République Algérienne Démocratique et Populaire
Université des Sciences et de la Technologie HOUARI BOUMEDIENE
Faculté d'Informatique

Architecture d'Apache Spark et Clustering de Big Data avec K-means

Présenté par

— Mouhoub Massinissa
— Bouchair Aya
— Hannoun Amar Amine
— Merbaet Katia
— Zigadi Sadji
— Mechidal Moncif

Table des matières

0.1 Introduction

La croissance rapide des données numériques a profondément transformé la manière dont les informations sont stockées, traitées et analysées. Les systèmes de bases de données traditionnels deviennent souvent insuffisants face au volume, à la variété et à la vitesse des données modernes, ce qui a conduit à l'émergence des technologies du Big Data. Parmi ces technologies, Apache Spark s'impose comme l'un des frameworks les plus puissants et les plus utilisés pour le calcul distribué, grâce à ses capacités de traitement en mémoire, sa scalabilité et son support de multiples types de traitements tels que le batch, le streaming, le machine learning et l'analyse de graphes.

Ce document présente une vue d'ensemble des systèmes Big Data et des architectures de bases de données, avec un accent particulier sur Apache Spark et son écosystème. Il débute par une introduction aux concepts fondamentaux du Big Data, aux bases de données relationnelles et NoSQL, ainsi qu'aux différences entre architectures traditionnelles et architectures Big Data. Il explore ensuite l'architecture de Spark en détaillant ses principaux composants et modules, notamment Spark SQL, Spark Streaming, MLlib, GraphX et SparkR. Enfin, ce travail met en évidence des applications concrètes de Spark et propose un cas d'utilisation basé sur l'algorithme de clustering K-means à l'aide de Spark MLlib.

L'objectif de ce travail est de mieux comprendre comment Spark peut être utilisé comme une plateforme unifiée pour le traitement efficace de données à grande échelle et pour l'apprentissage automatique. En combinant des aspects théoriques et pratiques, cette étude met en évidence la pertinence de Spark dans les applications modernes orientées données.

Chapitre 1

Big Data et Systèmes de Bases de Données

1.1 Introduction au Big Data

Le Big Data englobe le vaste ensemble d'informations avec lesquelles nous interagissons quotidiennement, comprenant de nombreux zettaoctets provenant de nos ordinateurs, appareils mobiles et divers capteurs.
Les entreprises exploitent ces données pour éclairer leurs décisions, affiner leurs processus et politiques, et concevoir des produits et services centrés sur le client. Le Big Data doit son nom non seulement à son volume considérable, mais aussi à sa diversité et sa complexité, dépassant souvent les capacités des bases de données traditionnelles en matière de capture, de gestion et de traitement de telles données.

1.1.1 Types de Big Data

1. **Données structurées :** Cette catégorie englobe les types de données facilement organisés et accessibles, tels que les dossiers financiers, les statistiques démographiques et les journaux de machines. Pensez à une feuille de calcul Excel avec ses colonnes et lignes bien ordonnées, facilitant la visualisation et la catégorisation.
2. **Données non structurées :** On y trouve les publications sur les réseaux sociaux, les fichiers audio, les images et les retours clients en format libre. Contrairement aux lignes et colonnes ordonnées des bases de données traditionnelles, les données non structurées trouvent souvent leur place dans des lacs de données, des entrepôts de données et des bases de données NoSQL en raison de leur format non conventionnel.

3. **Données semi-structurées :** Combinant des éléments de données structurées et non structurées, on peut citer par exemple les e-mails contenant des corps de messages non structurés accompagnés de métadonnées structurées telles que l'expéditeur, le destinataire et la date. De même, les appareils utilisant des géotags, des horodatages ou des marqueurs sémantiques offrent des données structurées intégrées dans un contenu non structuré.

1.1.2 Les 5 V du Big Data

Dans le paysage actuel axé sur les données, les 5 V du Big Data — Volume, Vélocité, Variété, Véracité et Valeur — définissent les défis et les opportunités inhérents à la gestion de grandes quantités d'informations. Du volume considérable de données nécessitant des analyses avancées au traitement en temps réel permis par les technologies modernes, la compréhension de ces aspects est cruciale pour les entreprises souhaitant tirer des insights exploitables et un avantage concurrentiel de leurs données.

1. **Volume :** Bien que ce ne soit pas le seul facteur déterminant, le volume considérable de données est indéniablement un aspect central du Big Data. L'exploitation de ces données nécessite des algorithmes avancés et des analyses pilotées par l'IA, appuyées par des mécanismes robustes de stockage, d'organisation et de récupération.
2. **Vélocité :** Les technologies modernes du Big Data permettent le traitement et l'analyse en temps réel, produisant des insights exploitables en quelques millisecondes après la génération des données.
3. **Variété :** Au-delà des ensembles de données structurées, la véritable essence du Big Data réside dans son amalgame de formats de données structurées, non structurées et semi-structurées.
4. **Véracité :** Les données n'ont de valeur que si elles sont précises, pertinentes et pratiques. Pourtant, des défis tels que les biais humains, le bruit social et les problèmes de provenance des données peuvent nuire à la qualité des données.
5. **Valeur :** Bien que le potentiel d'insights fascinants soit indéniable, la vraie valeur du Big Data réside dans sa capacité à fournir aux entreprises une intelligence exploitable, favorisant la compétitivité, la résilience et un meilleur service client.

1.1.3 Opérations sur le Big Data

— **Collecte du Big Data :** Une grande partie du Big Data est constituée d'énormes ensembles de données non structurées, provenant de sources disparates et incohérentes. Les bases de données traditionnelles sur disque et les mécanismes d'intégration de données ne sont tout simplement pas en mesure de gérer tout cela. La gestion du Big Data nécessite l'adoption de solutions de bases de données en mémoire et de solutions logicielles spécifiques au Big Data.

— **Stockage du Big Data :** Le Big Data est, comme son nom l'indique, volumineux. De nombreuses entreprises disposent de solutions de stockage sur site pour leurs données existantes et espèrent réduire les coûts en réutilisant ces dépôts pour traiter le Big Data. Cependant, le Big Data est plus efficace lorsqu'il n'est pas soumis à des contraintes de taille et de mémoire. Les entreprises qui n'intègrent pas de solutions de stockage en cloud dans leurs modèles Big Data dès le départ le regrettent souvent quelques mois plus tard.[1]

— **Analyse du Big Data :** Pour tirer le meilleur parti du Big Data, nous avons besoin de l'IA et du Machine Learning pour l'analyse. La « Vélocité », qui consiste à obtenir des insights rapidement, est cruciale. Pour cela, nous nous appuyons sur l'IA et les bases de données modernes pour optimiser les processus et apprendre de l'expérience.[1]

1.2 Bases de Données SQL et Systèmes de Gestion de Bases de Données

1.2.1 Définition d'un SGBD

Un Système de Gestion de Base de Données (SGBD) est un système logiciel qui facilite la création, l'organisation, la gestion et la manipulation des bases de données.

À sa base, un SGBD sert d'interface entre les utilisateurs et les bases de données, fournissant des méthodes efficaces pour stocker, récupérer, mettre à jour et gérer les données. Il agit comme un référentiel centralisé pour le stockage des données, garantissant l'intégrité, la sécurité et l'accessibilité des données.

Le rôle principal d'un SGBD est de faciliter la gestion efficace et efficiente de grands volumes de données structurées et non structurées.

1.2.2 Types de SGBD

1. **SGBD Relationnel (SGBDR) :** Le SGBD relationnel organise les données en tables avec des lignes et des colonnes, et établit des relations entre les tables à l'aide de clés. Il suit les principes de l'algèbre relationnelle et du SQL pour la manipulation des données. Les exemples incluent MySQL, Oracle Database et PostgreSQL. Le SGBDR est adapté aux applications transactionnelles, à l'entreposage de données et au traitement analytique en ligne (OLAP).
2. **SGBD Orienté Objet (SGBDOO) :** Le SGBD orienté objet stocke les données sous forme d'objets, encapsulant les données et les comportements ensemble. Il prend en charge l'héritage, l'encapsulation, l'abstraction et le polymorphisme, permettant une modélisation et une manipulation complexes des données. Les exemples incluent db4o (dataBase For Objects) et ObjectStore. Le SGBDOO est adapté aux applications avec des structures de données complexes et des paradigmes de programmation orientée objet.
3. **SGBD Hiérarchique :** Le SGBD hiérarchique organise les données en une structure arborescente avec des relations parent-enfant. Il représente les données de manière hiérarchique, où chaque enregistrement a un parent et plusieurs enfants. Les exemples incluent IBM Information Management System (IMS) et le Registre Windows. Le SGBD hiérarchique est adapté aux applications avec des structures de données hiérarchiques, telles que les systèmes de fichiers et les organigrammes.
4. **SGBD NoSQL (Not Only SQL) :** Le SGBD NoSQL englobe divers systèmes de bases de données non relationnels conçus pour gérer de grands volumes de données non structurées ou semi-structurées. Il offre une conception de schéma flexible, une évolutivité horizontale et une haute disponibilité. Les exemples incluent MongoDB, Cassandra et Redis. Les bases de données NoSQL sont adaptées à l'analytique en temps réel, à la gestion de contenu et au stockage de données distribué.

1.2.3 Fonctions d'un SGBD

— **Définition des données :** Le SGBD permet aux utilisateurs de définir la structure de la base de données, notamment la création de tables, la spécification des types de données, la définition des contraintes et l'établissement des relations entre les entités. Cette fonction garantit l'intégrité et la cohérence des données en appliquant des règles et des normes prédéfinies.

- **Manipulation des données :** Le SGBD permet aux utilisateurs de manipuler les données stockées dans la base de données par des opérations telles que l'insertion, la suppression, la modification et la récupération. Les utilisateurs peuvent exécuter des requêtes pour extraire des informations spécifiques de la base de données, effectuer des calculs et générer des rapports.
- **Stockage des données :** Le SGBD gère le stockage physique des données sur des dispositifs de stockage tels que des disques durs ou des disques SSD. Il optimise l'allocation du stockage, gère l'espace disque et assure une récupération efficace des données en organisant les données en structures de stockage telles que des tables, des index et des fichiers.
- **Récupération des données :** Le SGBD fournit des mécanismes pour récupérer des données de la base de données en fonction des requêtes et des critères des utilisateurs. Il prend en charge diverses opérations de récupération, notamment des requêtes simples et complexes, le tri, le filtrage et l'agrégation, pour récupérer efficacement les informations pertinentes.
- **Sécurité :** Le SGBD met en œuvre des mécanismes de sécurité pour protéger les données contre les accès non autorisés, la manipulation et la divulgation. Il prend en charge des mécanismes d'authentification, d'autorisation, de chiffrement et de contrôle d'accès pour garantir la confidentialité, l'intégrité et la disponibilité des données.

1.3 Introduction au SQL

1.3.1 Définition du SQL

Le Langage de Requête Structuré (SQL) est un langage de programmation standard utilisé pour gérer et manipuler les bases de données relationnelles. Il a été développé pour la première fois par IBM dans les années 1970 et est depuis devenu le langage incontournable pour interagir avec les bases de données. SQL fournit un moyen puissant et standardisé de définir, d'accéder et de manipuler les données stockées dans un système de gestion de base de données relationnelle.

1.3.2 Rôle du SQL dans la Manipulation et l'Interrogation des Données

SQL est essentiel pour extraire des insights précieux des bases de données par le biais d'interrogations et d'analyses. Il permet aux utilisateurs de récupérer

et de manipuler efficacement de grands volumes de données, d'effectuer des opérations complexes sur les données et de générer des rapports et des visualisations pertinents.
La flexibilité et la puissance du SQL en font un outil fondamental pour la gestion et l'analyse des données dans diverses industries et applications. SQL permet aux utilisateurs de :

— **Récupérer des données :** À l'aide d'instructions SELECT, qui permettent aux utilisateurs d'extraire des informations spécifiques d'une ou plusieurs tables de la base de données.

— **Manipuler des données :** À l'aide d'instructions INSERT, UPDATE et DELETE, permettant aux utilisateurs d'ajouter, de modifier ou de supprimer des enregistrements de la base de données.

— **Modifier le schéma de la base de données :** À l'aide d'instructions CREATE, ALTER et DROP, permettant aux utilisateurs de définir ou de modifier la structure des tables et d'autres objets de la base de données.

— **Contrôler l'accès aux données :** À l'aide d'instructions GRANT et REVOKE, permettant aux utilisateurs d'accorder ou de révoquer des autorisations d'accès ou de manipulation des données.

1.3.3 SQL et les Différents Systèmes de Gestion de Bases de Données

MySQL :

C'est un système de gestion de base de données relationnelle (SGBDR) open-source connu pour sa rapidité, sa fiabilité et sa facilité d'utilisation. Il est largement utilisé dans le développement web, le commerce électronique et d'autres applications nécessitant une solution de base de données robuste et évolutive. MySQL prend en charge diverses fonctionnalités, notamment :

1. **Types de données :** MySQL prend en charge une large gamme de types de données, notamment les entiers, les chaînes de caractères, les dates et les horodatages.
2. **Indexation :** MySQL permet aux utilisateurs de créer des index sur des colonnes pour améliorer les performances des requêtes et la récupération des données.

3. **Transactions :** MySQL prend en charge les opérations transactionnelles, garantissant l'intégrité et la cohérence des données dans les environnements multi-utilisateurs.
4. **Réplication :** MySQL prend en charge la réplication, permettant aux utilisateurs de créer des répliques de bases de données pour l'évolutivité et la haute disponibilité.

Oracle :

Oracle Database est un système de gestion de base de données relationnelle (SGBDR) propriétaire développé par Oracle Corporation. Il est connu pour son évolutivité, sa haute disponibilité et ses fonctionnalités avancées pour les applications d'entreprise. Oracle Database prend en charge diverses fonctionnalités, notamment :

1. **Évolutivité :** Oracle Database est conçu pour gérer de grands volumes de données et d'utilisateurs, ce qui le rend adapté aux applications à l'échelle de l'entreprise.
2. **Haute disponibilité :** Oracle Database fournit des fonctionnalités intégrées pour la réplication des données, le basculement et la récupération, garantissant une disponibilité et une fiabilité continues.
3. **Options de sécurité avancées :** Oracle Database offre des fonctionnalités de sécurité avancées, notamment le chiffrement, les contrôles d'accès et l'audit, pour protéger les données sensibles.

Oracle Database est largement utilisé dans des secteurs tels que la finance, les télécommunications et la santé, où la fiabilité, l'évolutivité et la sécurité sont des exigences critiques. Des entreprises telles qu'Amazon, Walmart et Bank of America s'appuient sur Oracle Database pour leurs applications critiques.

PostgreSQL :

PostgreSQL est un système de gestion de base de données objet-relationnel (SGBDOR) open-source connu pour son extensibilité, sa conformité aux normes et ses fonctionnalités avancées. Il prend en charge diverses fonctionnalités, notamment :

1. **Prise en charge du JSON :** PostgreSQL prend en charge nativement le stockage et l'interrogation de données JSON, ce qui le rend adapté aux applications avec des données semi-structurées.
2. **Recherche en texte intégral :** PostgreSQL fournit une prise en charge intégrée de la recherche en texte intégral, permettant aux utilisateurs d'effectuer des recherches textuelles avancées sur de grands volumes de données textuelles.

3. **Types de données avancés :** PostgreSQL offre un riche ensemble de types de données, notamment les tableaux, les types géométriques et les types définis par l'utilisateur, permettant aux utilisateurs de modéliser des structures de données complexes.

PostgreSQL est couramment utilisé dans des secteurs tels que la santé, la recherche et le gouvernement, où l'intégrité des données, la flexibilité et l'évolutivité sont primordiales. Des entreprises telles qu'Instagram, Spotify et GitHub s'appuient sur PostgreSQL pour gérer efficacement leurs données.[2]

1.4 Bases de Données NoSQL : Définitions et Différentes Implémentations

Une base de données NoSQL est un type de base de données qui n'utilise pas les relations tabulaires traditionnelles que l'on trouve dans les bases de données relationnelles. Au lieu de cela, les bases de données NoSQL utilisent une variété de modèles de données pour stocker et récupérer des données, tels que les bases de données orientées document, clé-valeur, famille de colonnes et graphe.

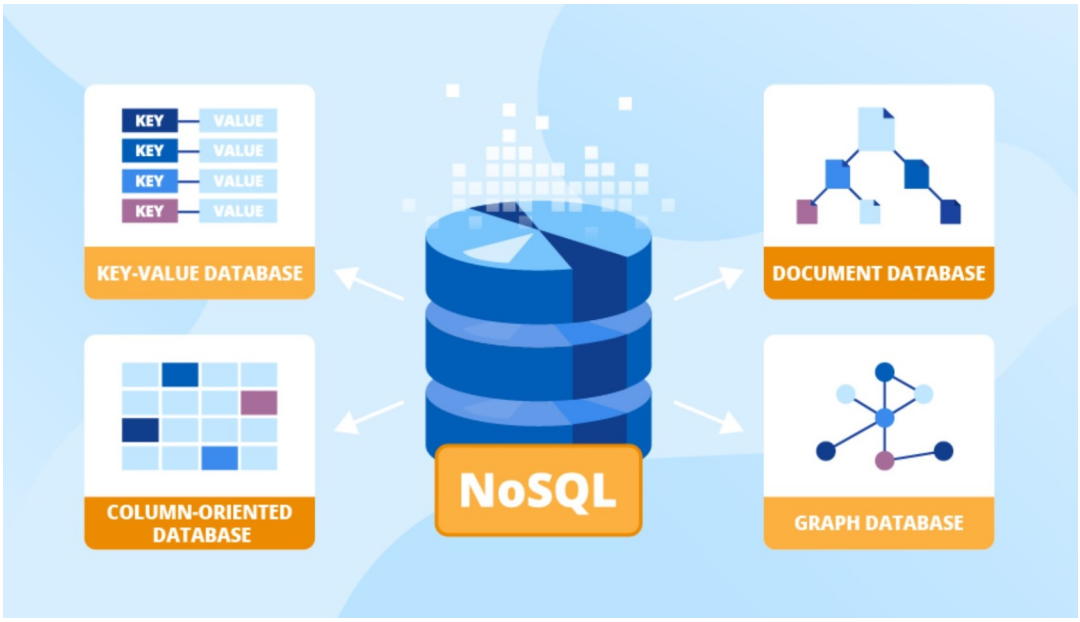

Figure 1.1 – NoSQL.

L'un des principaux avantages des bases de données NoSQL est leur capacité à évoluer horizontalement, ce qui signifie qu'elles peuvent gérer de

grandes quantités de données et de trafic en ajoutant davantage de serveurs à un cluster. Elles sont également plus flexibles que les bases de données relationnelles, car elles permettent le stockage de données non structurées ou semi-structurées, telles que des documents JSON ou XML.

1.4.1 Exemple de Base de Données NoSQL

Voici un exemple de base de données NoSQL utilisant MongoDB : Supposons que nous voulions stocker des informations sur des livres dans une base de données MongoDB. Nous pourrions créer une base de données appelée « **library** » et une collection appelée « **books** ». Chaque document de la collection « **books** » représenterait un seul livre et pourrait contenir les champs suivants :

```
{
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "published_year": 1925,
    "publisher": "Scribner",
    "genres": ["Classic", "Fiction"],
    "rating": 4.5,
    "reviews": [
        {
            "user": "john_doe",
            "rating": 4,
            "comment": "Great book, highly recommended!"
        },
        {
            "user": "jane_smith",
            "rating": 5,
            "comment": "One of the best books I've ever read."
        }
    ]
}
```

FIGURE 1.2 – Exemple de code NoSQL.

Dans cet exemple, chaque **document livre** possède un *titre*, un *auteur*, une *année de publication*, un *éditeur*, des *genres*, une *note* et des *avis*. Le champ « **genres** » est un tableau de chaînes de caractères, tandis que le champ « **reviews** » est un tableau de sous-documents, chacun contenant des informations sur un seul avis.
Avec cette structure orientée document, nous pouvons facilement récupérer tous les livres d'un certain **auteur**, filtrer les livres par **genre**, ou rechercher des livres avec une certaine **note**. Nous pouvons également ajouter ou

supprimer des champs selon les besoins sans nous soucier du maintien d'un schéma strict.

1.4.2 Avantages des Bases de Données NoSQL

Les bases de données NoSQL offrent plusieurs avantages par rapport aux bases de données relationnelles traditionnelles, notamment :

1. **Évolutivité :** Les bases de données NoSQL sont conçues pour évoluer horizontalement, ce qui signifie qu'elles peuvent gérer de grandes quantités de données et de trafic en ajoutant davantage de serveurs à un cluster. Cela les rend bien adaptées à la gestion d'applications à grande échelle nécessitant de traiter des volumes élevés de données.
2. **Flexibilité :** Les bases de données NoSQL peuvent gérer différents types de données et ne nécessitent pas de schéma fixe. Cela signifie qu'elles sont bien adaptées à la gestion de données non structurées ou semi-structurées, telles que des documents JSON ou XML.
3. **Haute performance :** Les bases de données NoSQL sont conçues pour optimiser les performances en lecture et en écriture, ce qui les rend idéales pour une utilisation dans des applications nécessitant un accès aux données à faible latence.
4. **Rentabilité :** Les bases de données NoSQL sont souvent open-source, ce qui signifie qu'elles peuvent être utilisées gratuitement ou à faible coût. De plus, comme elles sont conçues pour évoluer horizontalement, elles peuvent souvent fonctionner sur du matériel standard, réduisant ainsi davantage les coûts.
5. **Disponibilité et tolérance aux pannes :** Les bases de données NoSQL sont souvent conçues pour être hautement disponibles et tolérantes aux pannes, ce qui signifie qu'elles peuvent continuer à fonctionner même si des nœuds individuels tombent en panne. Cela les rend idéales pour une utilisation dans des systèmes distribués nécessitant des niveaux élevés de disponibilité.

1.4.3 Inconvénients des Bases de Données NoSQL

Bien que les bases de données NoSQL offrent de nombreux avantages, il existe également certains inconvénients à prendre en compte lors du choix d'une base de données NoSQL. Ces inconvénients incluent :

1. **Manque de conformité ACID :** Les bases de données NoSQL sacrifient souvent la conformité ACID (Atomicité, Cohérence, Isolation, Durabilité) pour atteindre une haute évolutivité et performance. Cela peut

les rendre moins adaptées aux applications nécessitant une cohérence stricte des données ou une intégrité transactionnelle.

2. **Fonctionnalité de requête limitée :** Les bases de données NoSQL manquent souvent de la sophistication des fonctionnalités de requête fournies par les bases de données relationnelles basées sur SQL. Cela peut rendre plus difficile l'exécution de requêtes de données complexes ou de jointures.
3. **Outils et support communautaire limités :** Les bases de données NoSQL ont souvent moins d'outils de développement et moins de support communautaire que les bases de données basées sur SQL. Cela peut rendre plus difficile la recherche d'aide ou l'intégration de la base de données avec d'autres technologies.
4. **Complexité :** Les bases de données NoSQL peuvent être plus complexes à concevoir et à gérer que les bases de données basées sur SQL, surtout si vous travaillez avec des bases de données distribuées ou des modèles de données complexes.
5. **Courbe d'apprentissage :** Les bases de données NoSQL nécessitent souvent l'apprentissage d'un nouvel ensemble de compétences et de langages de requête, ce qui peut être difficile pour les développeurs habitués à travailler avec des bases de données basées sur SQL.

1.4.4 Bases de Données NoSQL Populaires

Les bases de données NoSQL sont couramment utilisées dans les applications web et mobiles modernes, où les données sont souvent non structurées et où l'application doit gérer de grands volumes de trafic et de contenu généré par les utilisateurs. Parmi les exemples de bases de données NoSQL populaires, on trouve MongoDB, Cassandra, Redis et Neo4j.

MongoDB :

MongoDB est un système de base de données NoSQL orienté document open-source populaire, publié pour la première fois en 2009. Il est conçu pour être hautement évolutif, flexible et performant, ce qui le rend bien adapté à la gestion de grands volumes de données et de trafic. Parmi ses principales fonctionnalités, on trouve :

1. **Modèle de données orienté document :** MongoDB stocke les données dans des documents flexibles de type JSON, permettant une modélisation des données plus dynamique et expressive par rapport aux bases de données relationnelles traditionnelles.

2. **Haute disponibilité et évolutivité :** MongoDB prend en charge le basculement automatique et l'équilibrage de charge, lui permettant de gérer de grands volumes de données et de trafic avec une haute disponibilité.
3. **Langage de requête riche :** MongoDB prend en charge un puissant langage de requête qui inclut la prise en charge d'une large gamme d'opérations de requête et de pipelines d'agrégation.
4. **Indexation :** MongoDB fournit de nombreuses options d'indexation, vous permettant de créer des index sur n'importe quel champ d'une collection, y compris des index prenant en charge la recherche de texte et les requêtes géospatiales.
5. **Partitionnement intégré :** MongoDB prend en charge le partitionnement automatique, vous permettant de faire évoluer votre base de données horizontalement sur plusieurs serveurs ou clusters.
6. **Communauté open-source :** MongoDB possède une grande communauté open-source active qui fournit un support continu, une documentation et des outils.

Dans l'ensemble, MongoDB est un choix populaire pour de nombreux types d'applications, notamment le commerce électronique, les systèmes de gestion de contenu et l'analyse de données massives. Son modèle de données flexible, son puissant langage de requête et ses capacités d'évolutivité intégrées en font un bon choix pour une large gamme de cas d'utilisation.

Cassandra :

Apache Cassandra est un système de base de données distribuée NoSQL open-source populaire, publié pour la première fois en 2008. Il a été développé à l'origine par Facebook et est maintenant maintenu par la Apache Software Foundation.
Cassandra est conçu pour être hautement évolutif, tolérant aux pannes et performant, ce qui le rend bien adapté à la gestion de grands volumes de données et de trafic. Parmi ses principales fonctionnalités, on trouve :

1. **Architecture distribuée :** Cassandra est conçu pour fonctionner sur plusieurs serveurs, chaque nœud du cluster étant égal et indépendant. Cela permet une évolutivité et une tolérance aux pannes aisées.
2. **Aucun point de défaillance unique :** Cassandra est conçu pour être hautement tolérant aux pannes, sans point de défaillance unique dans le système. Les données sont répliquées sur plusieurs nœuds pour garantir leur disponibilité permanente.

3. **Cohérence configurable :** Cassandra fournit une cohérence configurable, vous permettant de choisir le niveau de cohérence dont vous avez besoin pour vos données.
4. **Modèle de données en famille de colonnes :** Cassandra stocke les données dans un modèle de données en famille de colonnes, ce qui permet une modélisation flexible des données et prend en charge les données semi-structurées et non structurées.
5. **CQL :** Cassandra fournit un puissant langage de requête appelé CQL (Cassandra Query Language) qui est similaire au SQL, ce qui le rend facile à apprendre et à utiliser pour les développeurs.
6. **Évolutivité linéaire :** Cassandra est conçu pour évoluer de manière linéaire, lui permettant de gérer de grands volumes de données et de trafic sans sacrifier les performances.

Dans l'ensemble, Cassandra est un choix populaire pour de nombreux types d'applications, notamment celles qui nécessitent une haute évolutivité et une tolérance aux pannes, comme le commerce électronique, les réseaux sociaux et l'analyse de données massives. Son architecture distribuée, sa cohérence configurable et son puissant langage de requête en font un bon choix pour une large gamme de cas d'utilisation.

Redis :

Redis est un magasin de structures de données en mémoire open-source populaire, publié pour la première fois en 2009. Il est conçu pour être hautement performant, avec une faible latence et un débit élevé, ce qui le rend bien adapté à la gestion de données à haute vitesse et d'applications en temps réel. Parmi ses principales fonctionnalités, on trouve :

1. **Magasin de données en mémoire :** Redis stocke les données en mémoire, ce qui permet des opérations de lecture et d'écriture extrêmement rapides.
2. **Structures de données :** Redis prend en charge une large gamme de structures de données, notamment les chaînes de caractères, les hachages, les listes, les ensembles et les ensembles triés. Cela permet une modélisation des données plus expressive et flexible par rapport aux magasins clé-valeur traditionnels.
3. **Messagerie pub/sub :** Redis fournit un système de messagerie pub/sub, permettant à plusieurs clients de s'abonner à des canaux et de recevoir des mises à jour en temps réel lorsque de nouvelles données sont publiées.

4. **Transactions :** Redis prend en charge les transactions, permettant l'exécution atomique de plusieurs commandes.
5. **Scripts Lua :** Redis fournit un moteur de scripts Lua, vous permettant d'écrire des scripts personnalisés pour effectuer des opérations complexes.
6. **Persistance :** Redis prend en charge la persistance, permettant l'écriture des données sur disque à des fins de durabilité et de sauvegarde.

Dans l'ensemble, Redis est un choix populaire pour de nombreux types d'applications, notamment l'analyse en temps réel, les jeux et les applications de messagerie instantanée. Son magasin de données en mémoire et ses structures de données flexibles en font un bon choix pour les applications en temps réel à haute vitesse.

Neo4j :

Neo4j est un système de gestion de base de données graphe open-source populaire, publié pour la première fois en 2007. Il est conçu pour stocker et traiter des données hautement interconnectées, ce qui le rend bien adapté aux cas d'utilisation tels que les réseaux sociaux, les moteurs de recommandation et les graphes de connaissances. Parmi ses principales fonctionnalités, on trouve :

1. **Modèle de données graphe :** Neo4j stocke les données dans un modèle de données graphe, ce qui permet de modéliser et d'interroger des relations complexes entre entités.
2. **Conformité ACID :** Neo4j est conforme ACID, ce qui signifie qu'il garantit la cohérence et la fiabilité des données.
3. **Langage de requête Cypher :** Neo4j fournit un puissant langage de requête appelé Cypher, spécifiquement conçu pour l'interrogation de données graphe.
4. **Haute disponibilité et évolutivité :** Neo4j prend en charge le clustering et le basculement automatique, lui permettant de gérer de grands volumes de données et de trafic avec une haute disponibilité.
5. **Données spatiales :** Neo4j prend en charge les données spatiales, vous permettant de stocker et d'interroger des données géospatiales.
6. **Intégrations :** Neo4j s'intègre avec une large gamme d'outils et de frameworks, notamment Python, Java et GraphQL.

Dans l'ensemble, Neo4j est un choix populaire pour de nombreux types d'applications, notamment les réseaux sociaux, les moteurs de recommandation et les graphes de connaissances. Son modèle de données graphe, son

puissant langage de requête et sa prise en charge des données spatiales en font un bon choix pour les cas d'utilisation nécessitant la modélisation de relations complexes entre entités.

En conclusion, la mise en œuvre d'une base de données NoSQL nécessite une réflexion approfondie sur vos besoins spécifiques, ainsi qu'une solide compréhension du type de base de données NoSQL que vous avez choisi d'utiliser.

Chapitre 2

Exploration de l'Architecture Spark

2.1 Vue d'ensemble des Architectures Big Data

L'avènement du Big Data a apporté une multitude de défis et d'opportunités pour les entreprises et les organisations.
Les systèmes de traitement de données traditionnels n'étaient pas équipés pour gérer le volume massif, la vélocité et la variété des données générées à l'ère moderne. Cela a conduit à l'émergence de nouvelles technologies et de nouveaux cadres, tels qu'Apache Spark, spécifiquement conçus pour répondre aux complexités du traitement Big Data.
En offrant un traitement plus rapide, un développement simplifié, une évolutivité améliorée et une rentabilité accrue, Spark est devenu un pilier des pipelines Big Data modernes dans de nombreux secteurs d'activité.

2.1.1 MapReduce : Le Modèle Fondateur du Traitement Distribué à Grande Échelle

MapReduce est un modèle de programmation et une implémentation associée pour le traitement et la génération de grands ensembles de données avec un algorithme parallèle et distribué sur le cluster. Il se compose de deux phases principales :

— **Map :** Les données d'entrée sont décomposées en petits morceaux, et chaque morceau est traité indépendamment par une fonction de mapper.

— **Reduce :** Les sorties des fonctions de mapper sont combinées en une sortie finale par une fonction de reducer.

MapReduce n'est pas bien adapté aux tâches nécessitant plusieurs passes sur les données, telles que les algorithmes itératifs ou l'analyse de données interactive [1]. De plus, il peut être inefficace pour les petits ensembles de données en raison de sa surcharge, et il exige que les données soient stockées dans un format spécifique, ce qui peut ajouter de la complexité au pipeline de traitement des données.

2.1.2 Hadoop : L'Écosystème Open-Source pour le Big Data

Hadoop est un framework logiciel open-source pour le stockage distribué et le traitement de grands ensembles de données. Il se compose de quatre composants principaux :

1. **Hadoop Distributed File System (HDFS) :** Un système de fichiers distribué qui stocke les données sur plusieurs nœuds dans un cluster.
2. **YARN :** Un gestionnaire de ressources qui planifie et gère les tâches sur un cluster.
3. **MapReduce :** Implémentation en tant que modèle de programmation pour le traitement de grands ensembles de données en parallèle sur un cluster Hadoop. Il est particulièrement adapté aux tâches pouvant être décomposées en unités de travail indépendantes, telles que le traitement de journaux, le filtrage de données, l'agrégation de données et les transformations de données de base.
4. **Hadoop Common :** Un ensemble d'utilitaires et de bibliothèques communs à tous les composants Hadoop.

La plateforme évolutive et rentable de Hadoop pour le stockage et le traitement de grands ensembles de données a joué un rôle déterminant dans l'essor du Big Data. Elle a permis à un large éventail d'organisations, notamment des entreprises, des gouvernements et des institutions de recherche, d'analyser des ensembles de données massifs et d'en extraire des informations précieuses à diverses fins.

2.1.3 Spark : Une Nouvelle Ère pour un Traitement Unifié et Performant

Spark est un moteur d'analyse unifié pour le traitement de données à grande échelle. Il peut être utilisé pour le traitement par lots, le streaming, l'apprentissage automatique et le traitement de graphes. Spark est plus rapide que MapReduce [3] car il utilise le calcul en mémoire et un modèle d'exécution plus efficace. Spark peut être intégré dans l'écosystème Hadoop pour fournir une plateforme unifiée de traitement Big Data.

2.2 Comprendre l'Écosystème Spark

Apache Spark est un puissant framework de calcul distribué open-source conçu pour traiter des données à grande échelle sur des clusters d'ordinateurs. Son écosystème se compose de divers composants offrant différentes fonctionnalités pour le traitement des données, l'analytique et l'apprentissage automatique. Plongeons dans les composants essentiels de l'écosystème Spark.

2.2.1 Spark Core

Toutes les fonctionnalités proposées par Apache Spark sont construites sur Spark Core. Il offre de la rapidité grâce à des capacités de calcul en mémoire. Spark Core constitue ainsi le fondement du traitement parallèle et distribué de grands ensembles de données.

Les principales caractéristiques d'Apache Spark Core sont :

— Il est responsable des fonctionnalités essentielles d'entrée/sortie.

— Il joue un rôle significatif dans la programmation et l'observation du cluster Spark.

— La répartition des tâches.

— La tolérance aux pannes.

— Il surmonte les limitations de MapReduce grâce au calcul en mémoire.

— Il intègre une collection spéciale appelée RDD (Resilient Distributed Dataset).

2.2.2 Spark SQL

Le composant Spark SQL est un framework distribué pour le traitement de données structurées. Grâce à Spark SQL, Spark obtient davantage d'in-

formations sur la structure des données et le calcul. Avec ces informations, Spark peut effectuer des optimisations supplémentaires. Il utilise le même moteur d'exécution lors du calcul d'une sortie. Il ne dépend pas d'une API ou d'un langage pour exprimer le calcul.

Spark SQL travaille sur l'accès aux informations structurées et semi-structurées. Il permet également de puissantes applications analytiques et interactives sur des données en streaming et des données historiques.

Spark SQL est un module Spark pour le traitement de données structurées. Il agit donc comme un moteur de requêtes SQL distribué.

Les fonctionnalités de Spark SQL incluent :

— Un optimiseur basé sur les coûts.

— Une tolérance aux pannes en cours de requête, réalisée en s'adaptant à des milliers de nœuds et des requêtes de plusieurs heures grâce au moteur Spark.

— Les DataFrames et SQL fournissent une manière commune d'accéder à diverses sources de données, notamment Hive, Avro, Parquet, ORC, JSON et JDBC.

— Une compatibilité totale avec les données Hive existantes.

— La possibilité d'intégrer des données structurées dans des programmes Spark, en utilisant soit SQL, soit une API DataFrame familière.

2.2.3 Spark MLlib

MLlib est une puissante bibliothèque d'apprentissage automatique au sein d'Apache Spark, conçue pour l'évolutivité et la rapidité. Elle offre une large gamme d'algorithmes de haute qualité pour des tâches telles que le clustering, la régression, la classification et le filtrage collaboratif. Initialement basée sur les RDD (Resilient Distributed Datasets), MLlib est passée à une API basée sur les DataFrames dans Spark Version 2.0, offrant une approche plus conviviale et tirant parti du moteur d'exécution optimisé de Spark.

MLlib simplifie les flux de travail d'apprentissage automatique en fournissant des API Java, Scala et Python pour des implémentations distribuées d'algorithmes d'apprentissage courants tels que les modèles linéaires, Naive Bayes, les ensembles d'arbres de décision et le clustering k-means. Elle comprend également des utilitaires pour l'optimisation convexe, l'algèbre linéaire distribuée, l'analyse statistique et l'extraction de caractéristiques.

De plus, MLlib bénéficie de l'écosystème Spark plus large. L'intégration de Spark SQL prend en charge le prétraitement des données et le traitement

de données structurées, tandis que Spark Streaming permet le développement d'algorithmes d'apprentissage en ligne pour le traitement de flux de données en direct. Le package spark.ml de MLlib simplifie davantage les pipelines d'apprentissage multi-étapes avec des API de haut niveau, tirant parti des vastes capacités de Spark pour transformer et traiter efficacement des ensembles de données à grande échelle.

2.2.4 Spark Streaming

Spark Streaming est une extension de l'API Spark principale qui permet le traitement de flux de données en direct de manière évolutive et tolérante aux pannes.

L'une des caractéristiques clés de Spark Streaming est le micro-batching, où le flux de données entrant est divisé en petits lots pour le traitement. Cette approche garantit la tolérance aux pannes et permet à Spark de gérer efficacement les données de streaming en temps réel.

Spark Streaming fonctionne en trois phases :

— **Collecte :** Les données sont collectées depuis diverses sources, notamment des sources basiques comme les systèmes de fichiers et les connexions socket, ainsi que des sources avancées comme Kafka, Flume et Kinesis.

— **Traitement :** Les données collectées sont traitées à l'aide d'algorithmes complexes exprimés avec des fonctions de haut niveau telles que map, reduce, join et les opérations de fenêtrage.

— **Stockage des données :** Les données traitées sont ensuite stockées ou transmises vers des systèmes de fichiers, des bases de données ou des tableaux de bord en direct.

Spark Streaming fournit également une abstraction de haut niveau appelée flux discrétisé, ou DStream. Un DStream représente un flux continu de données et peut être créé soit à partir de sources externes comme Kafka et Flume, soit via des opérations de haut niveau sur des DStreams existants. En interne, un DStream est une séquence de Resilient Distributed Datasets (RDD), qui sont la structure de données fondamentale dans Spark.

2.2.5 Graph X

GraphX dans Spark est une API essentielle conçue pour le traitement de graphes et l'exécution parallèle au sein du framework Spark. Il sert de puissant moteur d'analyse de graphes en réseau et de magasin de données, permettant diverses opérations telles que le clustering, la classification, le

parcours, la recherche et la recherche de chemins sur des graphes.

L'une des caractéristiques clés de GraphX est son extension du RDD (Resilient Distributed Dataset) de Spark en introduisant une nouvelle abstraction de Graphe. Cette abstraction représente un multigraphe dirigé, où des propriétés peuvent être attachées à chaque sommet et à chaque arête. Cette structure offre un moyen flexible et efficace de modéliser et d'analyser des relations complexes au sein des graphes.

GraphX propose également des optimisations pour la gestion des types de données primitifs lors de la représentation des sommets et des arêtes. Cette optimisation améliore les performances des calculs sur les graphes. De plus, GraphX prend en charge les opérateurs de graphes fondamentaux tels que subgraph, joinVertices et aggregateMessages. Il inclut également une variante optimisée de l'API Pregel, un modèle de programmation largement utilisé pour le traitement de graphes. Ces fonctionnalités font de GraphX un outil puissant pour l'analytique et le traitement de graphes au sein de l'écosystème Spark.

2.2.6 SparkR

SparkR a fait ses débuts avec la version Apache Spark 1.4, en introduisant le SparkR DataFrame comme élément central. En tant que structure de données essentielle en R pour le traitement des données, les DataFrames trouvent des équivalents dans d'autres langages comme Python avec des bibliothèques telles que Pandas. L'objectif principal de SparkR était de fusionner la facilité d'utilisation de R avec l'évolutivité de Spark, en présentant un package R servant d'interface légère pour utiliser Apache Spark directement depuis R.

SparkR offre plusieurs avantages :

— **API de Sources de Données :** En tirant parti de l'API de sources de données de Spark SQL, SparkR lit de manière transparente les données provenant de sources diverses telles que les tables Hive, les fichiers JSON et les fichiers Parquet.

— **Optimisations des DataFrames :** Les DataFrames SparkR bénéficient des optimisations apportées au moteur de calcul, notamment la génération de code améliorée et la gestion de la mémoire.

— **Évolutivité :** Les opérations exécutées sur les DataFrames SparkR sont distribuées sur tous les cœurs et machines disponibles dans le

cluster Spark. Cette évolutivité permet à SparkR de gérer de vastes ensembles de données, s'adaptant de téraoctets de données à des clusters composés de milliers de machines.

En résumé, SparkR permet aux utilisateurs R d'exploiter les capacités d'évolutivité et de performance d'Apache Spark pour des tâches de traitement de données à grande échelle, intégrant de manière transparente les capacités de manipulation de données et graphiques de R avec la puissance de calcul distribué de Spark.

Apache Spark améliore les outils Big Data existants pour l'analyse sans réinventer la roue. Sa popularité découle des solides composants de l'écosystème Apache Spark, ce qui en fait un choix privilégié par rapport aux autres frameworks Big Data. Spark sert de plateforme polyvalente pour diverses tâches de traitement des données, notamment l'analytique de données en temps réel, le traitement de données structurées et le traitement de graphes.

Avec sa dynamique croissante, Apache Spark émerge comme une alternative prometteuse pour prendre en charge les requêtes ad-hoc et le traitement itératif, remplaçant les approches MapReduce traditionnelles. Il facilite l'exécution interactive de code via les REPL (Read-Eval-Print Loop) Python et Scala, offrant la flexibilité d'écrire et de compiler des applications en Scala, Java ou d'autres langages pris en charge.

2.3 Différences entre le Big Data et les Architectures Traditionnelles

L'architecture Big Data et l'architecture traditionnelle sont deux approches différentes de la gestion et du traitement des données [4]. L'architecture Big Data est conçue pour gérer les grands volumes de données générés par les applications modernes, tandis que l'architecture traditionnelle est conçue pour des ensembles de données plus petits [5].

2.3.1 Architecture Big Data

Il s'agit d'une nouvelle approche de la gestion et du traitement des données, conçue pour gérer les grands volumes de données générés par les applications modernes. L'architecture Big Data est généralement basée sur un système de fichiers distribué, tel que Hadoop, et un framework de traitement distribué, tel que Spark.

2.3.2 Avantages de l'architecture Big Data :

— **Prise de décision :** L'architecture Big Data peut aider les entreprises à prendre de meilleures décisions en leur fournissant des informations sur leurs données. [6]

— **Efficacité accrue :** L'architecture Big Data peut aider les entreprises à améliorer leur efficacité en automatisant les processus et en réduisant le temps nécessaire à l'analyse des données.

— **Développement de nouveaux produits et services :** L'architecture Big Data peut aider les entreprises à développer de nouveaux produits et services en leur fournissant des informations sur les besoins des clients.

— **Réduction des coûts :** L'architecture Big Data peut aider les entreprises à réduire leurs coûts en améliorant l'efficacité et en réduisant le besoin de traitement manuel des données.

2.3.3 Défis de l'architecture Big Data :

— **Complexité :** L'architecture Big Data peut être complexe à mettre en œuvre et à gérer.

— **Coût :** L'architecture Big Data peut être coûteuse à mettre en œuvre et à maintenir.

— **Sécurité :** L'architecture Big Data peut présenter des risques de sécurité si elle n'est pas correctement mise en œuvre.

— **Compétences :** L'architecture Big Data nécessite des compétences spécialisées qui peuvent ne pas être disponibles en interne.

2.3.4 Architecture traditionnelle

L'architecture traditionnelle est la méthode traditionnelle de stockage et de traitement des données. Elle est généralement basée sur un système de gestion de base de données relationnelle (SGBDR). Les SGBDR sont conçus pour les données structurées, c'est-à-dire les données organisées sous forme de tableau.

2.3.4.1 Avantages de l'architecture traditionnelle :

— Elle est bien comprise et constitue une technologie mature.

— Elle est relativement facile à mettre en œuvre et à gérer.

— Elle est évolutive dans une certaine mesure.

— Elle est relativement sécurisée.

2.3.4.2 Inconvénients de l'architecture traditionnelle :

— Elle n'est pas conçue pour le Big Data.
— Elle n'est pas conçue pour le traitement des données en temps réel.
— Elle peut être coûteuse à faire évoluer.

2.3.5 Principales différences entre le Big Data et l'architecture traditionnelle

— **Volume de données :** L'architecture Big Data est conçue pour gérer de grands volumes de données, tandis que les architectures traditionnelles sont conçues pour des ensembles de données plus petits.
— **Variété des données :** L'architecture Big Data est conçue pour gérer une variété de types de données, notamment les données structurées, semi-structurées et non structurées [7]. L'architecture traditionnelle est généralement conçue pour les données structurées.
— **Vélocité des données :** L'architecture Big Data est conçue pour gérer les données générées en temps réel ou quasi-réel. L'architecture traditionnelle est généralement conçue pour le traitement par lots des données. [8]
— **Évolutivité :** L'architecture Big Data est conçue pour être évolutive afin de gérer des volumes de données croissants. L'architecture traditionnelle n'est généralement pas aussi évolutive.
— **Coût :** L'architecture Big Data peut être plus coûteuse à mettre en œuvre et à maintenir que l'architecture traditionnelle.
— **Complexité :** L'architecture Big Data peut être plus complexe à mettre en œuvre et à gérer que l'architecture traditionnelle.

En conclusion, l'architecture Big Data et l'architecture traditionnelle sont deux approches différentes de la gestion et du traitement des données.

L'architecture Big Data est conçue pour gérer les grands volumes de données générés par les applications modernes, tandis que l'architecture traditionnelle est conçue pour des ensembles de données plus petits. La meilleure approche pour une organisation particulière dépendra de ses besoins et exigences spécifiques.

Chapitre 3

Applications de Spark dans des Projets Réels

3.1 Projets et Implémentations Existants utilisant Apache Spark

Nous allons parler de quelques projets.

3.1.1 Pipeline ETL de Streaming de Données Twitter en Temps Réel

Résumé

Ce projet explore l'utilisation d'Apache Spark pour l'analyse des sentiments des tweets, mettant en évidence son efficacité et sa flexibilité dans le traitement de grands volumes de données en temps réel. Apache Spark sert de pilier central dans la collecte, le traitement et l'analyse des données, illustrant ainsi un cas d'utilisation puissant de Spark dans le domaine de l'analyse des données des réseaux sociaux.

Introduction

L'analyse des sentiments sur Twitter fournit des informations précieuses pour diverses applications, allant de la surveillance de la réputation des marques à la prise de décisions marketing basées sur les données. Ce projet utilise Apache Spark pour analyser les sentiments exprimés dans les tweets en temps réel, démontrant ainsi la puissance de Spark dans les applications de traitement de flux de données.

Contexte et Justification

Apache Spark a été choisi pour ce projet en raison de sa rapidité, de sa capacité à gérer les flux de données en temps réel et de son intégration facile avec d'autres technologies telles qu'Apache Kafka et Delta Lake. Ce rapport justifie le choix de Spark par ses capacités de traitement en mémoire et ses outils intégrés pour l'analyse des données.

Méthodologie

Premièrement, *ETL* signifie Extraire, Transformer, Charger (Extract, Transform, Load). Il s'agit d'un processus courant en ingénierie des données où les données sont extraites d'une source, transformées dans un format utilisable, puis chargées dans une destination telle qu'une base de données.

- **Composants utilisés**
  - **Tweepy** Une bibliothèque Python pour accéder à l'API de Twitter (pour obtenir les données Twitter).
  - **Apache Kafka** Une plateforme logicielle de streaming de données. Elle est utilisée pour gérer et organiser les données Twitter entrantes.
  - **Apache Spark** Un puissant moteur et framework d'analytique. Il est utilisé pour le traitement et l'analyse des données Twitter.
  - **Delta Lake** Une couche de stockage pour les lacs de données, utilisée pour stocker les données traitées.

Le flux de données est décrit, depuis l'extraction (API Twitter) jusqu'à la transformation (Spark) et le chargement (Delta Lake).

1. **Extraction des données :** Les données sont extraites en temps réel via l'API Twitter et dirigées vers un système de gestion de messagerie Kafka.
2. **Transformation des données avec Spark :** Le cœur de ce projet réside dans l'utilisation de Spark pour transformer et analyser les tweets. Les données provenant de Kafka sont consommées par Spark Streaming, qui les traite pour détecter et classifier les sentiments.
   - **Prétraitement :** Nettoyage et tokenisation des tweets.
   - **Pipeline ML :** Utilisation de MLlib pour créer un pipeline de traitement de texte, incluant TF-IDF et la régression logistique pour la classification des sentiments.
3. **Chargement des données :** Les données transformées sont chargées dans Delta Lake pour le stockage et une analyse ultérieure, tirant parti de la gestion transactionnelle et de l'évolutivité de Delta Lake.

Architecture du Système

Le système utilise une architecture distribuée où chaque composant joue un rôle clé dans le traitement et l'analyse des données. Kafka agit comme un hub pour les données en temps réel, Spark comme moteur de traitement et d'analyse, et Delta Lake comme couche de persistance.

Détails Techniques

— **Configuration de Spark Streaming :** Spark Streaming est configuré pour lire les données depuis le topic Kafka. Un schéma structuré est appliqué aux données pour faciliter la manipulation et le traitement.

— **Traitement des Tweets et Prédiction des Sentiments :** Les tweets sont nettoyés, tokenisés et transformés à l'aide d'un modèle TF-IDF suivi d'une régression logistique pour prédire le sentiment.

— **Enregistrement des Résultats :** Les résultats sont stockés dans Delta Lake, optimisé pour le traitement par lots et en streaming, offrant une plateforme robuste pour la gestion des données de sortie.

Analyse des Performances

Le modèle atteint une précision de 83,2% sur l'ensemble de données de test. Cette section aborde également les métriques de performance utilisées pour évaluer le modèle.

Conclusion

Ce projet d'analyse des sentiments sur Twitter a mis en évidence l'efficacité d'Apache Spark en tant que plateforme centrale pour le traitement des données en temps réel. Grâce à sa capacité à gérer de grands volumes de données rapidement et efficacement, Spark s'est révélé être un outil indispensable pour répondre aux exigences de ce projet ambitieux.

3.1.2 Utilisation d'Apache Spark pour l'Analyse des Sentiments de Tweets en Temps Réel

Résumé

Ce projet Spark vise à analyser et à comparer les performances de différentes régions dans la gestion de la pandémie de COVID-19. À l'aide de métriques et de statistiques pertinentes, l'objectif est d'identifier les régions les plus efficaces dans la limitation de la propagation du virus et la minimisation de

son impact. Le projet se concentrera également sur la surveillance en temps réel des données COVID-19, l'analyse de la popularité et du volume des discussions en ligne sur le sujet, et l'identification des hashtags les plus utilisés pour décrire la pandémie.

Introduction

La pandémie de COVID-19 a eu un impact profond sur le monde entier, bouleversant des vies et des économies. Alors que les pays ont mis en œuvre différentes stratégies et mesures pour combattre le virus, il est crucial d'évaluer l'efficacité de ces approches et d'identifier les meilleures pratiques. Ce projet Spark s'inscrit dans cet effort, en utilisant des outils d'analyse de données et de traitement du langage naturel pour fournir des informations précieuses sur la gestion de la pandémie.

Contexte et Justification

Comprendre les disparités régionales dans la gestion du COVID-19 est essentiel pour plusieurs raisons. Premièrement, cela aide à identifier les facteurs qui contribuent au succès ou à l'échec des différentes stratégies. Deuxièmement, cela peut informer les décideurs politiques et les responsables de la santé publique dans leurs efforts pour développer des interventions plus efficaces. Troisièmement, cela peut aider à orienter les ressources et les efforts de soutien vers les régions les plus touchées.

Méthodologie

Le projet Spark utilisera une combinaison de techniques d'analyse de données et de traitement du langage naturel (TLN) pour atteindre ses objectifs. La méthodologie peut être divisée en trois sous-sections principales :

1. **Collecte et préparation des données**
   - Identifier des sources fiables et reconnues de données COVID-19, telles que les organisations internationales de santé publique et les gouvernements nationaux.
   - Collecter des données sur les indicateurs clés de performance, tels que les taux d'infection, les taux de mortalité, les taux de vaccination et d'autres mesures pertinentes.
   - Nettoyer et prétraiter les données pour garantir leur qualité et leur cohérence.
   - Intégrer les données provenant de différentes sources dans un format unifié pour faciliter l'analyse.

2. **Analyse des données COVID-19**
   - Appliquer des techniques d'analyse statistique pour identifier les tendances et les modèles dans les données COVID-19.
   - Comparer les performances de différentes régions en termes d'indicateurs clés de performance.
   - Identifier les facteurs contribuant au succès ou à l'échec des différentes stratégies de gestion du COVID-19.
   - Visualiser les résultats de l'analyse pour faciliter la communication et la compréhension.
3. **Analyse des sentiments et des tendances dans les discussions en ligne**
   - Collecter des données sur les discussions en ligne concernant le COVID-19 à partir de sources telles que les réseaux sociaux, les forums en ligne et les articles de presse.
   - Appliquer des techniques de traitement du langage naturel pour analyser les sentiments et les tendances des discussions.
   - Identifier les sujets émergents, les hashtags populaires et les opinions dominantes sur le COVID-19.
   - Suivre l'évolution des perceptions et des préoccupations du public au fil du temps.

En combinant ces trois sous-sections, le projet Spark peut fournir une analyse complète et instructive de la gestion du COVID-19 à travers le monde.

Détails Techniques

Le projet Spark utilisera une variété d'outils et de technologies pour mener ses analyses. Ces outils peuvent inclure :

- Des bases de données et des entrepôts de données pour stocker et gérer les données COVID-19
- Des outils d'analyse statistique et de visualisation pour identifier les tendances et les modèles dans les données
- Des plateformes de traitement du langage naturel (TLN) pour analyser les discussions en ligne sur le COVID-19
- Des outils de fouille de données et d'apprentissage automatique pour découvrir des informations cachées dans les données

Conclusion

Ce projet Spark vise à fournir une analyse complète et instructive de la gestion du COVID-19 à travers le monde.

En identifiant les régions les plus performantes et en comprenant les facteurs qui contribuent à leur succès, le projet peut aider à améliorer les efforts mondiaux de lutte contre la pandémie.

De plus, l'analyse des discussions en ligne sur le COVID-19 peut fournir des informations précieuses sur les perceptions et les préoccupations du public, ce qui peut aider à orienter les stratégies de communication et de sensibilisation.

3.2 Configuration de l'Écosystème Spark

3.2.1 Environnement Windows

Étape 1 — Prérequis

Avant d'installer Apache PySpark, il est essentiel de s'assurer que certains prérequis sont satisfaits :

— **Java Development Kit (JDK) :** Il est nécessaire d'avoir le JDK version 8 ou supérieure installé sur le système.

— **Python :** La dernière version de Python doit également être installée.

Étape 2 — Installation du JDK

1. Téléchargez le fichier d'installation du JDK depuis le site officiel.
2. Exécutez le fichier d'installation et suivez les instructions.
3. Choisissez de spécifier un répertoire particulier pour l'installation de Java, généralement dans le dossier C :/Java/jdk.

Étape 3 — Installation de Python

1. Téléchargez Python depuis python.org
2. Exécutez le fichier d'installation et sélectionnez « Add python.exe to path »

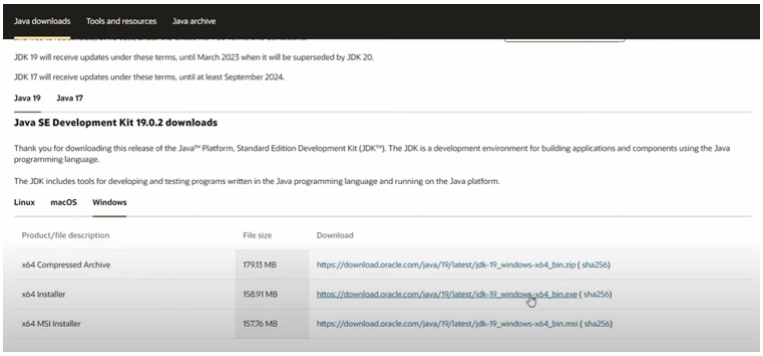

FIGURE 3.1 – Site Web du JDK

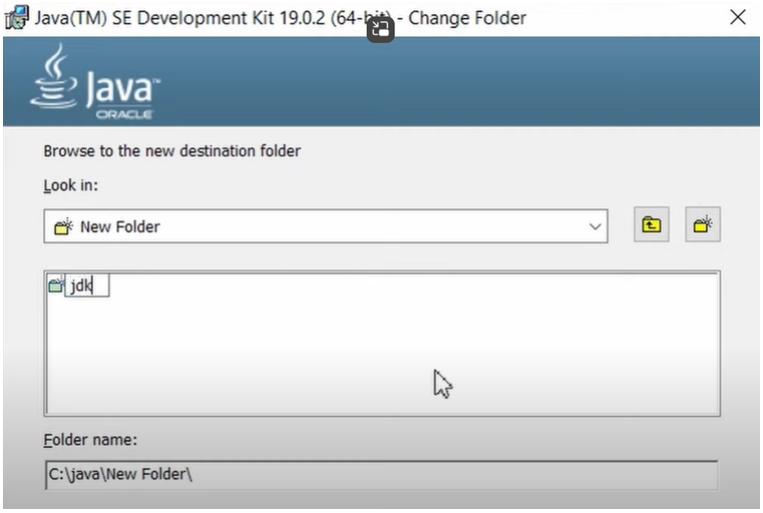

FIGURE 3.2 – Installation du JDK

Étape 4 — Téléchargement et installation d'Apache Spark

1. Rendez-vous sur le site officiel de Spark (spark.apache.org) et sélectionnez la version appropriée.
2. Téléchargez le fichier tar correspondant.
3. Extrayez le contenu du fichier tar dans un répertoire spécifique (par exemple, C :/spark).

Étape 5 — Téléchargement de WinUtils

1. Téléchargez le fichier WinUtils.exe depuis un dépôt Git.
2. Placez le fichier WinUtils.exe dans le répertoire bin du dossier Hadoop (par exemple, C :/Hadoop/bin).

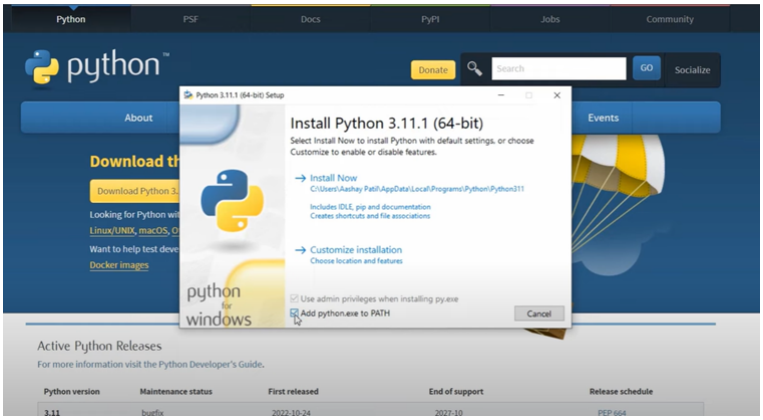

FIGURE 3.3 – python.org

FIGURE 3.4 – Site officiel d'Apache Spark

Étape 6 — Configuration des variables d'environnement

Accédez aux variables d'environnement système via les Paramètres Windows. Ajoutez les variables suivantes :

- **JAVA_HOME :** Chemin vers le répertoire d'installation de Java (par exemple, C :/Java/jdk).
- **HADOOP_HOME :** Chemin vers le répertoire d'installation de Hadoop (par exemple, C :/Hadoop).
- **SPARK_HOME :** Chemin vers le répertoire d'installation de Spark (par exemple, C :/spark).
- Chemin vers le répertoire d'installation de Python (par exemple, C :/Users/UserName/AppD
- Ajoutez également les chemins des répertoires bin correspondants pour JAVA_HOME, HADOOP_HOME et SPARK_HOME.

3.2.1.1 Étape 7 — Téléchargement de Pyspark

1. Ouvrez un terminal et lancez la commande suivante :

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

2. Puis lancez cette commande :

```
python get-pip.py
```

3. Après avoir installé *pip*, vous pouvez le vérifier en lançant cette commande :

```
python -m pip help
```

4. Installez maintenant *pyspark* à l'aide de la commande suivante :

```
pip install pyspark
```

Étape 8 — Vérification de l'installation

1. Ouvrez l'invite de commandes et vérifiez les versions de Java et Python.
2. Lancez le shell Spark en tapant « spark-shell » dans l'invite de commandes.
3. Vérifiez également que PySpark fonctionne en tapant « pyspark » dans l'invite de commandes.

Étape 9 — Visualisation de l'Interface Spark

1. Ouvrez un navigateur web et accédez à l'interface utilisateur Spark en entrant l'URL fournie dans la vidéo.
2. Explorez les différentes fonctionnalités de l'interface pour surveiller les applications Spark en cours d'exécution.

FIGURE 3.5 – Spark-shell

FIGURE 3.6 – Onglet Jobs d'Apache Spark

3.2.2 Environnement Linux

Étape 1 — Vérifier les prérequis

Vérifiez si Java est installé en utilisant :

```
java --version
```

ou :

```
which java
```

Étape 2 — Installer Java

Installez Java avec cette commande si vous utilisez une distribution basée sur Debian :

```
sudo apt-get install openjdk-17-jdk openjdk-17-jre
```

Ou celle-ci si vous utilisez Arch :

```
sudo pacman -S jdk17-openjdk openjdk17-doc jre17-openjdk
```

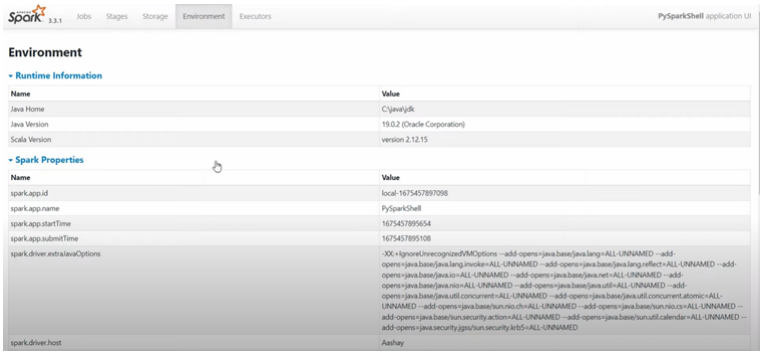

FIGURE 3.7 – Onglet Environment d'Apache Spark

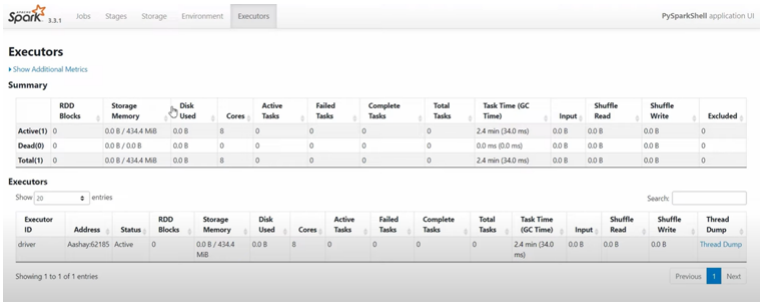

FIGURE 3.8 – Onglet Executors d'Apache Spark

Étape 2 — Configurer l'environnement pour Spark

Créez le dossier où stocker Apache Spark :

```
mkdir /opt/spark
```

Étape 3 — Installation de Spark

1. Dans le répertoire, lancez la commande suivante pour télécharger Spark :

   ```
   wget https://dlcdn.apache.org/spark/spark-3.5.1/
   spark-3.5.1-bin-hadoop3.tgz
   ```

2. Extrayez le fichier :

   ```
   tar -xvf spark_file_name
   ```

3. Supprimez le dossier tar :

```
sudo rm -r spark_file_name
```

4. Renommez le dossier extrait :

```
sudo mv spark_extracted_folder Spark
```

5. Ajoutez Spark à votre PATH en ajoutant la ligne ci-dessous à votre fichier shell (ex : .zshrc) :

```
export PATH="\$PATH:/opt/spark/Spark/bin"
```

3.2.2.1 Étape 4 — Installation de Pyspark

Lancez la commande suivante :

```
pip install pyspark
```

Si vous rencontrez des problèmes, installez-le avec la commande suivante sur Debian :

```
sudo apt get python-pyspark
```

Sous Arch, vous aurez besoin de *yay* :

```
git clone https://aur.archlinux.org/yay.git
cd yay
makepkg -si
yay -S python-pyspark
yay -S python-py4j
```

3.2.2.2 Étape 5 — Utiliser Spark

Vérifiez l'installation en exécutant cette commande :

```
spark-shell
```

3.2.3 Environnement MacOS

Étape 1 — Installer Homebrew

Pour installer *Homebrew*, exécutez la ligne ci-dessous :

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

3.2.3.1 Étape 2 — Installer Java (openjdk8)

Installez le *JDK* à l'aide de la commande ci-dessous :

```
brew install --cask homebrew/cask-versions/adoptopenjdk8
```

3.2.3.2 Étape 3 — Définir la variable d'environnement $JAVA_HOME

1. Trouvez votre $JAVA_HOME avec la commande ci-dessous :

   ```
   ls /Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/
   Contents/Home
   ```

2. Ajoutez-la à votre fichier shell (/.zshrc par exemple) :

   ```
   export JAVA_HOME=/Library/Java/JavaVirtualMachines/adoptopenjdk-
   8.jdk/Contents/Home
   ```

3. Sourcez votre fichier shell :

   ```
   source ~/.zshrc
   ```

3.2.3.3 Étape 4 — Installer Spark

1. Téléchargez la dernière version depuis https ://spark.apache.org/downloads.html
2. Décompressez le fichier :

   ```
   tar -xf spark-3.4.1-bin-hadoop3.tgz
   ```

3.2.3.4 Étape 5 — Définir la variable d'environnement $SPARK_HOME

L'emplacement de ce fichier correspond à votre $SPARK_HOME. Ajoutez cette variable d'environnement dans votre fichier /.zshrc comme vous l'avez fait pour $JAVA_HOME.

Définissez cette variable d'environnement :

```
export SPARK_HOME=~/spark-3.4.1-bin-hadoop3
```

FIGURE 3.9 – Spark Home

Étape 6 — Tester Spark

Pour lancer le shell Spark basé sur Scala, ouvrez un terminal et tapez :

```
spark-shell
```

Il est généralement recommandé de travailler avec des environnements virtuels afin d'éviter les conflits avec d'autres paquets.

Installez *py4j* :

```
pip install py4j
```

Créez un environnement virtuel dans le répertoire dans lequel vous souhaitez travailler.

```
python3 -m venv env
```

Pour activer l'environnement virtuel dans ce dossier, exécutez :

```
source env/bin/activate
```

Installez ensuite *pyspark* à l'aide de la commande ci-dessous :

```
pip install pyspark
```

J'ai procédé ainsi car j'ai obtenu une erreur indiquant que le module n'était pas trouvé ; espérons que vous ne rencontrerez pas ce problème.

Lancez Pyspark en tapant dans la ligne de commande :

```
pyspark
```

Étape 6 — Utiliser Spark

Le résultat final dans votre terminal, selon la commande utilisée, devrait ressembler à ceci :

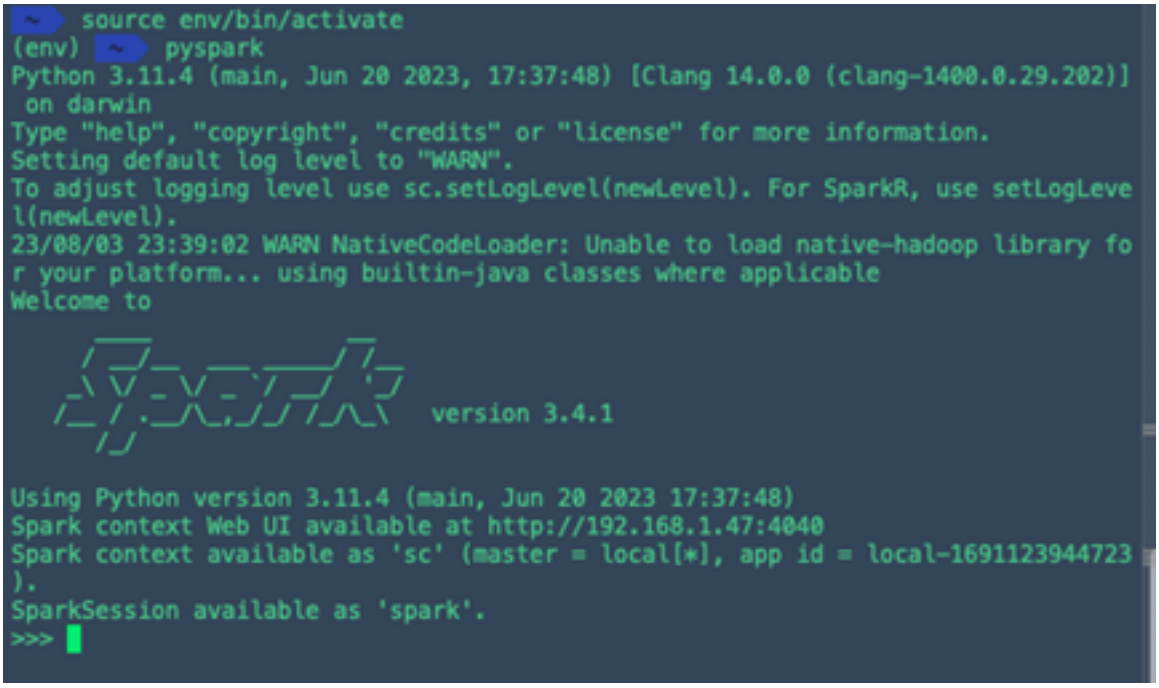

FIGURE 3.10 – Spark sur Mac

3.3 Notre Cas d'Utilisation : Exploiter Spark pour le Clustering K-means

Pour notre cas d'utilisation, nous avons choisi d'implémenter l'algorithme k-means inclus dans Spark MLlib afin de réaliser un clustering sur des données que nous avons générées, puis d'évaluer ce clustering.

Étape 1 — Importations

Nous commençons par importer simplement toutes les bibliothèques *pyspark* dont nous aurons besoin :

```
from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import ClusteringEvaluator
```

— **SparkSession :** Nous avons besoin de cette classe du sous-package *pyspark.sql* pour manipuler notre session Spark.

— **KMeans :** Depuis la *MLlib* de *pyspark*, nous importons la classe KMeans pour effectuer le clustering.

— **VectorAssembler :** Elle nous permettra d'effectuer les opérations de prétraitement appropriées pour appliquer des algorithmes d'apprentissage automatique aux données.
— **ClusteringEvaluator :** Cette classe nous permettra d'évaluer le clustering que nous avons effectué.

Étape 2 — Création de la session Spark et lecture des données

Dans cette étape, nous créons la session Spark à l'aide de la ligne ci-dessous :

```
spark = SparkSession.builder.appName("KMeansExample").getOrCreate()
```

Ensuite, nous lisons les données depuis un fichier donné :

```
data = spark.read.csv("data.csv", header=True, inferSchema=True)
```

Après cela, nous préparons les données pour qu'elles soient traitées par l'algorithme d'apprentissage automatique :

```
feature_cols = data.columns
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
data = assembler.transform(data)
```

Étape 3 — K-means

Ensuite, nous appliquons l'algorithme k-means sur les données :

```
kmeans = KMeans(k=3, seed=123)
model = kmeans.fit(data)
```

Nous pouvons afficher les clusters de chaque point :

```
data_clusterd = model.transform(data)
data_clusterd.show()
```

Étape 4 — Évaluation du Clustering

Enfin, nous pouvons évaluer notre clustering en utilisant la classe ClusteringEvaluator :

```
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(data_clusterd)

print("Silhouette score = " + str(silhouette))
```

3.4 Conclusion

Dans ce document, nous avons exploré les concepts fondamentaux du Big Data, des systèmes de bases de données ainsi que l'architecture d'Apache Spark. Nous avons montré que les approches traditionnelles de traitement des données ne sont plus adaptées face au volume et à la complexité des données actuelles, et que les frameworks distribués comme Spark offrent une solution efficace pour l'analyse à grande échelle. À travers l'étude de l'écosystème Spark, nous avons mis en évidence sa flexibilité, ses performances et sa capacité à prendre en charge plusieurs paradigmes de traitement au sein d'une même plateforme.

La partie pratique a permis de démontrer l'utilisation de Spark MLlib pour implémenter l'algorithme de clustering K-means sur des données distribuées. Ce cas d'utilisation confirme l'intérêt de Spark pour les tâches de machine learning, en particulier lorsqu'il s'agit de traiter de grands volumes de données nécessitant des performances élevées et une bonne scalabilité. Il illustre également comment Spark simplifie le développement d'applications orientées données grâce à ses abstractions de haut niveau et à son moteur d'exécution optimisé.

Dans l'ensemble, Apache Spark se distingue comme une plateforme polyvalente et performante pour le traitement et l'analyse du Big Data. Son adoption croissante dans le monde académique et industriel témoigne de son importance dans la résolution des défis liés à la gestion des données modernes. Des travaux futurs pourraient approfondir cette étude en explorant d'autres algorithmes de machine learning, des applications de streaming en temps réel, ou encore des techniques avancées de clustering et de classification.

Bibliographie

[1] Spark : Cluster Computing with Working Sets. Consulté le : 01-04-2024. URL : https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/Zaharia.pdf.

[2] Database with PostgreSQL. Consulté le : 16-04-2024. URL : https://www.oracle.com/cloud/postgresql/.

[3] Apache Spark Officially Sets a New Record in Large-Scale Sorting. Consulté le : 01-04-2024. URL : https://www.databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html.

[4] Learn Why Data Scientists and Developers Need More Than a Data Lake. Consulté le : 05-04-2024. URL : https://www.actian.com/blog/cloud-data-warehouse/learn-why-data-scientists-and-developers-need-more-than-a-data-lake/.

[5] Big data analytics. Consulté le : 05-04-2024. URL : https://www.ibm.com/analytics/big-data-analytics.

[6] Big Data Analytics. What it is and why it matters. Consulté le : 05-04-2024. URL : https://www.sas.com/en_us/insights/analytics/big-data-analytics.html.

[7] What is a Data Lake ? Consulté le : 05-04-2024. URL : https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-data-lake.

[8] Difference between Traditional Data and Big Data. Consulté le : 05-04-2024. URL : https://www.javatpoint.com/difference-between-traditional-data-and-big-data.