République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option: *Modèles Intelligents Et Décisions (M.I.D)*

Thème

Développement d'un cadre générique pour le partitionnement des données RDF

Réalisé par :

- **Boudghene Stambouli Mohammed Reda**

Présenté le 03 Juillet 2024 devant le jury composé de.

- *Mr Smahi Mohammed Ismail* *(Président)*
- *Mr Semmoud Abderraziq* *(Examinateur)*
- *Mr Matallah Houcine* *(Encadreur)*
- *Mr Saidi Boumediene* *(Co-encadreur)*

Année universitaire : 2023-2024

Remerciements

Je commence par exprimer ma profonde gratitude envers Allah, le Tout-Miséricordieux, pour m'avoir accordé la force, la patience et le courage nécessaires pour achever toutes ces longues années d'études.

Mes plus sincères remerciements s'adressent à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce projet.

Je tiens à remercier particulièrement mon encadrant, Monsieur Hocine MATALLAH, ainsi que mon superviseur de stage, Monsieur Amin MESMOUDI, qui m'ont prodigué des conseils précieux et un soutien constant tout au long du projet.

Je suis également reconnaissante envers mon co-encadrant, Monsieur Saidi BOUMEDIENE, qui a été à mes côtés à chaque étape et sans qui ce travail n'aurait jamais été possible.

Enfin, je remercie tous les enseignants du département informatique qui ont fourni des efforts considérables pour nous transmettre leurs connaissances durant tout notre cursus universitaire.

Dédicaces

Pour celle dont les prières ont été le secret de ma réussite et de ma tendresse, ma chère maman.

Pour celui qui m'a soutenu tout au long de ma vie, qui a porté pour moi la robe de gloire et m'a rempli de son amour, mon cher père.

À mes piliers, mes deux frères bien-aimés, Hakim et Imad. À toute ma famille et mes amis.

À tous les membres du Club Cheikh Muhammad Chiali pour l'enseignement du Coran et de ses sciences.

À mes camarades de classe de master 2ème année en MID. Nous avons partagé des moments précieux et construit des souvenirs inoubliables. Un merci tout particulier à mon ami Imad Chennafi.

A tous ceux qui m'ont connu, à tous ceux qui n'ont pas trouvé leur nom dans ma dédicace, à tous ceux qui ont atteint mon cœur et ne les ont pas écrits dans ma plume.

Résumé

Avec la croissance explosive des technologies web et l'avènement de l'ère numérique, le volume de données générées et partagées sur le web a atteint des niveaux sans précédent. Cette explosion de la création de données pose des défis significatifs en matière de gestion, de traitement et d'analyse de cet afflux massif d'informations. Ce projet de fin d'études s'inscrit dans le cadre du projet PQDAG, qui vise à développer des outils avancés de gestion des données spécifiquement conçus pour assurer la mise à l'échelle et les performances lors du traitement des Big RDF Data. Notre objectif principal était de proposer des approches de partitionnement facilitant l'évaluation d'un grand nombre de requêtes SPARQL sans nécessiter de jointures entre les partitions, réduisant ainsi le temps de communication sur le réseau. Pour ce faire, nous avons appliqué et comparé plusieurs stratégies de partitionnement, y compris MPC (Minimum Property Cut), Metis et K-means. Le but était de minimiser le nombre de jointures inter-partitions et d'améliorer les performances de traitement des requêtes dans un environnement distribué, tout en réduisant les coûts de communication entre les partitions. Grâce à cette recherche, nous visons à identifier la solution la plus efficace pour relever les défis de la gestion des Big Data dans le cadre du projet PQDAG, améliorant ainsi la capacité à gérer et analyser les données RDF à grande échelle sur le web en parallèle.

Mots-clés

Langage SPARQL, Triplestores, PQDAG, Partitionnement, MPC, RDF_QDAG, Optimisation, K-means

Abstract

With the explosive growth of web technologies and the advent of the digital age, the volume of data generated and shared on the web has reached unprecedented levels. This explosion in data creation presents significant challenges in managing, processing, and analyzing this massive influx of information. This thesis is part of the PQDAG project, which aims to develop advanced data management tools specifically designed to ensure scalability and performance when processing Big RDF Data. Our main objective was to propose partitioning approaches that facilitate the evaluation of a large number of SPARQL queries without requiring joins between partitions, thus reducing network communication time. To achieve this, we applied and compared several partitioning strategies, including MPC (Minimum Property Cut), Metis, and K-means. The goal was to minimize the number of inter-partition joins and improve query processing performance in a distributed environment while reducing communication costs between partitions. Through this research, we aim to identify the most efficient solution to address the challenges of Big Data management within the scope of the PQDAG project, thereby enhancing the ability to handle and analyze large-scale RDF data on the web in parallel.

Keywords

SPARQL language, Triplestores, PQDAG, Partitioning, MPC, RDF_QDAG, Optimization, K-means

Table des matières

Table des figures

Liste des tableaux

Table des abréviations

Abréviation	**Signification**
GCC	GNU Compiler Collection
IBM	International Business Machines
IDC	International Data Corporation
JSON	JavaScript Object Notation
MPC	Minimum Property cut
NT	N-Triples (format de fichier RDF)
OLD	Open Linked Data
RDF	Resource Description Framework
RDF_QDAG	RDF Querying Data As Graphs
RSG	Rooted Sub-Graph
SGBDR	Système de Gestion de Bases de Données Relationnelles
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
W3C	World Wide Web Consortium
WCC	Weakly Connected Component
XML	Extensible Markup Language

Chapitre 1

Introduction

1.1 Contexte et problématique

À l'ère numérique, le web est devenu un vaste réservoir de données en constante expansion et évolution. Cette prolifération de données, communément appelée Big Data, présente à la fois des opportunités et des défis, notamment dans le domaine de la sémantique web. La sémantique web, ou l'étude de la signification et de la structure du contenu web, joue un rôle crucial pour donner un sens à cet immense trésor d'informations.

Avec l'augmentation exponentielle du volume de données générées sur le web, les méthodes traditionnelles de traitement et d'analyse des données ne sont plus suffisantes. Les technologies et techniques du Big Data sont essentielles pour extraire des insights significatifs de cette avalanche d'informations. Selon une étude d'IDC [1], la quantité de données créées, capturées et répliquées dans le monde entier devrait atteindre 175 zettaoctets (ZB) d'ici 2025. Cette croissance exponentielle du volume de données nécessite des infrastructures et des algorithmes évolutifs capables de gérer

1. IDC. "Data Age 2025: The Digitization of the World from Edge to Core." IDC White Paper, Sponsored by Seagate, novembre 2018.

des ensembles de données aussi massifs.

Les données web se présentent sous divers formats, y compris les données structurées, semi-structurées et non structurées. Cette diversité de types de données pose des défis pour les méthodes traditionnelles de traitement des données. Selon un rapport d'IBM [2], 80 % des données mondiales sont non structurées, soulignant ainsi la nécessité de techniques avancées telles que les données liées ouvertes (OLD) et l'apprentissage automatique pour extraire des insights à partir de contenu web non structuré.

Les données sur le web sont générées à une vitesse sans précédent, les plateformes de réseaux sociaux produisant à elles seules d'énormes quantités de données en temps réel. Par exemple, les utilisateurs de Twitter génèrent environ 500 millions de tweets par jour [3]. Traiter et analyser ce flux de données en temps réel nécessite des plateformes et des algorithmes d'analyse de flux efficaces.

Cependant, la qualité et la fiabilité des données web peuvent varier considérablement, allant d'informations précises et dignes de confiance à de la désinformation et des fausses nouvelles. Assurer la véracité des données est essentiel pour prendre des décisions éclairées basées sur la sémantique web. Cela nécessite des mécanismes robustes de validation des données et d'assurance qualité.

Malgré l'abondance de données sur le web, extraire des insights exploitables qui apportent une valeur tangible reste un défi majeur. Selon une enquête de NewVantage Partners [4], 77 % des dirigeants estiment que leurs organisations ne réalisent pas encore pleinement la valeur potentielle de leurs données. Combler cet écart entre la collecte de données et la création de valeur nécessite des analyses avancées et des processus décisionnels basés sur les données. En résumé, gérer et exploiter efficace-

2. IBM. "Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data." IBM Redbooks, avril 2016.
3. Twitter. "Twitter Q1 2021 Shareholder Letter." Twitter, avril 2021.
4. NewVantage Partners. "Big Data and AI Executive Survey 2021." NewVantage Partners LLC, janvier 2021.

ment le Big Data sur le web est crucial pour tirer parti des opportunités offertes par cette ère numérique tout en surmontant les défis qu'elle présente.

Les travaux de ce stage s'inscrivent dans le cadre des travaux menés conjointement par le laboratoire LIAS (Poitiers - France) et le laboratoire LRIT (Tlemcen - Algérie) et qui visent à développer une nouvelle plateforme de gestion de données pour les grands base de données RDF. Une telle plateforme devrait pouvoir garantir non seulement le passage à l'échelle en permettant le stockage de plusieurs pétaoctets de données ainsi que les performances en garantissant une réponse aux requêtes des utilisateurs dans un délai acceptable. La plateforme actuellement développée au laboratoire LIAS s'appelle RDF_QDAG (RDF Query Data As Graph). Cette approche repose sur la représentation logique des données sous forme de graphe et permet d'évaluer des requêtes SPARQL en fonction du ratio d'exploitation des graphes. Une étude [8] a montré que cette approche surpasse de loin les approches existantes en termes de passage à l'échelle et de performances. La parallélisation d'une telle approche est l'un des principaux objectifs des chercheurs du LIAS et du LRIT. Afin d'atteindre cet objectif, il est nécessaire de proposer des extensions et des outils pour RDF_QDAG, lui permettant de gérer le partitionnement du graphe initial, l'optimisation des requêtes SPARQL et le transfert de données entre les différentes machines du cluster. Durant mon stage, j'ai participé à l'application et à l'évaluation de certaines stratégies de partitionnement initial.

1.2 Objectif

L'objectif du stage auquel j'ai participé à distance était de réaliser une étude sur le système de gestion de données RDF distribué, PQDAG, plus précisément sur les stratégies de partitionnement des graphes RDF.

Nous avons principalement appliqué les méthodes MPC [13], K-means, et Metis

[6] afin de comparer et évaluer leurs performances. En appliquant le partitionnement, nous visons à éviter les jointures inter-partitions coûteuses en termes de temps de communication réseau. Cela nous permet de gagner en performance et en évolutivité.

Nous avons également essayé de trouver une représentation appropriée pour les fragments que nous souhaitons partitionner, afin de pouvoir utiliser l'algorithme de clustering K-means.

1.3 Organisation du manuscrit

Le reste de ce manuscrit sera organisé comme suit : Dans le chapitre 2, nous présenterons les concepts du Web sémantique et du modèle RDF. Le chapitre 3 se concentrera sur les triplestores RDF distribués et les différentes méthodes de partitionnement, en particulier Metis, MPC et K-means. Le chapitre 4 détaillera le système de gestion de données PQDAG et les méthodes de partitionnement utilisées jusqu'à présent. Le chapitre 5 présentera nos contributions spécifiques, y compris l'utilisation de Metis avec les poids des arcs, MPC, K-means, ainsi que notre nouvelle contribution sur la représentation adéquate des fragments pour l'utilisation de K-means. Le chapitre 6 discutera et analysera les résultats obtenus pour chaque stratégie de partitionnement. Enfin, le chapitre 7 conclura ce projet de fin d'études avec une conclusion générale et quelques perspectives pour les recherches futures et les améliorations possibles.

Chapitre 2

Contexte de travail

2.1 Le Web sémantique

Le terme "web sémantique", ou "semantic web" en anglais, désigne un concept évolutif dans le domaine de l'informatique et des technologies de l'information, visant à ajouter du sens et du contexte au contenu du World Wide Web. En outre, nous pouvons considérer que Web sémantique [2] est un ensemble de normes et de bonnes pratiques de W3C pour le partage des données et la sémantique de ces données sur le Web, afin de les rendre utilisables par les applications (machines).

L'objectif ultime du web sémantique est de créer un web de données plus interconnecté et interopérable, connu également sous le nom de Données liées, où les informations peuvent être intégrées de manière transparente et accessibles tant par les humains que par les machines.

Les données liées fournissent les meilleures pratiques pour établir ces connexions. Lorsque ces données sont combinées avec celles qui peuvent être librement utilisées et distribuées (Open Data), on les appelle "données ouvertes liées" (LOD).

Les applications pratiques du web sémantique et LOD sont nombreuses [1]. Elles

1. https://www.dataversity.net/semantic-technology-trends-in-2024/

incluent des moteurs de recherche aux capacités améliorées, comme Google et DBpedia [2], ainsi que des expériences en ligne plus personnalisées, comme celles fournies par Amazon Alexa.

2.2 Le modèle de données RDF

Basée sur la définition de Lassila et Swick (1999)[10], le RDF (Resource Description Framework) est une norme du World Wide Web Consortium (W3C) conçue pour décrire des ressources web et échanger des données. Il permet de représenter des informations sous forme de triplets composés d'un sujet, d'un prédicat et d'un objet (<subject> <predicate> <object>).

Ces triplets forment ensuite un graphe orienté, facilitant la connexion et la compréhension des relations entre les données. Le RDF constitue un élément fondamental du web sémantique, offrant une structure pour modéliser et échanger des connaissances. Ceci est un exemple de triplet RDF :

Paris est la capitale de la France

Dans cet exemple : "Paris" représente le sujet, "est la capitale de" représente le prédicat, "la France" représente l'objet.

De nos jours, RDF est largement employé sur le Web, et de nombreuses applications tirent parti de ce type de données dans divers domaines. Voici quelques exemples de données RDF publiquement disponibles :

— Le projet DBpedia, qui extrait des contenus structurés RDF à partir des informations disponibles sur les différents sites de Wikipédia.

— Bio2RDF [3], une base de données biologique utilisant les technologies RDF et du Web sémantique.

2. https://www.dbpedia.org/about/
3. https://bio2rdf.org/

— LinkedGeoData [4], qui transforme les informations du projet OpenStreetMap en données RDF.

— Europeana [5], une plateforme qui donne accès à des millions d'objets numériques issus des collections culturelles européennes, disponibles sous forme de données RDF.

2.2.1 Graphe RDF

Un graphe de données RDF est une collection de triplets RDF. Il est constitué de sommets étiquetée, représentant les sujets et les objets, reliés par des arcs orientés. Chaque arc possède une étiquette qui représente sa propriété.

La figure 2.1 présente un graphe de données RDF contenant des informations relatives à certains films.

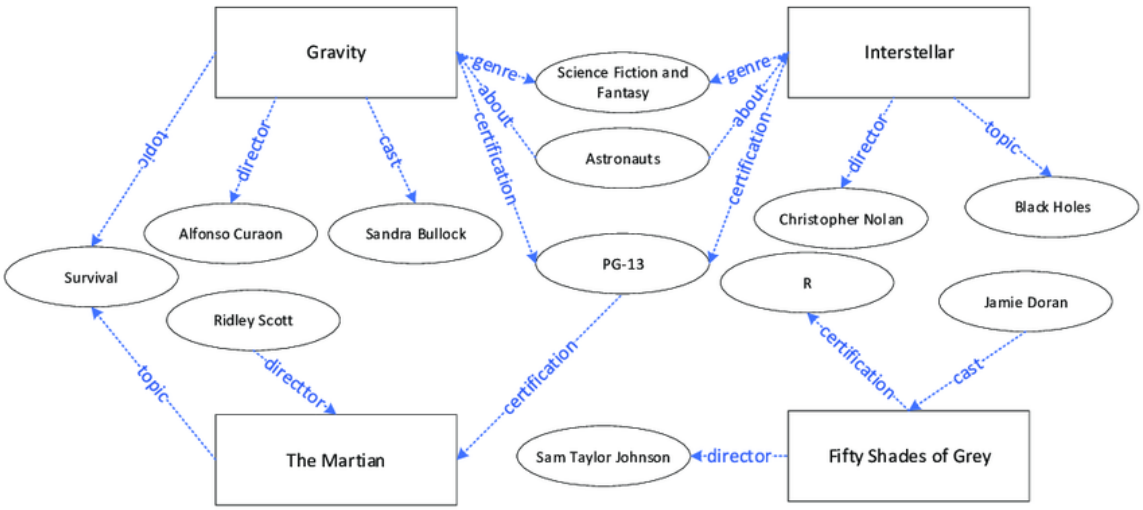


FIGURE 2.1 – Exemple de graphe RDF

4. http://linkedgeodata.org/
5. https://www.europeana.eu/en

2.2.2 Les syntaxes de RDF

Les syntaxes de RDF jouent un rôle crucial dans la représentation et le traitement des données RDF. Parmi ces syntaxes, on trouve RDF/XML, N-Triples, Turtle et RDFa, chacune offrant des méthodes standardisées pour sérialiser les données RDF.

Ces syntaxes sont essentielles pour stocker, partager et analyser les informations de manière efficace, en fournissant des moyens précis et normalisés de représenter les triplets RDF. Dans l'exemple de la syntaxe RDF/XML illustrée dans la figure2.2. Il décrit les informations suivantes :

Un article a un auteur qui a 27ans

```
<rdf:Description rdf:about="Paper">
    <author>
        <rdf:Description>
            <age rdf:type="Age">
                27
            </age>
        </rdf:Description>
    </author>
<rdf:Description>
```

FIGURE 2.2 – Exemple d'un syntaxe RDF/XML

2.2.3 Le langage de requête SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) est un langage de requête RDF, c'est-à-dire un langage de requête sémantique pour les bases de données, capable de récupérer et de manipuler des données stockées en RDF. SPARQL a été développé par le World Wide Web Consortium (W3C) et est devenu la recommandation officielle du W3C pour interroger les données RDF en 2008 [15].

SPARQL fonctionne en comparant des motifs de graphe à une base de données

RDF. Ce processus implique d'identifier des triplets correspondants dans la base de données qui s'alignent sur le motif spécifié. Des motifs de graphe complexes peuvent être construits en combinant des motifs plus simples à l'aide de divers opérateurs, permettant aux utilisateurs de formuler des requêtes précises qui capturent exactement les informations souhaitées.

2.2.4 La syntaxe de SPARQL

Les requêtes SPARQL se composent de plusieurs parties essentielles, chacune jouant un rôle spécifique dans la formulation de la requête pour interroger des données RDF. Voici une description de ces composants clés :

— **PREFIX** : Utilisé pour définir des préfixes pour les URI, facilitant la lisibilité des requêtes.
— **SELECT** : Spécifie les variables à retourner dans les résultats de la requête.
— **WHERE** : Contient les motifs de graphes qui doivent être satisfaits pour qu'un résultat soit inclus dans la sortie.
— **PREFIX** : Applique des conditions supplémentaires sur les résultats.

Voici quelques motifs de graphes couramment utilisés dans SPARQL :

— **Motif de triplet de base (BGP)**, est un ensemble de triplets RDF qui doivent tous correspondre aux données. Il est souvent utilisé dans la clause `WHERE`.
— **Motif de filtre**, est utilisé pour restreindre les résultats d'une requête en appliquant des conditions sur les variables. Il est défini à l'aide du mot-clé `FILTER`.
— **Motif optionnel**, permet d'inclure des triplets qui peuvent ou non être présents dans les données. Il utilise le mot-clé `OPTIONAL`.

— **Motif de groupe**, regroupe plusieurs motifs dans une seule unité logique à l'aide d'accolades ‘‘.

— **Motif de union**, combine plusieurs motifs et permet d'obtenir des résultats correspondant à l'un ou l'autre des motifs. Il est défini à l'aide du mot-clé `UNION`.

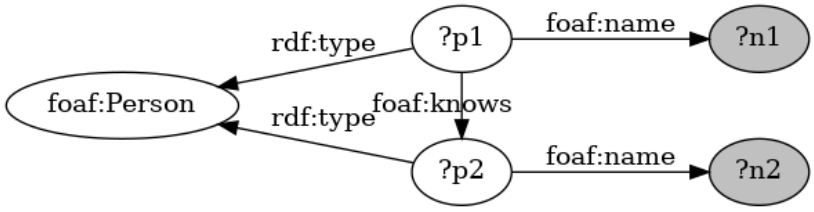


FIGURE 2.3 – Exemple de requêtes SPARQL

La représentation textuelle de la requête de la Figure 2.3 est la suivante :

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?n1 ?n2
WHERE {
    ?p1 a foaf:Person ;
        foaf:name ?n1 ;
        foaf:knows ?p2 .
    ?p2 a foaf:Person ;
        foaf:name ?n2 .
}
```

Cette requête SPARQL recherche des amis dans un réseau social. Elle cherche des

personnes (?p1) qui sont identifiées comme des personnes (foaf :Person) et qui ont un nom (?n1). Ensuite, elle vérifie si ces personnes connaissent quelqu'un d'autre (?p2), également identifiée comme une personne et ayant un nom (?n2). En résumé, la requête renvoie la liste des couples "nom de la personne 1 - nom de la personne qu'elle connaît".

2.3 Les triplestores RDF centralisés

La gestion des grands ensembles de données RDF présente de nombreux défis. Cela comprend le partitionnement efficace des données, le stockage approprié, l'indexation performante et l'évaluation optimale des requêtes SPARQL. Les systèmes RDF centralisés sont conçus pour gérer, stocker et interroger des données RDF de manière efficace dans un environnement centralisé, c'est-à-dire sur une seule machine. Cela offre des avantages en termes de cohérence et de facilité de gestion, mais peut également présenter des inconvénients en matière d'évolutivité et de points de défaillance uniques. Nous présentons brièvement leurs modèles de stockage.

— **Stockage relationnel**

Ce type de stockage utilise des bases de données relationnelles pour stocker les triplets RDF. Chaque triplet est représenté par une ligne dans une table, où les colonnes correspondent au sujet, au prédicat et à l'objet. Les requêtes SPARQL sont exécutées en traduisant les motifs de la requête en instructions SQL pour interroger les tables relationnelles. Hexastore [17] et Jena1 [11] sont des exemples notables de systèmes qui utilisent ce type de stockage.

— **Stockage en table de propriétés**

La table de propriétés est une façon de représenter les données RDF dans un

table a des dimensions déterminées par le nombre de sujets et de prédicats distincts. Chaque entrée dans la table contient la valeur de l'objet associé à un sujet et un prédicat particuliers. Cependant, cette représentation entraîne un coût de stockage élevé lorsque le nombre de prédicats uniques est important, et elle ne peut pas gérer les attributs multivalués, où un sujet est lié à plusieurs objets pour le même prédicat. Pour résoudre ces problèmes, Jena2 [18] a introduit deux autres représentations : la table de propriétés clusterisée et les tables de propriétés-classes.

— **Stockage par partitionnement vertical**
Dans ce modèle de stockage, la table de triplets est partitionnée en plusieurs tables distinctes, chacune composée de deux colonnes. Cette partition se fait en fonction des prédicats distincts présents dans les triplets. Chaque table représente ainsi l'ensemble des triplets partageant un même prédicat, regroupant les paires sujet-objet liées par ce prédicat spécifique. SW-Store [1] constitue un exemple concret de système utilisant ce modèle de partitionnement. Afin de rassembler des informations sur plusieurs propriétés pour des sous-ensembles de sujets, une jointure par fusion (Merge-join) est employée.

Les systèmes de gestion de bases de données RDF centralisés sont confrontés à des difficultés, surtout lors du passage à l'échelle. Les triplestores distribués apparaissent comme une solution prometteuse pour résoudre ce défi.

Chapitre 3

État de l'art

3.1 Les Systèmes RDF distribués

Ce type d'architecture arrive à s'adapter aux nouveaux besoins liés à la gestion des données RDF, surtout avec l'arrivée du Big Data qui nécessite généralement le traitement de très grands volumes de données. Un système de gestion de base de données distribuée (DDBMS) [7] peut être défini comme un système où la gestion des données est répartie sur plusieurs machines communiquant via un réseau informatique. Cela améliore non seulement l'évolutivité, mais aussi la tolérance aux pannes, car la défaillance d'un nœud n'affecte pas nécessairement l'ensemble du système. Les systèmes distribués offrent également des performances accrues pour les requêtes complexes, grâce à la parallélisation des tâches.

Les systèmes RDF distribués traitent de grands bases de données en distribuant le graphe RDF initial sur plusieurs machines (workers) et en évaluant les requêtes de manière distribuée. Lorsqu'une requête SPARQL est exécutée, elle est décomposée en sous-requêtes qui sont envoyées aux workers concernés. Chaque worker évalue sa sous-requête sur la partie du graphe qu'il stocke.

Avec la distribution des données, les 'workers' peuvent avoir besoin d'échanger des

résultats entre eux pour évaluer les requêtes nécessitant des données provenant de plusieurs machines. Cela peut générer un grand nombre de résultats intermédiaires, entraînant ainsi des coûts de communication importants.

Pour obtenir des temps de réponse acceptables, les systèmes distribués cherchent à minimiser les coûts de communication entre machines. Ceci peut être réalisé grâce à un partitionnement efficace des données, des implémentations de jointures optimisées et une gestion efficace des transferts de données entre les données d'une partition.

3.1.1 Quelques Triplestores Distribués

YARS2 [5] est un system RDF distribué conçu pour gérer des données RDF à grande échelle en utilisant du matériel standard sans ressources partagées. Il emploie trois types de méthodes d'indexation en raison de son stockage des données RDF en quads. La première méthode est un index de mots-clés utilisant Apache Lucene comme un index inversé pour faire correspondre les termes des objets RDF avec les sujets. La deuxième méthode implique six index clairsemés avec six fichiers triés stockés sur disque, encodés en utilisant le codage de Huffman. La troisième méthode est un index de jointure conçu pour accélérer l'exécution des requêtes contenant des combinaisons spécifiques de valeurs. YARS2 utilise SPARQL comme langage de requête et optimise l'exécution des requêtes par des méthodes d'évaluation de requêtes parallèles et une gestion du trafic réseau. De plus, il emploie une programmation dynamique pour déterminer l'ordre des jointures et utilise un placement de données basé sur le hachage pour l'indexation des quads.

SHARD [16] est un triplestore construit sur Hadoop qui utilise une distribution de données basée sur le hachage. Il stocke les données RDF dans des fichiers plats sur le système de fichiers distribué Hadoop (HDFS), chaque ligne contenant tous les

triplets associés à un sujet. Le jeu de données d'entrée est partitionné par hachage afin que chaque partition contienne un ensemble différent de triplets. Contrairement à d'autres systèmes, SHARD n'utilise pas d'indexation spécifique, mais s'appuie sur des itérations MapReduce pour l'exécution des requêtes. Initialement, il collecte les résultats des sous-requêtes, applique les jointures, puis filtre les résultats redondants ou ceux associés à des conditions spécifiques. Cette approche permet à SHARD de gérer efficacement de grands jeux de données RDF en tirant parti des capacités de traitement distribué de Hadoop.

Trinity.RDF [20] est un système de stockage de triplets RDF distribué, construit sur Trinity, une plate-forme de traitement des données clé/valeur en mémoire distribuée. Ce triplestore tire parti des capacités de Trinity pour fournir un stockage efficace et une récupération rapide des données RDF en utilisant des graphes de propriétés en mémoire. Trinity.RDF stocke les données RDF sous forme de graphes où chaque nœud et chaque arête sont directement accessibles en mémoire, ce qui améliore considérablement la performance des requêtes complexes par rapport aux triplestores traditionnels basés sur disque.

Virtuoso [3] stocke les données RDF dans une table unique avec quatre colonnes : Sujet (s), Prédicat (p), Objet (o) et Graphe (g), où s, p et g sont des IRI, et o peut être de n'importe quel type de données. Le système encode les IRI et les littéraux utilisés dans les triplets dans un dictionnaire. Virtuoso maintient quelques index clés, tels que l'index par défaut gspo et un index bitmap complémentaire opgs. L'exécution des requêtes dans Virtuoso implique la traduction des requêtes SPARQL en SQL, qui sont ensuite exécutées sur le système de base de données sous-jacent avec des plans d'optimisation basés sur les coûts. Le partitionnement des données dans Virtuoso est géré à l'aide d'une fonction de hachage sur les sujets, assurant une distribution

et une récupération efficaces des données dans tout le système. Cette intégration de SPARQL et SQL permet à Virtuoso de tirer parti des optimisations traditionnelles des bases de données pour la gestion des données RDF.

3.2 Positionnement par rapport à l'état de l'art

Dans les systèmes RDF distribués, le partitionnement de graphe est le processus de division d'un graphe RDF en sous-graphes plus petits, appelés partitions. Le partitionnement joue un rôle crucial dans la gestion et l'interrogation efficace des grandes bases de données RDF. Il permet de distribuer les données sur plusieurs machines afin d'exécuter des requêtes SPARQL en parallèle. L'objectif d'un bon partitionnement est de minimiser le coût des jointures entre les partitions de données afin d'optimiser le temps de communication réseau et, en fin de compte, d'améliorer les temps de réponse. Dans les sections suivantes, nous aborderons plus particulièrement trois stratégies de partitionnement : Metis, MPC et le clustering k-means.

3.3 Métis

METIS [6] est un algorithme de partitionnement de graphes qui comprend trois étapes : coarsening, le partitionnement initial et le refinement. L'idée derrière METIS est de créer des graphes G1, G2 successivement plus petits. . . ,Gk de G0. Ensuite, projetez à nouveau la partition sur le G0, en affinant à chaque étape. La figure 3.1 illustre la méthode.

METIS comporte trois étapes :

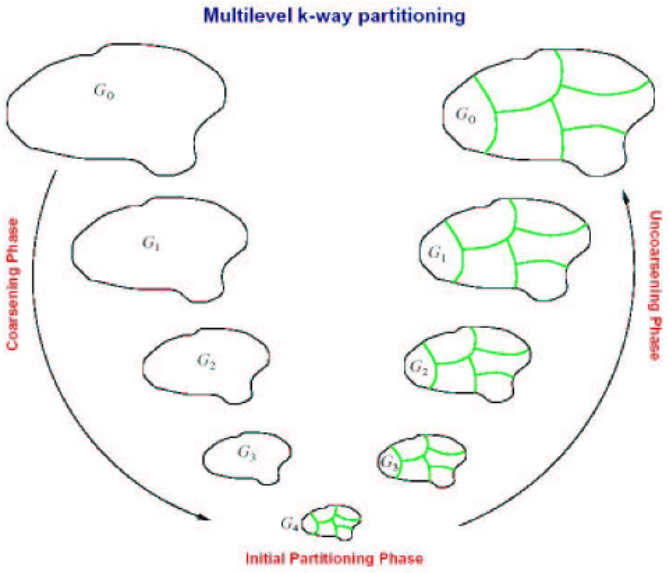


FIGURE 3.1 – Différentes phases de partitionnement de l'algorithme Metis

3.3.1 Coarsening

Le processus de coarsening dans Metis commence par la réduction de la taille du graphe d'origine. L'objectif est de simplifier le graphe en regroupant ses nœuds et arêtes pour former un graphe plus petit et plus maniable. À chaque itération, les nœuds du graphe sont regroupés en nœuds composites, souvent en utilisant des heuristiques basées sur les poids des arêtes pour choisir quels nœuds fusionner. La figure 3.2 illustre comment les nœuds sont regroupés en un nouveau nœud. Par exemple, deux nœuds connectés par une arête de fort poids sont plus susceptibles d'être contractés ensemble, car cela représente une relation forte entre ces nœuds.

3.3.2 Partition initiale

Après le coarsening, METIS procède à une première étape de partitionnement du graphe. Cette étape consiste à attribuer chaque nœud à l'une des partitions désirées.

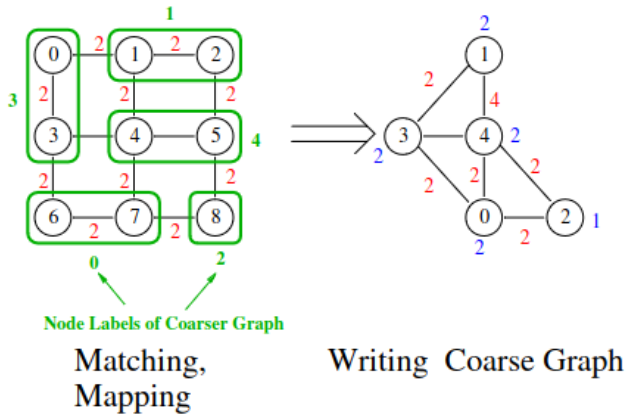


FIGURE 3.2 – Exemple de coarsening d'une grille en un graphe plus petit.

Deux méthodes sont utilisées [14] :

Partitionnement aléatoire : attribution des nœuds de manière aléatoire aux partitions.

Le partitionnement glouton : attribution itérative des nœuds à la partition qui minimise les connexions entre les partitions. L'objectif est de minimiser le nombre d'arêtes coupées entre les partitions tout en assurant que chaque partition contient un nombre de nœuds à peu près égal.

3.3.3 refinement

Après le partitionnement initial, Metis entre dans la phase d'uncoarsening et de raffinement. Cette étape implique de projeter la partition obtenue sur le graphe contracté vers les niveaux supérieurs du graphe, jusqu'à ce qu'elle soit appliquée au graphe d'origine. À chaque niveau de projection, la partition est ajustée et améliorée. Des algorithmes de raffinement, comme les déplacements de nœuds entre partitions, sont utilisés pour optimiser la qualité de la partition. Par exemple, déplacer un nœud d'une partition à une autre peut réduire le nombre d'arêtes coupées et équilibrer les

tailles des partitions.

Ce processus de raffinement est crucial pour obtenir une partition de haute qualité qui minimise les coûts de communication entre les partitions dans les applications parallèles. L'objectif final est d'atteindre une partition équilibrée et efficace pour le graphe d'origine, prête pour une utilisation dans des calculs distribués.

3.4 MPC :Minimum Property Cut

La technique Minimum Property-Cut (MPC) [13] est une nouvelle méthode de partitionnement des graphes RDF. Sa fonction objective est de minimiser les réductions du nombre de propriétés de liaison distinctes plutôt que de minimiser les les coupures d'arête. Cette approche évite ou réduire les jointures entre partitions rejoint un ensemble plus large de charges de travail SPARQL. Notre

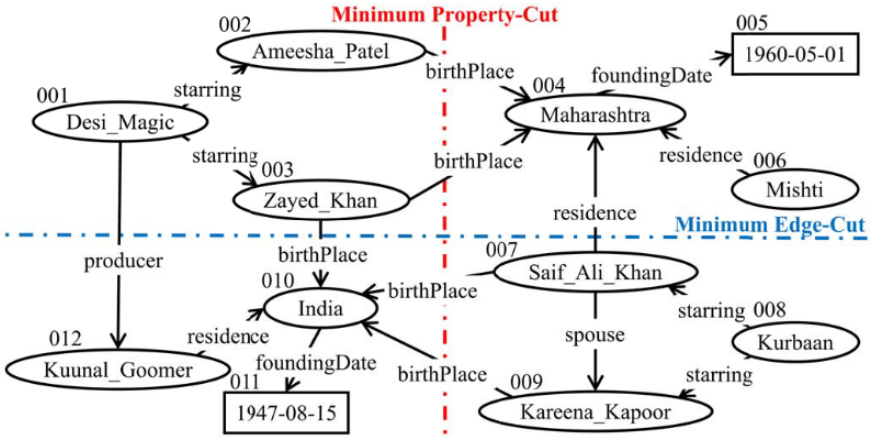


FIGURE 3.3 – Comparaison de MPC et une approche de partitionnement différente

Considérez le partitionnement en ligne pointillée rouge dans un graphe RDF sur la figure 3.3 qui est le résultat du partitionnement MPC. Dans ce cas il n'y a qu'un seul propriété de liaison, birthPlace. Les propriétés de liaison se situent à la frontière

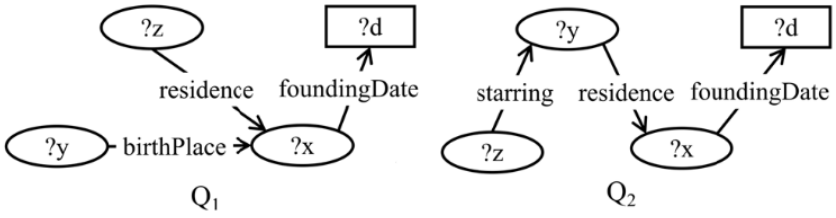


FIGURE 3.4 – Exemple de SPARQL requêtes

des partitions et assurent la connexion entre elles, tandis que les propriétés internes se trouvent dans la même partition.

La requête Q2 dans la figure 3.4 n'implique pas la propriété de liaison, BirthPlace, ce qui lui permet d'être exécutée indépendamment, sans nécessiter de jointure entre les partitions, même s'il ne s'agit pas d'une requête en étoile. Étant donné que le partitionnement MPC est également disjoint par sommet, il garantit toujours que toutes les requêtes en forme d'étoile (par exemple, Q1) peuvent être exécutées indépendamment sans nécessiter de jointures entre partitions.

3.4.1 Comment fonctionne l'approche MPC ?

Sélection des propriétés internes

En première étape, il faut sélectionner les propriétés internes du graphe RDF (G). Étant donné que ce problème est NP-complet, nous utilisons un algorithme heuristique glouton, illustré dans la figure 3.5 nommée 'Internal Property Selection Algorithm', pour effectuer cette sélection.

L'algorithme de sélection des propriétés internes est conçu pour identifier un ensemble de propriétés internes dans un graphe RDF donné. Voici une explication de cet algorithme :

1. **Initialisation** :

Algorithm 1: Internal Property Selection Algorithm

Input: An RDF graph $G = (V, E, L, f)$
Output: A set of internal properties $L_{in} \subseteq L$

$L_{in} \leftarrow \emptyset$;
for *each property* p *in* L **do**
  Compute $WCC(G[\{p\}])$, i.e., weakly connected components in $G[\{p\}]$;
  Compute the cost of $Cost(\{p\})$ according to Definition 4.2.
while $L \neq \emptyset$ **do**
  $mincost \leftarrow \infty$; $p_{opt} \leftarrow \phi$
  for *each property* p *in* L **do**
    Compute $WCC(G[L_{in} \cup \{p\}])$
    if $Cost(L_{in} \cup \{p\}) < (1 + \epsilon) \times |V|/k$ **then**
      if $Cost(L_{in} \cup \{p\}) < mincost$ **then**
        $mincost \leftarrow Cost(L_{in} \cup \{p\})$;
        $p_{opt} \leftarrow p$
  if $p_{opt} == \phi$ **then**
    BREAK
  $L_{in} \leftarrow L_{in} \cup \{p_{opt}\}$;
  $L \leftarrow L - \{p_{opt}\}$;
Return L_{in};

FIGURE 3.5 – Algorithme de sélection de propriété interne

— L'ensemble des propriétés internes, L_{in}, est initialement vide.

2. **Pré-calcul** :

— Pour chaque propriété p dans l'ensemble des propriétés L du graphe G, l'algorithme calcule les composants faiblement connectés $WCC(G[\{p\}])$ dans le sous-graphe induit par p. Un WCC est un sous-ensemble de sommets dans un graphe où chaque paire de sommets est connectée par un chemin, en tenant compte des directions des arêtes. Pour calculer les WCC, parcourez le graphe en effectuant une recherche en profondeur ou en largeur en ignorant les directions des arêtes.

— Il calcule également le coût $Cost(\{p\})$ qui est défini comme la taille du plus

grand composant faiblement connecté (WCC) dans le sous-graphe induit par un ensemble de propriétés internes.

3. **Boucle principale** :
   — Tant que L n'est pas vide, l'algorithme effectue les étapes suivantes :
   (a) Initialiser $mincost$ à l'infini et p_{opt} à une valeur vide (ϕ).
   (b) Pour chaque propriété p dans L :
      — Calculer les composants faiblement connectés $WCC(G[L_{in} \cup \{p\}])$ dans le sous-graphe induit par $L_{in} \cup \{p\}$.
      — Si le coût de $L_{in} \cup \{p\}$ est inférieur à $(1 + \epsilon) \times |V|/k$, alors :
         — Si ce coût est également inférieur à $mincost$, mettre à jour $mincost$ et définir p_{opt} comme étant p.
4. **Vérification et mise à jour** :
   — Si aucune propriété optimale p_{opt} n'a été trouvée (c'est-à-dire si $p_{opt} == \phi$), l'algorithme se termine.
   — Sinon, ajouter p_{opt} à L_{in} et retirer p_{opt} de L.
5. **Retourner l'ensemble des propriétés internes** :
   — L'algorithme retourne finalement l'ensemble L_{in}.

En résumé, cet algorithme itératif sélectionne de manière heuristique les propriétés internes du graphe RDF, en évaluant les coûts et en cherchant à minimiser le coût total tout en respectant certaines contraintes de connectivité.

Coarsening

Après la sélection des propriétés internes, chaque composante faiblement connexe (WCC) dans le graphe RDF est représentée par un super-sommet (Super-Vertex). Les super-sommets forment un graphe coarsened Gc. Chaque super-sommet représente un sous-graphe et est lié aux autres par des propriétés de liaison. Cela permet d'obtenir

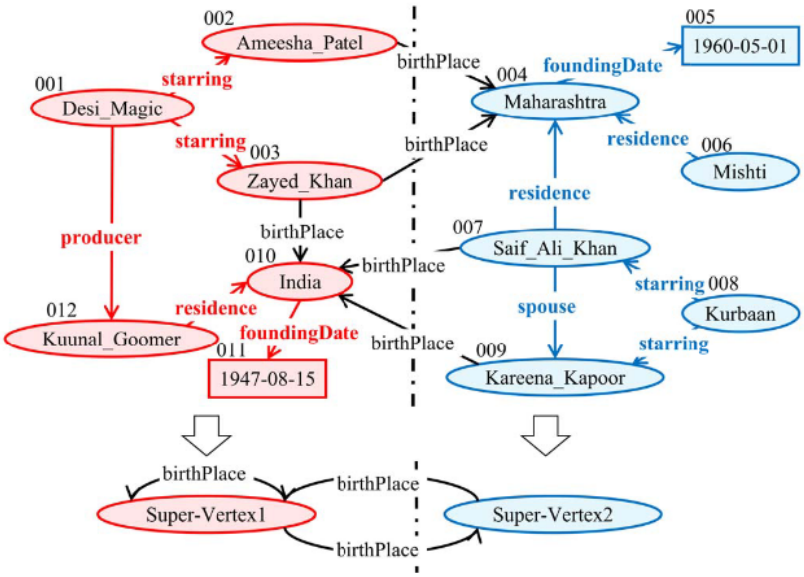


FIGURE 3.6 – Exemple de Coarsening

un nombre beaucoup plus petit de sommets, chacun représentant une WCC. Les WCC sont des sous-graphes dont les sommets sont liés par des propriétés internes. Un exemple de coarsening est illustré dans la figure, où "Birth place" est une propriété qui relie les différents WCC.

Le partitionnement final

Maintenant, nous pouvons utiliser n'importe quel algorithme de partitionnement spectral sur le graphe coarsened Gc. La complexité de ces algorithmes n'est pas un problème, car Gc est beaucoup plus petit que G. Enfin, le partitionnement de Gc est 'uncoarsened' pour obtenir un partitionnement du graphe original G.

3.5 K-means

L'algorithme K-means [9] est une méthode de partitionnement qui vise à regrouper un ensemble de données en un certain nombre de clusters, où chaque cluster est représenté par son centroïde. Ce centroïde est généralement la moyenne des points de données dans le cluster, d'où le nom "K-means" comme le montre la figure 3.7.

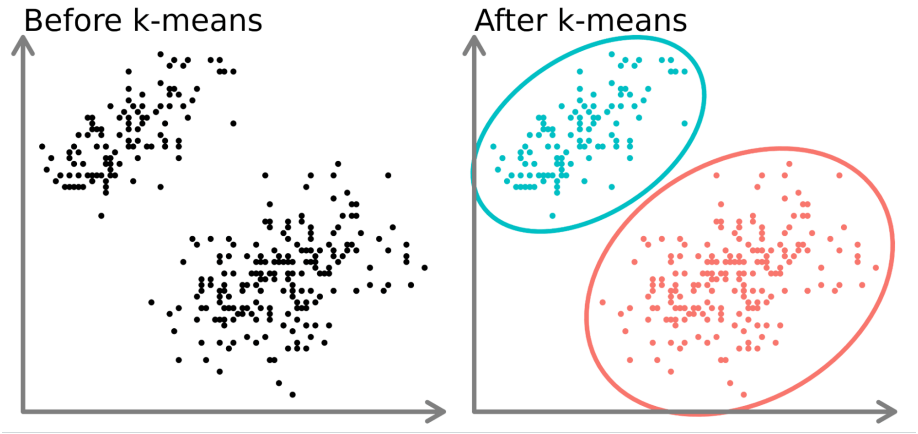


FIGURE 3.7 – Clustering K-means

3.5.1 Concept de base

L'algorithme K-means fonctionne en deux étapes principales : l'assignation des points de données aux clusters et la mise à jour des centroïdes. Voici les étapes détaillées du fonctionnement de l'algorithme :

1. **Initialisation :**
   — Choisir K centres initiaux de manière aléatoire parmi les points de données.
2. **Assignation :**
   — Pour chaque point de données, calculer la distance entre ce point et chaque centroïde.

— Assigner chaque point de données au cluster dont le centroïde est le plus proche.

3. **Mise à jour :**
   — Après l'assignation de tous les points de données, recalculer les centroïdes en prenant la moyenne des points dans chaque cluster.
4. **Répétition :**
   — Répéter les étapes d'assignation et de mise à jour jusqu'à ce que les centroïdes ne changent plus significativement ou qu'un nombre maximal d'itérations soit atteint.

3.5.2 Fonctionnement de K-means

L'algorithme K-means minimise la variance intra-cluster, c'est-à-dire qu'il réduit la somme des distances carrées entre les points de données et leur centroïde respectif. Cela conduit à des clusters où les points sont aussi proches que possible de leur centroïde, maximisant ainsi la similarité interne au cluster.

Le K-means est particulièrement efficace pour les jeux de données où les clusters ont une forme sphérique et sont à peu près de taille égale. Cependant, il peut être sensible aux points aberrants et aux valeurs initiales choisies pour les centroïdes, ce qui peut affecter le résultat final des clusters.

L'algorithme K-means est largement utilisé dans de nombreuses applications, y compris la segmentation de marché, la reconnaissance de motifs, et le traitement d'images, en raison de sa simplicité et de son efficacité. Cependant, dans le contexte des graphes RDF, des adaptations peuvent être nécessaires pour tenir compte des particularités structurelles et sémantiques des données.

Chapitre 4

Étude de l'existant

4.1 PQDAG

PQDAG est la nouvelle version distribuée du système RDF_QDAG RDF_QDAG [8], qui est un triplestore centralisé basé sur la fragmentation des données et tire parti de la notion de voisinage entre les nœuds liés à la représentation logique des données RDF. PQDAG est fondée sur le modèle BSP (Bulk Synchronous Parallel) [21], conçu spécifiquement pour le développement efficace d'algorithmes parallèles. Dans ce modèle, le calcul est divisé en plusieurs étapes appelées "super-étapes", ou, itérations. Les super-étapes sont séparées par une barrière de synchronisation pour garantir la réception des informations par chaque machine.

Chaque super-étape comporte trois phases : le calcul, la communication et la synchronisation. Les données sont réparties entre les différents processeurs (workers), qui exécutent simultanément des calculs locaux sur ces données. À la fin de chaque phase, les processeurs synchronisent leur progression pour échanger des données et se préparer à la phase suivante. Une illustration de ce fonctionnement est présente dans la figure 4.1.

En plus des machines workers, l'approche PQDAG inclut également une machine

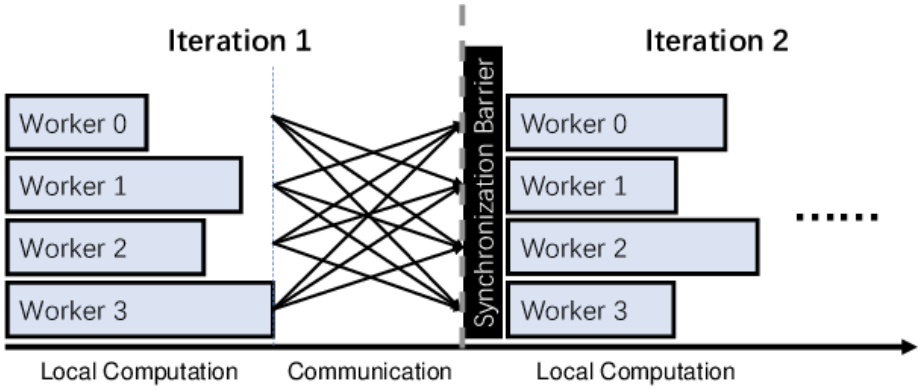


FIGURE 4.1 – Aperçu du modèle BSP

master qui gère tous les transferts.

Le master déclenche d'abord le traitement d'une requête en demandant à l'optimiseur de générer le plan optimal, lequel se présente sous la forme de plusieurs sous-requêtes. Sur cette base, le maître orchestre l'exécution des super-étapes. Il envoie le plan optimal aux machines workers et leur signale le début d'une super-étape. Dès que le signal est reçu, l'évaluation de la sous-requête à la position i dans le plan commence.

À la fin de cette étape, PQDAG examine les résultats intermédiaires pour déterminer s'il peut continuer avec les sous-requêtes aux positions supérieures à i + 1.

PQDAG arrête le traitement si aucun résultat ne peut être utilisé à des niveaux supérieurs. Sinon, la communication commence, où les résultats intermédiaires sont échangés entre les machines de calcul, terminant ainsi la super-étape de synchronisation. L'évaluation se poursuit de la même manière pour les autres super-étapes.

4.2 Méthodes de partitionnement utilisées jusqu'à présent dans PQDAG

Le PQDAG implique le tri des triplets RDF par sujet, suivi de la création d'étoiles de données associées à chaque sujet ou objet. Ensuite, des fragments de graphe sont créés en regroupant ces étoiles de données en fonction de leurs caractéristiques communes. Une fois les fragments créés, PQDAG utilise l'algorithme de METIS pour partitionner ces fragments sur chaque machine pour exécuter des requêtes SPARQL en parallèle.

L'algorithme de Metis est le seul qui a été mis en place. L'objectif de Metis est de minimiser les distances entre les fragments situés sur la même machine sans prendre en considération la localité des données. Par conséquent, le nombre de jointures entre les différentes machines n'est pas pris en compte, ce qui impacte énormément le temps de réponse final en raison des échanges coûteux de blocs de données.

Chapitre 5

Contribution

Pour remédier aux limitations identifiées dans l'étude de l'existant, nous proposons plusieurs approches de partitionnement dans PQDAG, incluant MPC, Metis et un clustering K-means. Pour K-means, nous appliquons deux représentations pour les fragments du graphe RDF que nous souhaitons partitionner, afin de pouvoir utiliser l'algorithme de clustering. L'objectif de toutes ces approches de partitionnement est de prendre en compte la localité des données (en essayant de placer des données avec des caractéristiques communes dans la même partition) afin de réduire le nombre de jointures nécessaires entre les machines et ainsi améliorer les performances globales.

5.1 Utilisation de Metis avec les poids des arcs

5.1.1 Représentation le graphe de fragment avec des poids d'arcs

Dans les graphes RDF, les nœuds représentent des entités et les arêtes représentent les relations (prédicats) entre ces entités. Le poids d'une arête peut représenter différentes caractéristiques, telles que la fréquence des interactions ou des connexions

entre les nœuds. Dans cette étape, nous construisons un RDF graphe dans lequel les nœuds représentent les sujets et les objets, et les arêtes pondérées représentent la fréquence de leurs connexions.

5.1.2 Un Exemple Réel de graphe de fragment avec des poids d'arcs

Considérons un graphe RDF représentant un réseau de citations scientifiques, où les nœuds sont des articles et les arêtes représentent les citations entre ces articles, pondérées par la fréquence des citations :

```
<article1 cite article2>    # Poids : 5 (cité 5 fois)
<article2 cite article3>    # Poids : 3
<article3 cite article4>    # Poids : 2
<article1 cite article4>    # Poids : 1
<article5 cite article1>    # Poids : 4
<article2 cite article4>    # Poids : 6
```

pour construire ce graphe :

```
Nœuds : {article1, article2, article3, article4, article5}


Arcs avec des poids :
article1 --(cite, 5)--> article2 (cité 5 fois)
article2 --(cite, 3)--> article3 (cité 3 fois)
article3 --(cite, 2)--> article4 (cité 2 fois)
article1 --(cite, 1)--> article4 (cité 1 fois)
article5 --(cite, 4)--> article1 (cité 4 fois)
article2 --(cite, 6)--> article4 (cité 6 fois)
```

5.1.3 Partitionnement avec Metis

Metis prend en compte la structure du graphe et les poids des arêtes (fréquences de connexion) afin de regrouper les nœuds ayant des connexions fréquentes dans la même partition. Cette approche optimise le traitement distribué en réduisant les communications entre les partitions.

Dans l'exemple ci-dessus, les poids des arêtes (5, 3, 2, 1, 4, 6) influencent le regroupement des articles dans les partitions. Metis assure que les articles ayant des fréquences de citation élevées sont regroupés ensemble dans la même partition. Par exemple, un article fortement cité sera plus susceptible d'être dans la même partition que les articles qui le citent fréquemment, ce qui minimise le coût des coupures d'arêtes et améliore l'efficacité globale du partitionnement.

5.2 MPC solution

5.2.1 Modélisation le graphe des fragments

Nous avons d'abord modélisé le graphe des fragments. Cette étape est cruciale car elle structure les données de manière à faciliter le partitionnement ultérieur. Chaque fragment représente un sujet ou un objet du graphe RDF original. Pour modéliser ces fragments, nous les représentons au format de fichier NT où chaque triple est sous la forme <sujet> <prédicat> <objet>. Pour modéliser le graphe, nous avons suivi ces étapes :

- **Identification des fragments** :
  - À partir de notre ensemble de données, nous extrayons tous les fragments qui sont soit un sujet soit un objet. Chaque fragment se voit attribuer un identifiant unique.
- **Création de liens entre les fragments** :

- Les fragments ne sont pas isolés. Ils interagissent les uns avec les autres via des entités ou des propriétés. Ces interactions sont capturées sous forme de liens entre les fragments. Chaque lien indique une relation entre deux fragments, représentée par un prédicat. Cela aide à comprendre comment les données sont interconnectées à travers différents fragments.

- **Stockage des fragments et des liens** :
  - Ce processus résultant à un ensemble de fragments et leurs liens correspondants. Ces données structurées doivent être stockées dans un format à la fois efficace pour le traitement et facile à interroger. Le format de fichier NT (N-Triples) est choisi pour sa simplicité et son acceptation large dans la communauté RDF. Chaque lien entre deux fragments est représenté sous forme de triple dans le format NT.

5.2.2 Création des partitions

Dans cette étape, nous avons appliqué l'algorithme MPC au graphe de fragments généré à l'étape précédente pour créer différentes partitions. Les résultats de l'algorithme MPC indiquent l'affectation de chaque super-sommet à une partition spécifique, et chaque partition est ensuite allouée à une machine spécifique. L'algorithme regroupe tous les fragments appartenant à la même partition, formant un ensemble de sommets (fragments), chaque groupe étant composé d'une liste d'identifiants de super-sommets.

5.3 K-means avec une représentation matricielle des fragments

Dans cette section, nous décrivons la solution de partitionnement K-means, en mettant l'accent sur la représentation des données et l'algorithme utilisé.

5.3.1 Représentation des données

Pour partitionner un graphe RDF avec l'algorithme K-means, il est essentiel de représenter les fragments de manière appropriée. Nous utilisons les sommets et les prédicats des arêtes qui les relient en entrée. Voici le processus suivi pour créer les vecteurs de caractéristiques des fragments :

1. **Création des vecteurs de caractéristiques** :

— **Identification des fragments** : Tous les sommets (ou fragments) du graphe sont identifiés et indexés.

— **Construction de la matrice des caractéristiques** : Une matrice carrée de dimension $n \times n$ (où n est le nombre de fragments) est construite comme le montre la figure 5.1. Chaque entrée de la matrice représente le prédicat de l'arête entre deux fragments. Si le graphe est non orienté, la matrice est symétrique.

$$\begin{bmatrix} & F_1 & F_2 & \cdots & F_n \\ F_1 & p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ F_2 & p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ F_n & p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{bmatrix}$$

FIGURE 5.1 – Matrice des poids entre les fragments du graphe RDF

5.3.2 Algorithme de partitionnement K-means

L'algorithme K-means est ensuite utilisé pour partitionner les fragments. Voici le processus suivi :

1. **Vecteurs de caractéristiques** : Chaque fragment est représenté par un vecteur de caractéristiques basé sur la matrice des caractéristiques générée précédem-

ment.

2. **Initialisation des clusters** : Les fragments sont initialement assignés aléatoirement à k clusters.

3. **Calcul des centroïdes** : Le centroïde de chaque cluster est calculé comme la moyenne des vecteurs de caractéristiques des fragments appartenant à ce cluster.

4. **Réassignation des fragments** : Les fragments sont réassignés au cluster dont le centroïde est le plus proche en termes de distance euclidienne entre les vecteurs de caractéristiques.

5. **Itération** : Les étapes de calcul des centroïdes et de réassignation des fragments sont répétées jusqu'à ce que les clusters se stabilisent et qu'il n'y ait plus de changement significatif dans l'affectation

5.4 K-means avec une représentation des fragments par RSGs

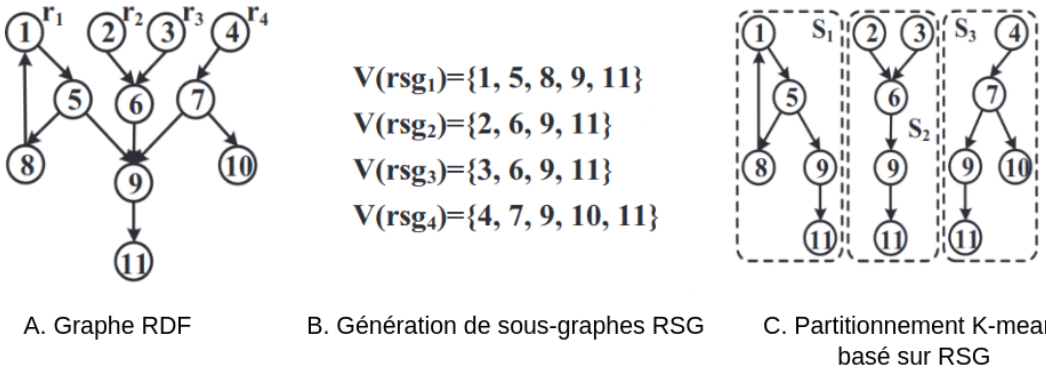


FIGURE 5.2 – Partitionnement K-Means basé sur RSG

Dans cette section, nous trouvons une autre représentation des fragments pour partitionner efficacement un graphe RDF avec de K-means. Pour ce faire, nous générons des sous-graphes RSGs (Rooted Sub-Graphs). Ensuite, nous appliquons le

clustering K-means sur les RSGs, comme illustré par l'exemple dans la figure 5.2.

5.4.1 Génération de sous-graphes RSGs

La génération de sous-graphes enracinés RSGs [19] vise à créer des sous-graphes enracinés à des sommets spécifiques, garantissant que les modèles de requêtes courants, tels que les requêtes en étoile, en chaîne, en arbre et en cycle, sont localisés dans ces sous-graphes. Cette approche permet de réduire les sommets en double et facilite une exécution efficace des requêtes.

Un RSG est défini dans le contexte d'un graphe RDF G, composé de sommets (V) et d'arêtes (E). Lorsqu'un sommet spécifique r de V est désigné comme racine, un sous-graphe noté rsg(r) = (Vrsg(r), Ersg(r)) est considéré comme un RSG enraciné à r si les conditions suivantes sont remplies :

- Pour chaque sommet v dans V, s'il existe un chemin dirigé de la racine r vers v, alors v doit appartenir à l'ensemble des sommets Vrsg(r).
- Pour chaque paire de sommets (u, v), si u et v sont tous deux des éléments de l'ensemble des sommets Vrsg(r) et s'il existe une arête dirigée (u, v) dans E, alors cette arête doit également être présente dans l'ensemble des arêtes Ersg(r).

En substance, un sous-graphe enraciné à un sommet r inclut tous les sommets pouvant être atteints depuis le sommet racine r via des chemins dirigés, ainsi que les arêtes correspondantes qui relient ces sommets au sein du sous-graphe.

Les étapes pour générer les RSGs sont les suivantes :

1. **Identification des Sommets Sources :**
   - Sélectionner les sommets avec un degré entrant de zéro (aucune arête en-

trante) comme sommets racines.

— Pour les sommets qui ne sont atteignables depuis aucun sommet source, sélectionner le sommet avec l'ID minimal dans de tels cycles comme sommets racines supplémentaires.

2. **Génération des RSGs :**
   — Pour chaque sommet racine, parcourir le graphe pour inclure tous les sommets atteignables, formant ainsi un sous-graphe enraciné RSG.
   — Assurer que tous les sommets soient couverts par au moins un RSG.

5.4.2 Clustering K-means sur les RSGs

Après avoir généré les RSGs, nous utilisons un algorithme de partitionnement k-means pour assigner ces RSGs aux nœuds de calcul (clusters), en optimisant l'allocation pour un traitement efficace des requêtes et une distribution optimale des données au sein des clusters.

Les étapes pour appliquer K-means sur les RSGs sont les suivantes :

1. **Conversion des RSGs en Vecteurs :**
   — Représenter chaque RSG comme un vecteur binaire

$$v_{rsg} = (v_1, v_2, ...v_V)$$

   de dimension $|V|$, indiquant la présence des sommets (vi = 1 si le fragment existe dans le RSG ou 0 sinon), où $|V|$ est le nombre total de sommets dans le graphe RDF G.

2. **Définition des Centroides et des Distances :**
   — Calculer le centroïde de chaque cluster comme la moyenne des vecteurs binaires des RSGs dans le cluster.

— Utiliser la distance euclidienne entre les vecteurs binaires pour définir la distance entre les RSGs et les centroides.

$$d(rsg, rsg') = d(v_{rsg}, v_{rsg'}) = \sum_{i=1}^{|V|} (v_{rsg}^{(i)} - v_{rsg'}^{(i)})^2$$

3. **Clustering K-means :**
   — Assigner aléatoirement les Rooted Sub-Graphs (RSGs) à k partitions initiales.
   — Mettre à jour de manière itérative les centroides en calculant la moyenne des RSGs dans chaque partition, puis réassigner les RSGs aux centroides les plus proches en fonction de la distance euclidienne jusqu'à ce que la convergence soit atteinte.

Chapitre 6

Réalisation

Dans ce chapitre, nous commencerons par présenter les outils et technologies utilisés ainsi que le dataset RDF utilisé dans cette étude. Ensuite, nous décrirons les étapes et le processus de partitionnement du dataset en utilisant les approches MPC, Metis et K-means avec une représentation matricielle des fragments et une représentation des fragments par RSGs. Pour chaque approche appliquée, nous générerons un fichier d'affectation où chaque fragment sera assigné à la partition correspondante.

6.1 Environnement

Pour mettre en œuvre nos systèmes de partitionnement, nous utilisons plusieurs technologies et bibliothèques :

6.1.1 Python3

Python3 [1] est l'une des principales technologies utilisées dans ce projet en raison de sa simplicité, de sa polyvalence et de sa vaste collection de bibliothèques, ce qui en

1. https://docs.python.org/3/

fait un choix idéal pour le traitement des données, l'utilisation d'algorithmes existants et la manipulation de graphes. Les bibliothèques que nous avons utilisées sont :

— **Jupyter** : C'est une application web open-source qui permet de créer et de partager des documents contenant du code en direct, des équations, des visualisations et du texte narratif. Il prend en charge divers langages de programmation, y compris Python, ce qui en fait un outil essentiel pour le calcul interactif et l'analyse des données.

— **NetworkX** : NetworkX [4] est une bibliothèque Python dédiée à la création, à la manipulation et à l'étude de la structure, de la dynamique et des fonctions des graphes complexes. Elle offre une interface intuitive pour travailler avec des graphes non orientés et orientés.

— **Scikit-learn** : C'est une bibliothèque Python puissante et polyvalente pour l'apprentissage automatique, offrant des outils simples et efficaces pour l'analyse et la modélisation des données. Elle inclut une large gamme d'algorithmes et d'utilitaires pour l'apprentissage automatique, y compris des implémentations pour le clustering.

— **Metis** : C'est une bibliothèque de haute performance pour la partition des graphes non structurés et des maillages. Il fournit des algorithmes efficaces pour la partition et le clustering de graphes, particulièrement adaptés aux problèmes de graphes à grande échelle.

— **SciPy** : C'est une bibliothèque Python utilisée pour le calcul scientifique et technique. La sous-bibliothèque scipy.sparse fournit des structures de données optimisées pour stocker des matrices creuses (matrices avec beaucoup de zéros). lil_matrix (List of Lists) et csr_matrix (Compressed Sparse Row) sont deux types de formats de matrices creuses qui permettent des opérations mathématiques et des manipulations efficaces en termes de mémoire et de temps de calcul.

6.1.2 Linux

Linux est un système d'exploitation open source largement utilisé pour le développement de logiciels en raison de sa stabilité et de sa flexibilité. Pour notre projet, nous avons utilisé GCC (GNU Compiler Collection) [12] sous Linux pour compiler et exécuter l'application MPC. GCC, un ensemble de compilateurs pour divers langages de programmation, nous a permis de transformer notre code source en un programme exécutable, facilitant ainsi le développement et les tests de l'application MPC.

6.1.3 Docker

Docker [2] est une plateforme logicielle open source permettant de créer, de déployer et de gérer des applications dans des conteneurs. Docker aide au démarrage et au déploiement d'un programme dans une autre machine complète différente. Nous utilisons Docker pour créer deux images Docker, une pour MPC et l'autre pour K-means adaptatif.

6.2 Dataset

Dans notre projet, nous avons utilisé le dataset WatDiv100M pour évaluer les performances de PQDAG. WatDiv100M est une base de données RDF générée qui contient 100 millions de triplets, conçue pour tester les performances des systèmes de gestion de données RDF à travers une variété de requêtes. Cette base de données a été essentielle pour plusieurs aspects de notre analyse et pour comparer différentes stratégies de partitionnement afin de déterminer la plus efficace.

Le dataset est représenté dans un fichier au format stat, comme le montre la figure 6.1. Pour décrire le contenu de ce fichier :

2. https://docs.docker.com/guides/docker-overview/

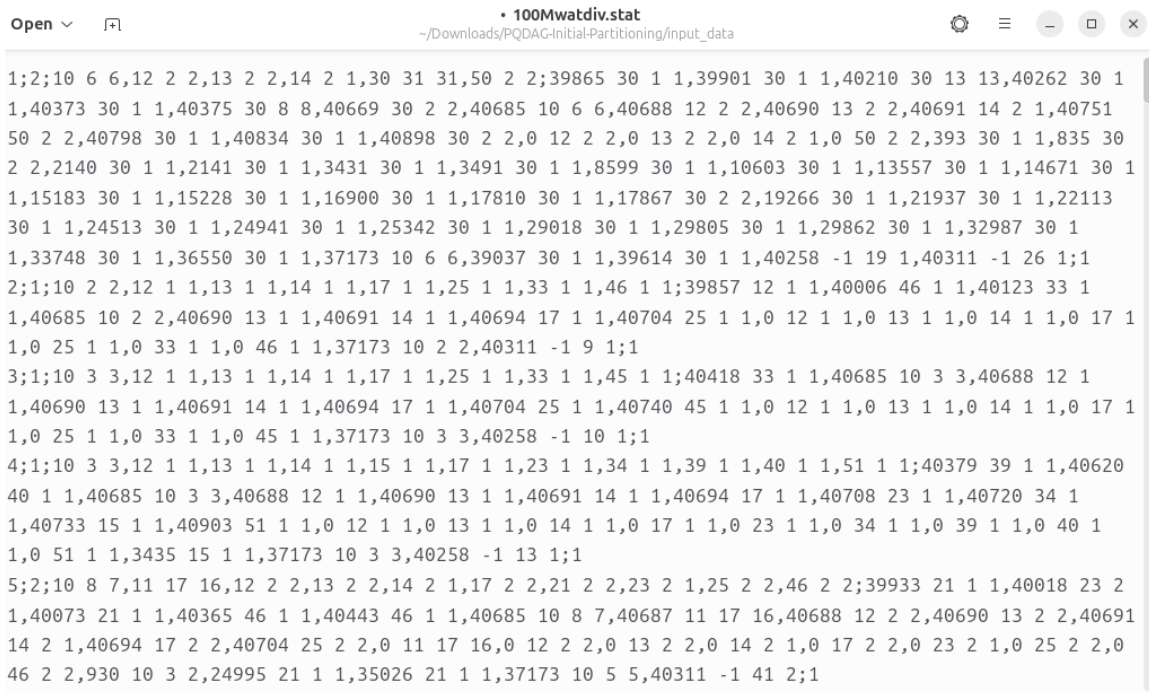


FIGURE 6.1 – Extrait du fichier stat montrant les premières lignes du dataset WatDiv100M.

— Les lignes sont séparées par un retour à la ligne.

— Les colonnes de chaque ligne sont séparées par un point-virgule (";").

— La première colonne contient l'identifiant d'un fragment (sujet).

— Les deuxième et troisième colonnes ne sont pas pertinentes pour notre analyse.

— La quatrième colonne relie l'ID du fragment de la première colonne aux autres fragments.

Plusieurs liens sont présents, chacun séparé par une virgule (","). Chaque lien comprend :

— L'identifiant du fragment cible (objet).

— Le prédicat.

— Le nombre de valeurs de ce prédicat (le poids).

6.3 Présentation de Metis Solution

6.3.1 Création graphe de fragments avec des poids d'arcs

Pour utiliser l'algorithme Metis, il faut d'abord représenter le graphe RDF dans un fichier où chaque ligne représente une arête du graphe sous la forme `poids objet sujet`. Chaque sujet et objet est traité comme un fragment, et une bordure est ajoutée entre eux. Le graphe est construit à l'aide de la bibliothèque Python `networkx`. Voici comment procéder :

```
import networkx as nx
def read_graph_from_file(file_path):
    graph = nx.Graph()
    with open(file_path, 'r') as file:
        for line in file:
            subject, _, obj = line.strip().split()
            graph.add_edge(subject, obj)  # Add edge between subject and object
    return graph
```

6.3.2 Partitionnement avec Metis

Une fois le graphe construit, nous utilisons l'algorithme Metis pour le partitionner en un nombre spécifié de partitions. Metis partitionne le graphe en minimisant les coupures d'arêtes entre les partitions, ce qui réduit les coûts de communication inter-partitions. Voici comment le partitionnement est réalisé :

```
import metis


def partition_graph(graph, num_partitions):
    """Partitions the graph into the specified number of partitions using Metis."""
```

```
    (edgecuts, parts) = metis.part_graph(graph, nparts=num_partitions)
    return parts
```

6.4 Présentation de MPC solution

6.4.1 Création de fichier NT format

L'approche MPC n'accepte que les données RDF sous forme de triplets NT, nous devons donc convertir notre base de données du format stat au format NT. Le format NT (N-Triples) est un format texte simple pour stocker des données RDF. La capture d'écran du code Python dans la figure 6.2 montre comment transférer notre dataset du fichier stat au fichier NT. Chaque ligne représente un triplet composé d'un sujet, d'un prédicat et d'un objet, séparés par des espaces et se terminant par un point. La figure 6.3 montre une partie du fichier résultant de cette étape où les triplets sont générés.

```python
def parse_file(input_file, output_file):
    with open(input_file, 'r') as f_in, open(output_file, 'w') as f_out:
        for line in f_in:
            col = line.strip().split(';')
            if len(col) >= 4:
                fragment_id = col[0]
                links = col[3].split(',')

                for link in links:
                    link_parts = link.strip().split()
                    if len(link_parts) >= 3 and link_parts[1] != '-1':
                        linked_fragment_id = link_parts[0]
                        if linked_fragment_id != '0':  # Exclude linked_fragment_id = 0
                            predicate = link_parts[1]
                            f_out.write(f"<{fragment_id}> <{predicate}> <{linked_fragment_id}> .\n")
```

FIGURE 6.2 – Fonction pour créer un graphe de fragments

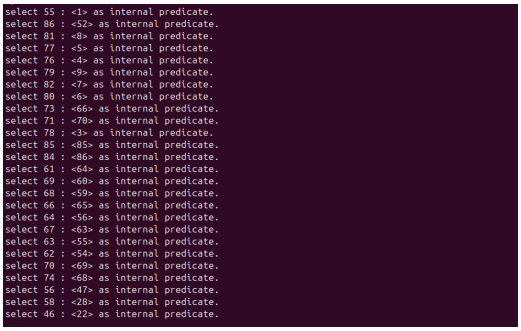
6.4.2 L'exécution de l'algorithme du MPC

MPC utilise le graphe RDF au format NT pour créer les partitions nécessaires. Le nombre de partitions est un paramètre d'entrée. Premièrement, MPC utilise l'al-

```
<1> <30> <39865> .
<1> <30> <39901> .
<1> <30> <40210> .
<1> <30> <40262> .
<1> <30> <40373> .
<1> <30> <40375> .
<1> <30> <40669> .
<1> <10> <40685> .
<1> <12> <40688> .
<1> <13> <40690> .
<1> <14> <40691> .
<1> <50> <40751> .
<1> <30> <40798> .
<1> <30> <40834> .
<1> <30> <40898> .
<1> <30> <393> .
<1> <30> <835> .
<1> <30> <2140> .
<1> <30> <2141> .
<1> <30> <3431> .
<1> <30> <3491> .
```

FIGURE 6.3 – Une partie du graphe de fragments dans au ficher format NT

gorithme 3.5 pour sélectionner les propriétés internes du graphe, comme le montre la figure 6.4a.



(a) Sélection des propriétés internes

(b) Sélection de propriétés croisées

FIGURE 6.4 – Comparaison de sélections de propriétés

Ensuite, MPC identifie les propriétés traversées pour appliquer une opération de coupe sur ces propriétés. La figure 6.4b illustre une partie de ces propriétés. Après découpe, MPC calcule le nombre total des fragments de chaque partition, comme le montre la figure 6.5.

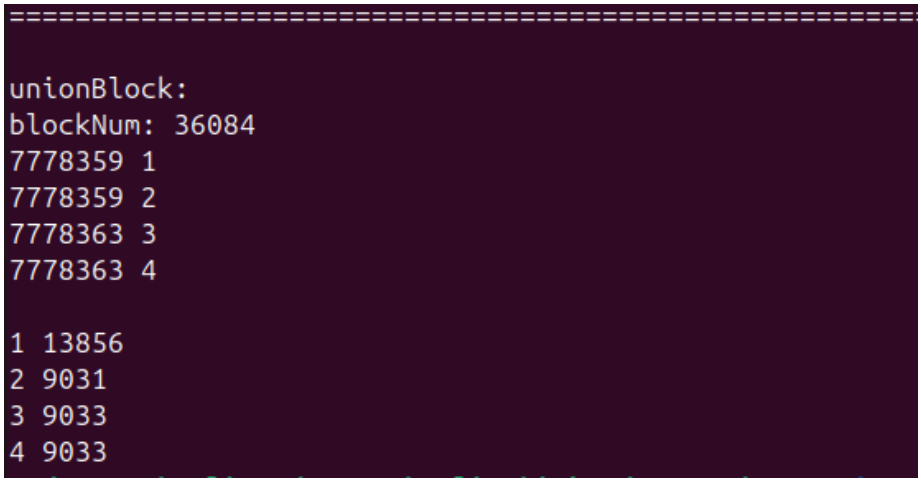


FIGURE 6.5 – Le nombre total des fragments pour chaque partition

Au final, MPC produit deux fichiers :

— **Le fichier OutputInternalPoints, ou fichier d'affectation** : Chaque ligne de ce fichier contient deux colonnes d'informations. La première colonne représente l'identifiant du fragment, et la seconde indique l'identifiant de la partition (de zéro jusqu'au nombre de partitions moins un). Ce fichier est utilisé pour créer les partitions et les distribuer sur les différentes machines du cluster. La figure 6.6a illustre une partie de ce fichier.

```
<1>     3
<39865> 0
<39901> 1
<40210> 1
<40262> 1
<40373> 3
<40375> 1
<40669> 2
<40685> 2
<40688> 3
```

(a) Une partie du fichier "outputInternalPoints"

```
<52>    15      0
<85>    107     0
<53>    30912   1
<5>     80      0
<2>     136667  1
<3>     1110    0
<68>    1569    0
<67>    43856   1
<59>    2080    0
<8>     26      0
```

(b) Une partie du fichier "outputcrossingEdges"

FIGURE 6.6 – Comparaison de parties de fichiers

— **Le fichier OutputcrossingEdges** : Chaque ligne de ce fichier comporte trois colonnes. La première colonne contient l'identifiant du prédicat, la deuxième le nombre d'occurrences de ce prédicat, c'est-à-dire le nombre de triplets qui utilisent ce prédicat, et la troisième est une valeur booléenne indiquant si ce prédicat a été sélectionné comme propriété croisée ou non. La figure 6.6b illustre une partie de ce fichier.

6.5 Présentation de K-means avec une représentation matricielle Solution

6.5.1 Lecture des données RDF

La première étape consiste à lire les données RDF du fichier fragments NT.

```
interactions = []
with open(file_fragments_path, 'r') as file:
    for line in file:
        parts = line.strip().split()
        if len(parts) != 4:
            continue  # Skip any malformed lines
        subject = parts[0].strip('<>')
        predicate = parts[1].strip('<>')
        object = parts[2].strip('<>')
        interactions.append((subject, object, predicate))
print("Lecture des interactions terminée.")
```

Cette étape nous permet de récupérer les triplets RDF et de les stocker sous forme

de tuples (sujet, objet, prédicat).

6.5.2 Création des vecteurs de caractéristiques

Ensuite, nous avons créé des vecteurs de caractéristiques pour chaque fragment du graphe RDF. Chaque fragment est représenté par une étoile de données associée à chaque sujet ou objet. Nous avons utilisé des matrices creuses (*sparse matrices*) pour stocker ces vecteurs efficacement en termes de mémoire.

```
fragments = set()
for subject, object, _ in interactions:
    fragments.add(subject)
    fragments.add(object)
fragments = list(fragments)
fragment_to_index = {fragment: idx for idx, fragment in enumerate(fragments)}
n = len(fragments)
features = lil_matrix((n, n))  # Use lil_matrix for efficient construction
predicate_to_index = {}  # To convert predicates to numerical indices
predicate_counter = 1
for subject, object, predicate in interactions:
    i, j = fragment_to_index[subject], fragment_to_index[object]
    # Assign a unique index to each predicate if not already done
    if predicate not in predicate_to_index:
        predicate_to_index[predicate] = predicate_counter
        predicate_counter += 1
    p_index = predicate_to_index[predicate]
```

```
    features[i, j] = p_index
    features[j, i] = p_index  # Assuming the graph is undirected
```

6.5.3 Partitionnement avec K-means

Nous avons ensuite utilisé l'algorithme K-means. Le nombre de clusters (ou partitions) est défini en fonction du nombre de machines disponibles pour le traitement distribué.

```
num_clusters = 10  # Nombre de machines
print(f"Partitionnement des fragments en {num_clusters} clusters avec K-means")
kmeans = KMeans(n_clusters=num_clusters, n_init=20, random_state=42)
kmeans.fit(features)
labels = kmeans.labels_
affectations = {fragments[i]: labels[i] for i in range(len(fragments))}
```

L'algorithme K-means attribue chaque fragment à un cluster spécifique. Le résultat est une affectation des fragments à différentes partitions.

6.6 Présentation de K-means avec une représentation des fragments par RSGs Solution

Dans cette section, nous avons mis en place une méthodologie pour générer des sous-graphes RSGs à partir de notre dataset. L'extrait de code des figures 6.7 et 6.8 montre la création et le traitement des triplets RDF, la construction du graphe

```python
def build_graph(triples):
    adjacency_list = defaultdict(list)
    vertices = set()
    source_vertices = set()

    for s, p, o in triples:
        adjacency_list[s].append(o)
        vertices.add(s)
        vertices.add(o)
        source_vertices.add(s)  # Treat all subjects as source vertices

    print(f"Source vertices (roots): {source_vertices}")

    return adjacency_list, source_vertices

def initialize_vertices(vertices, source_vertices):
    out = {}
    for v in vertices:
        if v in source_vertices:
            out[v] = (v, v)  # Root vertex is itself
        else:
            out[v] = (v, None)  # Non-root vertices
    return out

def propagation_step(adjacency_list, out):
    new_out = {}
    for v, (vm, r) in out.items():
        new_out[v] = (vm, r)
        if r is not None:  # If the vertex is active
            neighbors = adjacency_list.get(v, [])
            for w in neighbors:
                if w in out:
                    w_vm, w_r = out[w]
                    new_vm = min(vm, w_vm)
                    new_out[w] = (new_vm, r)
                else:
                    new_out[w] = (vm, r)
    return new out
```

FIGURE 6.7 – Le code de génération des RSGs 1

d'adjacence et la propagation des racines à travers les sommets pour identifier les sous-graphes enracinés (RSGs).

Tout d'abord, nous construisons une liste d'adjacence pour représenter le graphe et identifions les sommets sources, c'est-à-dire ceux sans prédécesseurs. En initialisant chaque sommet avec sa propre valeur minimale et sa racine, nous utilisons des étapes de propagation pour diffuser ces valeurs à travers le graphe. À chaque itération, les valeurs sont mises à jour pour refléter les connexions minimales, jusqu'à ce que les valeurs convergent.

Finalement, les sommets sont regroupés en sous-graphes enracinés (RSGs) en

```python
def update_vertices(out):
    updated_out = {}
    for v, (vm, r) in out.items():
        updated_out[v] = (min(vm, vm), r)
    return updated_out

def generate_rsgs(triples):
    adjacency_list, source_vertices = build_graph(triples)
    vertices = set(adjacency_list.keys())

    out = initialize_vertices(vertices, source_vertices)

    i = 0
    while True:
        new_out = propagation_step(adjacency_list, out)
        if new_out == out:
            break
        out = update_vertices(new_out)
        i += 1
        #print(f"Iteration {i}: {out}")

    rsgs = defaultdict(set)
    for v, (vm, r) in out.items():
        if r is not None:
            rsgs[r].add(v)
    return rsgs

def main(rdf_file):
    triples = parse_nt_file(rdf_file)
    rsgs = generate_rsgs(triples)
    return rsgs
```

FIGURE 6.8 – Le code de génération des RSGs 2

fonction de leurs racines identifiées, comme illustré dans la figure 6.9.

Après cela, nous appliquons l'algorithme de K-means sur les RSGs pour les partitionner en plusieurs clusters. Ce processus implique de calculer les centroïdes des partitions et de réaffecter les RSGs aux clusters les plus proches, en itérant jusqu'à convergence. Cela permet de regrouper les sous-graphes de manière optimale,

```
RSG 2086: {'40228', '2220', '3441'}
RSG 4185: {'40339', '4185', '39860'}
RSG 36755: {'4463', '4233', '40750'}
RSG 4864: {'40647', '4864', '4836'}
RSG 29471: {'40455', '39336', '38157', '37457', '31197', '38331', '40518', '40871', '35352', '29471', '15962', '27484', '39549',
279', '29902', '22130', '11341', '4867', '32848', '3331', '40778', '40017', '40603', '32821', '9837'}
RSG 40060: {'4870', '38580', '285', '35402', '29261', '38565', '40060', '28594', '20483', '16798', '39033', '26436', '23527'}
RSG 20194: {'40003', '5091', '20194'}
RSG 17462: {'34814', '35282', '25874', '30055', '40806', '5127', '40590', '40485'}
RSG 10947: {'14723', '21062', '34994', '35734', '22095', '11717', '40438', '22148', '17850', '5397'}
RSG 5982: {'5982', '40097'}
RSG 34645: {'40523', '6794', '11458', '34645', '39876', '35659'}
RSG 40166: {'40166', '8437'}
RSG 9035: {'9035', '40492'}
RSG 6760: {'9082', '40329', '9226', '34349', '5282'}
RSG 34987: {'26103', '9345', '27509', '11561', '40945', '34987', '17853', '26670', '35019', '16719'}
RSG 128: {'23973', '9346', '40080', '40804', '26261'}
RSG 2654: {'40120', '40949', '25490', '22174', '12036', '10588'}
RSG 10608: {'40946', '40213', '40860', '39956', '14905', '10608', '20954'}
RSG 40242: {'10611', '40242'}
RSG 40320: {'11029', '40320'}
```

FIGURE 6.9 – Un exemple des sous-graphes RSGs

6.7 Étude expérimentale

Pour évaluer nos stratégies de partitionnement, nous avons exécuté 17 requêtes SPARQL en utilisant notre triplestore distribué PQDAG sur le base de données Wat-Div100M pour chaque approche de partitionnement, à savoir Metis, Weighted Metis, K-means et MPC. Ensuite, nous avons comparé leurs temps d'exécution. Les expériences ont été réalisées en utilisant 10 machines (workers).

Stratégie	**Meilleur temps d'exécution**
Standard Metis	1
Weighted Metis	2
K-means	9
MPC	5

TABLE 6.1 – Comparaison des meilleurs temps d'exécution pour chaque stratégie

Comme le montre le tableau 6.1, les résultats indiquent que la stratégie K-means a obtenu les meilleurs temps d'exécution pour 9 des 17 requêtes SPARQL, démontrant ainsi son efficacité supérieure par rapport aux autres approches. La stratégie MPC a également montré des performances remarquables, avec les meilleurs temps d'exécution pour 5 des requêtes. Les stratégies Standard Metis et Weighted Metis ont obtenu

les meilleurs temps d'exécution pour 1 et 2 requêtes respectivement. Ces résultats montrent clairement que le partitionnement basé sur K-means offre une performance nettement améliorée en termes de temps d'exécution des requêtes SPARQL sur le dataset WatDiv100M.

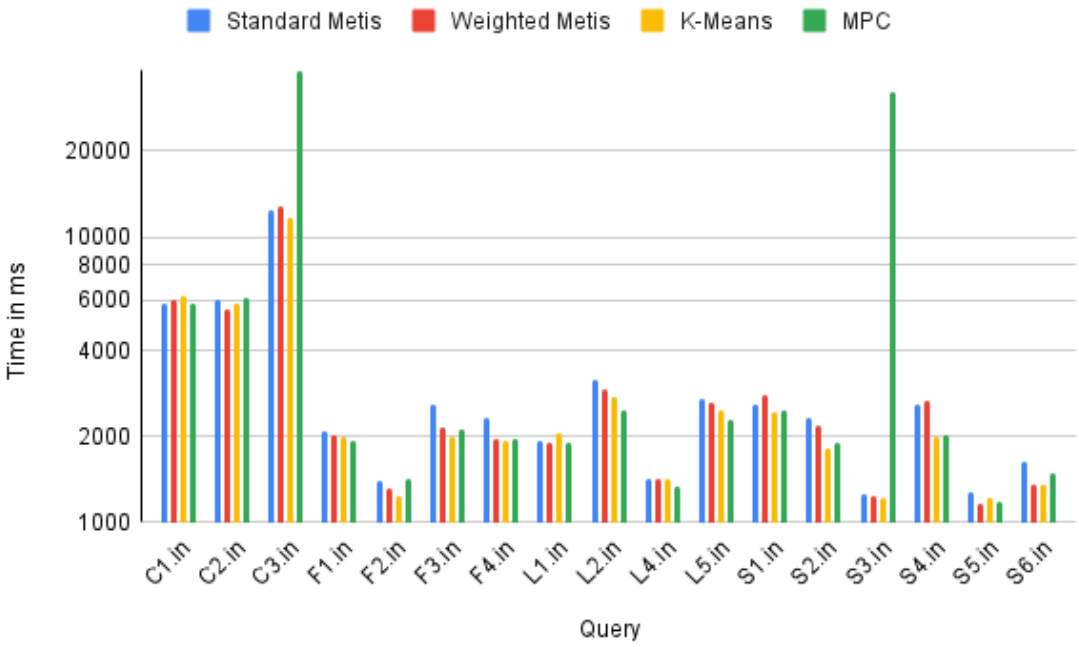


FIGURE 6.10 – Comparaison des performances

6.7.1 K-Means

L'algorithme de partitionnement k-means pour les données RDF a montré des résultats prometteurs par rapport à d'autres approches comme MPC et METIS. En regroupant les sous-graphes enracinés (RSGs) en fonction de leur similarité, k-means réduit la redondance des données en minimisant le nombre de triplets dupliqués. Cela permet une utilisation plus efficace de la mémoire et des ressources de traitement.

De plus, k-means améliore la localisation des requêtes en répartissant intelligem-

Query	Standard Metis	Weighted Metis	K-Means	MPC
C1.in	**5808**	6012	6238	5821
C2.in	5993	**5610**	5870	6076
C3.in	12491	12718	**11665**	38099
F1.in	2096	2024	2003	**1943**
F2.in	1405	1315	**1233**	1419
F3.in	2595	2150	**1999**	2115
F4.in	2315	1967	**1942**	1955
L1.in	1928	1918	2047	**1900**
L2.in	3171	2907	2761	**2472**
L4.in	1429	1418	1413	**1330**
L5.in	2699	2614	2488	**2295**
S1.in	2586	2798	**2438**	2475
S2.in	2341	2190	**1818**	1901
S3.in	1253	1237	**1218**	32118
S4.in	2598	2671	**2010**	2022
S5.in	1279	**1166**	1229	1188
S6.in	1624	1361	**1354**	1497

TABLE 6.2 – Comparaison des temps d'exécution en ms des requêtes SPARQL pour différentes stratégies de partitionnement

ment les RSGs sur les nœuds de calcul, maximisant ainsi la probabilité de traitement local des requêtes SPARQL sans nécessiter de communication entre les machines. Cette stratégie réduit les temps de traitement des requêtes et la surcharge du réseau, ce qui est crucial pour les systèmes de gestion de données distribués.

Enfin, l'algorithme k-means équilibre la charge de travail en s'assurant que chaque partition contient un nombre approximativement égal de RSGs. Cet équilibrage est essentiel pour éviter la surcharge de certains nœuds de calcul et garantir une répartition uniforme des tâches, améliorant ainsi la scalabilité et les performances globales du système.

6.7.2 MPC

L'approche MPC permet d'obtenir un temps d'exécution intéressant pour le traitement distribué de requêtes SPARQL grâce à plusieurs facteurs clés. Tout d'abord, MPC minimise les jointures inter-partitions en sélectionnant les propriétés internes et en agrégeant le graphe RDF avant le partitionnement. Cette réduction des jointures inter-partitions se traduit par un traitement des requêtes plus efficace.

De plus, MPC identifie les types de requêtes pouvant être exécutées indépendamment, notamment les requêtes internes et les requêtes étendues à exécutabilité indépendante. Grâce à cette catégorisation des requêtes, MPC peut éviter les jointures inter-partitions plus efficacement que les autres approches existantes, ce qui se traduit par de meilleures performances de requêtes.

Chapitre 7

Conclusion

7.1 Rétrospective

Avec l'augmentation exponentielle du volume de données sur le web au fil des ans, le traitement de ces Big Data est devenu une nécessité. De nombreux chercheurs ont développé des technologies pour traiter ces données en parallèle. Les bases de données relationnelles, traditionnellement utilisées pour stocker les données, ne peuvent plus faire face à la taille et à la complexité massives des Big Data. Notre recherche s'est concentrée sur la gestion des données structurées sous forme de graphes.

Bien que RDF_QDAG excelle dans la gestion des données de graphe massives avec des capacités de requête intégrées, il souffre de transferts réseau excessifs. Sa version distribuée, PQDAG, nécessite deux améliorations clés : des méthodes de transfert de données efficaces et une stratégie de partitionnement qui minimise les jointures entre partitions. Cela réduira considérablement le temps de communication réseau.

Pour analyser cela, nous avons exploré diverses stratégies de partitionnement pour les bases de données RDF afin d'optimiser les performances de traitement des requêtes SPARQL. Nous avons commencé par partitionner les données en utilisant différentes approches, y compris MPC, Metis et K-means. La comparaison entre ces

différentes approches de partitionnement était cruciale pour déterminer la méthode la plus efficace offrant les meilleurs temps de réponse pour le traitement des requêtes. Notre évaluation a révélé que l'approche K-means surpassait les autres en termes de temps de réponse, grâce à sa capacité à mieux regrouper les RSGs, ce qui minimise les duplications et optimise les jointures lors des requêtes.

Par ailleurs, l'intégration de l'intelligence artificielle (IA), notamment du machine learning (clustering), dans ces recherches pourrait ouvrir de nouvelles perspectives pour optimiser davantage les processus de partitionnement et d'analyse des données. L'utilisation d'algorithmes de clustering tels que K-means pourrait améliorer la précision et l'efficacité du traitement du Big Data, rendant les systèmes encore plus efficaces et adaptatifs aux besoins évolutifs du domaine.

7.2 Perspectives

En termes de perspectives, ce projet de fin d'études a fourni des solutions prometteuses pour le partitionnement et l'optimisation des requêtes SPARQL dans les bases de données RDF massives. Cependant, plusieurs défis restent à relever et des pistes d'amélioration doivent être explorées, comme l'optimisation des transferts de données, des protocoles de communication plus efficaces, et le renforcement de la scalabilité et de la tolérance aux pannes de PQDAG, ce qui est essentiel pour garantir la robustesse du système à très grande échelle.

De plus, l'apprentissage automatique pourrait offrir de nombreuses approches de partitionnement supplémentaires, adaptées à des types de données spécifiques ou à des charges de travail variées, et révolutionner le partitionnement et l'optimisation des requêtes en prédisant les schémas de requêtes et en ajustant dynamiquement les partitions.

Enfin, des évaluations rigoureuses et des benchmarks comparatifs avec d'autres

solutions de pointe permettront de mieux comprendre les performances et d'identifier les axes d'amélioration. En somme, bien que des progrès significatifs aient été réalisés, de nombreuses opportunités de recherche et d'optimisation demeurent pour relever les défis croissants du Big Data sur le web.

Bibliographie

[1] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Sw-store : a vertically partitioned dbms for semantic web data management. *The VLDB Journal*, 18 :385–406, 2009.

[2] B. DuCharme. *Learning SPARQL : querying and updating with SPARQL 1.1.* " O'Reilly Media, Inc.", 2013.

[3] O. Erling and I. Mikhailov. Virtuoso : Rdf support in a native rdbms. In *Semantic web information management : a model-based perspective*, pages 501–519. Springer, 2009.

[4] A. Hagberg and D. Conway. Networkx : Network analysis with python. *URL : https ://networkx. github. io*, 2020.

[5] A. Harth, J. Umbrich, A. Hogan, and S. Decker. Yars2 : A federated repository for querying graph structured data from the web. In *International Semantic Web Conference*, pages 211–224. Springer, 2007.

[6] G. Karypis and V. Kumar. Metis : A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.

[7] A. Khelil. *Gestion et optimisation des données massives issues du Web*. PhD thesis, Chasseneuil-du-Poitou, Ecole nationale supérieure de mécanique et d . . ., 2020.

[8] A. Khelil, A. Mesmoudi, J. Galicia, L. Bellatreche, M.-S. Hacid, and E. Coquery. Combining graph exploration and fragmentation for scalable rdf query processing. *Information Systems Frontiers*, 23(1) :165–183, 2021.

[9] K. Krishna and M. N. Murty. Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(3) :433–439, 1999.

[10] O. Lassila and R. R. Swick. Resource description framework (rdf) model and syntax specification. W3C Recommendation REC-rdf-syntax-19990222, W3C, 1999.

[11] B. McBride. Jena : A semantic web toolkit. *IEEE Internet computing*, 6(6) :55–59, 2002.

[12] D. Novillo. Gcc an architectural overview, current status, and future directions. In *Proceedings of the linux symposium*, volume 2, page 185, 2006.

[13] P. Peng, M. T. Özsu, L. Zou, C. Yan, and C. Liu. Mpc : Minimum property-cut rdf graph partitioning. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 192–204. IEEE, 2022.

[14] T. Project and Z. Kashe. Partial parallelization of graph partitioning algorithm metis. 2004.

[15] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. W3C Recommendation 15 January 2008, 2008.

[16] K. Rohloff and R. E. Schantz. High-performance, massively scalable distributed systems using the mapreduce software framework : the shard triple-store. In *Programming support innovations for emerging distributed applications*, pages 1–5. 2010.

[17] C. Weiss, P. Karras, and A. Bernstein. Hexastore : sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment*, 1(1) :1008–1019, 2008.

[18] K. Wilkinson, C. Sayers, H. A. Kuno, D. Reynolds, et al. Efficient rdf storage and retrieval in jena2. In *SWDB*, volume 3, pages 131–150. Citeseer, 2003.

[19] B. Wu, Y. Zhou, P. Yuan, H. Jin, and L. Liu. Semstore : A semantic-preserving distributed rdf triple store. In *Proceedings of the 23rd acm international conference on conference on information and knowledge management*, pages 509–518, 2014.

[20] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang. A distributed graph engine for web scale rdf data. *Proceedings of the VLDB Endowment*, 6(4) :265–276, 2013.

[21] I. Zouaghi. *Optimisation basée sur les graphes pour une évaluation efficace et scalable des requêtes*. PhD thesis, Chasseneuil-du-Poitou, Ecole nationale supérieure de mécanique et d ..., 2022.

Résumé

Avec la croissance explosive des technologies web et l'avènement de l'ère numérique, le volume de données générées et partagées sur le web a atteint des niveaux sans précédent. Cette explosion de la création de données pose des défis significatifs en matière de gestion, de traitement et d'analyse de cet afflux massif d'informations. Cette thèse s'inscrit dans le cadre du projet PQDAG, qui vise à développer des outils avancés de gestion des données spécifiquement conçus pour assurer la mise à l'échelle et les performances lors du traitement des Big RDF Data. Notre objectif principal était de proposer des approches de partitionnement facilitant l'évaluation d'un grand nombre de requêtes SPARQL sans nécessiter de jointures entre les partitions, réduisant ainsi le temps de communication sur le réseau. Pour ce faire, nous avons appliqué et comparé plusieurs stratégies de partitionnement, y compris MPC (Minimum Property Cut), Metis et K-means. Le but était de minimiser le nombre de jointures inter-partitions et d'améliorer les performances de traitement des requêtes dans un environnement distribué, tout en réduisant les coûts de communication entre les partitions. Grâce à cette recherche, nous visons à identifier la solution la plus efficace pour relever les défis de la gestion des Big Data dans le cadre du projet PQDAG, améliorant ainsi la capacité à gérer et analyser les données RDF à grande échelle sur le web en parallèle.

Mots-clès : Langage SPARQL, Triplestores, PQDAG, Partitionnement, MPC, RDF_QDAG, Optimisation, K-means

Abstract

With the explosive growth of web technologies and the advent of the digital age, the volume of data generated and shared on the web has reached unprecedented levels. This explosion in data creation presents significant challenges in managing, processing, and analyzing this massive influx of information. This thesis is part of the PQDAG project, which aims to develop advanced data management tools specifically designed to ensure scalability and performance when processing Big RDF Data. Our main objective was to propose partitioning approaches that facilitate the evaluation of a large number of SPARQL queries without requiring joins between partitions, thus reducing network communication time. To achieve this, we applied and compared several partitioning strategies, including MPC (Minimum Property Cut) , Metis and K-means. The goal was to minimize the number of inter-partition joins and improve query processing performance in a distributed environment while reducing communication costs between partitions. Through this research, we aim to identify the most efficient solution to address the challenges of Big Data management within the scope of the PQDAG project, thereby enhancing the ability to handle and analyze large-scale RDF data on the web in parallel.

Keywords : SPARQL language, Triplestores, PQDAG, Partitioning, MPC, RDF_QDAG, Optimization, K-means

ملخص

مع النمو الهائل لتقنيات الويب وظهور العصر الرقمي، بلغ حجم البيانات التي يتم إنشاؤها ومشاركتها على الويب مستويات غير مسبوقة. إن هذا الانفجار في إنشاء البيانات يطرح تحديات كبيرة في إدارة ومعالجة وتحليل هذا الكم الهائل من المعلومات. تندرج هذه الأطروحة ضمن مشروع PQDAG، الذي يهدف إلى تطوير أدوات متقدمة لإدارة البيانات مصممة خصيصًا لضمان القابلية للتوسع والأداء عند معالجة بيانات RDF الكبيرة. كان هدفنا الرئيسي هو اقتراح طرق لتجزئة البيانات تسهل تقييم عدد كبير من استعلامات SPARQL دون الحاجة إلى عمليات الربط بين الأجزاء، مما يقلل من وقت الاتصال بالشبكة. لتحقيق ذلك، طبقنا وقارننا عدة استراتيجيات للتجزئة، بما في ذلك MPC و Metis و K-means. كان الهدف هو تقليل عدد عمليات الربط بين الأجزاء وتحسين أداء معالجة الاستعلامات في بيئة موزعة، مع تقليل تكاليف الاتصال بين الأجزاء. من خلال هذا البحث، نسعى إلى تحديد الحل الأكثر كفاءة لمواجهة تحديات إدارة البيانات الكبيرة ضمن نطاق مشروع PQDAG، مما يعزز القدرة على إدارة وتحليل بيانات RDF واسعة النطاق على الويب بشكل متوازي.

الكلمات المفتاحية : لغة Sparql, متجر ثلاثي, PQDAG, التقسيم, MPC, RDF_QDAG, تحسين, K-means