers de
valeurs) ou une requête qui explore un client ou une transaction
données de niveau. Pour de telles requêtes granulaires, les cubes seront une mauvaise solution
car ils devront être très grands en raison des nombreuses cardinalités élevées
dimensions. Des cubes aussi gros seront peu pratiques et lents.
Utiliser à la fois des cubes et des index est le seul moyen de fournir rapidement
performances pour la gamme complète de requêtes BI – agrégées et
granulaire – et évitez complètement les analyses complètes lentes.

Cubes à Jethro

Les Jethro AutoCubes sont exactement comme leur nom l'indique : des cubes créés
automatiquement par Jethro au lieu de manuellement par un expert. Jéthro
surveille les requêtes en cours envoyées par les outils BI et examine chacune d'elles
pour voir s'il pourrait être servi de manière optimale par un cube. Si c'est le cas, Jethro créera
un tel cube en utilisant un processus en arrière-plan et stockez-le dans HDFS. Avenir
permutations de cette requête – différentes valeurs de filtre, n'importe quel sous-ensemble
combinaison des dimensions/mesures – sera servi à partir du
cube au lieu du tableau complet. Les cubes Jethro sont complètement transparents
à l'application BI qui ne connaît que les tables sous-jacentes.



Principales caractéristiques des AutoCubes Jethro

- - Cubes sur la table JOIN. Jethro crée deux types de cubes pour les requêtes incluant des tables jointes. Le premier est un cube « dimension » qui est construit sur la base des résultats de la requête, après la JOIN est effectué et peut être utilisé à la fois avec une étoile et un flocon de neige schémas. Le second est un cube « clé » qui est construit autour du JOIN clés des tables sous-jacentes. Un tel cube clé peut être utilisé pour un un grand nombre de requêtes futures car il est de nature générique.

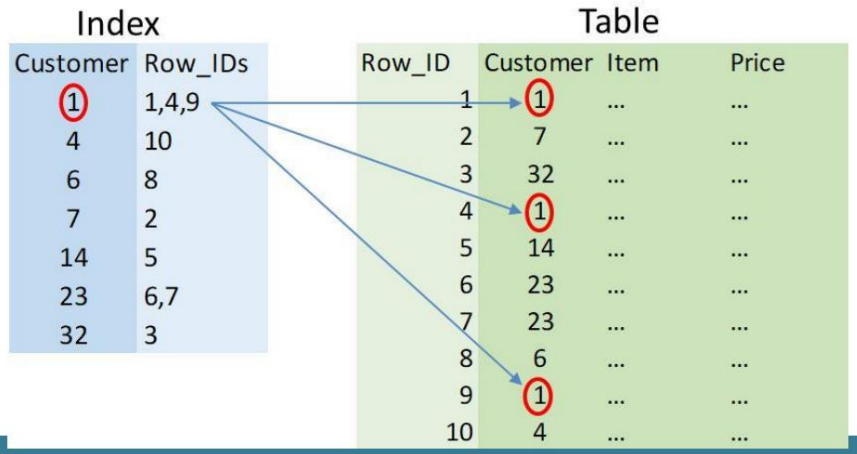
• Prise en charge des agrégations COUNT DISTINCT. Lorsque la mesure utilisé pour une requête est COUNTD Jethro stockera la liste des éléments distincts valeurs pour chaque entrée de cube au lieu du nombre total de ces entrées des valeurs uniques. Cela permet à Jethro d'utiliser le cube pour répondre non seulement combien de valeurs distinctes se trouvent dans une entrée spécifique (par exemple STATE='AZ") mais aussi des requêtes accédant à plusieurs entrées (par exemple état IN (« CA », « NY »)).

La maintenance du cube est également automatisée. Lorsque de nouvelles données arrivent, Jethro mettra progressivement à jour tous les cubes qu'il a construits. Comme le sont les applications modifié ou un nouveau ajouté, ou à mesure que de nouvelles colonnes sont ajoutées au modèle de données, Jethro adapte et crée automatiquement de nouveaux cubes pour les accueillir.

Parce que les cubes Jethro sont conçus pour fonctionner conjointement avec Index Jethro, Jethro ne créera que de petits cubes optimaux pour les requêtes agrégées. Gros cubes qui seraient nécessaires pour les requêtes granulaires sont évitées car ces requêtes sont optimisées à l'aide index à la place.

Index à Jethro

Jethro indexe automatiquement toutes les colonnes afin qu'aucun expert ne soit nécessaire pour le faire décidez quels cols sont les plus importants. Cette approche est essentielle pour l'auto-service BI et développement d'applications agiles, car les utilisateurs peuvent choisir librement comment pour accéder à leurs données et ne jamais vous inquiéter si une colonne spécifique est indexé ou non et quelles en seront les implications en termes de performances.



Les index Jethro sont des index DB typiques. Ils sont techniquement inversés
Listes – chaque valeur de colonne a une entrée avec la liste des lignes qui l'ont
valeur spécifique. Lorsqu'une requête arrive, Jethro évalue d'abord tous les filtres
et utilise l'index pour créer un ensemble de travail - une liste de toutes les lignes qui sont
nécessaire après avoir appliqué tous les filtres. Alors seulement, Jethro récupérera les données,
effectuez des JOINs et appliquez les agrégations pertinentes. Cela signifie que si
une requête est granulaire – pour accéder à un petit nombre de lignes, il faudra
peu de temps à réaliser quelle que soit la taille de l'ensemble de données.

Principales caractéristiques des index Jethro

• Les index sont implémentés sous forme de bitmaps compressés hiérarchiquement.
et sont stockés dans HDFS. Un index d'index est utilisé pour fournir un accès direct à chaque
entrée d'index, de sorte que les index n'ont pas besoin d'être stockés en mémoire.

• Les index JOIN sont utilisés pour créer un index sur une table de faits en utilisant les valeurs d'un
col d'une table de dimensions. De cette façon, lorsque la table de dimensions col est utilisée
comme filtre, aucun JOIN n'est nécessaire et
un index peut être utilisé directement

• Les index RANGE sont utilisés lorsqu'une colonne d'index comporte un grand nombre.
des valeurs et des requêtes sont généralement filtrées par une plage de valeurs

• Les index DATE sont créés automatiquement pour TIMESTAMP.
colonnes et créez des entrées pour les années, les mois, les jours, les RH, etc.
Il s'agit d'indices RANGE similaires dans la manière dont ils peuvent couvrir
une plage de dates utilisant une seule entrée d'index pour couvrir une année complète
au lieu de 365 entrées quotidiennes individuelles.

Les index dans Jethro sont mis à jour de manière incrémentielle. Lorsque incrémentiel
les données arrivent, Jethro ajoutera les index actuels. Avec ça
les index d'approche ne sont jamais verrouillés et donc même fréquents
le chargement de nouvelles données n'a aucun impact sur les performances des requêtes en cours.

Optimisation des requêtes dans Jethro

La meilleure façon d'accélérer une requête est de minimiser la quantité de travail réellement nécessaire pour le traiter. Jethro utilise une combinaison d'optimisation techniques allant des résultats de requêtes, aux cubes et aux index. Lorsqu'une requête arrive, il est d'abord divisé en sous-requêtes individuelles et chacune est évaluée séparément. La séquence d'optimisation :

• Cache des résultats de requête
un. Les résultats des requêtes passées sont enregistrés dans un stockage permanent.
Lorsqu'une nouvelle requête arrive, Jethro vérifie d'abord si la même requête est exacte.
la requête a déjà été exécutée et ses résultats ont été enregistrés. S'ils l'étaient, le
les résultats sont renvoyés sans aucun traitement supplémentaire.
b. Les résultats passés sont automatiquement mis à jour (incrémentalement ou
remplacement complet) lorsque de nouvelles données arrivent.

• AutoCubes
un. Jethro vérifie ensuite si une agrégation de cubes incluant le
Les dimensions et mesures de la requête existent déjà. Sinon, Jethro
vérifie si un cube clé peut être utilisé pour répondre à cette requête.
b. Le processus de mise en correspondance utilise les métadonnées des
cubes. Ils sont triés par taille, donc si une correspondance de cube est trouvée
utilisera le cube le plus petit et le plus efficace.

- Index

un. Si aucun résultat mis en cache et aucun cube n'est disponible, Jethro utilisera des index pour traiter la requête. Premièrement, les index sont utilisé pour appliquer tous les filtres, ce qui donne un ensemble de travail – liste de tous rangées nécessaires.

b. Ensuite, Jethro récupère les données de colonne pertinentes pour ces lignes et exécute la logique SQL sur ce sous-ensemble de données.

c. De nombreuses requêtes peuvent être servies directement à partir des index et ne nécessitent même pas l'accès aux données sous-jacentes. UN Un exemple courant est que les outils BI envoient un « SELECT DISTINCT col » requête pour obtenir la liste des valeurs distinctes pour un filtre. À Jethro, tel une requête est un simple accès à la liste des valeurs f à partir de l'index.

- Optimisation de l'exécution

un. Optimisation JOIN – Jethro utilise diverses techniques pour supprimer les JOIN. Ceux-ci incluent la transformation des étoiles, l'utilisation de Index JOIN, conversion des OUTER en INNER JOIN et changer l'ordre de la table de jointure après avoir appliqué des filtres.

b. Élagage de partition – si l'un des filtres est également une partition clé, Jethro ignorera tout traitement (par exemple, recherche d'index) du partitions inutiles. Cette fonctionnalité n'a pas autant d'impact dans Jethro tel qu'il est avec Full-Scan comme filtrage d'index déjà minimise la quantité de données consultées.

c. Multi-threading et pipelining – l'exécution est découpée de petits morceaux de données (c'est-à-dire des TupleSets) qui sont tous traités dans parallèle sur de nombreux threads. Cette approche optimise utilisation des ressources du système et permet également de revenir des résultats initiaux avant que la requête entière ne soit terminée.

d. Optimisation GROUP BY – en utilisant les informations du index, Jethro peut prédire la taille du GROUPE résultant BY et choisissez l'agrégation la plus optimale en termes de mémoire algorithme pour cela.

e. Parallélisme des sous-requêtes – Jethro traite chaque sous-requête de manière indépendante et les traiter (lorsque la logique le permet) en parallèle.

- Optimisations spécifiques à l'outil BI

un. Chaque outil BI a un certain modèle pour les requêtes SQL qu'il génère. Bien qu'elles soient pour la plupart optimisées, ces requêtes sont souvent complexes et difficiles à optimiser, même par des moteurs SQL matures.

b. Jethro a incorporé de nombreuses règles d'optimisation qui identifie des modèles de requêtes spécifiques et les transforme en un forme plus optimale. Par exemple, Jethro « poussera » les filtres de requête externe vers une requête interne ou pourra « extraire » des filtres à partir d'une instruction CASE et placez-les dans une clause WHERE.

Résumé

Les plateformes Big Data sont utilisées avec succès pour stocker une quantité croissante de données.
données d'entreprise précieuses. Une façon importante d'utiliser ces données est de
permettre aux utilisateurs BI d'accéder à ces données en libre-service. Une clé du
Le succès d'une telle BI sur le Big Data est de la réaliser à une vitesse interactive – comme
nous le savons, la BI lente n'est pas une BI. Il existe plusieurs approches et outils
utilisé pour améliorer les performances BI sur le Big Data. Ils comprennent des cubes,
Index et mise en cache des requêtes. La bonne solution doit combiner les 3
techniques car ni une (ni deux) à elles seules ne pourront
couvrent la gamme complète des requêtes BI granulaires et agrégées. Le
La solution gagnante doit également automatiser le processus d'application des
stratégies d'accélération telles que le recours à l'informatique pour définir manuellement les cubes ou
la construction d'index nécessitera d'énormes ressources et ralentira le
nature agile de la BI en libre-service. La solution de Jethro est unique par sa capacité
pour fournir une BI Interactive entièrement automatisée sur le Big Data.

Merci d'avoir lu!

Discutons et discutons de la façon dont vous pouvez accélérer votre BI à la vitesse de la pensée.

+1 (844) 384-3844

info@jethro.io