الجزائرية الديمقراطية الشعبية الجمهورية

République Algérienne Démocratique et Populaire

وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

المدرسة العليا للإعلام الآلي - 08 ماي 1945 – بسيدي بلعباس

Ecole Supérieure en Informatique

-08 Mai 1945- Sidi Bel Abbes

Mémoire de Fin d'étude

Pour l'obtention du diplôme d'**ingénieur d'état**

Filière : **Informatique**

Spécialité : **Ingénierie des Systèmes Informatiques (ISI)**

Thème

Conception et réalisation d'un outil d'aide à sélection de vues matérialisées dans un environnement Big Data : Cas de HIVE

Présenté par **:**

- Mr Ahmed DERKAOUI
- Mlle Asmaa BOUCHIKHI

Soutenu le : **30/09/2020** Devant le jury composé de :

-	M. AMAR BENSABER Djamel	MCA	Président
-	M. KECHAR Mohamed	MCB	Encadreur
-	M. BENCHERIF Khayra	MCB	Examinatrice

-

Année Universitaire : 2019 / 2020

Remerciements

”au nom de dieu clément et miséricordieux”

Nous tenons à exprimer notre gratitude et nos sincères remerciements à notre encadreur **Dr m.Kechar**, pour nous avoir guidé et orienté pendant toute la durée de la réalisation de notre projet, ainsi que pour ces précieux conseilles et ses encouragement.

Nous tenons aussi à remercier nos enseignants durant toutes nos années de formation, ainsi que staff administratif de notre école ESI-SBA.

Nous souhaitons remercier aussi nos familles respectives pour tout leurs sacrifices et leur soutien et encouragements durant toute notre vie.

Aux chers membres du Jury, nous vous remercions pour avoir accepter de d'évaluer notre travail et nous avoir accordé l'honneur de le juger.

Cordialement,

a.Derkaoui
a.Bouchikhi

Abstract

With the constant growth of data volume and its variety, big data technologies are emerging as the new norm for the data warehousing.

Due to the nature of big data technlogies being open source designed to run in a distributed system using commodity hardware, the cost of join and aggreagate operations are time demanding compared to traditional data warehousing solutions.

In regards of the above, we deploy a view suggestion mechanism to speed up an analytical workload and minimize the execution time of initial analytical queries.

keywords : View Selection, Dimensional Data, Data Warehouse, Big Data, Hive

Resumé

Avec la croissance constante du volume de données et sa variété, les technologies du Big Data émergent comme la nouvelle norme pour l'entreposage de données.

En raison de la nature des technologies de Big Data étant open source conçues pour fonctionner dans un système distribué utilisant du matériel de base, le coût des opérations de jointure et d'agrégation demande du temps par rapport aux solutions traditionnelles d'entreposage de données.

Au regard de ce qui précède, nous déployons un mécanisme de suggestion de vue pour accélérer une charge de travail analytique et minimiser le temps d'exécution des requêtes analytiques initiales.

Mots-clés: Sélection de vues, Données dimensionnelles, Entrepôt de données, Big Data, Hive

Contents

1 Introduction générale **1**
- 1.1 Introduction: . . . 1
- 1.2 Problématique: . . . 2
- 1.3 Objectifs: . . . 2
- 1.4 Organisation du mémoire: . . . 2

2 Background **3**
- 2.1 Data warehousing: . . . 3
  - 2.1.1 Modélisation d'un entrepôt de données . . . 4
  - 2.1.2 Approche de construction d'un entrepôt de données . . . 5
  - 2.1.3 Contraintes du data warehouse . . . 6
- 2.2 Big Data et analytique . . . 8
  - 2.2.1 Definition . . . 8
  - 2.2.2 Data Lake . . . 9
- 2.3 Hadoop . . . 11
  - 2.3.1 Componsant De Hadoop . . . 11
  - 2.3.2 Écosystème Hadoop . . . 14
- 2.4 Hive . . . 16
  - 2.4.1 Définition . . . 16
  - 2.4.2 Architecture du système . . . 17
- 2.5 Techniques d'optimisation: . . . 20
  - 2.5.1 Optimisation par indexation: . . . 20
  - 2.5.2 Optimisation par partitionnement: . . . 22
  - 2.5.3 Optimisation par matérialisation des vues : . . . 23

3 Approche Proposé **26**
- 3.1 Problème de sélection des vues a matérialiser : . . . 26
  - 3.1.1 Description du problème . . . 26
  - 3.1.2 Contexte . . . 26
- 3.2 Conception de la solution: . . . 27
  - 3.2.1 Diagramme De classes . . . 27
  - 3.2.2 Diagramme de cas d'utilisation . . . 29
  - 3.2.3 Principe de fonctionnement . . . 31
- 3.3 Environnement de travail: . . . 35
  - 3.3.1 Environnement matériel . . . 35
  - 3.3.2 Architecture et déploiement . . . 35

3.3.3 Environnement de développent et outils 37

4 Résultat et évaluation **39**
4.1 Matérialisation sans prédicat 42
4.2 Matérialisation avec prédicat 44
4.3 Étude comparative 47

5 Conclusion **48**
5.1 Perspective futures: 48

Bibliography **49**

List of Figures

2.1 Processus de création d'un entrepôt . . . 3
2.2 Exemple de table de faits et de dimensions . . . 5
2.3 Approche top-down . . . 5
2.4 Approche bottom-up . . . 6
2.5 Représentation d'une minute sur le net en 2020 . . . 7
2.6 HDFS DataNode . . . 12
2.7 HDFS NameNode . . . 12
2.8 Hadoop Yarn . . . 13
2.9 Hadoop Map Reduce . . . 14
2.10 Apache Hadoop Écosystème . . . 14
2.11 Architecture de apache Hive . . . 17
2.12 Hive Server. . . . 19
2.13 Partitionnement par plages . . . 23
2.14 Partitionnement par hachage . . . 23
2.15 Diagramme des vues à matérialiser . . . 24

3.1 Schéma TPCH. . . . 27
3.2 SSB-Schéma. . . . 28
3.3 Diagramme de cas d'utilisation. . . . 29
3.4 Diagramme de séquences. . . . 30
3.5 Principe de fonctionnement. . . . 31
3.6 Exemple de vue suggérée sans prédicat . . . 34
3.7 Exemple de vue suggérée avec prédicat . . . 34
3.8 Diagramme de déploiement . . . 36
3.9 Docker . . . 37
3.10 Prestodb . . . 37
3.11 Python . . . 37
3.12 Pycharm . . . 38
3.13 Git . . . 38
3.14 GitHub . . . 38

4.1 Requête Q1 . . . 40
4.2 Requête Q2 . . . 40
4.3 Requête Q3 . . . 41
4.4 Matérialisation sans prédicats . . . 42
4.5 Graphe de comparaison entre différentes exécutions de Q1 . . . 42
4.6 Graphe de comparaison entre différentes exécutions de Q2 . . . 43

4.7 Graphe de comparaison entre différentes exécutions de Q3 43
4.8 Matérialisation avec prédicats 44
4.9 Graphe de comparaison entre différentes exécutions de Q1 cas de vue avec prédicat 45
4.10 Graphe de comparaison entre différentes exécutions de Q2 cas de vue avec prédicat 45
4.11 Graphe de comparaison entre différentes exécutions de Q3 cas de vue avec prédicat 46

List of Tables

2.1 Tableau comparatif entre Data Lake et Data Warehouse 10

3.1 Explication du cas d'utilisation 29

4.1 Scaling factors utilisés. 39
4.2 Temps d'exécution en rapport du scaling factor (SF) 41
4.3 Tableau de gains en temps d'exécution 43
4.4 Tableau de gains en temps d'exécution cas de vue matérialisée 46
4.5 Tableau comparatif entre temps de génération des vue et le rapport espace . . . 47
4.6 Tableau récapitulatif des différences entre les approches 47

Chapter 1

Introduction générale

1.1 Introduction:

L'analyse des données est omniprésente dans toutes les disciplines et dans tous les domaines, elle permet d'extraire des connaissances et des information "utiles" en appliquant différentes méthodes d'analyse.Dans le cadre des entreprises, l'analyse des données permet d'extraire des informations clés qui constituent un avantage concurrentiel, la majorité des grandes entreprises se basent sur des systèmes de Business Intelligence pour le support de la prise de décision, d'où l'utilisation des techniques de Data Warehouse et OLAP (on-line Analytical Processing) pour la gestion et l'analyse des données.

De nos jours, l'évolution des données en matière de volume et variété est en croissance rapide, on prévoit 175 ZettaByte de données d'ici 2025 dont 30% doivent être traités en temps réel d'où l'émergence des techniques Big Data.

En raison de cette croissance ainsi que la nature open source de la majorité des techniques Big Data, les entreprises et les PME migrent leur système analytique aux technologies Big data pour optimiser à la fois les coûts et de s'adapter à la croissance des volumes de données. Dans le cadre de notre projet de fin d'études on s'intéresse au cas de Hive .

1.2 Problématique:

Hive se base sur l'écosystème Hadoop, ce qui permet d'utiliser un matériel de commodité pour exécuter des requêtes analytiques.

Dû à la nature distribuée imposée par le système de fichiers d'Hadoop HDFS, il existe plusieurs challenges en matière d'optimisation de temps d'exécution des charges analytiques dans un environnement distribué.

On cite différentes approches d'optimisations, telles que l'optimisation par partitionnement, Indexation et par sélection de vue matérialisée.Dans le cadre de notre travail on s'intéresse au problème de suggestion de vue matérialisée afin d'optimiser une charge analytique.

1.3 Objectifs:

l'objectif de notre projet est de suggérer un mécanisme pour accélérer la charge analytique dans le cas de Hive à travers la suggestion des vues à matérialiser.

A travers une analyse d'une charge analytique notre projet transformes les requêtes commune en vue candidate pour matérialisation permettant ainsi une accélération de la charge analytique sur un entrepôt de donne HIVE.

1.4 Organisation du mémoire:

Dans le chapitre 2 nous allons introduire des notions de bases sur les différents aspects de l'analyse données dans le contexte du Big Data puis dans le chapitre 3 nous allons détailler le principe utilisé par notre approche et comparer les différents résultats 4 et enfin conclure notre travail dans le chapitre 5 .

Chapter 2

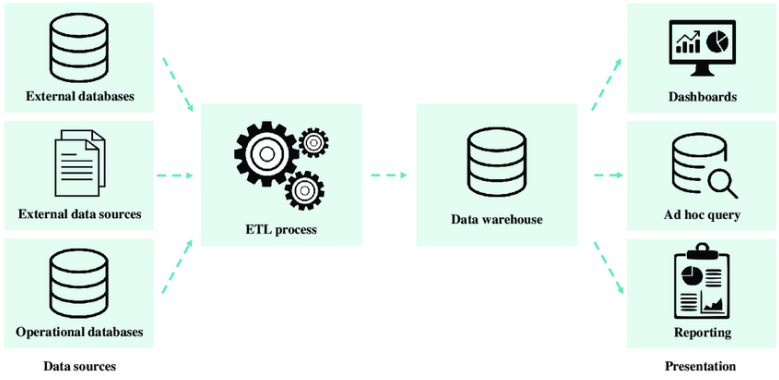
Background

Dans ce chapitre nous allons présenter les différentes notions liées aux entrepôts de données et les techniques Big Data, ainsi que les différentes approches d'optimisation existantes.

2.1 Data warehousing:

Un entrepôt de données est un système informatique dédié aux besoins analytiques, il est séparé du système transactionnel.Un entrepôt de données centralise et consolide de grandes quantités de données provenant de plusieurs sources. La capacité analytique d'un entrepôt de données permet aux décideurs d'avoir un accès à des informations stratégiques pour la prise de décision. Un entrepôt de données est souvent vu comme repaire historique d'une organisation et souvent considéré comme sa source unique de vérité.

Figure 2.1: Processus de création d'un entrepôt



Le terme Entrepôt de données a été formalisé par William H. (Bill) Inmon qui est considéré un des fondateurs. "Un entrepôt de données est une collection d'orientés sujets, intégrées,

non volatiles et variantes dans le temps qui sert à la prise de décision dans le management. L'entrepôt de données contient des données granulaires de l'entreprise" [1] .

2.1.1 Modélisation d'un entrepôt de données

Par définition les données d'un entrepôt de données sont :

- **Orientées sujet** : Les données sont modélisées au tour d'un besoin analytique spécifique, par exemple, L'analyse des ventes.
- **Intégrées** : Les données sont regroupées de différentes sources et transformées pour l'intégration avec le modèle du sujet à traiter **Non volatil** : les données ne changent pas suite aux résultats des traitements.
- **Variante dans le temps** : chaque nouvelle donnée est insérée. Un référentiel de temps doit être mis en place afin de pouvoir identifier chaque donnée dans le temps.

La modélisation des données dans un entrepôt est une modélisation multidimensionnelle qui repose sur le choix des :

- **Tables de faits** : les tables de faits contiennent les données que l'on souhaite voir apparaître dans les rapports d'analyse, sous forme de métriques. Les données des tables de faits sont agrégées à partir des tables de dimensions qui leur sont associées.
- **Tables de dimensions** : les tables de dimensions sont utilisées pour décrire les données que l'on souhaite stocker dans le Data Warehouse.
- **Les niveaux de granularité** : Déterminent les niveaux de détails des tables de faits et des tables de dimensions [3].

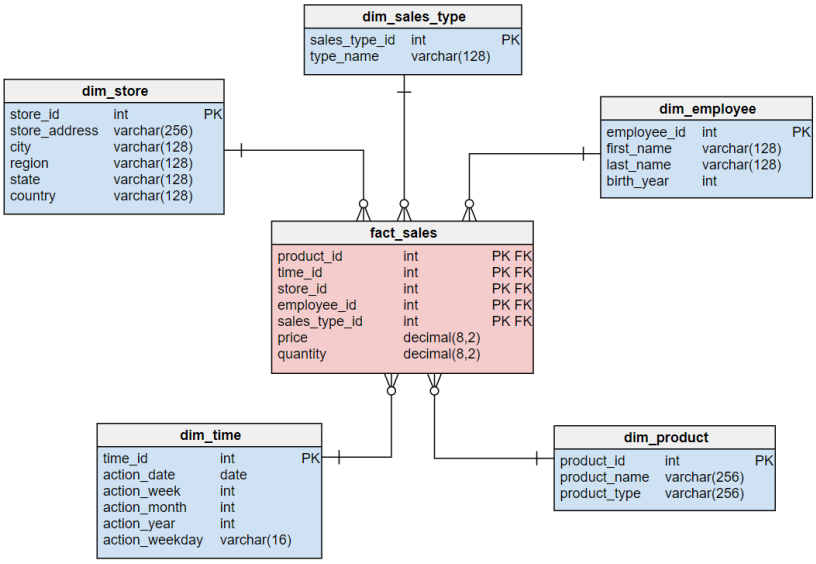


Figure 2.2: Exemple de table de faits et de dimensions

2.1.2 Approche de construction d'un entrepôt de données

Pour construire l'entrepôt de données, le processus ETL (Extract, Transforme and Load) se charge de collectionner les données des différentes sources et les transformer pour intégrer dans le modèle de l'entrepôt.

TOP-DOWN proposée par W.H.Inmon qui définit la création de l'entrepôt de données est de générer ensuite les data Mart (Magasin de données).

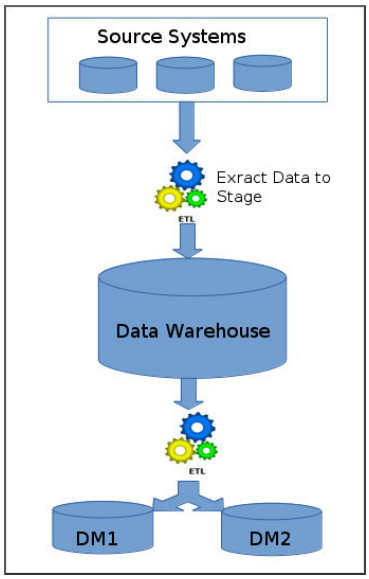


Figure 2.3: Approche top-down

BOTTOM-UP proposée par R. Kimball, qui définit la création de l'entrepôt de données à partir de la collection des différends data Mart (Magasin de données).

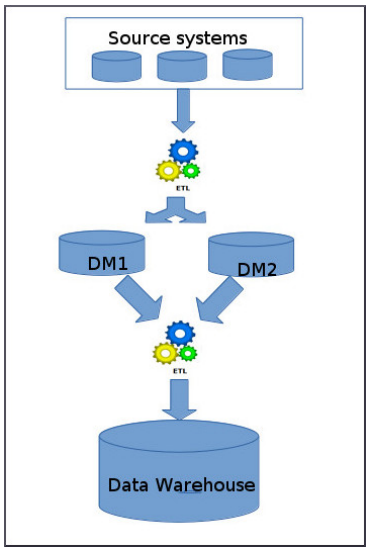


Figure 2.4: Approche bottom-up

2.1.3 Contraintes du data warehouse

Évolution des données: Les solutions Data Warehouse reposent sur des données de nature structure, avec l'évolution de la quantité de données à un taux de croissance annuel moyen de 61% on estime le volume de données à atteindre 175 ZettaByte en 2025 [23] et que 30% de ces données doivent être analysée en temps réel [22] .

Hétérogénéité des données IBM estime à estimé en 2016 que 80% des données quantifiées sont de nature non structurée [26], avec l'émergence de L'IoT qui constituera 90 ZettaByte en 2025, les majorités des données présentes seront de nature non-structure .

Coûts: Une contrainte majeure est la contrainte du coût d'implementation pour un entrepôt de données pour les petites et moyennes entreprises [16], le coût d'un entrepôt de données est proportionnellement lié à sa taille.

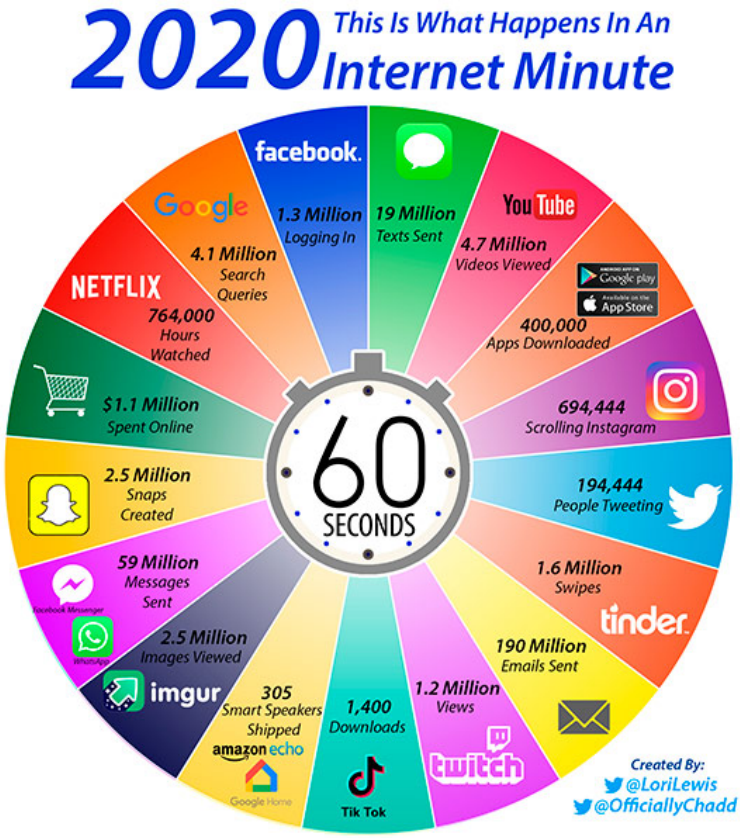


Figure 2.5: Représentation d'une minute sur le net en 2020

2.2 Big Data et analytique

Avec cette évolution constante de la taille des données, leur nature souvent hétérogène et la nécessité à analyser les données en temps réel, le Big Data est une réponse à ces problématiques. Dans cette section, nous allons expliquer le concept du Big Data et les techniques Big Data souvent utilisé dans l'analytique.

2.2.1 Definition

La plupart des data scientists et des experts définissent le Big Data par les trois principales caractéristiques suivantes (appelées les 3V) (Furht et Villanustre, 2016) [15] :

- **Volume** : De grands volumes de données numériques sont générés en continu à partir de millions d'appareils et d'applications (TIC, smartphones, codes produits, réseaux sociaux, capteurs, journaux, etc.). Selon McAfee et al. (2012)[4], on estime qu'environ 2,5 exaoctets ont été générés chaque jour en 2012. Ce montant double tous les 40 mois environ. En 2013, le total des données numériques créées, répliquées et consommées a été estimé par l'International Data Corporation (une société qui publie des rapports de recherche) à 4,4 zettaoctets (ZB). Il double tous les 2 ans. En 2015, les données numériques sont passées à 8 ZB . Selon le rapport d'IDC, le volume de données atteindra 40 octets Zeta d'ici 2020 et augmentera de 400 fois maintenant (Kune et al., 2016) [12].

- **Vitesse**: les données sont générées de manière rapide et doivent être traitées rapidement pour extraire des informations utiles et des informations pertinentes. Par exemple, Wallmart (une chaîne internationale de vente au détail à prix réduits) génère plus de 2,5 Po de données par heure à partir des transactions de ses clients. YouTube est un autre bon exemple qui illustre la vitesse rapide du Big Data.

- **Variété**: les Big Data sont générées à partir de diverses sources distribuées et dans plusieurs formats (par exemple, des vidéos, des documents, des commentaires, des journaux). Les grands ensembles de données sont constitués de données structurées et non structurées, publiques ou privées, locales ou distantes, partagées ou confidentielles, complètes ou incomplètes, etc.

  Emani et coll. (2015) [15] et Gandomi et Haider (2015) [9] indiquent que plus de V et d'autres caractéristiques ont été ajoutées par certains acteurs pour mieux définir le Big Data:**Vision** (un objectif),**Vérification** (données traitées conformes à certaines

spécifications), **Validation** (l'objectif est rempli), de la **valeur** (des informations pertinentes peuvent être extraites pour de nombreux secteurs), de la complexité (il est difficile d'organiser et d'analyser le Big data en raison de l'évolution des relations de données) et de l'immuabilité (les Big data collectées et stockées peuvent être permanentes si elles sont bien gérées).

2.2.2 Data Lake

Dans l'ère du Big Data, le terme data Lake a vu le jour, souvent confondu au Data Warehouse le Data Lake se différencie de ce dernier sur différents aspects dont on peut résumer sur ces points;

- **Données**: Le Data Warehouse utilise des données de nature structurées, par contre le Data Lake permet d'intégrer les donnes de natures semi-structurés ou non structuré grace aux technologies Big Data .
- **Traitement**: Le Data Warehouse repose sur le concept du schéma en écriture ce qui limite l'espace d'interactions avec le système aux requêtes analytiques dont le système a été conçu via le processus ETL, par contre le DATA Lake permet l'interaction avec diffère types de données sont possible, seul lorsque les données sont interrogées, ils seront transformée selon l'application de l'utilisateur via le processus ETL en se basant sur le concept de schéma en lecture.
- **Stockage et coûts** : les Data Lake est souvent conçu en se basant sur des techniques big data open source qui est conçu pour des serveurs et un matériel de commodité contrairement aux coûts élevés des licences et du stockage dans un environnement Data Warehouse classique.
- **Flexibilité**: Le data warehouse et un environnement hautement structuré avec le modèle schéma en écriture tout changement dans la conception de ce dernier engendre des coûts de maintenance. Le Data Lake se base sur le concept de schéma en lecture qui offre une flexibilité et une évolution plus souple .
- **Sécurité**: Les environnements Data Warehouse ont atteint un niveau de maturité en matière de mécanismes de sécurité, les data Lake est en constante amélioration et des recherches dans le domaine de la sécurité de ces derniers est en production.

- **Utilisateurs**: Plusieurs experts sont habitués aux concepts des Data Warehouse par contre les data scientists sont les principaux utilisateurs des datèrent Lake due à la nature et l'hétérogénéité de ces données [17].

Critère	Data Lake	Data Warehouse
Données	Structurées, semi-structurées et non structurées	Structuré, données traitées
Traitement coûts faible	Schema en Lecture coûteux	Schema en Écriture Stockage et coûts
Flexibilité	Agile, configuration flexible	Moins agile et configuration fixée
Sécurité	En développement	Mature
Utilisateurs	Décideurs et Professionnels du Business	Data Scientists et Analystes connaisseur du domaine

Table 2.1: Tableau comparatif entre Data Lake et Data Warehouse

Le Data Lake peut être vu comme la nouvelle approche du Data Warehouse en utilisant les techniques Big Data et le traitement distribué et le stockage distribué offert par le paradigme MapReduce et le system de stockage HDFS.Dans la prochaine section nous allons détailler sur Hadoop et son écosystème tel que Hive, qui constituent la base des data Lake récents .

2.3 Hadoop

Hadoop est un framework open source qui permet de stocker et de traiter des big data dans un environnement distribué sur des clusters d'ordinateurs à l'aide de modèles de programmation simples. Il est conçu pour passer d'un serveur unique à des milliers de machines, chacune offrant un calcul et un stockage locaux [7].

2.3.1 Componsant De Hadoop

les principales composantes de Hadoop sont:

- Hadoop Common : contient les librairies et les utilitaires nécessaires pour les autres modules d'Hadoop .
- Hadoop Distributed File System (HDFS): un système de fichiers distribué qui stocke les données sur les machines de base, fournissant une bande passante globale très élevée à travers le cluster.
- Hadoop Yarn: une plate-forme de gestion des ressources chargée de gérer les ressources de calcul dans les clusters et de les utiliser pour la planification des applications des utilisateurs [6].
- Hadoop MapReduce: L'implémentation de du paradigme MapReduce dans Hadoop, pour le traitement des travaux en mode distribué.

HDFS :

HDFS stocke des données sur plusieurs machines. Les données sont automatiquement répliquées sur diverses machines pour éviter la perte de données. Dans HDFS, les données sont divisées en plusieurs blocs (afin d'être compatibles avec le stockage du matériel de base.); HDFS fournit un accès à haut débit aux blocs de données. Lorsque des données non structurées sont téléchargées sur HDFS, elles sont converties en blocs de données de taille fixe par défaut de 128 Mo [25] .

HDFS fournit une interface limitée pour gérer le système de fichiers. Il garantit que l'on peut effectuer une augmentation ou une réduction des ressources dans le cluster Hadoop.

HDFS crée plusieurs répliques de chaque bloc de données et les stocke dans plusieurs systèmes à travers le cluster pour permettre un accès fiable et rapide aux données.

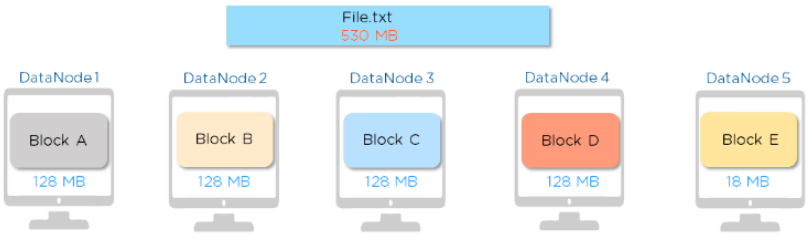


Figure 2.6: HDFS DataNode

HDFS a deux composants qui s'exécutent sur différentes machines. Elles sont:

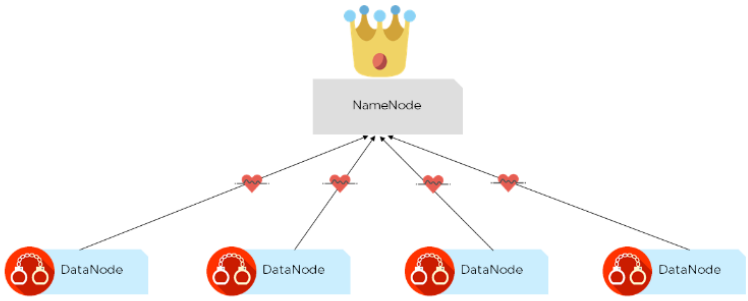


Figure 2.7: HDFS NameNode

- **NameNode** - NameNode est le maître de la couche de stockage HDFS. Il stocke toutes les métadonnées. Si la machine sur laquelle traite le NameNode tombe en panne, le cluster sera indisponible.

- **DataNode** - Les DataNodes sont appelés nœuds esclaves. Ils stockent les données réelles et effectuent les opérations de lecture / écriture. Fondamentalement, le NameNode gère tous les DataNodes. Les signaux appelés pulsations sont envoyés par les DataNodes au NameNode pour fournir des mises à jour de statut [25].

Yarn :

une plate-forme de gestion des ressources chargée de gérer les ressources de calcul dans les clusters et de les utiliser pour la planification des applications des utilisateurs [6] .
Apache YARN comprend:

- **Gestionnaire de ressources -** Il agit comme le démon maître. Il examine l'affectation du processeur, de la mémoire, etc.

- **Node Manager -** Il s'agit du démon esclave. Il signale l'utilisation au gestionnaire de ressources.
- **Maître d'application -** Cela fonctionne à la fois avec le gestionnaire de ressources et le gestionnaire de nœuds lors de la négociation des ressources [25].

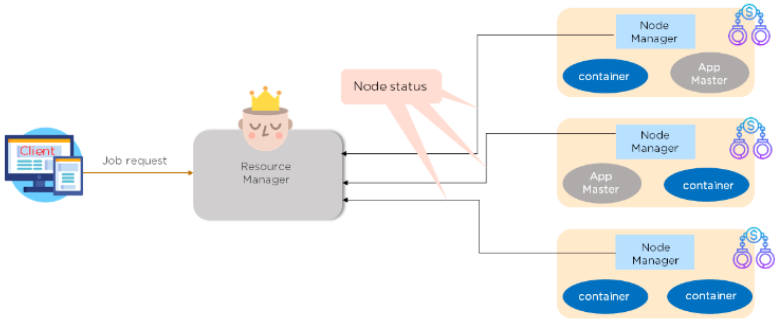


Figure 2.8: Hadoop Yarn

certaines des fonctionnalités de YARN:

- YARN est responsable du traitement des demandes d'emploi et de l'allocation des ressources.
- Différentes versions de MapReduce peuvent s'exécuter sur YARN, ce qui permet de gérer une mise à niveau de MapReduce.
- Selon nos besoins, on peut ajouter des nœuds à volonté [25].

MapReduce : Certaines des principales fonctionnalités du composant Hadoop MapReduce sont les suivantes:

- Il effectue un traitement de données distribué à l'aide du paradigme de programmation MapReduce.
- Il vous permet de posséder une phase de mappage définie par l'utilisateur, qui est un traitement parallèle et sans partage des entrées.
- Il agrège la sortie de la phase de mappage, qui est une phase de réduction définie par l'utilisateur après un processus de mappage [24].

À partir du diagramme suivant, c'est comment chaque étape est exécutée dans MapReduce [25] :

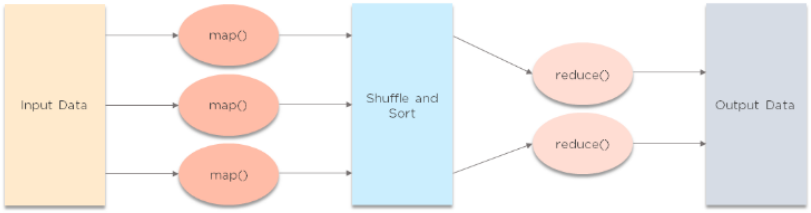


Figure 2.9: Hadoop Map Reduce

2.3.2 Écosystème Hadoop

En addition à ces composants, il existe de nombreux projets liés à Hadoop, ce qui construit un écosystème. une partie de cet écosystème est présente dans le schéma suivant:

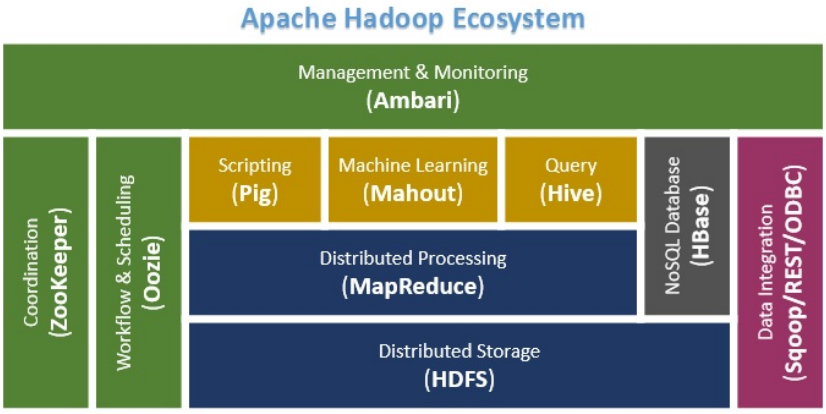


Figure 2.10: Apache Hadoop Écosystème

Parmi les projets liées a Hadoop :

- **Ambari ™:** un outil Web pour l'approvisionnement, la gestion et la surveillance des clusters Apache Hadoop qui inclut la prise en charge de Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig et Sqoop. Ambari fournit également un tableau de bord pour visualiser l'état de santé du cluster, comme les cartes thermiques et la possibilité de visualiser visuellement les applications MapReduce, Pig et Hive, ainsi que des fonctionnalités permettant de diagnostiquer leurs caractéristiques de performance de manière conviviale.

- **Avro ™:** un système de sérialisation de données.

- **Cassandra ™:** une base de données multi-maître évolutive sans point de défaillance unique.

- **Chukwa ™:** Un système de collecte de données pour la gestion de grands systèmes distribués.
- **HBase ™:** Une base de données distribuée évolutive qui prend en charge le stockage de données structuré pour les grandes tables.
- **Hive ™:** Une infrastructure d'entrepôt de données qui fournit une synthèse des données et des requêtes ad hoc.
- **Mahout ™:** une bibliothèque évolutive d'apprentissage automatique et d'exploration de données.
- **Pig ™:** un langage de flux de données de haut niveau et un cadre d'exécution pour le calcul parallèle.
- **Spark ™:** un moteur de calcul rapide et général pour les données Hadoop. Spark fournit un modèle de programmation simple et expressif qui prend en charge un large éventail d'applications, notamment ETL, l'apprentissage automatique, le traitement de flux et le calcul de graphes. Sous-marin: une plate-forme d'IA unifiée qui permet aux ingénieurs et aux scientifiques des données d'exécuter une charge de travail d'apprentissage automatique et d'apprentissage en profondeur dans un cluster distribué.
- **Tez ™:** Un cadre de programmation de flux de données généralisé, construit sur Hadoop YARN, qui fournit un moteur puissant et flexible pour exécuter un DAG arbitraire de tâches pour traiter les données pour les cas d'utilisation par lots et interactifs. Tez est adopté par Hive ™, Pig ™ et d'autres frameworks de l'écosystème Hadoop, ainsi que par d'autres logiciels commerciaux (par exemple les outils ETL), pour remplacer Hadoop ™ MapReduce comme moteur d'exécution sous-jacent.
- **ZooKeeper ™:** un service de coordination hautes performances pour les applications distribuées [21].

Dans la section suivante nous allons détailler sur Hive, car il fait l'objet de notre travail, nous allons introduire ces concepts de bases et son fonctionnement.

2.4 Hive

2.4.1 Définition

Hive est défini comme un système d'entrepôt de données pour Hadoop qui facilite les requêtes ad hoc et l'analyse de grands ensembles de données stockées dans Hadoop. Hive est un système de gestion et d'interrogation de données non structurées dans un format structuré.Il utilise le concept de MapReduce pour l'exécution de ses scripts et le système de fichiers distribués Hadoop ou HDFS pour le stockage et la récupération des données.Les performances sont meilleures dans Hive puisque le moteur Hive utilise le meilleur script intégré pour réduire le temps d'exécution, permettant ainsi une sortie élevée en moins de temps [10].

Construit sur Apache Hadoop ™, Hive fournit les fonctionnalités suivantes:

- Outils pour permettre un accès facile aux données via SQL, permettant ainsi des tâches d'entreposage de données telles que l'extraction / transformation / chargement (ETL), la création de rapports et l'analyse des données.
- Un mécanisme pour imposer une structure à une variété de formats de données Accès aux fichiers stockés directement dans Apache HDFS ™ ou dans d'autres systèmes de stockage de données tels que Apache HBase ™.
- Exécution de requêtes via Apache Tez ™, Apache Spark ™ ou MapReduce Langage procédural avec HPL-SQL.
- Récupération de requête en sous-seconde via Hive LLAP, Apache YARN et Apache Slider.

2.4.2 Architecture du système

Dans cette partie, nous présentons brièvement l'architecture de Hive. La figure illustre les principaux composants du système [19].

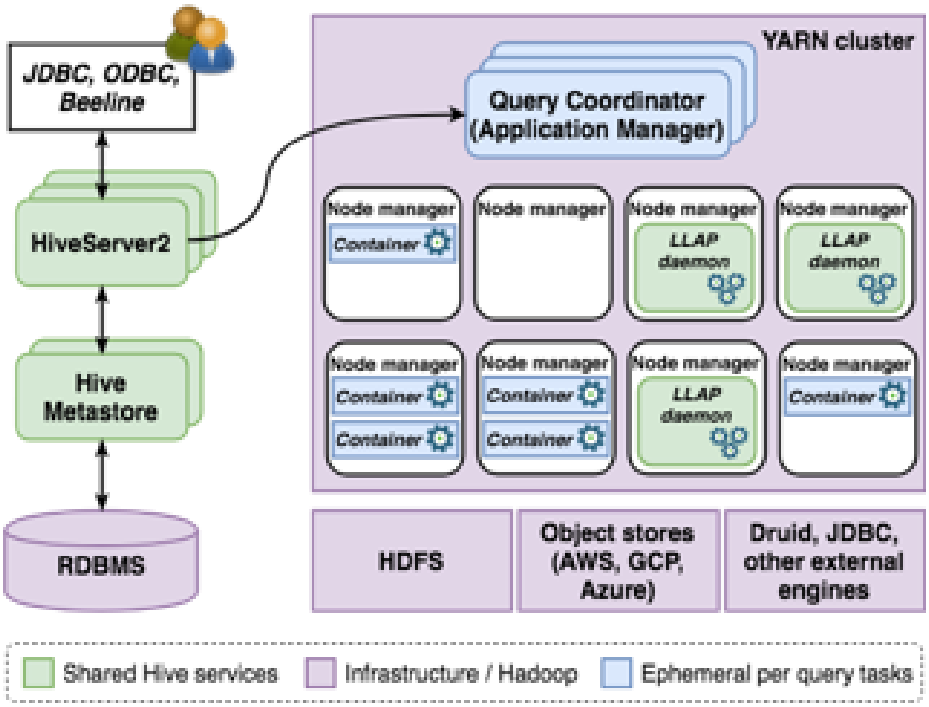


Figure 2.11: Architecture de apache Hive

Stockage de données

Les données dans Hive peuvent être stockées à l'aide de l'un des formats de fichiers pris en charge dans n'importe quel système de fichiers compatible avec Hadoop. À ce jour, les formats de fichiers les plus courants sont ORC et Parquet. À leur tour, les systèmes de fichiers compatibles incluent HDFS, qui est l'implémentation de système de fichiers distribué la plus couramment utilisée, et tous les principaux magasins d'objets cloud commerciaux tels que AWS S3 et Azure Blob Storage. En outre, Hive peut également lire et écrire des données sur d'autres systèmes de traitement autonomes, tels que Druid ou HBase.

Catalogue de données

Hive stocke toutes les informations sur ses sources de données à l'aide du Hive Metastore (ou HMS, en bref). En un mot, HMS est un catalogue de toutes les données interrogeables par Hive. Il utilise un SGBDR pour conserver les informations telles que My Sql ,POSTGRESQL ou Derby (Par défaut) et s'appuie sur Data Nucleus, une implémentation de mappage objet relationnel Java, pour simplifier la prise en charge de plusieurs SGBDR au niveau du back-end. Pour les appels qui nécessitent une faible latence, HMS peut contourner DataNucleus et interroger directement le RDMBS. L'API HMS prend en charge plusieurs langages de programmation et le service est implémenté à l'aide de Thrift, un framework logiciel qui fournit un langage de définition d'interface, un moteur de génération de codes et une implémentation de protocole de communication binaire.

Traitement des données

Hive est devenu l'un des moteurs SQL les plus populaires sur Hadoop et s'est progressivement éloigné de MapReduce pour prendre en charge des temps d'exécution de traitement plus flexibles compatibles avec YARN. Bien que MapReduce soit toujours pris en charge, actuellement le runtime le plus populaire pour Hive est Tez. Tez offre plus de flexibilité que MapReduce en modélisant le traitement des données sous forme de DAG avec des sommets représentant la logique d'application et des arêtes représentant le transfert de données .

Serveur de requêtes

HiveServer2 abrégé HS2 permet aux utilisateurs d'exécuter des requêtes SQL dans Hive. HS2 prend en charge les connexions JDBC et ODBC locales et distantes; La distribution Hive comprend un client léger JDBC appelé Beeline.

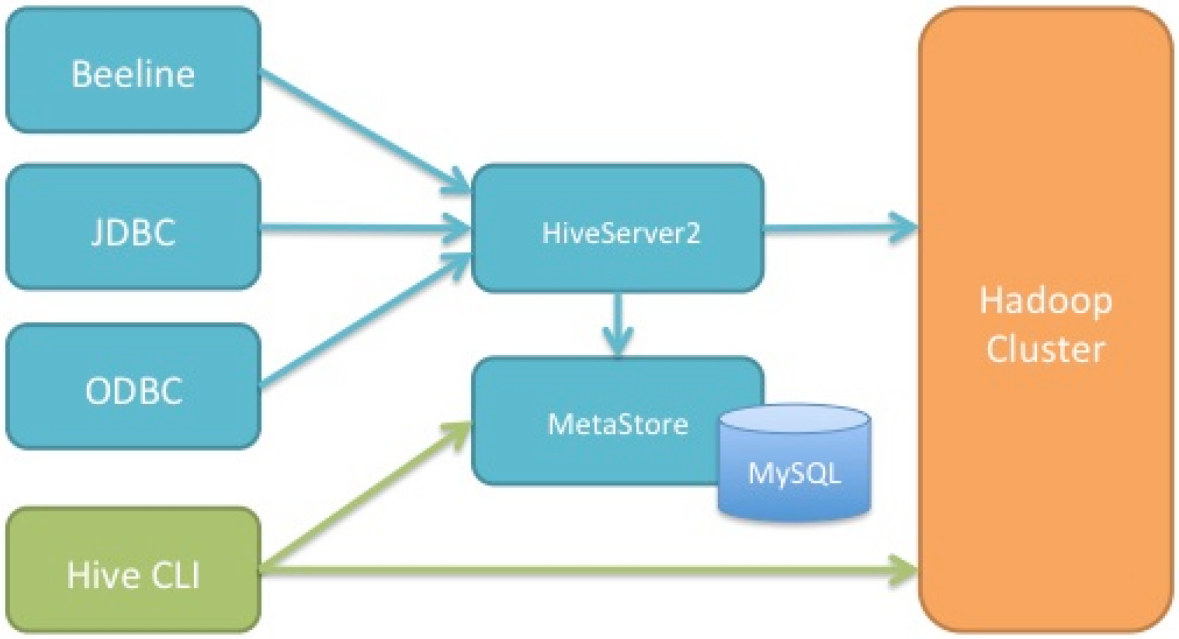


Figure 2.12: Hive Server.

2.5 Techniques d'optimisation:

Dans les bases de données analytiques et les data warehouse, les requêtes analytique sont souvent coûteuse dans leur temps d'exécution.

Dans le contexte du big data, le volume et la charge analytique est importante, d'où la nécessiter de l'optimisation. Il existe plusieurs approches, nous allons les définir et les détailler dans cette section.

2.5.1 Optimisation par indexation:

Définition d'un index : L'indexation est une méthode utilisée dans plusieurs disciplines en informatique, elle a pour objectif de minimiser le temps de recherche nécessaire pour accéder à une information.[11]

Une stratégie d'indexation est la conception d'une méthode d'accès à un élément recherché, ou simplement, un index. Il décrit également comment les données sont organisées dans un système de stockage pour faciliter la recherche d'informations.

L'idée de l'indexation Big Data est de fragmenter les ensembles de données selon des critères qui seront fréquemment utilisés dans la requête . Les fragments sont indexés avec chaque valeur contenant satisfaisant certains prédicats de requête. Cela vise à stocker les données de manière plus organisée, facilitant ainsi la récupération d'informations [8].

Types d'indexes : Il existe plusieurs types d'indexés dans un environnement big datèrent on peut citer comme exemples:

- **B-Tree :** Un B-tree fonctionne comme la recherche d'arbre binaire, mais d'une manière plus complexe. En effet, les nœuds de B-tree ont de nombreuses branches, contrairement à l'arbre binaire qui a deux branches par nœud.
  Les index B-tree satisfont les requêtes de plage et les requêtes de similarité.

- **R-Tree :** il s'agit d'une stratégie d'indexation utilisée pour les requêtes spatiales ou de plage. Il est principalement appliqué dans les systèmes géos spatiaux avec chaque entrée ayant des coordonnées X et Y avec des valeurs minimales et maximales .
  L'avantage d'utiliser un R-tree sur un B-tree est que, le R-tree satisfait les requêtes multidimensionnelles ou de plage, alors que le B-tree ne le fait pas. Étant donné une plage de requêtes, l'utilisation de l'arborescence R permet de trouver rapidement des réponses aux requêtes.

- **X-Tree :** L'arbre X est similaire à l'arbre R, bien que contrairement à l'arbre R qui satisfait les requêtes de plage de 2 à 3 dimensions, l'arbre X satisfait les requêtes de nombreuses dimensions. Cela implique que le X-tree est une version plus compliquée du R-tree.
  L'avantage de l'arbre X par rapport à l'arbre Rest où'il couvre plus de dimensions.

- **Hash Index:** Le hachage permet une comparaison d'égalité. L'indexation par hachage accélère la recherche d'informations en détectant les doublons dans un grand ensemble de données.
  Le hachage est utilisé dans l'indexation Big Data pour indexer et récupérer des éléments de données (dans un ensemble de données) qui sont similaires à l'élément recherché. Il utilise une clé hachée (qui est calculée par la fonction de hachage et généralement plus courte que la valeur d'origine) pour stocker et récupérer les index .

- **Stratégie d'indexation personnalisée :** (Custom Indexing Strategy) : L'indexation personnalisée prend en charge l'indexation de champs multiples basée sur des index arbitraires ou définis par l'utilisateur . Ils sont généralement basés sur des stratégies d'indexation telles que B-tree, R-tree, index inversé et stratégie d'indexation par hachage. Deux types de stratégies d'indexation personnalisées sont l'arbre de recherche généralisé (Generalized Search Tree)(GiST) et l'index inversé généralisé (Generalized Inverted Index) (GIN).

- **d'indexation du modèle de Markov caché (HMM):**est une méthode d'accès développée à partir du modèle de Markov. le HMM utilisent la reconnaissance de formes et la relation entre les données. Dans l'approche d'indexation HMM, les données ou caractéristiques dont les états dépendent la requête sont classées et stockées à l'avance. Les résultats de la requête sont généralement des prédictions des états futurs d'un élément, en fonction de l'état actuel.

- **Indexation sémantique latente :** en abrégé LSI, est une stratégie d'indexation (méthode de récupération / d'accès)((retrieval/access method) qui identifie les modèles(paterns) entre les termes d'un ensemble de données non structuré (en particulier, du texte). Il utilise une approche mathématique connue sous le nom de décomposition en valeurs singulières (SVD) pour l'identification du modèle ou de la relation.

les stratégies d'indexation peuvent être classées en approche d'intelligence artificielle (IA) (Indexation sémantique latente, d'indexation du modèle de Markov cache (HMM) et en ap-

proche d'intelligence non artificielle (NAI): (B-tree, R-tree, X-tree, hash index, Stratégie d'indexation personnalisée) [8] .

2.5.2 Optimisation par partitionnement:

Définition de partitionnement : Le partitionnement consiste à diviser les tables en tables plus petites et plus faciles à gérer, appelées partitions.
Un partitionnement s'effectue toujours en fonction d'une clé, soit un ou plusieurs attributs dont la valeur sert de critère à l'affectation d'un document à un fragment. La première décision à prendre est donc le choix de la clé.
Un partitionnement doit être dynamique: en fonction de l'évolution de la taille de fragments doit pouvoir évoluer. C'est important pour optimiser l'utilisation de l'espace disponible et obtenir les meilleures performances. C'est aussi, techniquement, la propriété la plus difficile à satisfaire [20].

Type de partitionnement : La plupart des systèmes NoSQL utilisent soit partitionnement basé sur le hachage, soit sur plage (ou un mélange des deux) .

- **Partitionnement basée sur plages :**Dans le partitionnement basé sur des plages, l'espace de clés est divisé en plages et chaque plage est attribuée à un serveur et potentiellement répliquée sur d'autres. Le principal avantage du partitionnement par plage est que deux clés consécutives sont susceptibles d'apparaître dans la même partition, ce qui est avantageux lorsque les requêtes de type scan de plage sont fréquentes. Les schémas de partitionnement basés sur des plages maintiennent généralement une carte qui stocke des informations sur les serveurs responsables de quelles plages de clés [5].

  Certains bases de données NoSQL comme Apache HBase , MongoDB utilisent le partitionnement par plage .

- **Partionnement basée sur le hachage :** Le partitionnement basé sur le hachage utilise simplement le hachage des données pour déterminer le serveur responsable du stockage de ces données.
  d'autres bases de données NoSQL comme Google BigTable, Amazon Dynamo , Cassandra utilisées par Facebook utilisent le partitionnement par hachage.

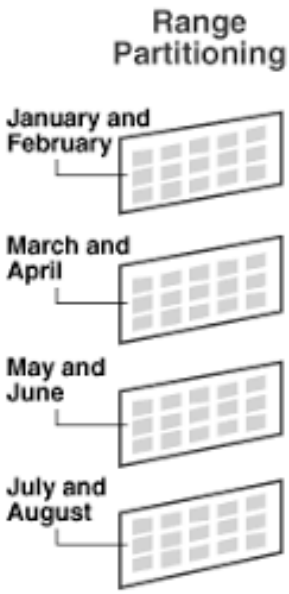


Figure 2.13: Partitionnement par plages

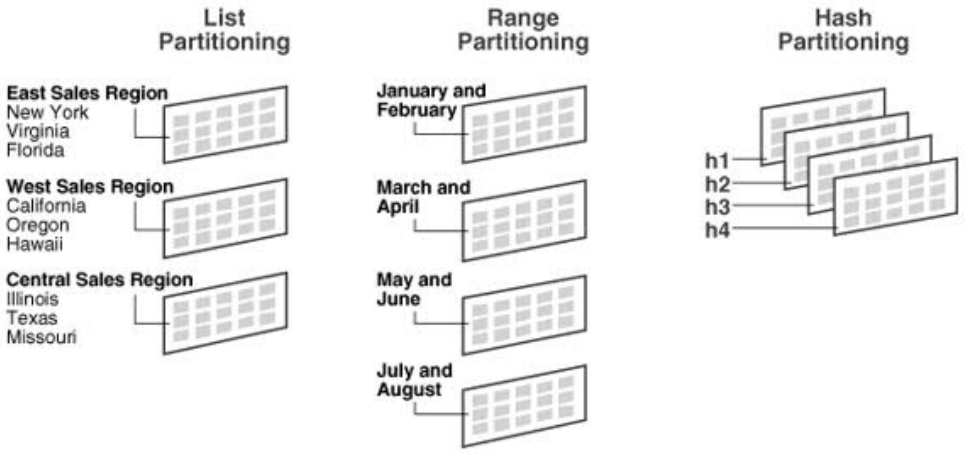


Figure 2.14: Partitionnement par hachage

2.5.3 Optimisation par matérialisation des vues :

Défition des vues matérialisées : Les vues sont un ensemble de données relationnelles logiques dérivées des tables de base afin que les requêtes complexes puissent être simplifiées en accédant à ces vues. Si les données extraites des vues intermédiaires de ces requêtes sont sauvegardées ou matérialisées, au lieu de générer et de récupérer des données des tables de base pour ces relations dérivées encore et encore, elles peuvent être directement lues à partir des vues matérialisées.

La matérialisation de sous-requêtes et de vues fréquentes signifie que l'ensemble de données résultant des vues réside dans la mémoire d'un ou plusieurs nœuds du cluster, ce qui réduit le

coût MapReduce, le coût de soumission et de planification des travaux du système de fichiers distribués pour le traitement des requêtes.

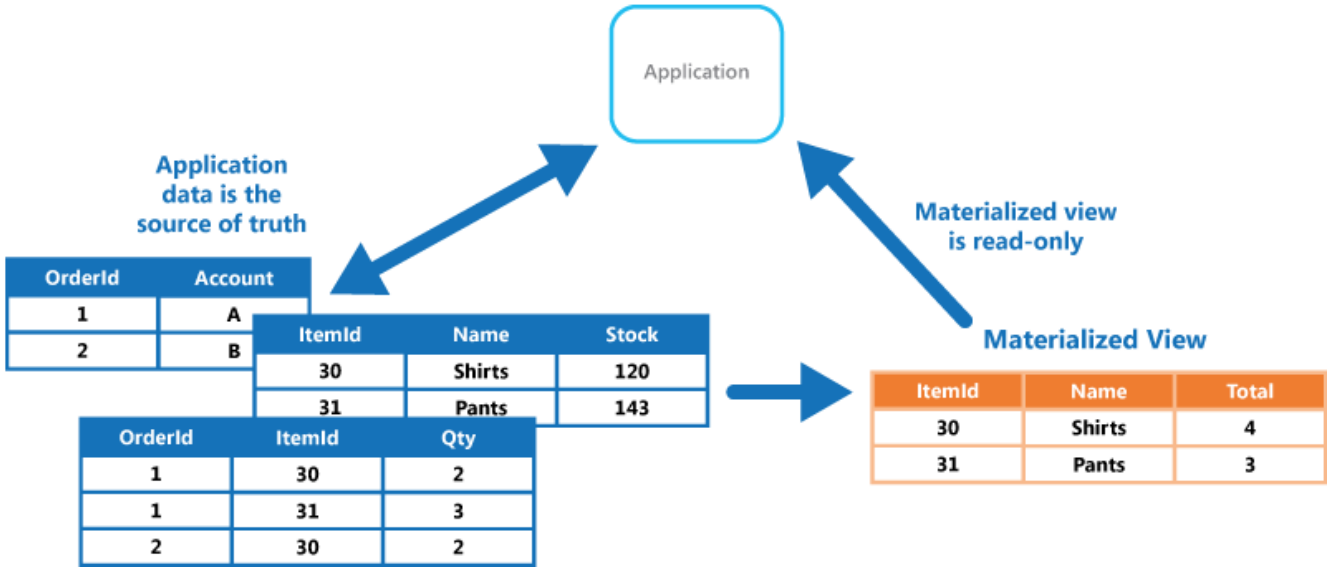


Figure 2.15: Diagramme des vues à matérialiser

Problème de sélections des vues : En basant sur la definition de Chirkova et al. (Chirkova et al., 2001): Soit R l'ensemble des relations de base (comprenant les tables de faits et de dimensions), S l'espace de stockage disponible, Q une charge de travail sur R, L la fonction d'estimation du coût du traitement des requêtes . Le problème de sélection de vue est de trouver l'ensemble des vues V (configuration de vue) sur R dont la taille totale est au plus S et qui minimise L (R, V, Q). [2] Il existe plusieurs approches pour résoudre ce problème, qui peut être classifié comme approche heuristique, approche à algorithme aléatoire et approche utilisant le data mining [18].

Modèle de coût : Le modèle de coût à utiliser pour traiter le problème de sélection de vue matérialisée pour HDFS est différent des modèles de coût utilisés dans les approches utilisées pour l'architecture client-serveur conventionnelle avec l'entreposage de données basé sur le SGBDR. La principale raison derrière cela est que, dans un système classique basé sur un SGBDR, le modèle d'accès aux données est principalement dominé par les recherches et le temps de recherche, tandis que dans HDFS ou dans un cadre distribué similaire, le modèle d'accès aux données est principalement dominé par le taux de transfert de données et les couts de MapReduce [14] .

paragraphLes différents coûts et bénéfices à optimiser pour matérialiser les vues afin d'améliorer le traitement des requêtes dans l'entrepôt Big Data sont d éfinit :

- **Coût de traitement le requêtes :**Le coût total de traitement des requêtes d'un ensemble de requêtes fréquentes peut être considéré comme le total des sur coûts MapReduce pour l'exécution de l'ensemble de requêtes.

- **Coût de maintenance des vues matérialisées :**Dans l'analyse Big Data sur un entrepôt basé sur HDFS, les opérations de mise à jour sont généralement très rares. Mais chaque fois qu'il y a un changement dans les données de base, les vues matérialisées doivent être mises à jour. En cas de matérialisation des requêtes en mémoire, un rafraîchissement fréquent est nécessaire car dans ce cas les requêtes peu fréquentes doivent être rejetées après chaque période de temps fixe [14].

Chapter 3

Approche Proposé

3.1 Problème de sélection des vues a matérialiser :

Dans un environnement Big Data, en vue sa nature en terme volume et de distribution des données, les opérations d'Agrégation, Sélection, Projection et Jointure (ASPJ) sont coûteuses en matière de temps d'exécution Les vues matérialisées sont fréquemment utilisées pour accélérer des requêtes du type ASPJ. Dans notre cas on s'intéresse à développer un outil d'aide à sélection de vues matérialisées dans un environnement Big Data dans le cas de HIVE.

3.1.1 Description du problème

Comme décrit dans section 2.5.3, le problème de sélection de vue à matérialiser est un problème NP-Complex. Il existe plusieurs approches qui peuvent être classifiées en trois catégories:

- Approches Heuristiques
- Approche à Algorithmes aléatoires
- Approches Data Mining

Dans notre cas on s'intéresse à développer un outil d'aide à la sélection des vues à matérialiser. Où à partie d'une analyse syntaxique d'une charge analytique sur une des données stockées dans HIVE on développe un outil qui suggère des vues à matérialiser.

3.1.2 Contexte

Il existe different contexte possible pour le problème de sélection de vue (PSV) à matérialiser, en peut classifier le problème en dépendance de la connaissance préalable ou pas des charges

analytiques d'où le PSV Statique et PSV dynamique Dans notre cas de Hive nous supposons avoir un schéma en étoile à évolution lente. Ainsi qu'une liste de requête analytique en préalable, on s'intéresse à au problème de sélection de vue statique.

3.2 Conception de la solution:

3.2.1 Diagramme De classes

: Le schéma utilisé SSB Star Schema Benchmark, qui est un standard dérive du TPC-H. Le schéma contient 4 tables de dimensions et une table de faits.

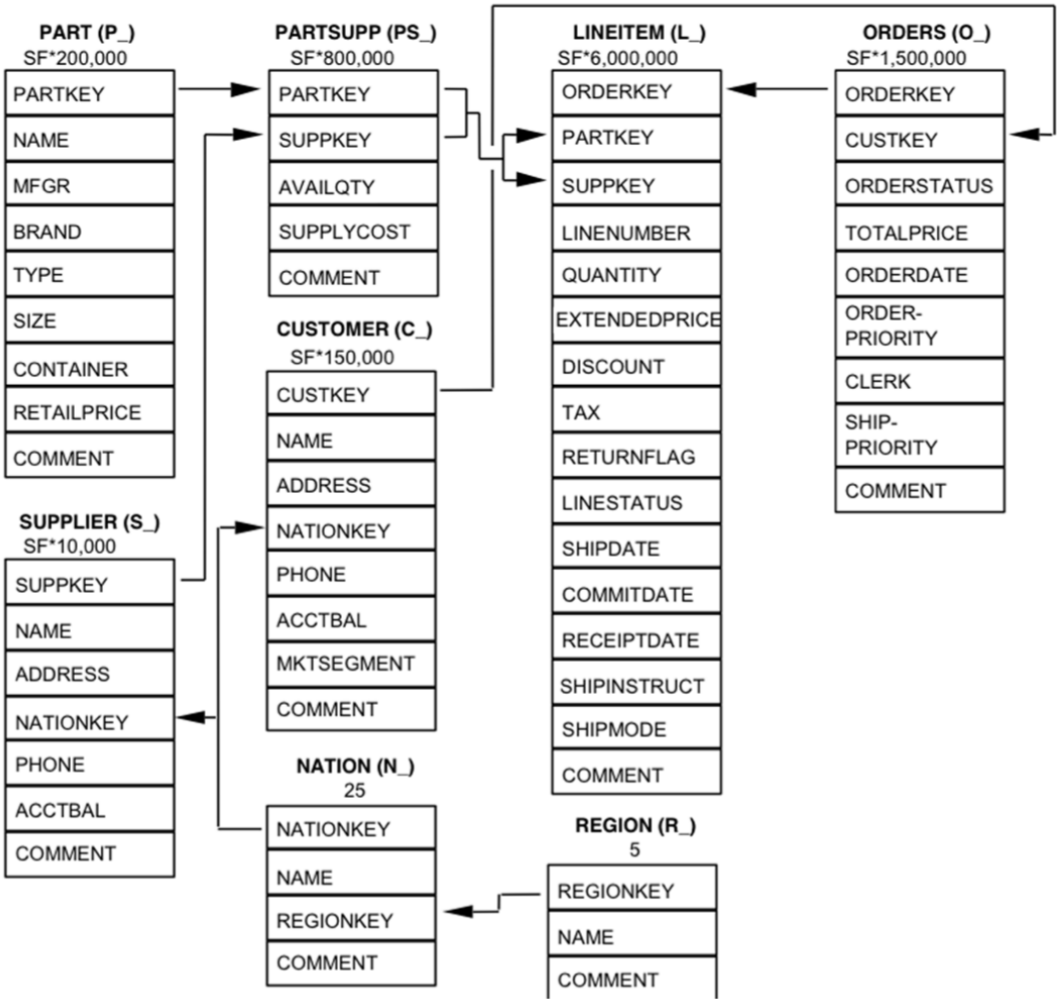


Figure 3.1: Schéma TPCH.

Le schéma SSB est génère a travers un utilitaire on note Scale Factor (SF) un Facteur de scalabilité. La taille de la table des faits est par exemple SF 6000000 de lignes.

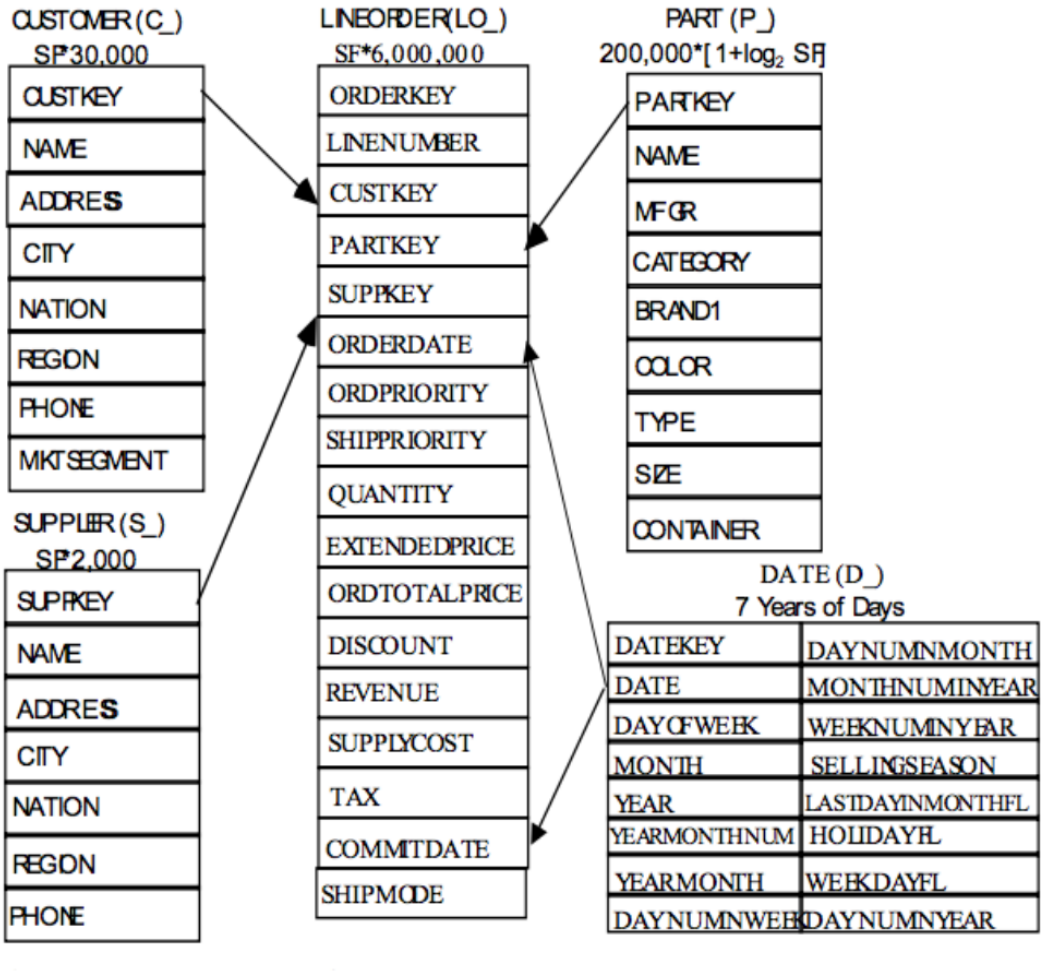


Figure 3.2: SSB-Schéma.

Dans notre solution nous avons utilisé un SSB de Scale Factore 20, soit 120 millions de lignes. Les formules pour les autres dimensions sont mentionnée dans le schéma [13].

3.2.2 Diagramme de cas d'utilisation

Notre solution est un outil de suggestion que l'administrateur de la base de données utilise pour prendre sa décision sur quelle vue à matérialiser à partir de la charge analytique la plus fréquente sur la base de données .

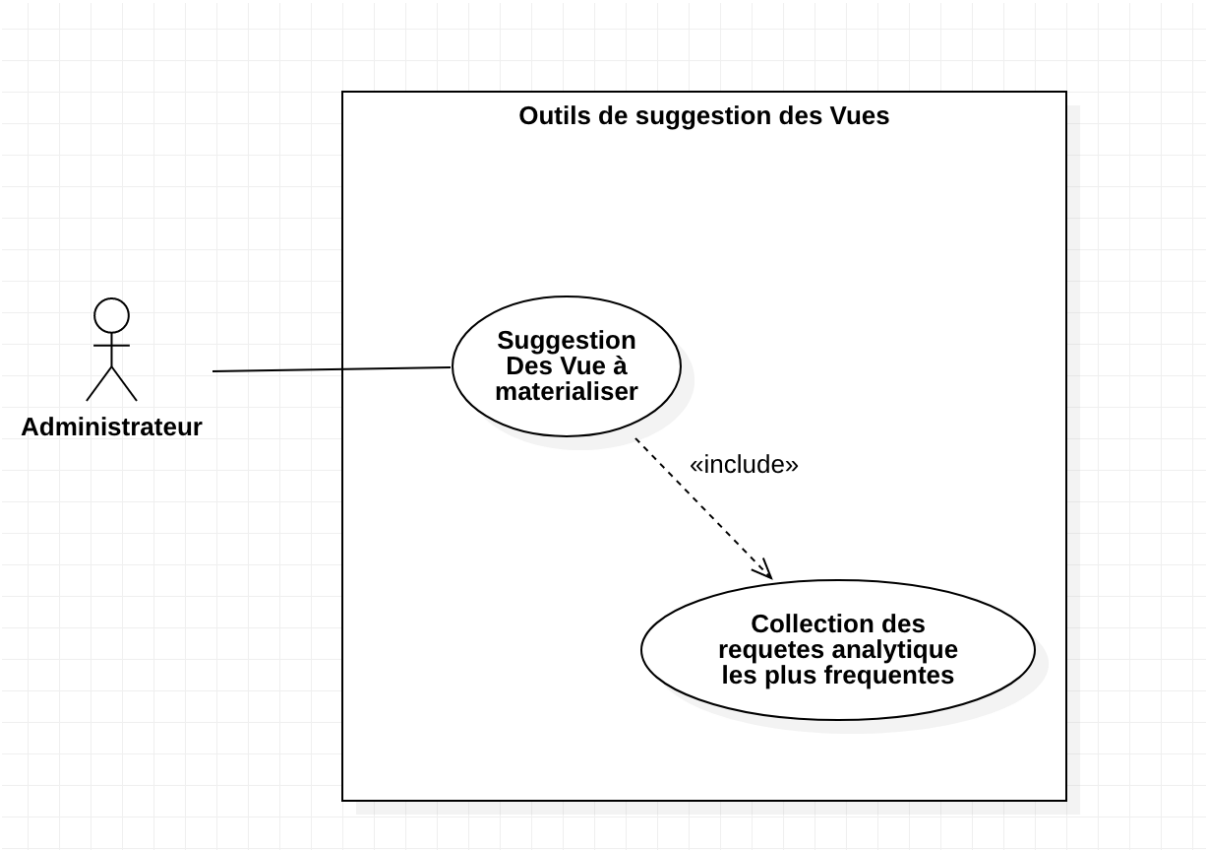


Figure 3.3: Diagramme de cas d'utilisation.

Titre: Suggestion des vues à Matérialiser
Déscription: l'administrateur fait appel à l'outil développé pour avoir des vue candidate à matérialiser par suite
Acteur: administrateur
Enchaînement: l'administrateur fait appel au script de l'outil de suggestion des vues à matérialiser, ce qui déclenche une lecture des requêtes analytiques fréquemment utilisées
Pré-conditions: Droits d'accès, liste des requêtes fréquemment utilisées
Post-conditions: Résultat des vues suggéré pour la matérialisation

Table 3.1: Explication du cas d'utilisation

pour la partie de la collection des requêtes analytiques elle est détaillée dans le schéma de séquence suivante;

Diagramme de séquences :

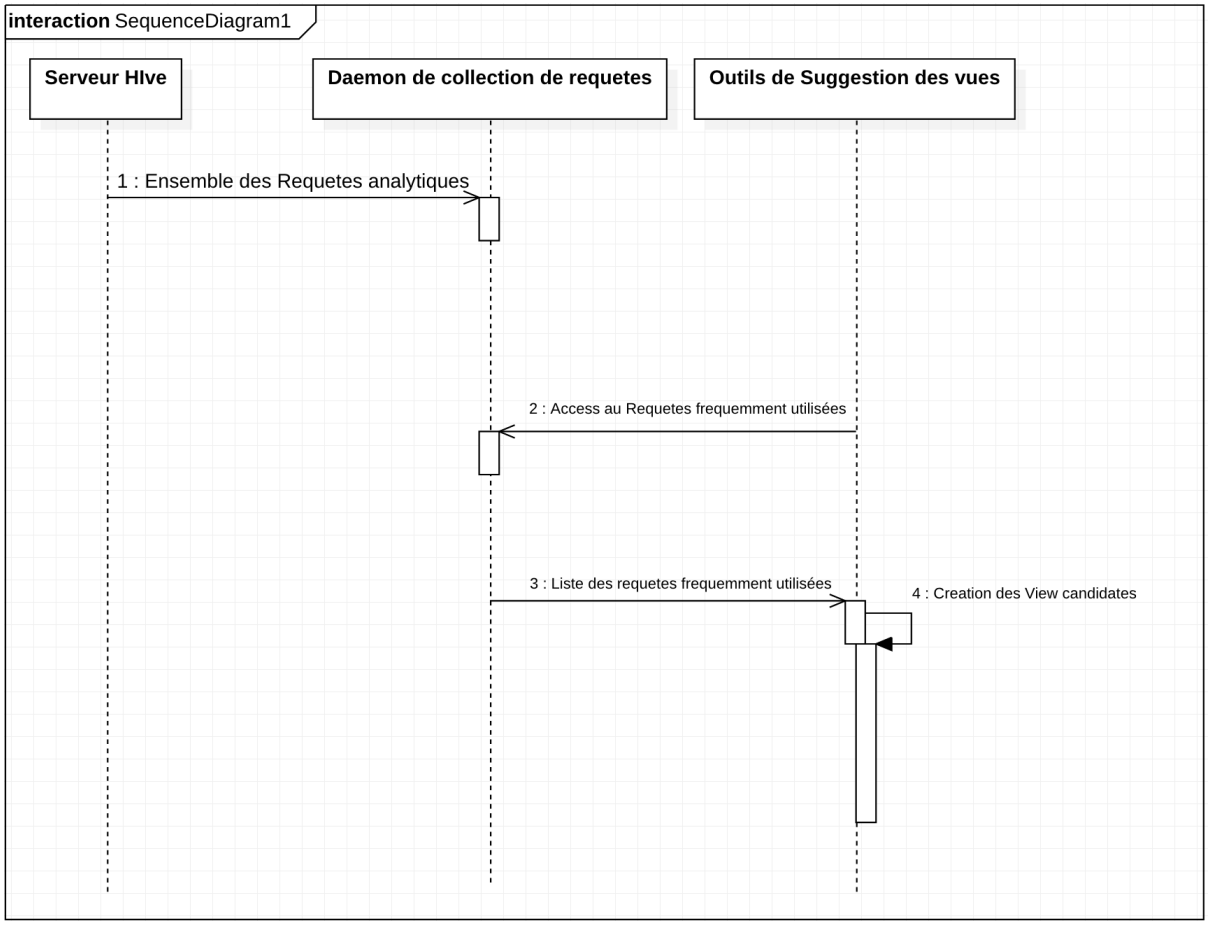


Figure 3.4: Diagramme de séquences.

3.2.3 Principe de fonctionnement

l'outil prend en entrée une charge analytique et applique une analyse syntaxique pour regrouper les requêtes et suggérer les vues à matérialiser.

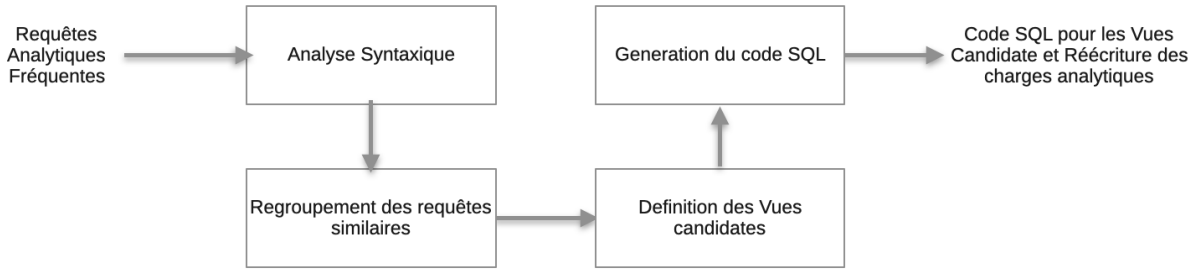


Figure 3.5: Principe de fonctionnement.

Suivant le schéma indiqué ci-dessus, l'algorithme consiste à suivre les phases suivantes:

- Analyse syntaxique des requêtes: pour chaque requête analytique on extrait les opérations de projection, restrictions, jointure et agrégation.

Algorithm 1: Analyse syntaxique

Require: Q Un Ensemble de Requetes $Q = [q_1, q_2...., q_n]$
  for q_i in Q **do**
    $jn \leftarrow q_i.joins$ {récupère les tables de jointures}
    $rg \leftarrow q_i.predicates$ {récupère attributs des prédicats}
    $agg \leftarrow q_i.aggregation$ {récupère les agrégations}
    $prj \leftarrow q_i.projections$ {récupère les projections}
    $q_{ia} \leftarrow [agg, prj, jn, rg]$
    $Q_a \leftarrow Q_a + q_{ia}$
  end for
  return Qa {Retourne un l'ensemble des requêtes sous forme algébrique }

- Regroupement des requêtes similaires: dans notre approche, on regroupe les requêtes ayant les jointures similaires.

- Définition des Vues candidates a matérialisé: On définit les projections, restrictions, agrégations et jointures des vues candidates.

- génération du code SQL: création du script SQL des vues candidates et réécriture des requêtes utilisant ses vues.
  Il existe deux types de vues candidates pour chacun n-groupe de requêtes similaires. Vue

Algorithm 2: Définition de la vue candidate

Require: Q_a Un Ensemble de Requêtes sous forme Algébrique $Q_a = [q_{a1}, q_{a2}...., q_{an}]$
$View_{a1} \leftarrow [agg_v, prj_v, jn_v, groupby_v]$ {Initialisation de la forme algébrique de la vue Sans predicat}
$View_{a2} \leftarrow [agg_v, prj_v, jn_v, rg_v, groupby_v]$ {Initialisation de la forme algébrique de la vue Avec predicat}
for q_{ai} in Q_a **do**
$jn_v \leftarrow jn_v \cup jn_{qai}$
$rg_v \leftarrow rg_v$**or**rg_{qai} {ou logique entre les expressions des predicat}
$agg_v \leftarrow agg_v \cup agg_{qai}$ {récupère les agrégations}
$prj_v \leftarrow prj_v \cup prj_{qai} \cup rg_{qai}$
$groupby_v \leftarrow groupby_v \cup prj_{qai} \cup rg_{qai}$
end for
$View_{a1} \leftarrow [agg_v, prj_v, jn_v, groupby_v]$
$View_{a2} \leftarrow [agg_v, prj_v, jn_v, rg_v, groupby_v]$
return $View_{a1}, View_a$

matérialiser avec prédicat et vue matérialisé sans prédicat. Les vue Matérialises avec prédicat consistent à faire un ou logique entre toutes les requêtes similaires.

Un exemple de requête Similaires:

```
-- Query1
SELECT
sum(lo.extendedprice * lo.discount) as revenue
FROM
lineorder lo
JOIN date_dim d on lo.orderdate = d.datekey
WHERE
d.weeknumberinyear = 6
and d.year = 1994
and lo.discount between 5 and 7
and lo.quantity between 26 and 35;



--Query2
SELECT
sum(lo.extendedprice * lo.discount) as revenue
FROM
lineorder lo
JOIN date_dim d on lo.orderdate = d.datekey
WHERE
d.yearmonthnum = 199401
and lo.discount between 4 and 6
and lo.quantity between 26 and 35;
```

Selon le pseudo code :

- **QUERY1 Q1**
  - **Q1.jn** = [lineorder,date_dim]
  - **Q1.rg** = [d.weeknumberinyear,d.year, lo.discount, lo.quantity]
  - **Q1.agg** = [sum(lo.extendedprice - lo.discount)]
  - **Q1.prj** = [ ]
- **QUERY1 Q2**
  - **Q1.jn** = [lineorder,date_dim]
  - **Q1.rg** = [d.yearmonthnum, lo.discount, lo.quantity]
  - **Q1.agg** = [sum(lo.extendedprice - lo.discount)]
  - **Q1.prj** = [ ]

Le résultat donnée par l'outil est :

```
create materialized view View0 as
select
    d.weeknuminyear as d_weeknuminyear,
    d.year as d_year,
    d.yearmonthnum as d_yearmonthnum,
    lo.discount as lo_discount,
    lo.quantity as lo_quantity,
    sum(lo.extendedprice * lo.discount) as revenue,
    sum(lo.revenue - lo.supplycost) as profit
from
    lineorder lo
    join date_dim d on lo.orderdate = d.datekey
group by
    d.weeknuminyear,
    d.year,
    d.yearmonthnum,
    lo.discount,
    lo.quantity
```

Figure 3.6: Exemple de vue suggérée sans prédicat

```
create materialized view View0p as
select
    d.weeknuminyear as d_weeknuminyear,
    d.year as d_year,
    d.yearmonthnum as d_yearmonthnum,
    lo.discount as lo_discount,
    lo.quantity as lo_quantity,
    sum(lo.extendedprice * lo.discount) as revenue,
    sum(lo.revenue - lo.supplycost) as profit
from
    lineorder lo
    join date_dim d on lo.orderdate = d.datekey
where
    (
        d.weeknuminyear = 6
        and d.year = 1994
        and lo.discount between 5
        and 7
        and lo.quantity between 26
        and 35
    )
    or (
        d.yearmonthnum = 199401
        and lo.discount between 4
        and 6
        and lo.quantity between 26
        and 35
    )
group by
    d.weeknuminyear,
    d.year,
    d.yearmonthnum,
    lo.discount,
    lo.quantity
```

Figure 3.7: Exemple de vue suggérée avec prédicat

Dans la section suivante nous allons mettre en détaille l'implementation de notre solution et l'environnement de travail .

3.3 Environnement de travail:

3.3.1 Environnement matériel

:

Une Machine Macbook Pro 13 pouce 2014 ayant les caractéristiques suivants:

- Processeur: i5 Dual Core
- Ram : 16 GB
- Stockage : 256 SSD
- System D'exploitation: OS X Catalina

Pendant toute la période du travail le déploiement du système en expérimentation a été en single-node implétentant yarn.

3.3.2 Architecture et déploiement

Le déploiement d'Hive a été réalise en single-node en utilisant une image Docker.
La base de données SSB a été générée en utilisant le ssb-dbgen.
Le script SQL pour générer le schéma SSB sur HIVE et les tables générées ont été intégrés dans la machine docker a travers l'utilitaire docker CP.
Presto Permet l'interrogation d'Hive en utilisant le terminal de la machine hôte sous Mac Os.

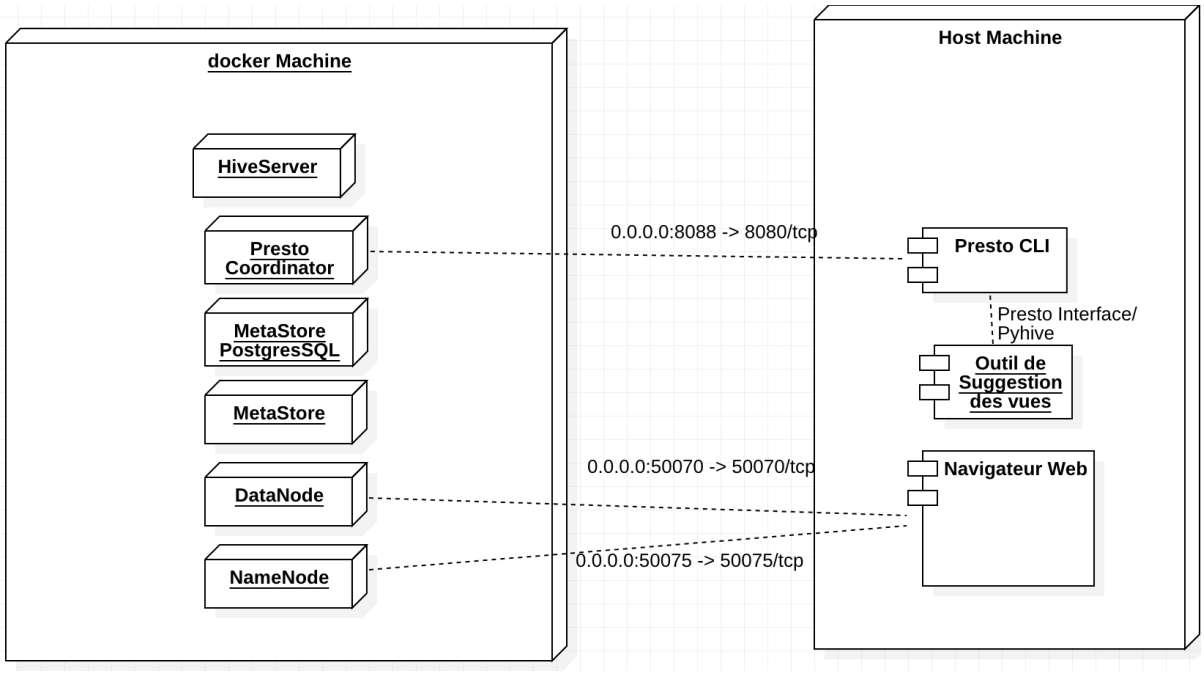


Figure 3.8: Diagramme de déploiement

3.3.3 Environnement de développent et outils

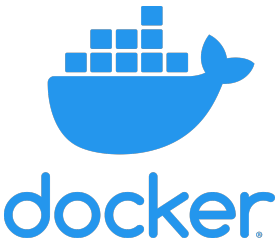


Figure 3.9: Docker

Docker à était lancé comme projet open source en 2013. Docker permet le lancement d'application dans un conteneur indépendamment de la machine hôte. Il offre plusieurs avantages notamment le déploiement facile

Dans le cas de notre projet, Docker nous a permis de lancer Hive et ses services sans avoir à configurer le tout.

Presto est un moteur de requête SQL distribué open source pour exécuter des requêtes analytiques interactives sur des sources de données de toutes tailles allant de gigaoctets à pétaoctets.Presto permet aussi d'avoir une interface web pour le monitoring d'un cluster, cette interface permet d'avoir un dashboard et des informations nécessaires sur l'exécution des requêtes analytiques.

Figure 3.10: Prestodb

Figure 3.11: Python

Python : est l'un des langages de programmation les plus utilisés, nous avons utilisé Python3.X lors de la conception de notre solution. Python supporte plusieurs paradigme de programmation, il est polyvalent et simple à utiliser grâce a ses différentes librairies telles que PyHive et Sqlparse, Re .. utilisé dans le cadre de notre projet.

PyCharm: Pour l'écriture du code de l'outil de suggestion des vue a matérialisé nous avons choisi PyCharm comme Environnement de développement intégré.PyCharm a été developpé par Jetbrains, il contient des fonctionnalités permettant d'économiser du temps dans le processus de développement, il permet d'intégrer d'autres environnements et outils tels que Git.Il permet aussi de détecter les packages et de les installer si nécessaire.

Figure 3.12: Pycharm

Figure 3.13: Git

Git utilisé pour le visionnement et le contrôle de l'historique de nos documents.
Git permet de gérer les différents changements dans le projet et de restaurer une version antécédente en cas de nécessité.

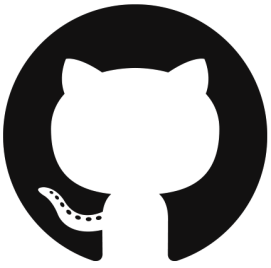


Figure 3.14: GitHub

GitHub utilisé pour le visionnement et le partage de code.
GitHub à été utilisé dans le cadre de notre projet pour la collaboration et l'acquisition des différents projets open source servant à la construction de notre système.

Chapter 4

Résultat et évaluation

Dans ce chapitre nous allons visualiser les résultats de notre approche, nous nous sommes basé sur le SSB benchmark, on exécute différentes requêtes analytiques sur notre système et on compare le résultat en temps d'exécution et son amélioration suivant les recommandations de notre outil. Nous avons créé 3 bases de données avec 3 facteurs de scalabilité différentes (Scale Factor SF).

	Row Count	**Taille en GB**
Scaling Factor=1	6 Millions	0.5 GB
Scaling Factor=5	30 Millions	2.9 GB
Scaling Factor=20	120Millions	12 GB

Table 4.1: Scaling factors utilisés.

Nous prenons dans notre exemple 3 requêtes ayant des jointures similaires, le code des requêtes est le suivant:

```
select
    sum(lo.revenue) as revenue,
    d.year,
    p.brand1
from
    lineorder lo
    join date_dim d on lo.orderdate = d.datekey
    join part p on lo.partkey = p.partkey
    join supplier s on lo.suppkey = s.suppkey
where
    p.category = 'MFGR#12'
    and s.region = 'AMERICA'
group by
    d.year,
    p.brand1
order by
    d.year,
    p.brand1;
```

Figure 4.1: Requête Q1

```
select
    sum(lo.revenue) as revenue,
    d.year,
    p.brand1
from
    lineorder lo
    join date_dim d on lo.orderdate = d.datekey
    join part p on lo.partkey = p.partkey
    join supplier s on lo.suppkey = s.suppkey
where
    p.brand1 = 'MFGR#2221'
    and s.region = 'EUROPE'
group by
    d.year,
    p.brand1
order by
    d.year,
    p.brand1;

select
```

Figure 4.2: Requête Q2

```
select
    sum(lo.revenue) as revenue,
    d.year,
    p.brand1
from
    lineorder lo
    join date_dim d on lo.orderdate = d.datekey
    join part p on lo.partkey = p.partkey
    join supplier s on lo.suppkey = s.suppkey
where
    p.brand1 between 'MFGR#2221'
    and 'MFGR#2228'
    and s.region = 'ASIA'
group by
    d.year,
    p.brand1
order by
    d.year,
    p.brand1;
```

Figure 4.3: Requête Q3

Lors de l'exécution de ces requêtes sur notre système nous obtenons le résultat suivant:

	Temps Q1	**Temps Q2**	**Temps Q3**
Scaling Factor=1	24s	16s	22s
Scaling Factor=5	1m15s	0m59s	1m06s
Scaling Factor=20	14m35s	5m09s	6m50s

Table 4.2: Temps d'exécution en rapport du scaling factor (SF) .

Notre outil alors génère ainsi des vue à matérialiser pour ces requêtes, on distingue de notre solution deux types de vue suggérées pour ces requêtes **Vue Matérialises Sans Prédicat** décrit dans section 4.2 et **Vue Matérialises Avec Prédicat** décrit dans section 4.2

4.1 Matérialisation sans prédicat

En exécutant notre outil sur les requêtes précédentes on obtient comme résultat la vue suggérée suivante.

```
create materialized view View2 as
select
    d.year as d_year,
    p.brand1 as p_brand1,
    p.category as p_category,
    s.nation as s_nation,
    s.region as s_region,
    sum(lo.revenue) as revenue
from
    lineorder lo
    join date_dim d on lo.orderdate = d.datekey
    join part p on lo.partkey = p.partkey
    join supplier s on lo.suppkey = s.suppkey
group by
    d.year,
    p.brand1,
    p.category,
    s.nation,
    s.region
```

Figure 4.4: Matérialisation sans prédicats

Nous optons pour le choix de cette vue, et on applique à nouveau les requêtes Q1 - Figure 4.1, Q2 - Figure 4.2, Q3 - Figure 4.3 sur cette vue. La réécriture des requêtes est donnée par notre outil. En calculant le temps moyen des exécutions et on comparant avec le baseline de l'exécution de ces requêtes sans vue matérialisée et avec vue matérialisée en obtient les résultats suivants.

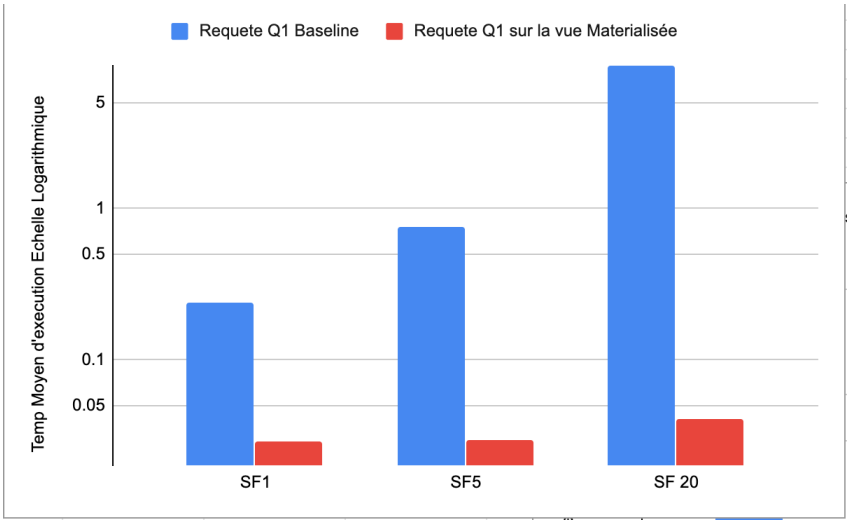


Figure 4.5: Graphe de comparaison entre différentes exécutions de Q1

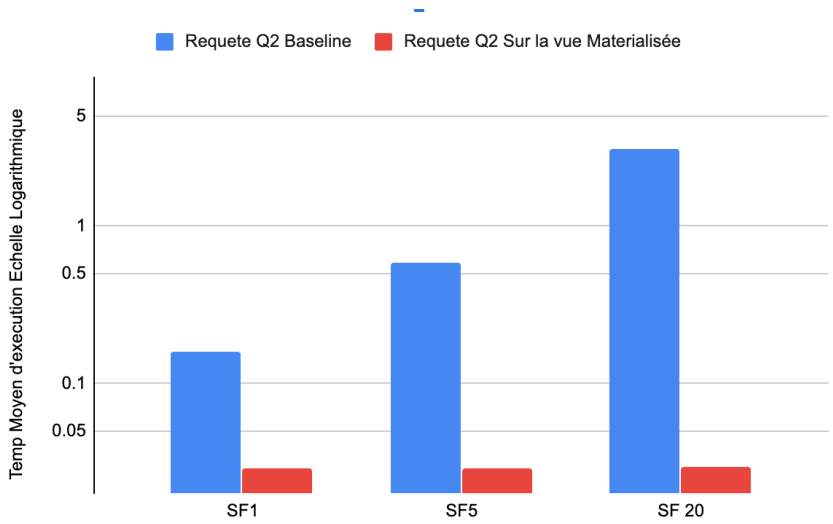


Figure 4.6: Graphe de comparaison entre différentes exécutions de Q2

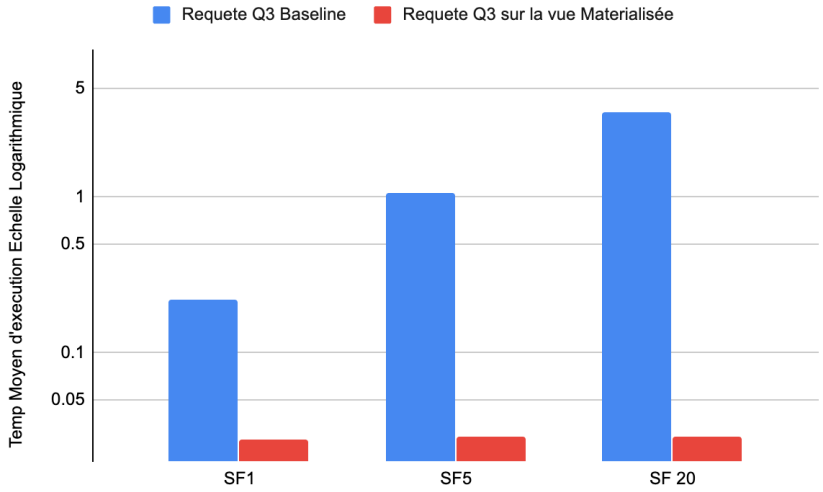


Figure 4.7: Graphe de comparaison entre différentes exécutions de Q3

Dans l'axe Y nous avons le temps d'exécution en échelle logarithmique, et dans les axes X nous avons les différents facteurs de Scalabilité (SF). Nous constatons un gain considérable dans le temps d'exécution. Par moyenne on obtient le gain résumé par ce tableau.

	Gain Q1	**Gain Q2**	**Gain Q3**
Scaling Factor=1	87.85%	81.87%	87.72%
Scaling Factor=5	96.06%	95.05%	97.27%
Scaling Factor=20	99.53%	99.04%	99.16%

Table 4.3: Tableau de gains en temps d'exécution

4.2 Matérialisation avec prédicat

En exécutant notre outils sur les requêtes précédentes Q1 - Figure 4.1, Q2 - Figure 4.2, Q3 - Figure 4.3 on obtient comme résultat la vue suggérée suivante.

```
create materialized view View2p as
select
    d.year as d_year,
    p.brand1 as p_brand1,
    p.category as p_category,
    s.nation as s_nation,
    s.region as s_region,
    sum(lo.revenue) as revenue
from
    lineorder lo
    join date_dim d on lo.orderdate = d.datekey
    join part p on lo.partkey = p.partkey
    join supplier s on lo.suppkey = s.suppkey
where
    (
        p.category = 'MFGR#12'
        and s.region = 'AMERICA'
    )
    or (
        p.brand1 = 'MFGR#2221'
        and s.region = 'EUROPE'
    )
    or (
        p.brand1 between 'MFGR#2221'
        and 'MFGR#2228'
        and s.region = 'ASIA'
    )
group by
    d.year,
    p.brand1,
    p.category,
    s.nation,
    s.region
```

Figure 4.8: Matérialisation avec prédicats

Nous optons pour le choix de cette vue, et on applique les requêtes Q1 - Figure 4.1, Q2 - Figure 4.2, Q3 - Figure 4.3 sur cette vue. La réécriture des requêtes est donnée par notre outil. En calculant le temps moyen des exécutions et on comparant avec le baseline de l'exécution de ces requêtes sans vue matérialisée et avec vue matérialisée en obtient les résultats suivants.

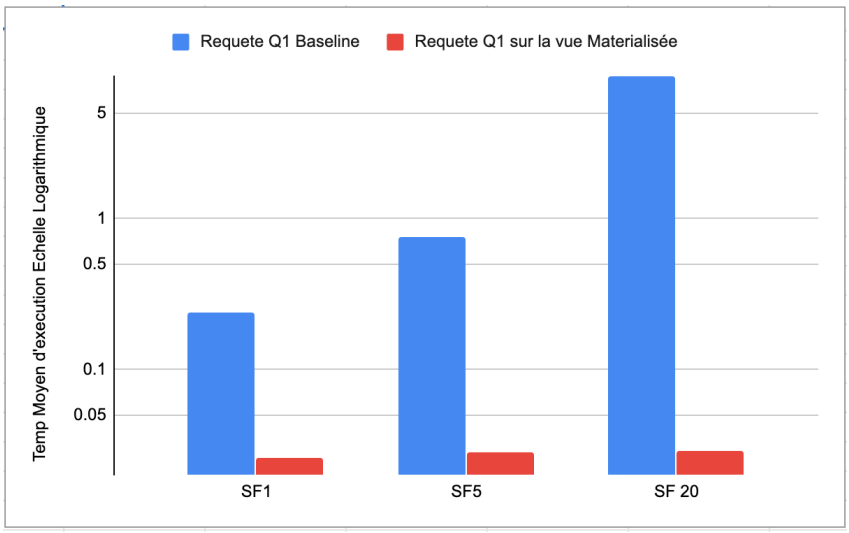


Figure 4.9: Graphe de comparaison entre différentes exécutions de Q1 cas de vue avec prédicat

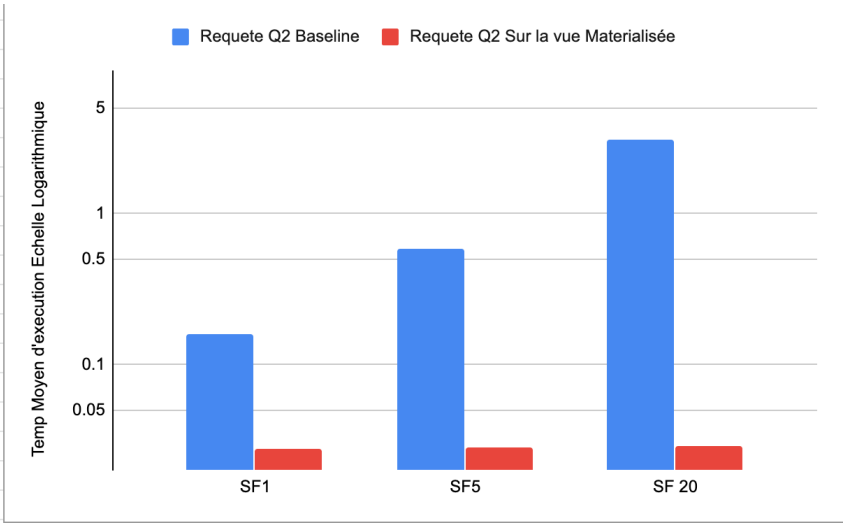


Figure 4.10: Graphe de comparaison entre différentes exécutions de Q2 cas de vue avec prédicat

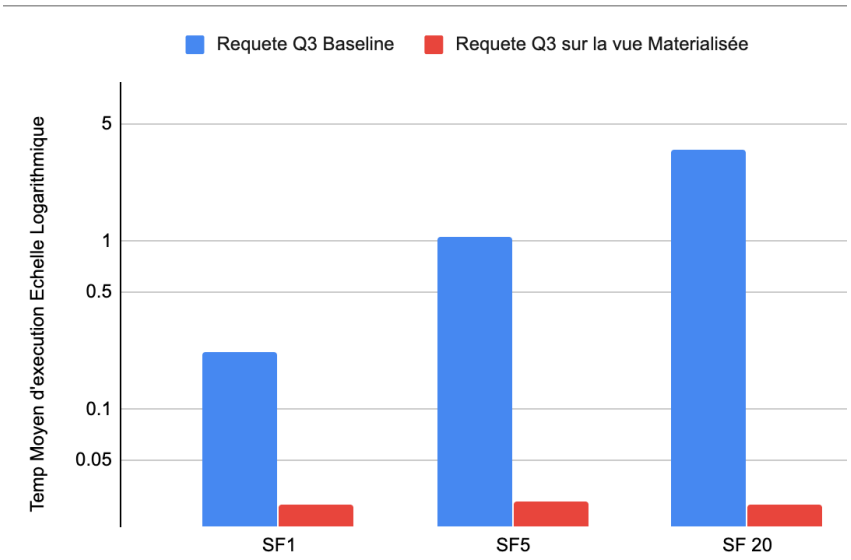


Figure 4.11: Graphe de comparaison entre différentes exécutions de Q3 cas de vue avec prédicat

Dans l'axe Y nous avons le temps d'exécution en échelle logarithmique, et dans les axes X nous avons les différents acteurs de Scalabilité (SF). Nous constatons un gain considérable dans le temps d'exécution. Par moyenne on obtient le gain résumé par ce tableau.

	Gain Q1	**Gain Q2**	**Gain Q3**
Scaling Factor=1	89.12%	82.5%	87.72%
Scaling Factor=5	96.62%	95.23%	97.32%
Scaling Factor=20	99.66%	99.06%	99.22%

Table 4.4: Tableau de gains en temps d'exécution cas de vue matérialisée

4.3 Étude comparative

Dans notre expérimentation nous avons explorer les deux options de génération de vue à matérialiser avec et sans prédicat. Pour chaque approche il existe plusieurs contraintes telles que le coûts de génération et de maintenance des vues et l'espace occupé par ces dernières.

Nous appelons V1 la vue matérialisée Sans Prédicat décrite dans Figure 4.2 Nous appelons V2 la vue matérialisée Avec Prédicat décrite dans Figure 4.2

	Temps pour générer V1	**Temps pour générer V2**	**Rapport Espace V1/V2**
Scaling Factor=1	2m11s	2:42s	102
Scaling Factor=5	7m02s	9m35s	102
Scaling Factor=20	40m05s	1h01m02s	102

Table 4.5: Tableau comparatif entre temps de génération des vue et le rapport espace

Contrainte Temps: Il est plus coûteux de générer et de maintenir une vue avec prédicat comparé à une vue sans prédicat.

Contrainte Espace : une vue avec prédicat occupe moins d'espace qu'une vue sans prédicats mais cela dépend aussi de la sélectivité des prédicats des requêtes qui contribuent à la génération de cette dernière.

Contrainte Évolution De la Charge analytique : la limitation de l'approche avec prédicat est l'évolution de la charge analytique, en cas d'introduction de nouvelles charges analytiques le modèle sans prédicant permet d'intégrer ses requêtes et d'utiliser les vues pour satisfaire une partie ou la totalité de la requête.

Le choix de l'approche à utiliser dépend alors du contexte et des contraintes système. nous pouvons résumer les différences entre les deux approches dans le tableau suivant :

Critère	**Matérialisation sans prédicat**	**Matérialisation avec Predicat**
Création et maintenance	Moins coûteuse	coûteuse
Espace	Occupe plus d'espace	Occupe moins d'espaces
Évolution	Possibilité d'intégration nouvelle charges analytique	Moins flexible adapté au cas de charge non évolutive

Table 4.6: Tableau récapitulatif des différences entre les approches

Chapter 5

Conclusion

Dans le but d'accélérer une charge analytique et Dans le cadre de notre projet de fin d'études nous avons implémenté un système de suggestion de vue à matérialiser.Cette thèse a été structurée pour citer les différentes approches possible d'optimisation de requêtes analytique dans le cas du big data et la conception de notre contribution dans le cadre de la technique HIVE.

5.1 Perspective futures:

l'amélioration de notre outil peut prendre plusieurs approches nous pouvons citer:

- utilisation de méthodes de data Mining pour l'analyse de sélectivité des requêtes.
- utilisation d'algorithmes de clustering et machine learning pour la classification et regroupement des requêtes analytiques.
- inclusion des contraintes système dans la classification et le regroupement de requête .

Bibliography

[1] William H. Inmon. *Building the Data Warehouse*. USA: John Wiley Sons, Inc., 1992. ISBN: 0471569607.

[2] Rada Chirkova, Alon Halevy, and Dan Suciu. "A Formal Perspective on the View Selection Problem". In: *VLDB Journal* 11 (Sept. 2001). DOI: 10.1007/s00778-002-0070-0.

[3] W. H. Inmon. In: *Building the Data Warehouse*. Robert Ipsen, 2008, p. 43. ISBN: 0-471-08130-2.

[4] Andrew McAfee and Erik Brynjolfsson. "Big Data: The Management Revolution". In: *Harvard business review* 90 (Oct. 2012), pp. 60–6, 68, 128.

[5] A.Turk et al. "Temporal Workload-Aware Replicated Partitioning for Social Networks". In: vol. 26. 11. 2014, pp. 2832–2845.

[6] Sachin P Bappalige. *An introduction to Apache Hadoop for big data*. Aug. 2014.

[7] Alex Holmes. *Hadoop in Practice: Includes 104 Techniques*. Manning Publications, Oct. 2014. Chap. 1. ISBN: 1617292222. URL: https://www.xarg.org/ref/a/1617292222/.

[8] Fatima Adamu et al. "A Survey On Big Data Indexing Strategies". In: Dec. 2015. DOI: 10.13140/RG.2.1.1844.0721.

[9] Haider andomi. "Beyond the hype: Big data concepts, methods, and analytics". In: *International Journal of Information Management* 35 (Apr. 2015), pp. 137–144.

[10] Tom white. *hadoop , the definitive guide*. 4th ed. 2015.

[11] Abdullah Gani et al. "A Survey on Indexing Techniques for Big Data: Taxonomy and Performance Evaluation". In: *Knowl. Inf. Syst.* 46.2 (Feb. 2016), pp. 241–284. ISSN: 0219-1377. DOI: 10.1007/s10115-015-0830-y. URL: https://doi.org/10.1007/s10115-015-0830-y.

[12] Raghavendra Kune et al. "The anatomy of big data computing". In: *Softw. Pract. Exp.* 46 (2016), pp. 79–105.

[13] Jimi Sanchez. "A Review of Star Schema Benchmark". In: (June 2016).

[14] Dutta Goswami Bhattacharyya. "Materialized view selection using evolutionary algorithm for speeding up big data query processing". In: *Intell Inf Syst* 49 (2017), p. 27.

[15] Ahmed Oussous et al. "Big Data technologies: A survey". In: *Journal of King Saud University - Computer and Information Sciences* 30 (2017), pp. 431–448.

[16] Umar Bin Qushem et al. “The trend of business intelligence adoption and maturity”. In: *2017 International Conference on Computer Science and Engineering (UBMK)*. IEEE, Oct. 2017, pp. 532–537. DOI: 10.1109/ubmk.2017.8093455. URL: https://doi.org/10.1109/ubmk.2017.8093455.

[17] Pwint Phyu Khine and Zhao Shun Wang. “Data lake: a new ideology in big data era”. In: *ITM Web of Conferences* 17 (2018). Ed. by Kei Eguchi and Tong Chen, p. 03025. DOI: 10.1051/itmconf/20181703025. URL: https://doi.org/10.1051/itmconf/20181703025.

[18] Leandro Ante et al. “Automatic View Selection for Distributed Dimensional Data”. In: May 2019. DOI: 10.5220/0007555700170028.

[19] J. Camacho-Rodríguez et al. “Apache Hive: From MapReduce to Enterprise-grade Big Data Warehousing”. In: *Proceedings of the 2019 International Conference on Management of Data* (2019).

[20] Philippe Rigaux. *Systèmes NoSQL*. 2019. URL: http://b3d.bdpedia.fr.

[21] *Apache Hadoop*. https://hadoop.apache.org/. (Accessed on 09/11/2020).

[22] *DataAge 2025 - La numérisation du monde — Seagate France*. https://www.seagate.com/fr/fr/our-story/data-age-2025/. (Accessed on 09/10/2020).

[23] *dataage-idc-report-07-2020.pdf*. https://www.seagate.com/files/www-content/our-story/trends/files/dataage-idc-report-07-2020.pdf. (Accessed on 09/10/2020).

[24] *Introduction to Big Data and Hadoop*. https://www.simplilearn.com/introduction-to-big-data-and-hadoop-tutorial.

[25] Shruti M. *Hadoop Tutorial: A Beginner’s Guide*. https://www.simplilearn.com/hadoop-tutorial-article.

[26] *The biggest data challenges that you might not even know you have - Watson Blog*. https://www.ibm.com/blogs/watson/2016/05/biggest-data-challenges-might-not-even-know/. (Accessed on 09/10/2020).