Gestion et optimisation des données massives issues du Web

Abdallah Khelil

▶ **To cite this version:**

Abdallah Khelil. Gestion et optimisation des données massives issues du Web. Autre [cs.OH]. ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique - Poitiers; Université d'Oran, 2020. Français. NNT : 2020ESMA0009 . tel-03079086

HAL Id: tel-03079086

https://theses.hal.science/tel-03079086

Submitted on 17 Dec 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE en cotutelle

pour l'obtention du Grade de

DOCTEUR DE L'ÉCOLE NATIONALE SUPÉRIEURE DE MÉCANIQUE ET D'AÉROTECHNIQUE

(Diplôme National – Arrêté du 25 mai 2016)

Ecole Doctorale : Sciences et Ingénierie des Systèmes, Mathématiques, Informatique

Secteur de Recherche : Informatique et Applications

Et

de l'Université d'Oran 1 Ahmed BENBELLA

Présentée par :

Abdallah KHELIL

****

Gestion et Optimisation des Données Massives Issues du Web

****

Directeurs de Thèse : **Ladjel BELLATRECHE et Mohamed SENOUCI**
Co-directeur : **Amin MESMOUDI**

****

Soutenue le 08/10/2020
devant la Commission d'Examen

****

JURY

Président	**M. Yamine AIT AMEUR**	Professeur, INPT-ENSEEIHT, Toulouse, France
Rapporteurs :	**Mme Fatima DEBBAT**	Professeur, Université de Mascara M. STAMBOULI, Algérie
	M. Djamal BENSLIMANE	Professeur, Université Claude Bernard Lyon 1, France
Membres du jury :	**M. Ghalem BELALEM**	Professeur, Université Oran1, Ahmed BENBELLA, Algérie
	M. Mohamed SENOUCI	Professeur, Université Oran1, Ahmed BENBELLA, Algérie
	M. Amin MESMOUDI	Maître de conférences, Université de Poitiers, France
	M. Ladjel BELLATRECHE	Professeur, ISAE-ENSMA, Poitiers, France

Remerciement

Soyons reconnaissants aux personnes qui nous donnent du bonheur ; elles sont les charmants jardiniers par qui nos âmes sont fleuries

— Marcel Proust

A l'issue de la rédaction de ce manuscrit, je suis convaincu que la thèse est loin d'être un travail solitaire. En effet, je n'aurais jamais pu réaliser ce travail doctoral sans le soutien de plusieurs personnes que je voudrais, à travers ces quelques lignes, remercier du fond du cœur.

Les travaux présentés dans cette thèse ont fait l'objet d'une convention de cotutelle entre l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique(ISAE-ENSMA) et l'Université d'Oran1 - Ahmed Benbella, Algérie.

En premier lieu, je tiens à remercier mon directeur de thèse, M. Ladjel BELLATRECHE pour la confiance qu'il m'a accordée en acceptant d'encadrer ce travail de thèse.

Je souhaite remercier aussi mon directeur de thèse du côté algérien, M. Mohamed SENOUCI, Professeur des Universités et directeur de l'Ecole Nationale Polytechnique d'Oran pour avoir accepté à codiriger cette thèse en cotutelle.

Je remercie très chaleureusement mon co-encadrant de thèse, M. Amin MESMOUDI, Maître de conférences du Laboratoire d'Informatique et d'Automatique pour les Systèmes, pour son attention de tout instant sur mes travaux, pour ses conseils avisés et son écoute qui ont été prépondérants pour la bonne réussite de cette thèse. Son énergie et sa confiance ont été des éléments moteurs pour moi. J'ai pris un grand plaisir à travailler avec lui.

J'exprime tous mes remerciements aussi à l'ensemble des membres de mon jury : Professeure Fatima DEBBAT de l'université de Mustapha STAMBOULI, Mascara et Professeur Djamel BENSLIMANE de l'Université Claude Bernard, Lyon d'avoir rapporté mon manuscrit, et Professeur Yamine AIT AMEUR de INPT-ENSEEIHT, Toulouse et Professeur Ghalem BELALEM de l'Université Oran1,Ahmed BENBELLA,d'avoir examiné le travail effectué durant ma thèse.

Mes remerciement vont également : Au directeur adjoint du laboratoire LIAS professeur Emmanuel GROLLEAU pour ses qualités humaines hors du commun, son écoute et son dévouement, et pour son aide et sa sympathie.

Au professeur Allel HadjAli de l'ISAE-ENSMA, pour ses conseils qui m'ont été extrêmement précieux.

Je souhaite adresser mes remerciements à tous les membres du laboratoire LIAS qui ont facilité mon intégration.

Par ailleurs, je tiens également à adresser mes sincères remerciements aux personnes non encore citées dont les permanents, les anciens et futurs docteurs du LIAS ainsi que les stagiaires pour leur soutien mutuel, leurs encouragements, leurs savoirs scientifiques et leurs échanges amicaux pendant ces années de dur labeur.

Je souhaite remercier spécialement M. Sidahmed MAKHLOUF de l'université d'Oran 1 pour son soutien tout au long de la thèse.

Mes remerciements vont aussi à ma famille et mes amis qui, avec cette question récurrente, "quand est-ce que tu la soutiens cette thèse ?", bien qu'angoissante en période fréquente de doutes,m'ont permis de ne jamais dévier de mon objectif final.

Ces remerciements ne peuvent s'achever, sans une pensée pour mon premier fan : mon père. Sa présence et ses encouragements sont pour moi les piliers fondateurs de ce que je suis et de ce que je fais.

A mes très chers parents
A mon épouse pour son soutien irremplaçable
A mon fils Abdelbacet
A mes frères et à ma sœur
A toute ma famille ainsi qu'à mes amis

Table des matières

Table des matières **7**

Liste des figures **9**

Liste des tableaux **13**

1 Introduction Générale **15**
1.1 Contexte de la Thèse 15
1.2 Notre Démarche et Contributions 23
1.3 Organisation du manuscrit 25
1.4 Publications 27

I État de l'Art **29**

2 Le monde des données ouvertes liées **31**
2.1 Le Web sémantique 33
2.2 Données Ouvertes Liées 34
2.2.1 Freebase 36
2.2.2 DBpedia 36
2.2.3 Wikidata 37
2.3 Le modèle de données RDF 39
2.3.1 Syntaxe RDF 43
2.3.2 Le schéma RDF 45
2.4 Langages de requête RDF 50
2.4.1 Le langage de requête SPARQL 50
2.4.2 Évaluation de requêtes 54
2.5 Conclusion 55

3 Panorama des Systèmes de Stockage et de Traitement des LOD **57**
3.1 Évolution de la gestion de données 58
3.2 Stockage des données RDF 61
3.2.1 Traitement non-natif de RDF 62
3.2.2 Le stockage natif de RDF 68
3.2.3 Traitement natif de RDF 68
3.3 Systèmes RDF distribués 73
3.3.1 Propriétés des systèmes distribués 74
3.3.2 Avantages des systèmes distribués 75

3.3.3 Limites des systèmes distribués 76
3.3.4 Les systèmes homogènes 76
3.3.5 Les systèmes hétérogènes 78
3.3.6 Traitement massivement parallèle pour RDF 80
3.3.7 Fragmentation des données 84
3.3.8 Allocation des fragments 85
3.4 Évaluation des systèmes de gestion des LOD 86
3.5 Évaluation expérimentale des systèmes existants 89
3.6 Conclusion 90

II L'approche RDF_QDAG 91

4 L'approche RDF QDAG : bases théoriques, mise en œuvre et optimisation 93
4.1 Bases théorique de l'évaluation de requêtes 94
4.1.1 Représentation logique de données 94
4.1.2 Représentation logique de requêtes 96
4.1.3 Évaluation de requêtes 99
4.2 Mise en oeuvre et optimisation 103
4.2.1 Stockage de graphes RDF 103
4.2.2 Opérations d'évaluation de requêtes 107
4.2.3 Adaptation du modèle Volcano 113
4.2.4 Traitement des requêtes wildcard 117
4.2.5 Modules complémentaires 118
4.2.6 Exemple d'exécution 119
4.3 Conclusion 125

5 L'approche RDF QDAG : Validation expérimentale 127
5.1 Introduction 129
5.2 Environnement d'expérimentations 129
5.3 Évaluation du pré-traitement 130
5.4 Performances des requêtes 130
5.4.1 Basic Graph Pattern queries 131
5.4.2 Aggregations 133
5.4.3 Requêtes Wildcard 133
5.5 RDF_QDAG vs. les systèmes parallèles 135
5.6 Conclusion 139

III Conclusion et perspectives 141

6 Conclusion & Perspectives 143
6.1 Conclusion 143
6.2 Perspectives 146

Bibliographie 149

Liste des figures

1.1 Évolution du cloud LoD de mai 2007 à mars 2019 19
1.2 Plan de thèse 26

2.1 Le "layer cake" du Web sémantique 34
2.2 Évolution du cloud $\mathcal{LOD}$ de 2007 à 2019 38
2.3 Exemple de graphe RDF 42
2.4 Syntaxe concrète RDF (RDF/XML) 44
2.5 Syntaxe concrète RDF (N-triple) 46
2.6 Syntaxe concrète RDF (N3) 47
2.7 Syntaxe concrète RDF (Turtle) 48
2.8 Syntaxe concrète RDF (RDFa) 49
2.9 Exemple de requêtes SPARQL 53

3.1 Architecture du System R 60
3.2 Approches de stockage RDF 62
3.3 Table de triplets 64
3.4 Table de propriétés 66
3.5 Tables binaires 67
3.6 Taxonomie des systèmes RDF distribués 74

4.1 Exemple de découpage de données 97
4.2 Exemple de treillis 98
4.3 Flux d'exécution 103
4.4 Exemple de stockage sur disque 106
4.5 Illustration de la méthode de compression 108
4.6 Apperçu d'un arbre B+. 109
4.7 Exemples de plan d'exécution de requête 114
4.8 Arbre de décisions 121
4.9 Exemple d'un plan physique d'exécution 122
4.10 Arbre de décisions (suite) 123
4.11 Schéma d'exécution 123
4.12 Traitement des filtres 124
4.13 Flux de traitement de requêtes et de données 124

5.1 Requêtes BGP sur Yago2 (Temps logarithmique) 133
5.2 Résultat aggrégations - WatDiv100M 134
5.3 Résultat aggrégations - WatDiv1B 134

5.4 Résultats des requêtes BGP sur LUBM 20M 136
5.5 Résultats des requêtes BGP sur LUBM100M 137
5.6 Résultats des requêtes BGP sur WatDiv 100M 138
5.7 Résultats de requêtes BGP sur YAGO 139
5.8 Résultats des requêtes BGP (DBLP) avec systèmes distribués 139

Liste des tableaux

1.1 Taille et nombre de triplets des jeux de données 19
1.2 Résultats expérimentaux des systèmes 24

2.1 Table de triplets RDF 41
2.2 Un exemple d'appariement de graphe 55

3.1 Résultats expérimentaux de comparaison 90

4.1 Exemple : Étoiles de données 95
4.2 Exemple : Etoiles de requêtes 99
4.3 Résultats de la requêtes 99
4.4 Diagramme de Compression 105

5.1 Jeux de données expérimentaux 130
5.2 Jeux de données testés par les systèmes 130
5.3 Résultats des requêtes BGP en secondes sur WatDiv100M 131
5.4 Résultats des requêtes BGP en secondes sur WatDiv1B 132
5.5 Résultats des requêtes BGP en secondes sur LUBM500M 132
5.6 Résultats des requêtes BGP en secondes (DBLP) 132
5.7 Requêtes Wildcards sur WatDiv100M (résultats en secondes) 135
5.8 Requêtes Wildcards sur WatDiv1B (résultats en secondes) 135

Chapitre 1

Introduction Générale

1.1 Contexte de la Thèse

Les Services d'Internet et les Données. Le développement d'Internet a fortement impacté l'ensemble des acteurs de notre société : les citoyens, les scientifiques, les universités, les entreprises, les gouvernements, les agences scientifiques, etc. Il est devenu *indispensable* à nos vies de tous les jours. Cela grâce aux services qu'il offre sur le Web tels que World Wide Web (W3), la messagerie électronique, le transfert/-téléchargement des fichiers, les discussions instantanées, etc. Ces services ont fait le bonheur des citoyens, des communautés de recherche (Bases de Données, Services Web, W3, Smart-applications, etc.), des industriels, les enseignants, les formateurs, les éditeurs des produits et logiciels Open Source, etc. Ce bonheur a été accentué après la résistance de l'Internet (et ses services) lors de la crise du COVID-19. Malgré une demande mondiale exponentielle, il a su assurer une qualité de service exceptionnelle.

Une des caractéristiques principales des services de l'Internet est qu'ils génèrent d'une manière continue un déluge de données, surtout depuis quelques années. En effet, selon une étude de l'IBM Marketing Cloud [1], 90% des données sur Internet ont été créées depuis 2016. Cela est dû au fait que les personnes, les entreprises et les appareils sont tous devenus des usines (producteurs) de données qui diffusent chaque jour une quantité importante d'informations sur le Web. Selon les statistiques publiées par le groupe Radicati [2], il est stipulé qu'en 2019, environ 293 milliards de courriers électroniques ont été envoyés chaque jour, ce qui devrait augmenter de 4,2% par an pour atteindre 347 milliards en 2023. Le même rapport prévoit 4,4 milliards d'utilisateurs d'ici la fin de 2023. Les réseaux sociaux sont un autre fournisseur de données. Considérons quelques statistiques de trois réseaux sociaux importants : *Twitter*, *Instagram* et *Facebook*. *Twitter* poste environ 682 millions de tweets par jour. Les utilisateurs du réseau *Instagram* téléchargent plus de 67 millions/jour. 4,3 milliards de messages Facebook sont publiés quotidiennement et 5,76 milliards de mentions de type "j'aime" sont collectés chaque jour. L'Internet des Objets est une autre usine de données. A titre d'exemple, le moteur GTF (Geared Turbo Fan) de Pratt & Whitney, est équipé de 5 000 capteurs générant jusqu'à 10 Go de données

1. https ://curious.stratford.edu/2017/10/10/iot-big-data-and-ai-the-new-superpowers-in-the-digital-universe/
2. `www.radicati.com`

par seconde, ce qui représente une masse importante de données à gérer, stocker, etc.

La pandémie du COVID-19 n'a fait qu'augmenter les statistiques présentées ci-dessus. En effet la période du confinement a poussé les états, les gouvernements, les entreprises, les instituions à favoriser le télétravail de leur employés et collaborateurs. En conséquence, le nombre de courriels, partages/téléchargements/publication des documents ont fortement augmenté. Une autre conséquence de cette pandémie est la naissance de besoins de collaborations entre les institutions de santé nationaux, européens et internationaux afin de trouver des solutions pour contrôler la propagation de ce virus. Cette collaboration passe principalement par le partage des données liées à la COVID-19, des connaissances, des expertises, etc. La plateforme de partage de données pour les chercheurs et qui est lancée par la Commission Européenne [3] est un exemple d'une telle collaboration. Cette plateforme est alimentée en données par les centres de recherche et les hôpitaux. Ces dernières sont analysées et adaptées à des standards généraux avant qu'elles soient mises à la disposition de la communauté scientifique internationale.

La présence de cette mine de données volumineuses, hétérogènes et évolutives, a fait naître une industrie florissante autour de ces données. Les Big Five firmes américaines Google, Amazon, Facebook, Apple, et Microsoft (GAFAM), trois firmes chinoises Baidu, Alibaba, Tencent sont des exemples des entreprises qui alimentent cette industrie et créent une richesse gigantesque. Les Big Five représentent à elles seules un chiffre d'affaires annuel qui avoisine les 500 milliards de dollars. Ces dernières sont à la fois des producteurs et des gestionnaires des données qu'elles génèrent. Ces montants colossaux s'expliquent par les efforts déployés pour gérer, stocker, exploiter, analyser, visualiser ce déluge de données et en mettre sur les marchés des systèmes et des produits appropriés.

Cette industrie traite principalement des données Internet du Web. Après une analyse rapide de ce type de données, nous constatons qu'initialement elles étaient représentées généralement sous forme de *documents textuels*. Cette vision a favorisé le développement des moteurs de recherche d'information. Cette vision textuelle a évoluée après le constat stipulant qu'une grande quantité de *données structurées* intégrées dans des textes en langage naturel, des tableaux à deux dimensions, et d'autres formulaires existent. Ce constat a motivé l'entreprise Google à exploiter cet enjeu pour mieux exploser et traiter ce type de données. La firme Google a alors recruté le Professeur Alon Y. Halevy, un spécialiste dans les bases de données pour diriger un *Structured Data Group of Google Research* à Mountain View, California, USA dans les années 2000. Récemment, un autre type de données structurées a vu le jour : les *Données Ouvertes Liées* (Linked Open Data) ($\mathcal{LOD}$). Cette technologie a présenté un intérêt important auprès des gouvernements, des usagers, des chercheurs, etc. Pour comprendre les motivations et les enjeux, un bref historique est nécessaire. En 2009, Tim Berners-Lee animait une conférence lors de laquelle il militait sur l'ouverture des données brutes. En 2010, avec son modèle en cinq étoiles, il a proposé une approche progressive pour que les gouvernements adoptent des standards ouverts de données [Janowicz et al. (2014)]. Tim Berners-Lee a suivi l'application de ses préconisations en conseillant le gouvernement britannique dans

3. `https://ec.europa.eu/france/news/20200421/plateforme_partage_donnees_coronavirus_fr`

sa politique d'open data [Goeta (2016)]. En 2012, l'Open Knowledge Foundation a créé un outil de benchmarking, l'Open Data Index qui classe les États selon la publication d'une sélection de données. En 2013, une charte a été adoptée par les chefs d'État du G8 pour faire de l'Open Data la pratique par défaut des administrations qu'ils dirigent. Ce rappel historique montre l'intérêt des $\mathcal{LOD}$ dans la vie des gouvernements/ entreprises/ usagers. Ces $\mathcal{LOD}$ représentent alors une valeur ajoutée pour les gestionnaires des données.

L'ère des Données Ouvertes Liées. Comme nous l'avons indiqué dans le paragraphe précédent, ces données sont devenues un phénomène en pleine croissance auprès de différentes organisations. Prenons l'exemple de la commission européenne qui a fait une communication en décembre 2011 pour indiquer que l'ouverture des données personnelles est un "moteur pour l'innovation, la croissance et la transparence des gouvernements". En conséquence, son portail Open data a été lancé en décembre 2012. Il fournit aux européens les données publiques détenues par les institutions et organes de l'Union Européenne (EU). En 2013, l'UE a adopté la directive 2003/98/CE concernant la réutilisation des informations du secteur public (directive PSI), actuellement transposée en France à travers le Projet de loi pour une République numérique. La France a été classée au 4e rang mondial de l'ouverture des données publiques par l'Open Data Index, derrière le Taiwan, le Royaume-Uni et l'Australie. Cette ouverture est actuellement dirigée, sous l'autorité du Premier ministre, par la mission Etalab [4], à l'origine de la refonte du portail `data.gouv.fr`. Ce dernier offre aux chercheurs et citoyens plusieurs services de données générales et thématiques. Pendant la période du COVID-19, ce portail publie régulièrement les données et des indicateurs de suivi du virus en France par région et par département [5].

L'écosystème des $\mathcal{LOD}$ est composé de plusieurs ressources : des bases de données, des ontologies, des méta données (comme Dublin Core), etc. L'ensemble des bases interconnectées du Linked Data forment le Linked Data Cloud contenant 142 milliards de triplets issus de 2973 bases [6]. Ces bases contiennent des données se rapportant à de nombreux *domaines différents* tels que la biologie, les données gouvernementales, géographiques, bibliographiques ou encore des connaissances encyclopédiques. Récemment, G. Gottlob [Bellomarini et al. (2017)] a classé ces bases en trois catégories :

1. Les bases généralistes comme *DBpedia* [Auer et al. (2007)], *FreeBase* [Bollacker et al. (2008)] et *Google Knowledge Graph* [Singhal (2012)],
2. Les bases spécialisées associées à un domaine bien précis. Nous pouvons citer l'exemple de Facebook Knowledge Graph, Amazon *Knowledge Graph* et *Central Banks*,
3. Les bases d'entreprises comme *Entreprise Knowledge* graph [7].

En plus des données, les $\mathcal{LOD}$ utilisent également des ontologies comme *Friend Of A Friend* [8], et en même temps les concepteurs des ontologies font souvent appel aux

4. `https://www.etalab.gouv.fr/`
5. `https://www.data.gouv.fr/fr/datasets/indicateurs-de-suivi-de-lepidemie-de-covid-19/`
6. `http://stats.lod2.eu/`
7. `https://www.slideshare.net/LuMa921/enterprise-knowledge-graph`
8. `http://www.foaf-project.org/`

$\mathcal{LOD}$ pour construire de nouvelles ontologies de domaine [Thorsen and Pattuelli (2013)] (l'exemple du projet Linked Jazz, où une ontologie est construite à partir des $\mathcal{LOD}$). Cette forte interaction bidirectionnelle entre les $\mathcal{LOD}$ et les ontologies augmente le pouvoir de partage, d'intégration, de fusion, et le raisonnement automatiques et conforte la vision du Web Sémantique [Berners-Lee et al. (2001a)].

Rappelons que le Web sémantique étend le Web actuel en interconnectant des *données* et des *métadonnées* lisibles par les machines. Il devient alors une réalité par la publication de grands ensembles de données en utilisant les principes de l'initiative des données ouvertes. Cela offre une nouvelle façon d'interagir avec les données, grâce des au modèle graphe *Resource Description Framework* (RDF) [Lassila et al. (1998)]. Ce dernier fournit une représentation des connaissances simple et abstraite pour toute ressource du Web. Cette dernière est identifiée de manière unique par les identificateurs universels de ressources (URI). Chaque fait du Web sémantique peut être encodé avec un triplet RDF. Les langages RDFS et OWL sont souvent utilisés pour coder les ontologies du Web. Le RDFS [Brickley (2004)] est le langage de vocabulaire pour RDF. Il définit les termes qui sont utilisés dans les faits RDF, ce qui donne un sens spécifique. Le langage d'ontologie Web (OWL) [Hitzler et al. (2009)] peut également être utilisé pour la conception et fournit une plus grande expressivité des relations entre les ressources. Le langage SPARQL est devenu depuis janvier 2008, le langage de recommandation officiel du W3C pour l'interrogation des données RDF [Prud'hommeaux et al. (2017)].

Des bonnes pratiques ont été émises pour bien publier les données structurées sur le Web [Heath and Bizer (2011); Bizer et al. (2011)]. La disponibilité et l'appropriation des technologies autour des données du Web a poussé les trois principaux moteurs de recherche Google, Yahoo ! et Bing à lancer le 2 juin 2011 une initiative (appelée `schema.org`) visant à créer et à prendre en charge un vocabulaire commun (référentiel) dédié aux développeurs pour des fins d'annotation de leurs pages Web. Plus concrètement, cette initiative fournit une collection de schémas ouverts pour décrire le contenu des pages Web HTML en utilisant le format *Microdata* [Hickson (2013)] – un modèle de données similaire à RDF.

Les données ouvertes liées sont là : pensez à les stocker, gérer et exploiter. Depuis sa création, le Cloud de $\mathcal{LOD}$ [Harris et al. (2009)] a été un point de référence pour la communauté des données liées, comprenant un certain nombre de jeux de données liés exposés sur le Web des données. Le $\mathcal{LOD}$ n'a pas cessé d'augmenter le nombre de jeux de données qui en font partie. Le nombre de jeux de données a augmenté d'une manière significative. Il est passé en effet de quelques dizaines lors de la présentation de la première version du $\mathcal{LOD}$ à plus de 1406 jeux de données. la Figure 1.1 montre cette évolution spectaculaire.

Il faut rappeler que cette augmentation ne concerne pas que les jeux de données mais également le nombre de triplets et le volume de données (voir le Tableau 1.1). Pour les jeux de données (Wikidata, DBpedia, DBLP et Freebase), le nombre de triplets varient de 800 millions à plus de 2 milliards. De plus, le volume de données pourrait atteindre rapidement plusieurs centaines de giga-octets.

Les différents jeux de données sont généralement accessibles par des points d'entrée SPARQL. En effet, chaque jeu de données fournit sa propre interface pour interroger les données via SPARQL. À titre d'exemple, *DBpedia* compte actuelle-

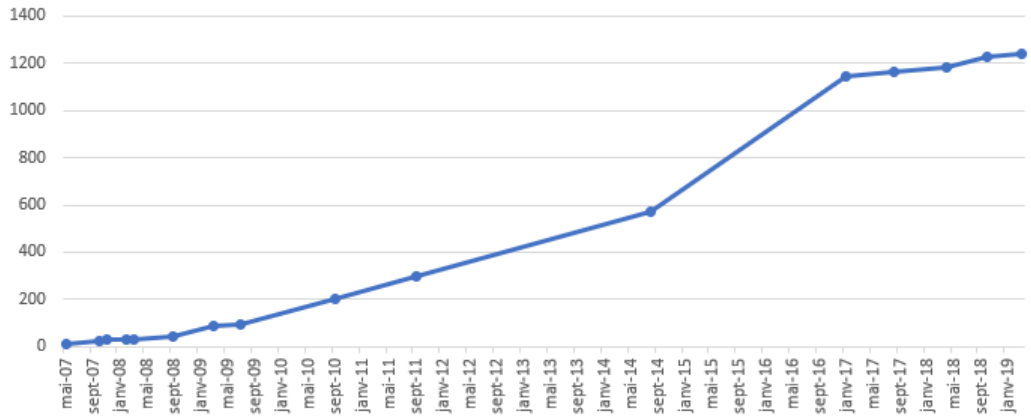


FIGURE 1.1 – Évolution du cloud LoD de mai 2007 à mars 2019

TABLEAU 1.1 – Taille et nombre de triplets des jeux de données

Jeu de données	Taille	Nombre de triplets (Millions)
Wikidata	119 Go	12 000
DBpedia	34 Go	1 800
DBLP	1 Go	882
Freebase	11Go	2 000

ment environ 177 mille requêtes par jour [Yoose and Perkins (2013)]. Avec un tel nombre de requêtes, le point d'entrée SPARQL pourrait rapidement devenir le goulot d'étranglement et par conséquence ralentir le développement du Web de Données. *Il est nécessaire que les producteurs de ces données liées utilisent des systèmes de stockage capable de gérer une quantité importante de données et aussi répondre efficacement aux requêtes SPARQL*. Notons que ce besoin ne concerne pas uniquement les *producteurs de ces données*, mais également leurs *consommateurs* autre que les usagers traditionnels. Récemment plusieurs communautés de recherche ont identifié de l'intérêt, la richesse et la qualité des $\mathcal{LOD}$ pour augmenter la valeur ajoutée de leurs produits (par exemple, la correction des erreurs des données, l'expansion de requêtes, etc.). Nous pouvons citer le cas des entrepôts de données [Lanasri et al. (2019); Berkani et al. (2019)], des systèmes de recommandation [Bellomarini et al. (2017)], les services Web [Ghazouani et al. (2020)]).

Une bonne exploitation de ces données passe par des processus de stockage et d'interrogation efficaces.

Forte Demande sur les Systèmes de Gestion Efficace des $\mathcal{LOD}$. La nécessité de gérer et interroger efficacement les données RDF a conduit au développement de nouveaux systèmes conçus spécialement pour traiter ce format de données. Deux catégories principales de ces systèmes existent : (i) les *systèmes non natifs*) et (ii) les *systèmes natifs*.

1. **Les systèmes non natifs** sont basés principalement sur les SGBDs relationnels qui utilisent une représentation tabulaire pour stocker les données. En

analysons la littérature, les systèmes non natifs peuvent être classés en trois catégories principales selon le type de la table utilisée : *la table de triplets*, *la table de propriété* et *la table binaire*.

— Les systèmes utilisant la table de triplets proposent de stocker et d'interroger les données organisées en une seule très grande table de trois attributs composant un triplet (Sujet, Prédicat, Objet). De nombreux systèmes adoptent cette approche, à l'image de Redland (2002) [Beckett (2002)], qui est l'un des premiers moteurs de stockages des données RDF et 3Store [Harris and Gibbins (2003)] proposée en 2003. Ce dernier prend en compte également les données RDF représentées avec le vocabulaire XML. En 2005, Oracle [Chong et al. (2005)] a enrichi son système en introduisant la fonction SQL "RDF_MATCH" permettant d'évaluer des requêtes SPARQL. Dès 2007, le système Virtuoso [Erling (2012)] a proposé de stocker, en plus des triplets, des quadruplets, qui permettent d'entendre les triplets pour intégrer une information sur le contexte lié à chaque triplet. Le système RDFKB [McGlothlin and Khan (2009)], lancé en 2009, prend en charge l'inférence au moment du stockage des données et non pas pendant le processus de traitement des requêtes. Enfin, les concepteurs de système JenaSDB (2013) [Wilkinson et al. (2003)] ont adopté une approche générique permettant de rester indépendant du SGBD utilisé pour stocker physiquement les données.

— La table de propriétés permet de regrouper les triplets ayant les mêmes propriétés. Chaque groupe est stocké ensuite séparément dans une table relationnelle. Sesame (2002) [Broekstra et al. (2002)] est le premier système à adopter cette stratégie de stockage. Il est conçu en effet pour interagir avec n'importe quel système de stockage relationnel. Jena [McBride (2002)], dont le développement a été initié en 2006, est l'un des principales approches adoptant la table de propriété. Cependant il s'appuie sur deux types de tables de propriétés : (1) une basée sur le clustering de propriétés et (2) une autre basée sur les classes.

— Dans une table binaire [Abadi et al. (2007)], les données RDF sont partitionnées verticalement. L'ensemble de triplets est organisé en n tables (où n représente le nombre de propriétés distinctes) à deux colonnes. Dans chacune de ces tables, la première colonne regroupe les sujets des triplets tandis que la deuxième colonne regroupe les objets des mêmes triplets.

Naïvement, la réutilisation des SGBDs relationnels pour stocker les données RDF est possible et ne nécessite pas d'efforts importants. Mais cette réutilisation peut donner lieu à des systèmes moins performants. En effet, les représentations tabulaires du modèle relationnel ne permettent pas de prendre en compte la particularité des données RDF (sous forme de graphe). Les techniques d'optimisation classiques offertes par les SGDBs relationnels comme les index, le partitionnement, les vues matérialisées, etc. sont inadaptées pour mieux optimiser des requêtes formulées à l'aide du langage SPARQL.

2. **Les systèmes natifs** sont basés sur des approches spécifiques conçue pour la gestion des triplets RDF. Deux tendances caractérisant ses approches se distinguent selon le type d'optimisation utilisée : les approches utilisant d'une manière intensive les structures d'indexation et les approches utilisant les techniques de partitionnement de graphes.

   - **Les approches Dirigées par l'Indexation Intensive**. Pour illustrer le principe de ces systèmes, considérons quelques exemples : *Allegrograph* [Watson (2009)] est l'un des premiers systèmes à avoir adapté à la fois la présentation graphe des données et l'indexation intensive. D'autres systèmes ont été proposés et suivant la même vision, mais ils varient principalement selon les types d'index utilisés. Le système *YARS* proposé en 2005 [Harth and Decker (2005)] utilise six index de type arbre B+ [Graefe et al. (2011)]. Le système *KOWARI* [Wood (2005)] propose une approche hybride combinant les deux types d'arbres AVL [Foster (1973), Knuth (1997)] et un arbre B [Graefe et al. (2011)]. Le système *TipleT* [Fletcher and Beck (2009)] adopte une technique d'indexation partielle afin de limiter la taille de l'index et cela contribue à sa maintenance. *iStore* [Tran et al. (2009)] exploite une structure d'indexation construite automatiquement à partir des données. Cet index peut agir comme un schéma, permettant la navigation et l'interrogation de données Web sans schéma. Quant au système RDFCube [Matono et al. (2006)] un index de trois dimensions qui correspondent aux sujets, prédicats et objets. Le système *gStore* [Zou et al. (2011)] propose de garder la structure graphe en stockant les données RDF sous forme de listes adjacentes. Contrairement aux autres systèmes qui s'appuient sur des index du monde relationnel, gStore bénéficie des listes adjacentes pour définir des index binaires de type *arbre S* [Deppisch (1986)]

   - **Les approches dirigées par le Partitionnement**. Dans ces approches, nous trouvons GRIN [Udrea et al. (2007)] considéré comme le premier système d'indexation RDF à utiliser le partitionnement et les distances dans les graphes comme base d'indexation pour les requêtes SPARQL. Cependant, GRIN n'utilise pas des structures disques et se contente de la mémoire principale pour gérer les données. DOGMA [Bröcheler et al. (2009)] étend GRIN pour assurer la persistance des données sur disque.

   - **Les approches Inspirées des SGBDs Traditionnels.** Un des systèmes populaires suivant ces approches, nous citons l'exemple de RDF3X [Neumann and Weikum (2008)]. Il se distingue des autres approches car sa conception est inspirée de SGBD relationnel. En d'autres mots, il intègre toutes ses composantes. L'utilisation intensive des jointures caractérise ce système, ce qui implique une dégradation de performances de certains types de requêtes.

   - **Les approches en Mémoire Principale.** Elles se positionnent comme une alternative intéressante des approches orientées disque. Certains systèmes privilégient l'utilisation de la mémoire principale dans la gestion des données RDF afin de maintenir une bonne performance de requêtes.

BRAHMS [Janik and Kochut (2005)] suit cette logique et permet d'identifier les associations sémantiques entre les différents triplets. Hexastore [Weiss et al. (2008) se base sur l'indexation multidimensionnelle en mémoire principale. *BitMat* [Atre et al. (2008)] s'appuie sur une structure matricielle et binaire. *Parliament* [Kolas et al. (2009)] décrit un schéma de stockage et d'indexation basé sur les listes adjacentes. *TripleBit* [Yuan et al. (2013)] introduit deux structures d'indexation afin de minimiser le coût de la sélection de l'index lors de l'évaluation de la requête. Ces systèmes sont intéressants pour gérer un volume raisonnable de données RDF. Une fois la taille de ces données s'accroit pour atteindre des millions de triplets, ils deviennent rapidement inefficaces.

- **Les approches dirigées par le Calcul distribué**. Afin d'assurer le passage à l'échelle de ces systèmes, d'autres approches, s'appuyant sur le calcul distribué, ont été proposées. Deux types de distribution ont été utilisés jusqu'à maintenant, à savoir la distribution homogène et la distribution hétérogène. Un système homogène repose sur le fait que toutes les machines d'un cluster utilisent la même configuration logicielle et matérielle. Dans ce contexte, l'architecture réseau (e.g, type de communication) du système peut jouer un rôle important lors du traitement des requêtes. Deux architectures sont principalement utilisées, à savoir, l'architecture Pair-à-Pair et l'architecture client-Serveur. Concernant le premier type d'architecture, nous pouvons citer *RDFPeers* [Cai and Frank (2004)] qui est considéré comme le premier système RDF distribué à utiliser une communication Pair-à-Pair. Il fournit différents composants pour le stockage, l'indexation et l'interrogation à l'aide du langage *RDQL* [Seaborne (2004)]. *MIDAS-RDF* [Tsatsanifos et al. (2011)] fait aussi partie de cette catégorie. Il est basé sur une structure d'index multidimensionnelle distribuée. Il propose une récupération rapide des triplets RDF satisfaisant les différents modèles de requêtes en les traduisant en requêtes de plages multidimensionnelles.

  L'architecture Client-serveur a été aussi utilisée pour gérer les données RDF, où un nœud joue le rôle du maitre afin d'orchestrer l'exécution des tâches assignées aux autres nœuds (considérés comme des esclaves). Nous pouvons citer ainsi, le système *YARS2* [Harth et al. (2007)] basé principalement sur *YARS*. Il fournit des méthodes distribuées d'indexation et d'évaluation de requêtes.

  Les systèmes hétérogènes correspondent généralement à un moteur de requêtes fédéré ou à base de médiateur. L'approche fédérée de requêtes a beaucoup attiré l'attention des chercheurs. Cela est dû au mouvement des données liées ainsi qu'au développement des points d'entrées SPARQL [Ibragimov et al. (2015)]. *Splendid* [Görlitz and Staab (2011)] est l'un des premiers systèmes fédérés dédié aux requêtes SPARQL. *FedX* [Schwarte et al. (2011)] permet aux utilisateurs du Web sémantique d'intégrer à la demande des points d'entrées SPARQL existants. Les systèmes basés sur les médiateurs correspondent principalement aux systèmes OBDA (Ontology-Based Data Access) [Calvanese et al. (2018)] et peu d'entre eux sont capables d'interagir avec plusieurs sources de données existantes.

Un autre problème de ces systèmes est la gestion de la non-disponibilité des sources de données. Le système de matérialisation MARVEL (MAterialized Rdf Views with Entailment and incompLetness) traite cette problématique importante [Ibragimov et al. (2016)].

Pour faire face aux données massives, un nouveau type d'approches a été proposé. Ces dernières s'appuient sur un traitement massivement parallèle (en anglais MPP - Massively Parallel Processing). Le traitement massivement parallèle est un moyen de croquer d'énormes quantités de données en distribuant le traitement sur des centaines ou des milliers de processeurs. Ces derniers peuvent fonctionner dans la même boîte ou dans des ordinateurs séparés et éloignés. Chaque processeur d'un système MPP possède sa propre mémoire, disques, applications et instances du système d'exploitation. Parmi les systèmes qui utilisent cette approche on peut citer *SHARD* [Rohloff and Schantz (2011)] qui s'appuie sur Hadoop pour les aspects de persistance des données et de traitement des requêtes. *HadoopRDF* [Huang et al. (2011)] combine le framework Hadoop distribué avec le système RDF centralisé *RDF3X*. *CliqueSquare* [Djahandideh et al. (2015)] limite le nombre de tâches MapReduce et le transfert de données entre les nœuds pendant l'évaluation des requêtes. *SparkRDF* [Chen et al. (2015)] se base sur Spark et s'appuie sur le partitionnement du graphe RDF en sous-graphe selon les relations et les classes. *S2RDF* [Schätzle et al. (2016)] utilise l'interface relationnelle de Spark pour l'exécution des requêtes et étend le schéma de partitionnement vertical.

En s'appuyant sur les approches MPP, les systèmes de gestion de données RDF sont devenus en mesure de garantir le passage à l'échelle (en termes de volume de données à traiter). Par contre, cela a impliqué une dégradation de performances. En effet, l'inconvénient majeur de ces approches réside dans leur inefficacité de traiter efficacement (dans un temps acceptable) certains types de requêtes représentées par des motifs de graphes complexes.

Si nous analysons les systèmes que nous avons revus, nous remarquons qu'ils font appel à une panoplie de méthodes d'optimisation que nous pouvons diviser en cinq méthodes principales : (i) des méthodes utilisant intensivement de techniques d'optimisation comme les index, (ii) des méthodes inspirées du SGBG relationnel, (iii) des méthodes favorisant des traitements distribués et parallèles, (iv) des méthodes dirigées par le type de systèmes hébergeant les données RDF (orientés disque/orientés mémoire), et (v) des méthodes qui se focalisent sur stockage physique de graphe RDF (le cas de listes adjacentes utilisées dans le système gStore). Cette analyse nous pousse à mettre tout à plat et à choisir une méthode qui prend en compte les avantages de chaque type de méthodes proposées afin d'assurer le passage à l'échelle en termes de données et aussi une bonne performance lors du traitement des requêtes SPARQL.

1.2 Notre Démarche et Contributions

Pour mener à bien nos travaux de thèse, nous avons suivi la démarche suivante qui se base sur le principe suivant : tester, analyser et proposer. Tester signifie évaluer les

systèmes existants selon les deux critères : la performance et le passage à l'échelle. Cela est possible parce que plusieurs jeux de données RDF existent accompagnés par des requêtes SPARQL. Une fois testés, les résultats obtenus sont analysés afin d'identifier les causes du respect/non-respect de nos critères. Cette analyse nous facilite la proposition d'un nouveau système.

Tester : Nous avons mené des expérimentations pour évaluer le critère de passage à l'échelle dont les résultats sont décrits dans Tableau 1.2.

Jeu de données	gStore	Rdf-3X	Virtuso	gStoreD	CS
WatDiv100M	✓	▲	✓	▲	▲
WatDiv1B	✘	▲	✓	▲	▲
LUBM1B	✘	✘	✘	▲	▲
LUBM500M	✘	▲	✓	▲	▲
Yago	✘	▲	✓	▲	✘
DBLP	✘	▲	✓	▲	▲

✓ : Supporte les requêtes BGP, expressions régulières, agrégation et tri, , ▲ : Chargé avec succès mais ne supporte pas toutes les requêtes, ✘ : Incapable de charger, CS : Cliquesquare

Tableau 1.2 – Résultats expérimentaux des systèmes

Afin de mieux mener à bien cette phase de "tester", *nous avons passé beaucoup de temps à comprendre en détails les systèmes que nous avons choisis et évalués.* Après cette compréhension, une compagne de test a été mise en place en utilisant les infrastructures du laboratoire LIAS adaptées pour cette tâche. Nous avons évalué plusieurs systèmes et cinq ont été retenus pour une comparaison à large échelle. Chaque système est considéré comme l'état de l'art pour une catégorie donnée : *Virtuoso* pour les approches non natives relationnelle, *g-Store* pour les approches à base de graphes, RDF-3X pour les approches natives basées sur l'indexation intensive et cliquesquare pour les approches MPP. Les jeux données et requêtes sont issues de benchmarks populaires (Watdiv, LUBM, Yago et DBLP). Afin de montrer les limites des systèmes comparés, nous avons varié le nombre de triplets pour WatDiv et LUBM. Pour le premier, deux jeux de données ont été utilisé, avec respectivement 100 millions et 1 milliard de triplets, tandis que, pour le deuxième, nous avons utilisé deux jeux de données avec respectivement 500 millions et 1 milliard de triplets.

Analyser : D'après les résultats obtenus, nous remarquons qu'il est clair qu'aucune approche n'est en mesure de gérer des volumes importants de données RDF tout en garantissant à la fois le passage à l'échelle et les performances des requêtes.

Proposer : L'analyse fine de ces résultats nous a poussé à proposer notre système, baptisé "RDF_QDAG" qui utilise des structures de données scalables pour stocker les données RDF et revisite les techniques d'optimisation traditionnelles comme la fragmentation, l'indexation, la répartition et la gestion de la mémoire pour les adapter à ces structures.

La satisfaction des deux besoins non fonctionnels est assurée par la combinaison de la fragmentation physique de données et le processus d'exploration du graphe de données. Cette dernière permet de profiter de la nature graphe des données tandis que la fragmentation permet de regrouper les nœuds du graphe ayant les mêmes propriétés ce qui offre des possibilités d'élagages en fonction de la sémantique implicite

des données. Notre système permet de supporter plusieurs types de requêtes basées non seulement sur les motifs basiques de graphes mais aussi qui intègrent des filtres à base d'expression régulière et aussi des fonctions d'agrégation et de tri. L'efficacité de ces requêtes est garantie à l'aide de notre modèle de contrôle de la mémoire principale pour éviter tout débordement et garantir des meilleures performances même si la configuration matérielle est limitée. Ce modèle est inspiré du système *Volcano* [Graefe (1994)]. Malgré qu'il ait été développé dans les années 90, *Volcano* offre un environnent riche pour les chercheurs et les industriels pour le développement d'heuristiques pour l'optimisation des requêtes, l'exécution parallèle des requêtes et l'allocation des ressources. Avant de détailler nos solutions, un cadre formel lié aux représentations logiques des données et des requêtes est proposé.

Une batterie d'expérimentations est proposée pour valider l'efficacité de notre système. Ces expérimentations ont concerné les quatre types principaux de requêtes SPARQL : (1) BGP, (2) expressions régulières, (3) agrégation et (4) tri. Nous avons comparé notre approche avec quatre autres systèmes qui représentent l'état de l'art en matière de gestion de données RDF : *Virtuoso*, *g-Store*, *RDF-3X* et *cliquesquare* [Goasdoué et al. (2013)]). Nos résultats montrent l'intérêt de notre système RDF_-QDAG, car il surclasse les systèmes étudiés en termes de passage à l'échelle et de performances de requêtes.

1.3 Organisation du manuscrit

Le reste de la thèse est structurée en deux parties : une partie état de l'art et une partie contributions comme le montre la Figure 1.2.

La première partie présente les concepts fondamentaux permettant d'élaborer nos propositions et comporte deux chapitres.

Dans le Chapitre 2, nous introduisons le monde des Données Ouvertes Liées et les Knowledge Graphs, en dégageant ses concepts et technologies fondamentaux. Trois exemples de données ouvertes sont discutés, à savoir *Freebase*, *Wikidata*, et *DBpedia*. Nous détaillons par la suite le Resource Description Framework (RDF) dédié à modéliser les ressources Web sémantiques ainsi que le langage SPARQL. Ces concepts sont illustrés par des exemples. Le Chapitre 3 analyse et compare les systèmes principaux de stockage des données RDF. Une classification de ces systèmes est proposée incluant des systèmes non natifs et natifs. Des efforts de distribution de ces systèmes ainsi que leur traitement de requêtes sont largement commentés, en s'appuyant sur des systèmes opérationnels. Nous présentons aussi certains outils de benchmarking permettant d'évaluer le comportement de ces systèmes vis-à-vis les performances et le passage à l'échelle. Un point important de chapitre est lié à l'étude expérimentale que nous avons menée pour évaluer les systèmes étudiés et ce en s'appuyant sur des benchmarks connus (Watdiv, LUMB, Yago, DBLP) et en variant la taille des données.

La deuxième partie de ce mémoire est dédiée aux contributions liées à la définition et à la mise en oeuvre de notre système RDF_QDAG.

Dans le chapitre 4, nous commençons par présenter le cadre théorique permettant d'évaluer les requêtes en s'appuyant sur la fragmentation et l'exploration de graphes. Nous détaillons ensuite les composants principaux de notre système RDF_QDAG. Nous nous étalons sur le modèle de stockage de données basé sur les graphes ainsi que

FIGURE 1.2 – Plan de thèse

les structures d'optimisation permettant de satisfaire les besoins non fonctionnels que nous avons fixés : la performance des requêtes et le passage à l'échelle. Ces techniques permettent d'appuyer notre approche d'évaluation de requêtes. Notre modèle, inspiré du système *Volcano*, de gestion de la mémoire lors de l'exécution de la requête est aussi discutée. Enfin, nous présentons des modules complémentaires permettant d'assurer le chargement de données et aussi de trouver un bon plan d'exécution. Dans le chapitre 5, nous présentons notre étude expérimentale permettant de valider nos propositions vis-à-vis le passage à l'échelle et les performances. Nous comparons en effet notre système avec d'autres systèmes qui représentent l'état de l'art en matière de gestion de données RDF : une approche relationnelle (Virtuoso), une approche à base de graphes (g-Store), une approche d'indexation intensive (RDF-3X) et une approche MPP (CliqueSquare).

Le dernier chapitre conclut ce mémoire de thèse, établit un bilan des contributions apportées et trace différentes perspectives de recherche.

1.4 Publications

La liste suivante présente les publications concernant le travail de cette thèse.

— **Article dans des Revues Internationales avec Comité de Lecture**

1. **Abdallah Khelil**, Amin Mesmoudi, Jorge Galicia Auyon, Ladjel Bellatreche, Hacid Mohand-Said, Emmanuel Coquery, *Combining graph exploration and fragmentation for scalable rdf query processing*, Information Systems Frontiers Journal, Springer, pp. 1–19, 2020, **[Q1 : SJR]**.

— **Article dans des Conférences Internationales avec Comité de Lecture**

1. **Abdallah Khelil**, Amin Mesmoudi, Jorge Galicia, Mohamed Senouci : Should We Be Afraid of Querying Billions of Triples in a Graph-Based Centralized System ? *in Proceedings of 9th International Conference on Model and Data Engineering (MEDI)*, pages 251-266, Toulouse, France, October 28-31, 2019. Lecture Notes in Computer Science 11815, Springer 2019, ISBN 978-3-030-32064-5

— **Article dans des Conférences Nationales avec Comité de Lecture**

1. **Abdallah Khelil**, Amin Mesmoudi, Jorge Galicia, Ladjel Bellatreche, EXGRAF :Exploration et Fragmentation de Graphes au Service du Traitement Scalable de Requêtes RDF, *dans 16ème Edition de la conférence de Business Intelligence Big Data (EDA)*, Lyon, France, August 25-27, 2020

Première partie

État de l'Art

Chapitre 2

Le monde des données ouvertes liées

Sommaire

2.1 Le Web sémantique 33
2.2 Données Ouvertes Liées 34
2.2.1 Freebase 36
2.2.2 DBpedia 36
2.2.3 Wikidata 37
2.3 Le modèle de données RDF 39
2.3.1 Syntaxe RDF 43
2.3.2 Le schéma RDF 45
2.4 Langages de requête RDF 50
2.4.1 Le langage de requête SPARQL 50
2.4.2 Évaluation de requêtes 54
2.5 Conclusion 55

Dans le Chapitre précédent nous avons présenté le contexte global de notre problématique ainsi que ses contraintes et ses objectifs. La proposition d'un système de traitement de données RDF, optimisant à la fois le passage à l'échelle et les performances des requêtes SPARQL, passe principalement par la compréhension du monde des données liées et le processus de leur interrogation. Comme nous l'avons indiqué précédemment, le monde des $\mathcal{LOD}$ est devenu un phénomène planétaire grâce à l'évolution des technologies du Web Sémantiques.

Ce chapitre est structuré comme suit : La Section 2.1 présente une synthèse concise sur le Web Sémantique et ses briques technologiques. Dans la Section 2.2 des exemples de bases de connaissances sont décrits afin de comprendre l'enjeu de leur stockage et interrogation. Le framework RDF est présenté dans la Section 2.3 où ses composants sont illustrés par des exemples. La Section 2.4 décrit en détail le langage d'interrogation SPARQL, ainsi que le processus d'évaluation de requêtes. La Section 2.5 conclut ce chapitre.

2.1 Le Web sémantique

Le Web sémantique a été introduit pour la première fois dans un article publié par Tim Berners-Lee dans Scientific American 2001[Berners-Lee et al. (2001b)]. Le Web sémantique utilise l'ontologie pour l'organisation de données et identifie également les connexions en enregistrant les relations entre ces données. Il organise les données dans une structure logique à base de graphes. Il est vrai que de nombreux formats de données existants (tels que les données tabulaires et les données relationnelles) compréhensibles par l'homme, mais en contrepartie elles sont difficiles à manipuler par les machines. Les technologies du Web sémantique facilitent le traitement des données par des machines.

Le terme Web sémantique fait référence à la vision du W3C liée au réseau de données liées. Les technologies du Web sémantiques permettent aux gens de créer des systèmes de gestion de données sur le Web, de construire des vocabulaires et d'écrire des règles pour le traitement de ces données. Il existe de nombreuses applications pratiques utilisant les technologies du Web sémantique. Aujourd'hui, la British Broadcasting Corporation (BBC) utilise le Web sémantique sur son service Web et fournit des ontologies pour le sport, l'éducation, la musique, etc. Certains grands moteurs de recherche tels que Google et Yahoo utilisent également le Web sémantique pour améliorer les résultats de la recherche.
Dans cette section nous présentons le Web sémantique comme une extension du Web actuel dans lequel l'information a un sens bien défini, permettant aux ordinateurs et aux gens de mieux travailler en coopération.

La description et l'interprétation des données du Web sont étayées par une pile de technologies. Cette pile est souvent appelée le "layer cake" du Web sémantique. L'une de ses versions actuelles est présentée à la Figure 2.1. L'organisation des couches de cette pile technologique implique que les éléments décrits à une couche donnée sont conformes aux normes définies aux couches inférieures. Chaque couche est moins générale que les couches ci-dessous.

— *La couche la plus basse* de cette pile fournit une solution d'identification globale pour les ressources trouvées sur le Web. Dans la Figure 2.1, ils sont appelés identificateurs de ressources uniformes/internationalisés (URI / IRIs). Les URI et les IRIs sont maintenant utilisés de manière interchangeable.

— *La deuxième couche* prend en charge la définition d'une syntaxe basée sur XML, un métalangage basé sur la notion de tags. Notez que XML est livré avec certaines technologies associées qui permettent la définition de schémas pour les instances des documents, telles que la définition de type de document (DTD) ou le schéma XML. Une convention de nommage (c'est-à-dire les espaces de noms) est prise en charge pour désambiguïser l'utilisation de balises qui se chevauchent et proviennent de différentes langues.

— Dans la *troisième couche*, nous commençons notre voyage dans le Web Sémantique avec le langage RDF. Dans la Figure 2.1, il est qualifié d'échange de données parce que son objectif principal est de permettre l'échange de faits entre agents. Notez que RDF repose à la fois sur XML et sur la couche URI.

— La *quatrième couche* permet d'étendre RDF pour décrire, en plus des faits, des métadonnées. Avec RDF Schema (RDFS), une première solution est proposée pour définir des métadonnées sur certains éléments d'une instance au sein d'un

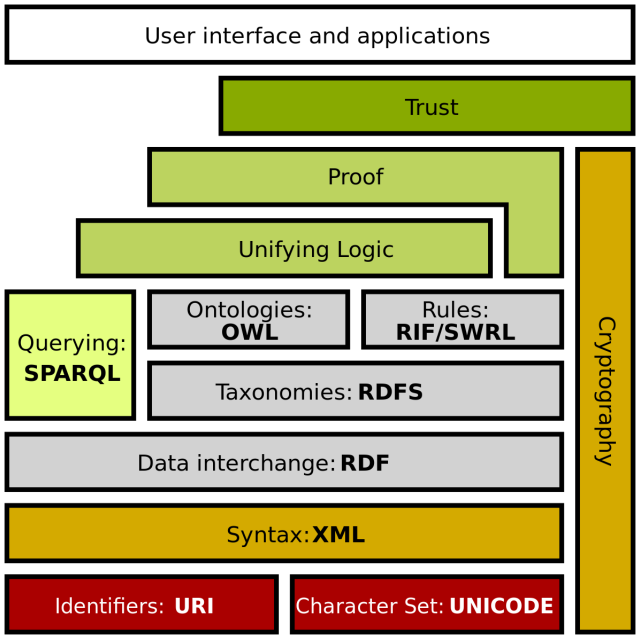


FIGURE 2.1 – Le "layer cake" du Web sémantique

document RDF. La spécification de ce langage ne permet pas de supporter des vocabulaires très expressifs, ce qui est pris en compte dans la couche suivante.

— La *cinquième couche* est composée du langage d'ontologie Web (OWL). OWL permet de définir des ontologies plus expressives que RDFS. Il engendre malheureusement une complexité de calcul lors qu'il s'agit de faire un raisonnement poussé. Pour résumer :

> *"Le Web sémantique n'est pas un nouveau Web, mais une extension du Web actuel, dans lequel l'information a un sens bien défini, permettant mieux aux ordinateurs et aux gens de travailler en coopération."* Tim Berners Lee (2001)
>
> *"Le Web sémantique fournit un cadre commun qui permet de partager et de réutiliser les données entre les applications, les entreprises et les communautés. C'est un effort de collaboration mené par le W3C avec la participation d'un grand nombre de chercheurs et de partenaires industriels."* World Wide Web Consortium (W3C)

D'autres défis du Web sémantique comprennent l'immensité, l'imprécision, l'incertitude, l'incohérence et la tromperie [Djebri et al. (2019); Karanikola and Karali (2018)].

2.2 Données Ouvertes Liées

Les données ouvertes liées ont émergé suite à l'adoption massive des concepts du Web sémantique. Les données ouvertes sont des données (connaissances communes ou informations) que toute personne peut utiliser, modifier et partager. Le mouvement des données ouvertes, également appelé Open Data, préconise un accès gratuit

aux données sous licence ouverte. Toutefois, la provenance des données doit être indiquée et la transparence des données doit être préservée.

Par conséquent, les données ouvertes, qu'elles soient brutes ou bien structurées, devraient être publiques. Ainsi, le Web est devenu le moyen approprié et simple de partager des données ouvertes dans plusieurs domaines comme la recherche scientifique, les sciences de la vie, la santé publique, et les données gouvernementales. Les données liées ou réseau de données, en revanche, visent à relier les données connexes disponibles sur le Web. L'objectif est de créer des liens entre ces données pour faciliter leur exploration par les humains et les machines. En effet, l'établissement de liens entre différentes données crée un réseau mondial de données qui permet aux utilisateurs d'avoir accès à toutes les données connectées et connexes à partir d'une donnée. Cela contribue à transformer le Web en un espace de données global. Les données liées, introduites par Tim Berners-Lee en 2006, préconisent plusieurs standards Web : URI (Uniform Resource Identifier), HTTP (Hypertext Transfer Protocol), RDF (Resource Description Framework) et SPARQL (SPARQL Protocol and RDF Query Language) pour identifier, décrire, connecter et interroger des données sur le Web. Un fournisseur de données devrait donc utiliser :

— Le concept d'URI pour identifier n'importe quoi (tous les objets et concepts).

— Le protocole HTTP pour interroger les données.

— RDF pour décrire les informations utiles avec des URI et SPARQL pour les interroger.

Sur la base des principes précédents, de plus en plus de sources de données liées ont été mises à disposition au cours de la dernière décennie, d'où l'utilisation du terme "réseau de données" en parallèle avec le réseau de documents. La combinaison de données ouvertes et de données liées a donné naissance à Linked Open Data ($\mathcal{LOD}$). Le $\mathcal{LOD}$ est défini par Tim Berners-Lee comme "les données liées publiées sous une licence ouverte, qui n'empêche pas leur réutilisation gratuitement". Il a également proposé une échelle de notation de 5 étoiles pour aider les éditeurs de données à faciliter l'utilisation de leurs données par les gens, et donc les rendre plus puissants :

— Les données sont disponibles sur le Web (quel que soit le format), avec une licence ouverte pour être Open Data. Il s'agit ici de la première étoile.

— La mise en place d'un format structuré lisible par machine donne droit à une deuxième étoile.

— La mise en place d'un format non-propriétaire donne aussi droit d'une autre étoile.

— L'adoption des standards ouverts du W3C (URI et RDF) pour nommer, représenter et interroger les données donne droit aussi à une autre étoile.

— Lorsque les données sont liées aux données d'autres personnes pour fournir un contexte, une autre étoile est donnée.

Grâce à l'échelle de notation de 5 étoiles, une pléthore de systèmes de gestion de données RDF a été développée et connectées au Web. Le projet Linking Open Data [1] identifie les sources de données ouvertes qui sont conformes aux principes $\mathcal{LOD}$ et

1. https ://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

montre les liens existants entre elles. Ces sources sont principalement publiées par les membres du projet, mais d'autres organisations et individus peuvent également y participer. Le projet publie occasionnellement un diagramme en Cloud représentant les jeux de données, appelés aussi "knowledge graph" comme vous pouvez le voir dans la Figure 2.2. À ce jour, environ 1239 jeux de données sont interconnectés dans le Cloud de données ouvertes et liées avec 16147 liens, la majorité des liens reliant des entités identiques dans deux jeux de données. Les données publiées couvrent divers domaines et sujets tels que les sciences de la vie, la musique, la géographie, les données gouvernementales et médiatiques. Le projet publie également des statistiques sur les jeux de données, les vocabulaires utilisés et des informations sur les points d'accès SPARQL.

Le cloud $\mathcal{LOD}$ a actuellement plus de 1000 bases de connaissances. La majorité de ces bases de connaissances contiennent des informations sur les entités à partir desquelles les moteurs de recherche peuvent apprendre. Les bases de connaissances considérées comme les plus significatives sont représentées près du centre du nuage $\mathcal{LOD}$ (voir Figure 2.2), et comprennent DBpedia, Wikipedia, YAGO, the CIA World Fact Book, Freebase, et Wikidata.

Dans les sections suivantes, nous présentons quelques exemples de bases de connaissances les plus connues.

2.2.1 Freebase

En 2007, *Freebase* a été lancé par *MetaWeb* en tant que base de connaissances ouverte et collaborative. Sur le site `freebase.com`, les utilisateurs pourraient enregistrer un compte gratuit et modifier les données de diverses entrées, créant des liens entre les données liées aux entités. *Freebase* était exclusivement concernée par le codage des liens et des relations entre les entités de la base de connaissances. En 2010, *MetaWeb* a été acquis par Google. *Freebase* a été utilisée chez Google pour alimenter certaines parties de leur graphe interne "Google Knowledge Graph", qui prenait en charge les fonctionnalités de recherche Google tels que les cartes de connaissances ou les panneaux. Le 16 décembre 2014, il a été annoncé sur la communauté *Freebase Google Plus* que *Freebase* serait progressivement arrêtée au cours des six prochains mois et que ses données seraient transférées vers la plate-forme *Wikidata*. Les données *Freebase* sont disponibles en téléchargement dans un format RDF N-Triples.

Quelques informations chiffrées sur Freebase : Selon le site Web des développeurs de Google, la dernière version de *Freebase* contient environ 50 millions d'entités et 3 milliards de triplets. Le schéma de Freebase comprend environ27000 types d'entités et 38000 types de relations.

2.2.2 DBpedia

Le projet communautaire *DBpedia* a permis de créer un graphe de connaissances basé sur les données structurées publiées sur *Wikipedia*. Ce graphe a permis de rendre les informations de *Wikipedia* disponibles via les normes du Web sémantiques. Cela a permis de profiter des meilleures pratiques développées pour les données liées. L'un des objectifs du projet *DBpedia* est de fournir des capacités d'interrogation et de recherche à une large communauté en extrayant des données structurées de *Wikipedia*

qui peuvent ensuite être utilisées pour répondre à des requêtes expressives. Son projet *DBpedia* a été lancé en 2006 et a entre-temps suscité un intérêt important pour la recherche dans divers domaines d'application. *DBpedia* a été un facteur clé du succès de l'initiative *Linked Open Data* et joue le rôle d'un centre d'interconnexion pour d'autres jeux de données. Le noyau de *DBpedia* consiste en un processus d'extraction d'infobox, qui a été décrit pour la première fois dans [Auer and Lehmann (2007)]. Les infoboxes sont des modèles de contenus dans de nombreux articles de *Wikipédia*. L'extracteur d'infobox traite une infobox comme suit : L'URI DBpedia, qui est créé à partir de L'URL de L'article Wikipedia, est utilisé comme sujet. L'une des raisons pour lesquelles la qualité des données de DBpedia s'est améliorée au cours des dernières années est que la structure de ses connaissances est maintenue par sa communauté.

Quelques informations chiffrées sur DBpedia : La version la plus récente de *DBpedia* se compose de 13,1 milliards d'informations (triplets RDF) dont : 1,7 milliard ont été extraites de l'édition anglaise de Wikipedia, 6,6 milliards ont été extraites d'autres langues et 4,8 milliards de *Wikipedia* Commons et *Wikidata*.

2.2.3 Wikidata

Wikidata est une base de données lisible par l'homme, lisible par la machine, multi-lingue, multidisciplinaire, centralisée, modifiable, structurée et liée avec une diversité croissante de cas d'utilisation. Elle rassemble des données structurées qui peuvent être lues et éditées par des humains et des machines. Contrairement à *DBpedia*, *Wikidata* n'extrait pas directement des données de *Wikipedia*. Au lieu de cela, il s'agit d'une base de connaissances ouverte où chacun a le droit de la modifier. Lancé en octobre 2012 par la fondation *Wikimedia* qui héberge également les différentes éditions linguistiques de *Wikipedia*, elle utilise un modèle de *crowdsourcing* pour créer et éditer des enregistrements de données structurées tout en réconciliant les données de différentes versions linguistiques de Wikipédia.

Quelques informations chiffrées sur Wikidata : Elle compte plus de 3 500 contributeurs actifs qui effectuent un demi-million de modifications par jour. Les données sont exposées dans des formats lisibles par machine tels que JSON, XML et RDF. Ses administrateurs reconnaissent son rôle en aidant à remplir le graphe de connaissances de Google. *Wikidata* a actuellement plus de 40 millions de ressources et plus de 350 millions de déclarations, avec environ 14 millions de modifications par mois par plus de 30 mille éditeurs actifs.

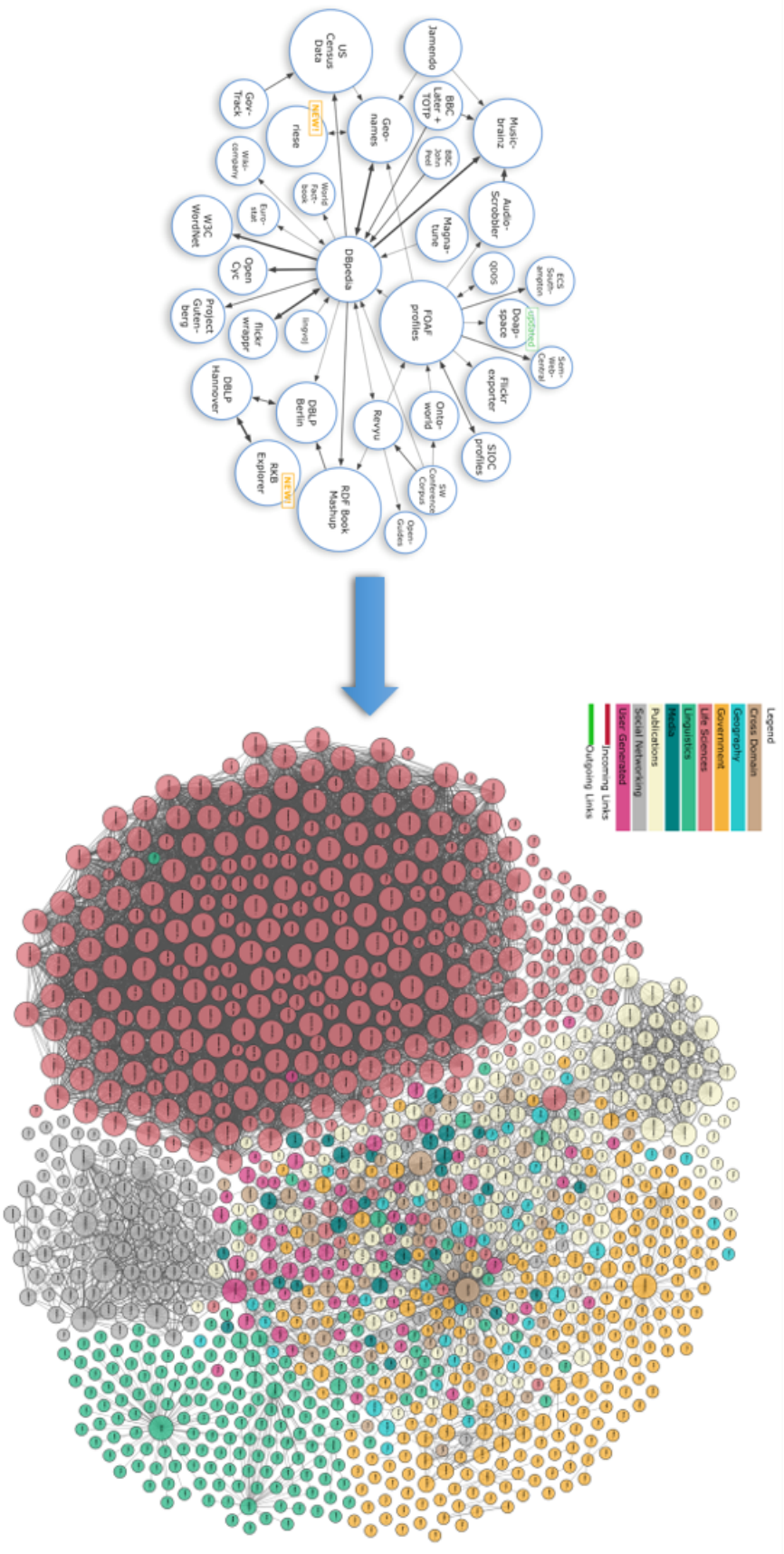


FIGURE 2.2 – Évolution du cloud *LOD* de 2007 à 2019

2.3 Le modèle de données RDF

Dans la section précédente, nous avons commenté la philosophie adoptée par le Web sémantique et les données ouvertes liées. Dans cette section, nous présentons le langage qui permet de représenter une donnée RDF.

Le premier objectif du framework RDF (Ressource Description Framework) était de fournir un modèle de données et de métadonnées au Web. Aujourd'hui, il est considéré comme une norme W3C et émerge comme un modèle de données pour le Web de données et le Web sémantique. Cela est dû au fait qu'il fournit une organisation logique définie en termes de certaines structures de données, ce qui facilite la représentation, l'accès et la spécification de contraintes et de relations entre objets d'intérêt dans un domaine d'application donné.

Le Resource Description Framework (RDF) est un modèle standard pour l'échange de données sur le Web. Les ressources décrites par RDF peuvent être n'importe quoi, y compris des documents, des personnes, des objets physiques et des concepts abstraits.

RDF décrit les données en identifiant les ressources, en leur attribuant des propriétés et en établissant des relations entre elles. Une ressource est définie à l'aide d'identificateurs - appelés identificateurs de ressources internationalisés (IRIs) - ou littéraux. Une relation entre deux ressources est connue sous le nom de triplet (<sujet> < prédicat> <objet>), faisant une relation dirigée du < sujet > au < objet > en utilisant le < prédicat>. Cette structure de liaison forme un multigraphe dirigé et étiqueté, où les arêtes représentent le lien nommé entre deux ressources, représentées par les nœuds de graphe.

RDF fournit un moyen conforme aux normes pour l'échange de données. En utilisant ce modèle simple, il permet aux données structurées et semi-structurées d'être mélangées, exposées et partagées entre différentes applications. Des exemples d'utilisation de RDF peuvent être l'ajout d'informations lisibles par machine à des pages Web et l'enrichissement d'un jeu de données en le reliant à des jeux de données tiers. RDF a des fonctionnalités qui facilitent la fusion des données même si les schémas sous-jacents diffèrent, et il prend spécifiquement en charge l'évolution des schémas au fil du temps sans nécessiter de modification de tous les consommateurs de données (c-à-d que les données ont toujours la structure triplet).

RDF se base sur une suite de recommandations du W3C publiées en 2004 (connu sous le nom RDF 1.0). En 2014, une deuxième suite de recommandations et de notes du groupe de travail du W3C a été publiée. Elle permet de définir une version plus récente connue sous le nom RDF 1.1. Cette dernière suite comprend (mais sans s'y limiter) les Concepts RDF 1.1, la syntaxe abstraite, la syntaxe XML et aussi sa sémantique [Consortium et al. (2014)] et d'autres recommandations liées à la sérialisation de RDF. La mission du groupe de travail était d'étendre RDF pour y inclure des caractéristiques souhaitables et importantes pour l'interopérabilité, mais sans effet négatif sur le déploiement.

RDF est actuellement largement répandu sur le Web. Plusieurs applications exploitent ce type de données dans divers domaines. Des exemples de données RDF disponibles publiquement comprennent :

— Le projet DBpedia qui vise à extraire du contenu structuré RDF à partir des informations disponibles sur les différents sites Web de Wikipédia.

— Bio2RDF qui est une base de données biologique basée sur les technologies RDF et Web sémantique.

— UniProt qui fournit une ressource complète, de haute qualité et librement accessible de la séquence protéique.

— LinkedGeoData qui utilise les informations du projet OpenStreetMap et les rend disponibles sous forme de données RDF.

L'unité de base d'information dans RDF est donnée comme un triplet (s, p, o), composé d'un sujet (s), d'un prédicat (p), et un objet (o).

— **Le sujet :**La ressource sur laquelle une affirmation est faite. Seuls les URI et les nœuds vides sont autorisés à être utilisés comme sujet d'un triplet.

— **Le prédicat :**Un attribut d'une ressource ou d'une relation binaire qui relie cette ressource à une autre. Seuls les URI sont valides pour être utilisés comme prédicat d'un triplet.

— **L'objet :** Généralement, la valeur de l'attribut ou une autre ressource. Les objets valides sont des URI et des nœuds vides, mais aussi des chaînes de caractères. Ces chaînes, sont également appelées littéraux.

Chaque triplet représente un fait sur une chose qu'on veut décrire (c-à-d., le sujet, qui est également appelé la ressource), sur une propriété spécifique (c.-à-d., le prédicat), et avec une valeur donnée (c.-à-d., l'objet). Voici un exemple RDF :

```
Al Pacino a gagné le Golden globe
```

Pour cet exemple : "Al Pacino" représente le sujet, "a gagné" représente le prédicat tant dis que "le Golden globe" représente l'objet.

Le modèle de données RDF offre les concepts de base suivants :

— **Ressources :** Dans RDF, une ressource est tout ce que nous voulons décrire dans le World Wide Web. Une ressource peut être une page Web, un livre, un auteur, un article ou un fichier informatique. Chaque ressource est identifiée de façon unique par un identifiant de ressource universel (URI)

— **Propriétés :** une propriété est une caractéristique d'une ressource. Par exemple, "situé" peut être utilisé pour localiser l'emplacement. Les propriétés sont également identifiées par les URI.

— **Littéraux :** les littéraux sont des valeurs constantes de toute propriété. Par exemple, "Futuroscope" peut être la valeur de la propriété "situé".

— **Déclarations :** Les déclarations sont les constructions proposées par RDF pour représenter des informations sur un domaine. Une instruction se compose de trois parties : la ressource sur laquelle porte l'instruction, la propriété de la ressource à laquelle l'instruction se réfère et la valeur de cette propriété. Les trois parties d'une instruction sont nommées, respectivement, sujet, prédicat et objet. L'objet d'une instruction peut être une autre ressource ou un littéral.

Nous pouvons maintenant fournir une notation formelle pour faire référence aux différents ensembles de termes RDF :

Definition 2.3.1. (Termes RDF :) L'ensemble des termes RDF est l'union de trois paires d'ensembles disjoints : l'ensemble de tous les IRIs (I), l'ensemble de

tous les littéraux (L) et de l'ensemble des nœuds vides (B). L'ensemble de tous les littéraux peut être décomposée dans l'union de deux ensembles disjoints : l'ensemble des littéraux typés (L_t) et non typés (L_p).

La première représentation possible des données RDF consiste à représenter l'ensemble des triplets, toutes les instructions sont représentées par des "triplets" sous la forme sujet-prédicat-objet (ressource, propriété, valeur). La table 2.1 fournit un aperçu simplifié des triplets RDF.

Sujet	**Prédicat**	**Objet**
AlPacino	a-gagné	Oscar
AlPacino	né-à	New York
La loi et l'ordre	vedette	AlPacino
Jon Avnet	est-nommé	Golden globe
La loi et l'ordre	Genre	Drama

Tableau 2.1 – Table de triplets RDF

Une façon plus compacte de représenter un ensemble de triplets consiste à utiliser une représentation à base de graphe orienté et étiqueté. Dans un tel graphe RDF, un triplet est représenté par un arc entre deux nœuds. Le nœud source correspond au sujet, le nœud de destination à l'objet et l'arc au prédicat. Comme illustré dans la Figure 2.3 tout sujet ou objet est représenté par un seul nœud et toutes ses informations associées correspondent à un sous-graphe.

Une autre caractéristique utilisée dans RDF est liée aux nœuds vides. Les nœuds vides sont des ressources anonymes qui ne sont pas exprimées par un URI mais sous la forme de : bnodeID. Le but des nœuds vides est double. Tout d'abord, ils peuvent être utilisés pour coder n-aire relations. Un autre but des nœuds vides est de faire des déclarations sur les ressources qui peuvent ne pas avoir d'URI mais qui peuvent être décrites en termes de relation avec d'autres ressources. Par exemple, imaginez que nous voulons coder la relation entre une ressource qui représente une personne et l'adresse de cette personne qui est composée par le nom de la rue, le code postal et la ville. Ainsi, nous pouvons utiliser un nœud vide pour représenter la notion de l'adresse.

Maintenant, nous introduisons la terminologie d'une façon formelle pour les éléments sur lesquels les graphes RDF sont construits.

Definition 2.3.2. (Terminologie RDF :) Trois ensembles de termes infinis disjoints par paires sont définis comme suit :

— I : l'ensemble des IRIs identifiant les ressources RDF.

— L : L'ensemble des littéraux, où un littéral peut être soit typé ou non.

— B : l'ensemble des nœuds vides, où un nœud vide est localement étendu pour le graphe RDF, et n'identifie aucune ressource.

Nous donnons maintenant la définition d'un triplet RDF dont les éléments appartiennent aux ensembles précédemment définis.

Definition 2.3.3. (Triplet RDF :) Un triplet RDF se compose de trois composants :

— Le sujet s $\in I \cup B$.

— Le prédicat p $\in I$.

— L'objet o $\in I \cup L \cup B$.

Un triplet RDF est généralement écrit dans l'ordre (s, p, o), comme dans les recommandations du W3C pour les syntaxes concrètes RDF

Definition 2.3.4. (Graphe de données :) Un graphe de données est noté $G = \langle V_c, L_V, E, L_E \rangle$ est une collection de sommets qui correspondent à tous Sujets et objets dans un graphe de données, L_V est une collection d'étiquettes de sommets, E est une collection d'arcs orientés qui connectent les sujets et objets correspondants, et L_E est une collection d'étiquettes d'arcs. Étant donné un arc $e \in E$, son étiquette d'arc est sa propriété.

Comme les graphes RDF sont des ensembles de triplets, ils peuvent être combinés facilement, supportant l'utilisation de données provenant de sources multiples. Néanmoins, il est parfois souhaitable de travailler avec plusieurs graphes RDF tout en gardant leur contenu séparé. Les jeux de données RDF supportent cette exigence.

la Figure 2.3 présente un graphe de données contenant des faits liés au film "la loi et l'ordre".

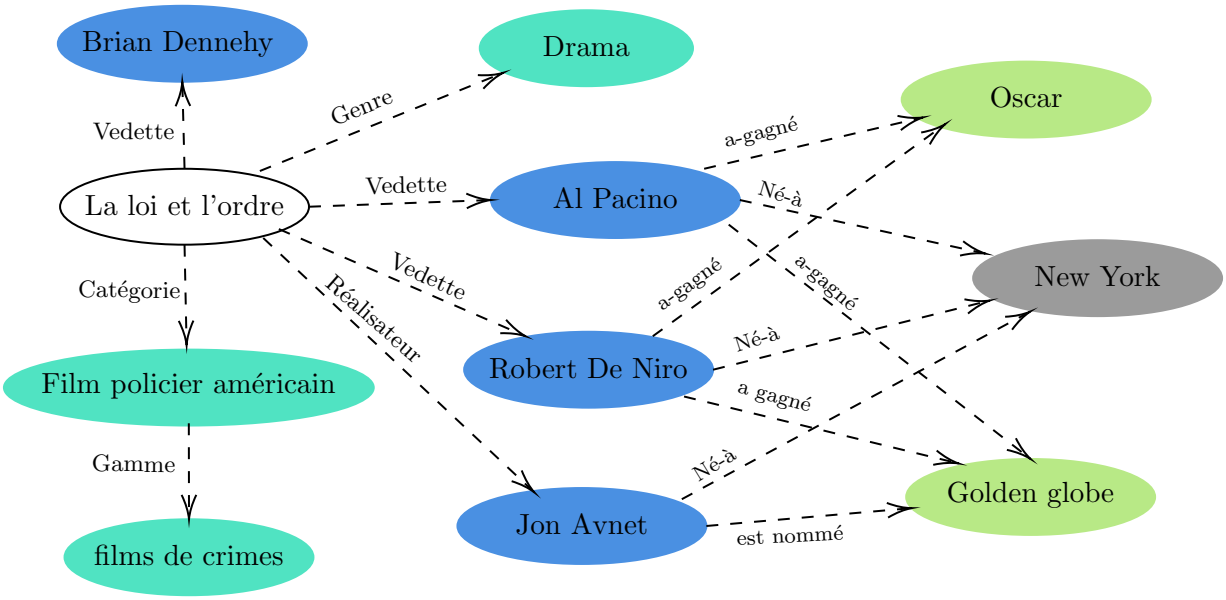


FIGURE 2.3 – Exemple de graphe RDF

Definition 2.3.5. (Jeu de données RDF :) est une collection de graphes RDF, et comprend :

— un graphe par défaut, étant un graphe RDF. Le graphe par défaut n'a pas de nom et peut être vide.

— Zéro ou plusieurs graphes. Chaque graphe nommé est une paire composée d'un nom n $\in$ (I $\cup$ B) (le nom du graphe) et d'un graphe RDF. Les noms de graphe sont uniques dans un jeu de données RDF.

2.3.1 Syntaxe RDF

RDF propose plusieurs syntaxes de sérialisation (syntaxes concrètes), définies dans sa suite de recommandations W3C. Un document RDF en syntaxe concrète code un graphe RDF ou un jeu de données RDF qui permet leur stockage et leur échange entre les systèmes. Cependant, différentes manières d'écrire le même graphe conduisent exactement aux mêmes triplets, et sont donc logiquement équivalentes. Le W3C définit quatre formats de sérialisation pour les données RDF qui sont : RDF/XML, Turtle, N-triples et N3. Il existe également d'autres formats de sérialisation comme RDFa, microdata, RDF-json, etc.

2.3.1.1 RDF/XML

RDF/XML sérialise les données RDF (graphe) en tant que fichier XML dans lequel les nœuds et les arcs du graphe RDF sont représentés à l'aide des éléments XML, d'attributs et de valeurs de texte, comme montre la Figure 2.4. RDF/XML est le premier format de sérialisation RDF adopté par le W3C et le plus utilisé. Le principal avantage de RDF/XML par rapport aux autres formats de sérialisation est qu'il peut être utilisé facilement avec des systèmes et de programmes basés sur XML. Cependant, tous les triplets RDF ne peuvent pas être représentés dans RDF/XML en raison d'une limitation que XML impose à sa syntaxe.

2.3.1.2 N-triple

N-triple est le format le plus simple de représentation textuelle des données RDF, mais c'est aussi la plus difficile à utiliser dans une version imprimée car elle ne permet pas l'abréviation d'URI. Les triplets sont donnés dans l'ordre du sujet, du prédicat et de l'objet sous la forme de trois URI complets séparés par des espaces et entourés de crochets (< >). Chaque déclaration est présentée sur une seule ligne terminée par un point (.). la Figure 2.5 montre un exemple de la représentation syntaxique N-triple de la Figure 2.3.

2.3.1.3 Notation 3

La Notation 3, ou N3 pour faire court, a été proposée par Tim Berners-Lee comme un compromis entre la simplicité des N-triples et l'expressivité de RDF/XML. La notation est très similaire à N-triples, comme illustre la Figure 2.6. Les principales différences syntaxiques sont :

— Les crochets ont été supprimés.

— Les URI peuvent être abrégés par des noms de préfixe.

— Il n'y a aucune restriction sur le nombre d'espaces de séparation.

— Une syntaxe de raccourci pour les instructions partageant un prédicat et/ou un objet a été introduite.

2.3.1.4 Turtle

Turtle (the Terse RDF Triple Language) est une version simplifiée de N3 qui devient de plus en plus populaire et est maintenant une recommandation du W3C.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE RDF [
    <!ENTITY movies "http://isae-ensma.fr/Movies#">
    <!ENTITY local "http://isae-ensma.fr/vocabulaire#">
]>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:movies="http://isae-ensma.fr/Movies#"
    xmlns:local="http://isae-ensma.fr/vocabulaire#">

    <rdf:Description rdf:about="&movies;La_loi_et_l'ordre">
        <local:Vedette rdf:resource="&movies;Brian_Dennehy"/>
        <local:Categorie rdf:resource="&movies;Film_Policier_américain"/>
        <local:Genre rdf:resource="&movies;Drama"/>
        <local:Vedette rdf:resource="&movies;Al_Pacino"/>
        <local:Vedette rdf:resource="&movies;Robert_De_Niro"/>
        <local:Réalisateur rdf:resource="&movies;Jon_Avnet"/>
    </rdf:Description>
    <rdf:Description rdf:about="&movies;Film_Policier_américain">
        <local:Gamme rdf:resource="&movies;Films_de_crimes"/>
    </rdf:Description>
    <rdf:Description rdf:about="&movies;Al_Pacino">
        <local:a-gagné rdf:resource="&movies;Oscar"/>
        <local:a-gagné rdf:resource="&movies;Golden_globe"/>
        <local:Né-à rdf:resource="&movies;New_York"/>
    </rdf:Description>
    <rdf:Description rdf:about="&movies;Robert_De_Niro">
        <local:a-gagné rdf:resource="&movies;Oscar"/>
        <local:a-gagné rdf:resource="&movies;Golden_globe"/>
        <local:Né-à rdf:resource="&movies;New_York"/>
    </rdf:Description>
    <rdf:Description rdf:about="&movies;Jon_Avnet">
        <local:est-nommé rdf:resource="&movies;Golden_globe"/>
        <local:Né-à rdf:resource="&movies;New_York"/>
    </rdf:Description>
</rdf:RDF>
```

FIGURE 2.4 – Syntaxe concrète RDF (RDF/XML)

Dans Turtle, le prédicat est équivalent à l'URI complet correspondant à RDF :type. Les nœuds vierges RDF peuvent soit être explicitement exprimés comme _ :id, où id est l'identifiant du nœud vide, soit anonymement en plaçant tous les triplets qui sont des sujets entre crochets([ ]).

Enfin, Turtle fournit une structure de collection pour les listes sous la forme d'une séquence d'éléments séparés par des espaces et entourés de parenthèses ( ), comme montré dans la Figure 2.7.

2.3.1.5 RDFa

Resource Description Framework in Attributes (RDFa), est une recommandation du W3C qui permet d'ajouter des informations de méta-données aux documents HTML (ou XHTML) en étendant les attributs des éléments. RDFa a été adopté par les moteurs de recherche tels que Google, Bing, Yandex et Yahoo ! pour permettre l'extraction des données par les agents. De plus, RDFa n'est pas limité à un domaine particulier et peut être lié à un vocabulaire conforme, comme montré dans la Figure 2.8.

2.3.2 Le schéma RDF

Comme RDF ne fournit que le modèle de données de base et n'a aucune sémantique orientée domaine, le W3C définit RDF-S au-dessus de RDF et fournit une modélisation primitive qui organise les hiérarchies des objets. RDF-S fournit un vocabulaire de modélisation des données pour les données RDF. RDF-S est une extension sémantique de RDF. Il fournit des mécanismes pour décrire les groupes de ressources et les relations entre ces ressources.

Tout ce qui est décrit par RDF est appelé une ressource RDF ; il est identifié par un URI, et c'est une instance de la classe <RDFs :Resource>, qui est la classe racine de tout. Toutes les autres classes sont des sous-classes de cette classe. <RDFs : Resource > est une instance de la classe RDF principale des ressources <RDFs :Class>. Une propriété RDF <RDF :Property> **est une relation entre les ressources objet et les ressources objet.** La propriété RDF <RDF :type> est utilisée pour définir le type de données d'une ressource RDF en l'associant à une ou plusieurs classes. La propriété de domaine <RDFs :domain> identifie la ou les classes pour lesquelles une propriété est définie. La propriété range < RDFs :range> définit les valeurs d'une propriété qui sont des instances d'une ou plusieurs classes.

Par conséquent, RDFS peut être considéré comme une extension du vocabulaire RDF, et selon les règles d'inférence RDF-S, il a une capacité préliminaire de sémantique et de raisonnement pour des domaines spécifiques.

```
<http://ensma.fr/Movies#La_loi_et_l'ordre> <http://ensma.fr/vocabulaire#Réalisateur> <http://ensma.fr/Movies#Jon_Avnet> .
<http://ensma.fr/Movies#La_loi_et_l'ordre> <http://ensma.fr/vocabulaire#Vedette> <http://ensma.fr/Movies#Robert_De_Niro> .
<http://ensma.fr/Movies#Al_Pacino> <http://ensma.fr/vocabulaire#a-gagné> <http://ensma.fr/Movies#Golden_globe> .
<http://ensma.fr/Movies#La_loi_et_l'ordre> <http://ensma.fr/vocabulaire#Vedette> <http://ensma.fr/Movies#Brian_Dennehy> .
<http://ensma.fr/Movies#Film_Policier_américain> <http://ensma.fr/vocabulaire#Gamme> <http://ensma.fr/Movies#Films_de_crimes>
<http://ensma.fr/Movies#La_loi_et_l'ordre> <http://ensma.fr/vocabulaire#Vedette> <http://ensma.fr/Movies#Al_Pacino> .
<http://ensma.fr/Movies#Robert_De_Niro> <http://ensma.fr/vocabulaire#Né-à> <http://ensma.fr/Movies#New_York> .
<http://ensma.fr/Movies#Al_Pacino> <http://ensma.fr/vocabulaire#Né-à> <http://ensma.fr/Movies#New_York> .
<http://ensma.fr/Movies#La_loi_et_l'ordre> <http://ensma.fr/vocabulaire#Categorie> <http://ensma.fr/Movies#Film_Policier_am\u(
<http://ensma.fr/Movies#Jon_Avnet> <http://ensma.fr/vocabulaire#est-nommé> <http://ensma.fr/Movies#Golden_globe> .
<http://ensma.fr/Movies#La_loi_et_l'ordre> <http://ensma.fr/vocabulaire#Genre> <http://ensma.fr/Movies#Drama> .
<http://ensma.fr/Movies#Jon_Avnet> <http://ensma.fr/vocabulaire#Né-à> <http://ensma.fr/Movies#New_York> .
<http://ensma.fr/Movies#Robert_De_Niro> <http://ensma.fr/vocabulaire#a-gagné> <http://ensma.fr/Movies#Golden_globe> .
<http://ensma.fr/Movies#Robert_De_Niro> <http://ensma.fr/vocabulaire#a-gagné> <http://ensma.fr/Movies#Oscar> .
<http://ensma.fr/Movies#Al_Pacino> <http://ensma.fr/vocabulaire#a-gagné> <http://ensma.fr/Movies#Oscar> .
```

FIGURE 2.5 – Syntaxe concrète RDF (N-triple)

```
@prefix local: <http://isae-ensma.fr/vocabulaire#> .
@prefix movies: <http://isae-ensma.fr/Movies#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://isae-ensma.fr/Movies#La_loi_et_l'ordre> local:Categorie movies:Film_Policier_américain ;
    local:Genre movies:Drama ;
    local:Réalisateur movies:Jon_Avnet ;
    local:Vedette movies:Al_Pacino,
        movies:Brian_Dennehy,
        movies:Robert_De_Niro .

movies:Al_Pacino local:Né-à movies:New_York ;
    local:a-gagné movies:Golden_globe,
        movies:Oscar .

movies:Film_Policier_américain local:Gamme movies:Films_de_crimes .

movies:Jon_Avnet local:Né-à movies:New_York ;
    local:est-nommé movies:Golden_globe .

movies:Robert_De_Niro local:Né-à movies:New_York ;
    local:a-gagné movies:Golden_globe,
        movies:Oscar .
```

FIGURE 2.6 – Syntaxe concrète RDF (N3)

```
@prefix ns0: <http://isae-ensma.fr/vocabulaire#> .

<http://isae-ensma.fr/Movies#La_loi_et_l'ordre>
    ns0:Vedette <http://isae-ensma.fr/Movies#Brian_Dennehy>, <http://isae-ensma.fr/Movies#Al_Pacino>
    ns0:Categorie <http://isae-ensma.fr/Movies#Film_Policier_américain> ;
    ns0:Genre <http://isae-ensma.fr/Movies#Drama> ;
    ns0:Réalisateur <http://isae-ensma.fr/Movies#Jon_Avnet> .

<http://isae-ensma.fr/Movies#Film_Policier_américain> ns0:Gamme <http://isae-ensma.fr/Movies#Films_de_crimes> .
<http://isae-ensma.fr/Movies#Al_Pacino>
    ns0:a-gagné <http://isae-ensma.fr/Movies#Oscar>, <http://isae-ensma.fr/Movies#Golden_globe> ;
    ns0:Né-à <http://isae-ensma.fr/Movies#New_York> .

<http://isae-ensma.fr/Movies#Robert_De_Niro>
    ns0:a-gagné <http://isae-ensma.fr/Movies#Oscar>, <http://isae-ensma.fr/Movies#Golden_globe> ;
    ns0:Né-à <http://isae-ensma.fr/Movies#New_York> .

<http://isae-ensma.fr/Movies#Jon_Avnet>
    ns0:est-nommé <http://isae-ensma.fr/Movies#Golden_globe> ;
    ns0:Né-à <http://isae-ensma.fr/Movies#New_York> .
```

FIGURE 2.7 – Syntaxe concrète RDF (Turtle)

```
<div xmlns="http://www.w3.org/1999/xhtml"
  prefix="
    rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
    local: http://isae-ensma.fr/vocabulaire#
    rdfs: http://www.w3.org/2000/01/rdf-schema#"
  >
  <div typeof="rdfs:Resource" about="http://isae-ensma.fr/Movies#La_loi_et_l'ordre">
    <div rel="local:Réalisateur">
      <div typeof="rdfs:Resource" about="http://isae-ensma.fr/Movies#Jon_Avnet">
        <div rel="local:Né-à" resource="http://isae-ensma.fr/Movies#New_York"></div>
        <div rel="local:est-nommé" resource="http://isae-ensma.fr/Movies#Golden_globe"></div>
      </div>
    </div>
    <div rel="local:Vedette">
      <div typeof="rdfs:Resource" about="http://isae-ensma.fr/Movies#Al_Pacino">
        <div rel="local:a-gagné" resource="http://isae-ensma.fr/Movies#Oscar"></div>
        <div rel="local:a-gagné" resource="http://isae-ensma.fr/Movies#Golden_globe"></div>
        <div rel="local:Né-à" resource="http://isae-ensma.fr/Movies#New_York"></div>
      </div>
    </div>
    <div rel="local:Vedette" resource="http://isae-ensma.fr/Movies#Brian_Dennehy"></div>
    <div rel="local:Categorie">
      <div typeof="rdfs:Resource" about="http://isae-ensma.fr/Movies#Film_Policier_américain">
        <div rel="local:Gamme" resource="http://isae-ensma.fr/Movies#Films_de_crimes"></div>
      </div>
    </div>
    <div rel="local:Genre" resource="http://isae-ensma.fr/Movies#Drama"></div>
    <div rel="local:Vedette">
      <div typeof="rdfs:Resource" about="http://isae-ensma.fr/Movies#Robert_De_Niro">
        <div rel="local:a-gagné" resource="http://isae-ensma.fr/Movies#Golden_globe"></div>
        <div rel="local:Né-à" resource="http://isae-ensma.fr/Movies#New_York"></div>
        <div rel="local:a-gagné" resource="http://isae-ensma.fr/Movies#Oscar"></div>
      </div>
    </div>
  </div>
</div>
```

FIGURE 2.8 – Syntaxe concrète RDF (RDFa)

2.4 Langages de requête RDF

Plusieurs langages de requête ont été proposés pour RDF [Haase et al. (2004)]. La plupart d'entre eux utilisent un modèle de requête basé sur l'algèbre relationnelle [Codd (1970)], où les graphes RDF sont considérés comme une collection de triplets et les requêtes sont des formules basées sur des triplets exprimées sur une seule relation. Malgré les avantages tirés des systèmes de base de données relationnelles existants tels que les mécanismes d'indexation, de stockage sous-jacent des triplets comme relations [Harris and Shadbolt (2005)], les techniques d'optimisation des requêtes, les requêtes relationnelles ne peuvent pas exprimer des relations récursives et même les formes les plus simples tel que la fermeture transitive d'une relation [Aho and Ullman (1979)], directement héritée de la nature graphe des triplets RDF. Il existe de nombreuses applications du monde réel, à l'intérieur et à l'extérieur du domaine du Web sémantique, nécessitant une représentation récursive de données. Pour cette raison, il existe plusieurs tentatives pour étendre l'algèbre relationnelle pour exprimer une modélisation de requête complexe. En dehors du domaine du Web sémantique, nous mentionnons [Agrawal (1988)] qui étend l'algèbre relationnelle pour représenter la fermeture transitive et [Jagadish (1989)] pour représenter les hiérarchies de requêtes. Dans le domaine de RDF, certains langages de requête tels que RQL [Karvounarakis et al. (2002)] tente de combiner l'algèbre relationnelle avec certaines hiérarchies de classes. Il prend en charge une forme d'expressions transitives sur les propriétés transitives RDFS (c-à-d, subPropertyOf et subClassOf) pour naviguer dans les hiérarchies de classes et de propriétés. Versa, RxPath, PRDF [Alkhateeb et al. (2007) et Matono et al. (2005)] sont tous des langages de requête basés sur des chemins pour RDF qui sont bien adaptés à la traversée de graphe mais ne supportent pas les fonctionnalités de type SQL. WILBUR [Lassila (2002)] est une boîte à outils qui intègre des expressions de chemin pour la navigation dans les graphes RDF. [Zhang and Yoshikawa (2008)] discute de l'utilisation d'une description limitée concise (CBD) d'un graphe RDF, qui est défini comme un sous-graphe constitué de ces instructions qui constituent ensemble un ensemble ciblé de connaissances sur une ressource donnée (ou un nœud) dans un graphe RDF donné. Il définit également une version dynamique (DCBD) de CBD et propose un langage de requête pour RDF appelé DCBDQuery, qui aborde principalement le problème de trouver des chemins significatifs (les plus courts) par rapport à DCBD.

Les langages de requête de type SQL pour RDF incluent SeRQL [Broekstra and Kampman (2004)], RDQL [Seaborne (2004)] et son successeur actuel – une recommandation du W3C – SPARQL [Prud'hommeaux et al. (2017)]. Comme il est défini par le groupe de travail sur l'accès aux données (DAWG) du W3C et devient le langage de requête le plus populaire pour RDF, nous avons basé notre travail sur SPARQL. SPARQL sera présenté ci-dessous en plus détail.

2.4.1 Le langage de requête SPARQL

Comme expliqué précédemment, de nombreux langages de requête ont été proposés pour le modèle de données RDF. En 2004, le W3C a lancé le groupe de travail sur l'accès aux données pour la conception d'un langage de requête RDF, appelé SPARQL [Prud'hommeaux et al. (2017)], qui est l'acronyme de simple Protocol et RDF Query Language. Le 15 janvier 2008, SPARQL est devenu le langage de re-

commandation officiel du W3C pour interroger les données RDF.

Dans cette section, nous définissons la syntaxe abstraite pour le langage de requêtes SPARQL.

Definition 2.4.1. (Variable de requête :) Une variable de requête est un membre d'un ensemble infini qui est disjoint de l'ensemble des termes RDF. Nous désignons par V l'ensemble des noms de variables.

Dans SPARQL, une variable de requête est marquée par l'utilisation de "?" ou "$" ; **là "?"ou "$" ne fait pas partie du nom de la variable.** Dans une requête, $abc et ?abc identifient les mêmes variables.

Definition 2.4.2. (Motif de triplets :) Un motif de triplets se compose de trois éléments :

— Le sujet s $\in I \cup L \cup V$.

— Le prédicat p $\in I \cup V$.

— L'objet o $\in I \cup L \cup V$.

Definition 2.4.3. (Motif de graphe basique (BGP) :) Un motif de graphe basique est un ensemble de motifs de triplets.

SPARQL est basé sur la recherche de correspondances de motifs de graphes. Des motifs de graphes complexes peuvent être formés en combinant des motifs plus petits de diverses manières. Nous définissons les motifs de graphes suivants utilisés dans SPARQL :

— **Motif de graphe basique (BGP)**, où un ensemble de modèles triplets doit correspondre. Il combine des motifs de triplets par conjonction.

— **Motif de graphe de groupe**, où un ensemble de motifs de graphe doit tous correspondre. Il combine des motifs de graphes par conjonction. (Pour cela, nous utilisons le mot-clé AND).

— **Motif de graphe optionnel**, où des motifs supplémentaires peuvent étendre la solution. (Pour cela, nous utilisons le mot clé OPTIONAL ou OPT pour plus de concision).

— **Motif de graphe alternatif**, où deux ou plusieurs motifs possibles sont essayés. Il fournit un moyen de combiner des motifs de graphe afin que l'un des motifs de graphe alternatifs puisse correspondre. Si plusieurs alternatives correspondent, toutes les solutions de motifs possibles sont récupérées. (Pour cela, nous utilisons le mot-clé UNION)

— **Modèle de graphe exclu (MINUS)**, où une exclusion des résultats se produit sur la base de la suppression des correspondances de l'évaluation d'un motif de graphe par rapport à un autre modèle de graphe. (Pour cela, nous utilisons le mot-clé MINUS).

— **Motif de graphe filtré**, où les expressions à valeur booléenne (ressemblant à des contraintes) limitent le nombre de réponses à renvoyer. Chaque réponse a un ensemble de liaisons de variables aux termes RDF. Les filtres limitent les solutions à celles pour lesquelles l'expression de filtre a la valeur TRUE. (Pour cela, nous utilisons le mot-clé FILTER).

Definition 2.4.4. (Motif de graphe SPARQL). Un motif de graphe SPARQL est défini de manière inductive de la manière suivante :

— Chaque motif de graphe basique est un motif de graphe SPARQL.

— Si P et P' sont des motifs de graphe SPARQL et que K est une contrainte SPARQL, alors (P AND P'), (P UNION P'), (P OPT P'), (P moins P') et (P filtre K) sont des motifs de graphe SPARQL.

SPARQL fournit également des modificateurs de solution, qui permettent de modifier l'ensemble de résultats en appliquant des opérateurs classiques comme ORDER By pour ordonner l'ensemble de résultats dans un ordre croissant (asc (.) Par défaut) ou décroissant (desc (.)), Distinct pour la suppression des réponses en double, limite pour limiter le nombre de réponses à un nombre fixe (choisi par un utilisateur), projection pour choisir certaines variables et en éliminer d'autres des solutions, ou offset pour définir la position des premières réponses à retourner.

Une requête SPARQL classique a la forme générale ci-dessous, où le préfixe de clause est utilisé pour abréger les URI (qui seront omis dans les exemples suivants), la clause select permet de spécifier quelles variables doivent être retournées, la clause *From* définit les jeux de données à interroger, et la clause *Where* contient les motifs de triplets recherchés.

```
prefix ... #Déclarations de préfixe
select ... #Résultats
from ... #Définition du jeu de données
where ... #Motifs
order by ..., distinct ..., limit ..., offset ..., projection ... #Modificateurs
```

2.4.1.1 Requêtes SPARQL

Les requêtes SPARQL sont composées de deux clauses principales : la première spécifie la forme de requête, tandis que la seconde est la clause WHERE.

SPARQL a quatre formes de requête. Ces formes de requêtes utilisent les solutions de correspondance des motifs de graphes pour former des ensembles de résultats ou des graphes RDF comme montre la Figure 2.9. La forme de requêtes est comme suite :

— SELECT : renvoie toutes les variables ou un sous-ensemble des variables liées dans une correspondance de motifs de requête.

— CONSTRUCT : applique le motif de requête pour remplir les valeurs des variables du motif de résultat et renvoie un graphe RDF. Si nécessaire, les fonctions d'agrégation peuvent être appelées dans le cadre d'une sous-requête dans le motif de requête de type CONSTRUCT.

— ASK : renvoie une réponse booléenne à une requête SPARQL en fonction de la correspondance possible d'un motif de requête avec l'jeu de données.

— DESCRIBE : renvoie un graphe RDF qui décrit les ressources trouvées. Cette approche est particulièrement utile lorsque l'on essaie de découvrir des informations précieuses en suivant les URI des jeux de données.

Dans les travaux de cette thèse, nous ne travaillons qu'avec des requêtes SELECT, car la complexité de requêtes vient de la recherche de motifs de graphe (problème NP-Hard).

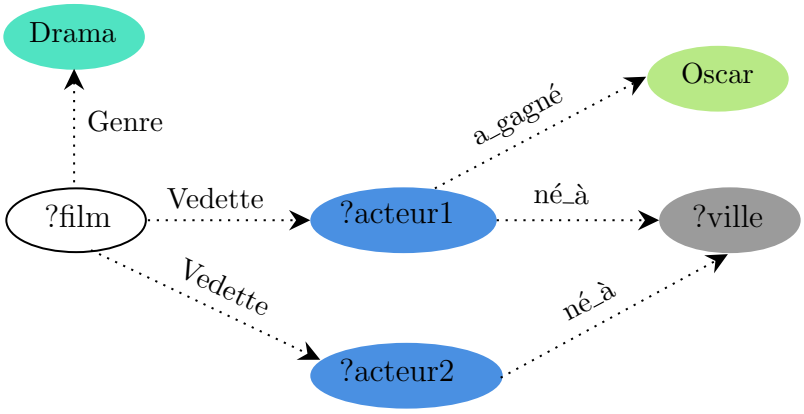


FIGURE 2.9 – Exemple de requêtes SPARQL

Alors que la partie WHERE (c'est-à-dire le motif de requête) de la requête nous permet de restreindre le sous-graphe RDF d'intérêt qui devrait correspondre aux conditions, la partie *SELECT* (c'est-à-dire le motif de résultat) nous permet de sélectionner la partie du résultat que nous voulons exposer. Les clauses de la partie *WHERE* sont définies comme le motif de triplets RDF basique, y compris les variables. Les Variables non présentes dans la partie *SELECT* sont utilisées pour connecter différents motifs de triplets (similaires à la jointure de SQL) et sont qualifiées de non distinguées.

La clause *WHERE* est obligatoire pour tous les formes de requêtes, à l'exception des requêtes *DESCRIBE*, qui peuvent être directement utilisées avec les URI des ressources (par exemple *DESCRIBE* <http ://isae-ensma.fr/>)

Definition 2.4.5. (Requête SPARQL :) Étant donné un modèle de graphe SPARQL P, une séquence $\overrightarrow{B}$ de variables dans P, un IRI µ, et un motif de graphe de base Q,

— ASK FROM µ WHERE P

— SELECT $\overrightarrow{B}$ FROM µ WHERE P

— CONSTRUCT Q FROM µ WHERE P

— DESCRIBE $\overrightarrow{B}$ FROM µ WHERE P

sont des reqêtes SPARQL.

La représentation textuelle de la requête de la Figure 2.9 est la suivante :

```
SELECT ?ville, ?actor1, ?actor2  WHERE {
?film  genre Drama          #TP1
?film Vedette  ?actor1      #TP2
?film Vedette ?actor2       #TP3
?actor1 a_gagné      Oscar #TP4
?actor1 né_à  ?ville        #TP5
```

```
?actor2 né_à  ?ville       #TP6
}
```

En outre, le langage SPARQL fournit des opérateurs pour modifier les résultats de requête :

- **DISTINCT :** pour supprimer les solutions en double.
- **OFFSET :** ignore un nombre donné de résultats.
- **LIMIT :** permet de limiter le nombre de résultats.
- **ORDER BY :** trie les solutions en fonction d'un ensemble d'expressions (variables, fonctions ASC() et DESC (), etc.).
- **GROUP BY :** regroupe des jeux de données pour effectuer des fonctions d'agrégation telles que AVG (), MIN (), MAX () ou COUNT().
- **HAVING :** spécifie une condition limitant les valeurs à apparaître dans le résultat.

SPARQL fournit également les mots-clés supplémentaires suivants : *OPTIONAL*, *FILTER* et *UNION*.

- **OPTIONAL :** permet de récupérer des données même en l'absence de quelque chose correspondant à certains modèles triplets. Les triplets qui ne sont pas obligés de trouver une liaison sont précisément exprimés dans la clause facultative, qui est dans la clause WHERE. Ainsi, il correspond à une jointure externe en algèbre relationnelle et SQL.
- **FILTER :** permet de vérifier davantage si une variable remplit une certaine condition (par exemple, la correspondance de chaînes à l'aide d'expressions regex).
- **UNION :** permet de définir des résultats intermédiaires, fournis par des sous-requêtes, qui seront combinés pour produire un résultat contenant toutes les données satisfaisant au moins un de ces modèles. Chaque sous-requête doit être placée dans son propre ensemble d'accolades ().

2.4.2 Évaluation de requêtes

Dans cette section, nous détaillons le mécanisme d'évaluation de requêtes SPARQL. Nous nous appuyons sur la définition de Zou et al. (2011). Nous nous concentrons seulement sur le traitement des requêtes avec un motif basique de graphes (BGP) et des filtres (de type Wildcards). Les autres opérateurs (Group By, Order By) apportent juste des transformations. Nos requêtes sont formalisées dans la définition 2.4.6.

Definition 2.4.6. (Graphe de requête) Un graphe de requête est désigné par $Q = \langle V, L_V, E, L_E \rangle$, où $V = V_p \cup V_c$ est un ensemble de sommets correspondant à tous les sujets et objets d'une requête, où V_p est une collection de sommets de paramètres, et V_c est tel que défini dans la définition 2.3.4. En tant que convention de dénomination, nous distinguons les variables des éléments de V_c par le biais d'un symbole de point d'interrogation (*e.g.*, ? Nom, ? X), L_V est une collection d'étiquettes de sommet. Pour un sommet $v \in V_p$, son étiquette de sommet est $\emptyset$ [2]. Un sommet $v \in V_c$, E et L_E sont déjà discutés dans la définition 2.3.4.

2. $\emptyset$ est utilisé pour désigner un élément vide

Definition 2.4.7. (Appariement de graphes RDF) Considérons un graphe RDF G (définition 2.3.4) et une requête graphe Q (définition 2.4.6) qui a n sommets $\{v_1, \ldots, v_n\}$. Un ensemble de n sommets distincts $\{u_1, \ldots, u_n\}$ dans G représente un appariement de Q si et seulement si les conditions suivantes sont réunies :

1. Si v_i est un sommet littéral, v_i et u_i ont la même valeur littérale ;
2. Si v_i est une entité ou un sommet de classe, v_i et u_i ont le même URI ;
3. Si v_i est un sommet de paramètre (variable), il n'y a pas de contrainte sur u_i ;
4. Si v_i est un sommet générique, v_i est une sous-chaîne de u_i et u_i est une valeur littérale.
5. S'il y a un arc de v_i à v_j dans Q avec la propriété p, il y a aussi un arc de u_i à u_j dans G avec la même propriété p.

Considérons le graphe de requête décrit dans la Figure 2.9. Les nœuds (*La loi et l'ordre*, *Drama*, *Al Pacino*, *Robert De Niro*, *New York*, *Oscar*) dans le graphe de la Figure 2.3 représentent un appariement de cette requête. Les différentes associations sont montrées dans le Tableau 2.2.

?film	Drama	?acteur1	?acteur2	?ville	Oscar
La loi et l'ordre	Drama	Al Pacino	Robert De Niro	New York	Oscar

Tableau 2.2 – Un exemple d'appariement de graphe

Le processus de recherche de réponses à une requête SPARQL est équivalent à la recherche de tous les appariements d'un graphe de requête dans un graphe RDF. L'évaluation de requêtes RDF peut être défini comme suite :

Definition 2.4.8. (Évaluation de requêtes RDF) Étant donné un graphe de requête Q sur un graphe RDF G, l'évaluation permet de trouver tous les appariements de Q sur G selon la définition 5.

2.5 Conclusion

Ce chapitre présente une compilation de tous les notions, les concepts, et les briques technologiques importants pour comprendre le monde des données ouvertes. Nous avons concentré notre description sur trois éléments principaux qui sont en relation avec notre problématique, à savoir la nature graphe de la donnée traitée, son langage d'interrogation et le processus d'évaluation de requêtes. Cette présentation est alors vue comme un prérequis pour décrire et analyser les systèmes de traitement dédiés aux données liées qui feront l'objet du chapitre suivant.

Chapitre 3

Panorama des Systèmes de Stockage et de Traitement des LOD

Sommaire

3.1 Évolution de la gestion de données 58
3.2 Stockage des données RDF 61
3.2.1 Traitement non-natif de RDF 62
3.2.2 Le stockage natif de RDF 68
3.2.3 Traitement natif de RDF 68
3.3 Systèmes RDF distribués 73
3.3.1 Propriétés des systèmes distribués 74
3.3.2 Avantages des systèmes distribués 75
3.3.3 Limites des systèmes distribués 76
3.3.4 Les systèmes homogènes 76
3.3.5 Les systèmes hétérogènes 78
3.3.6 Traitement massivement parallèle pour RDF 80
3.3.7 Fragmentation des données 84
3.3.8 Allocation des fragments 85
3.4 Évaluation des systèmes de gestion des LOD 86
3.5 Évaluation expérimentale des systèmes existants 89
3.6 Conclusion 90

Dans l'Introduction Générale, nous avons établi une vue d'ensemble sur les systèmes existants de traitement de données RDF et identifié leurs forces et leurs limites. Nous réclamons que nous avons donné des indicateurs solides en faveur de la nécessité de proposer un nouveau système intégrant les forces des systèmes existants. Le fait que la plupart de ces indicateurs ne sont pas quantitatifs, nous nous sommes fixés un objectif de les améliorer avec une approche quantitative. Cette méthode passe par l'audit des systèmes de stockage de données d'une manière générale, et les systèmes de gestion de données RDF en particulier, tout en identifiant leurs composantes. En effet, dans cette thèse, nous nous focalisons principalement sur deux composantes, à savoir le stockage de données RDF et le traitement de requêtes SPARQL.

L'intérêt direct de cette opération d'audit réside dans la facilité de la mise en place d'une compagne de tests et d'évaluation de certains systèmes de gestion de données RDF en considérant deux critères : la performance des requêtes SPARQL et le passage à l'échelle en termes de données. Afin d'avoir une comparaison objective, nous utilisons le même environnement de test (la plateforme matérielle, les jeux de données et requêtes).

Ce chapitre illustre notre démarche de comparaison et d'analyse de systèmes de gestion de données RDF existants, et surtout il confirme la nécessité de proposer un nouveau système performant qui offre un compromis entre les deux critères d'évaluation.

Ce chapitre est structuré comme suit : La Section 3.1 montre l'évolution spectaculaire des systèmes de gestion de données. Cette évolution est motivée par la naissance de nouveaux types de données. La Section 3.2 présente les systèmes de traitement centralisé de données RDF. Chaque type de stockage est illustré par des exemples. Un effort d'analyse et de comparaison selon des critères pertinents sont donnés. La Section 3.3 décrit les systèmes de traitement distribué accompagnées par une analyse. La Section 3.4 présente des benchmarks populaires qui permettent de quantifier une analyse comparative entre les systèmes de gestion de données RDF. Dans la section 3.5, nous présentons notre compagne initiale d'expérimentation couvrant les systèmes suivants : gStrore, RDF3X et Virtuoso. La section 3.6 conclut le chapitre et introduit le chapitre 4.

3.1 Évolution de la gestion de données

La gestion des données, en tant que concept, a commencé dans les années 1960 dans le but d'offrir des techniques d'organisation des données, les étapes utilisées pour atteindre l'efficacité et collecter des informations à partir de ces données. La gestion des données est devenue un problème dans les années 50, lorsque les ordinateurs étaient lents, maladroits et nécessitaient une énorme quantité de travail manuel.

A l'époque, plusieurs entreprises informatisées ont utilisé des étages entiers pour stocker et gérer "uniquement" les cartes perforées stockant leurs données. Ces mêmes entreprises utilisaient d'autres étages pour entretenir des trieuses, des tabulatrices et des banques de perforations de cartes. Les programmes de l'époque étaient configurés sous une forme binaire ou décimale et étaient lus à partir de commutateurs marche/arrêt sur une bande magnétique ou même sur des cartes perforées. Cette forme de programmation s'appelait à l'origine "Absolute Machine Language" (et plus tard a été changée en Langages de programmation de première génération). Les langages de programmation de deuxième génération (anciennement appelés langages d'assemblage) ont été utilisés comme une première méthode d'organisation et de gestion des données. Ces langues sont devenues populaires à la fin des années 50 et utilisaient des lettres de l'alphabet pour la programmation, plutôt qu'une chaîne complexe d'uns et de zéros. De ce fait, les programmeurs peuvent utiliser des mnémoniques d'assemblage, ce qui facilite la mémorisation des codes. Ces langages sont maintenant obsolètes, mais ont contribué à rendre les programmes beaucoup plus lisibles pour les humains et ont libéré les programmeurs de calculs fastidieux et sujets à erreurs.

Les langages de haut niveau (HLL) sont des langages de programmation plus an-

ciens et plus faciles à lire pour les humains. Certains sont encore populaires, d'autres ne le ne sont pas. Ils permettent à un programmeur d'écrire des programmes génériques qui ne dépendent pas complètement d'un type spécifique d'ordinateur. Bien que ces langues mettent l'accent sur la facilité d'utilisation, leur objectif principal est d'organiser et de gérer les données. Différentes langues de haut niveau ont différentes forces : FORTRAN, Lisp, COBOL, BASIC, C et C++. Jusqu'à la fin des année 60, pour gérer les données, un développeur doit gérer tous les aspects d'accès physiques aux données.

Avec la publication de l'article d'Edgar Codd [Codd (1970)] sur l'algèbre relationnelle, une nouvelle étape dans la gestion de données a été franchie. En effet, nous sommes passés d'une gestion fonctionnelle de données à une gestion déclarative. Les systèmes de gestion de données relationnelles (SGBDR) ont d'ailleurs fait leur réputation grâce à la facilité de représentation et de manipulation des données. Les données sont représentées logiquement comme des relations et un utilisateur exprime ses besoins de manière déclarative avec un langage de haut niveau (*e.g.,* SQL [1][Chamberlin and Boyce (1974)]) basé sur l'algèbre relationnelle. Le SGBDR est alors chargé de trouver le meilleur moyen pour stocker et manipuler ces relations.

Historiquement, Le premier SGBDR qui a implémenté l'algèbre relationnelle est le System R d'IBM. Il a vu la lumière en 1974 [Chamberlin et al. (1981)]. Il est d'ailleurs le premier à proposer SQL, qui est devenu le langage standard pour requêter de données relationnelles. Il a également été le premier système à démontrer qu'un système de gestion de bases de données relationnelles pouvait fournir de bonnes performances lors qu'il s'agit de traitements transactionnels. Les décisions de conception dans "System R", ainsi que certains choix d'algorithmes fondamentaux (tels que l'algorithme de programmation dynamique utilisé dans l'optimisation de requêtes[Stonebraker et al. (1976)]), ont influencé ultérieurement de nombreux systèmes relationnels.

Au cours de la même période, deux scientifiques, Michael Stonebraker et Eugene Wong, de l'Université de Californie à Berkeley ont lancé le projet Ingres. Ce projet a duré 15 ans de 1970 à 1985 [Stonebraker et al. (1976)]. L'objectif était de fournir un outil de gestion des données suivant la philosophie d'Edgar Codd. Le code d'origine, comme celui d'autres projets de Berkeley, était disponible à un coût minimal sous une version de la licence BSD. [2] Ingres a influencé un certain nombre d'applications de bases de données commerciales, dont Sybase [McGoveran (1999)], Microsoft SQL Server [McGoveran (1999)], NonStop SQL [Chen et al. (1993)] et plusieurs autres. Ingres assure les propriétés ACID [3][Gray (1981)] et il est entièrement transactionnel (y compris toutes les instructions DDL [4]).

Oracle a été le premier SGBDR à proposer une version commerciale. Il a été initié par Larry Ellison qui a commencé à partir d'une chaîne différente, basée sur les documents d'IBM sur System R [Sumathi and Esakkirajan (2007)]. Oracle a d'ailleurs battu le produit d'IBM avec sa première version sortie en 1978[Kedar (2007)]. Stonebraker a profité de l'expérience acquise lors du développement d'INGRES pour proposer un nouveau SGBDR, Postgres (Post Ingres) [Stonebraker et al. (1990)]

1. Structured Query Language
2. Berkeley Software Distribution License
3. atomicité, cohérence, isolation et durabilité)
4. Data Description Language

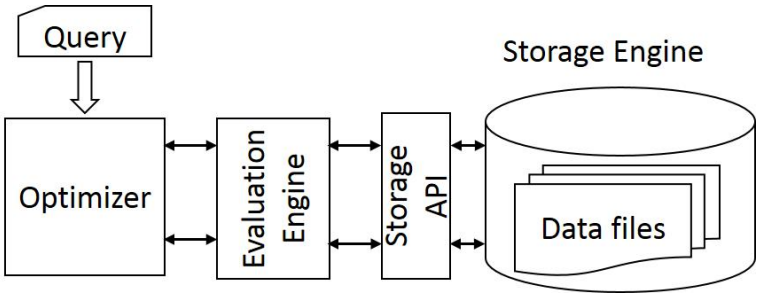


FIGURE 3.1 – Architecture du System R

lancé au milieu des années 80, et qui est connu maintenant sous le nom de PostgreSQL. MySQL [5] a été créé par une société suédoise, MySQL AB. La première version de MySQL est apparue en 1995. Elle a été initialement créée pour un usage personnel à partir de mSQL [6] basé sur le langage de bas niveau ISAM[Ramakrishnan and Gehrke (2000)]. Ses créateurs ont proposé une nouvelle interface SQL, tout en conservant la même API que mSQL. En gardant l'API cohérente avec le système mSQL, de nombreux développeurs ont pu utiliser MySQL au lieu de l'antécédent mSQL (sous licence propriétaire).

De la publication de l'article d'Edgard Codd en 1970 jusqu'à la fin des années 90, de nombreux travaux de normalisation ont été effectués afin d'arriver à l'adoption de SQL comme langage de requête. Les efforts des concepteurs de SGBDR se sont principalement concentrés sur l'intégration de nouvelles extensions à SQL et sur la façon de traiter efficacement les extensions proposées. Il convient de noter que la conception initiale proposée par IBM dans System R n'a pas beaucoup changé.

la Figure 4.13 montre les principaux composants d'un SGBDR. Comme nous pouvons le voir, nous avons trois composants principaux[Chaudhuri (1998)] :

— Moteur de stockage : Ce composant propose des mécanismes permettant de lire et d'écrire des données. Le stockage ne concerne pas uniquement les données de tables (relation) mais également les données liées aux structures complémentaires (*e.g.,* Index) permettant d'accélérer l'accès aux données. Dans la majorité des cas, ce composant est accessible via une API (Application Programming Interface). Pour les principaux SGBDR, au cours de la période 1970-2000, les relations sont stockées au format ligne. Avec ce format, les valeurs du même tuple sont stockées ensemble au même emplacement. Une autre stratégie, basée sur le stockage colonne a été proposée afin de répondre à quelques besoins spécifiques qui sont liés principalement au traitement de données analytiques. Avec ce type de stockage les valeurs de la même colonne sont stockées ensemble.

— Moteur d'évaluation : le moteur d'exécution des requêtes implémente un ensemble d'opérateurs physiques (*e.g.*, tri externe, scan séquentiel, scan d'index). De nombreuses implémentations pourraient être proposées pour chaque opérateur logique. Par exemple, l'opérateur de jointure peut être implémenté à l'aide

5. https ://www.mysql.com
6. http ://www.hughes.com.au/

d'une boucle de hachage ou imbriquée. L'implémentation permet d'accéder aux données à l'aide du moteur de stockage.

— Le composant qui fait la réputation du SGBDR est l'optimiseur. En effet, ce module offre les capacités déclaratives de traitement des requêtes. L'optimiseur prend en entrée une représentation logique de la requête et produit en sortie un plan qui indique la manière dont la requête est évaluée. De plus, les algorithmes (implémentations), l'ordre d'évaluation des différents opérateurs et les structures physiques à utiliser sont indiqués. Il est évident que pour une requête nous pourrions avoir plusieurs plans possibles. Il est nécessaire que l'optimiseur soit capable d'énumérer tous les plans possibles et de comparer les coûts pour choisir le meilleur (ou éviter le pire). L'optimiseur s'appuie sur des statistiques pour calculer le coût d'un plan.

L'évolution du Web, fin des année 90, a impliqué un changement radical dans la gestion de données. En effet, une nouvelle mouvance, appelée NoSQL, est apparue à cause de l'incapacité des SGBDR de gérer d'une manière efficaces les données engendrées par le Web. Les systèmes appartenant à cette catégorie (i.e., NoSQL) combinent des fonctionnalités issues à la fois des moteurs de recherches et des SGBD. L'interface d'interrogation pourrait changer en fonction du domaine métiers et de la particularité du système. L'architecture d'un système NoSQL n'est pas standard et ne suit pas l'architecture des SGBDR. Avec la mouvance Big Data, les systèmes NoSQL ont confirmé leur position en matière de gestion non conventionnelle de données et ce en intégrant des paradigmes des traitements parallèles permettant de garantir le passage à l'échelle.

Dans cette thèse, nous nous intéressons aux systèmes NoSQL orientés graphes, et plus particulièrement la gestion de donnes RDF.

3.2 Stockage des données RDF

Dans cette thèse, nous nous focalisons sur les différentes organisations physiques pour le stockage et l'indexation des données RDF. En plus d'avoir un impact sur la taille requise pour stocker les ensembles de données, ces choix de conception ont une influence importante sur les performances des principaux opérateurs utilisés lors du traitement des requêtes.

La recherche d'un moyen efficace pour stocker et traiter les données RDF est toujours considérée comme un problème ouvert. En effet, cela est dû à la jeunesse du modèle de données RDF (La première recommandation du W3C est arrivée en 1999). Par rapport aux SGBDRs où les premiers systèmes ont été lancés au début des années 70, les systèmes de gestion de données RDF ont vu le jour qu'à partir de 2002 (Sesame 2002). La communauté "bases de données" contribue activement et massivement au développement des systèmes RDF en consacrant plusieurs sessions de ces conférences majeures (VLDB, SIGMOD, EDBT) à l'étude des problèmes liés à l'exploitation de données RDF. De plus, l'implication récente des géants de l'informatique (Oracle, IBM, Microsoft) souligne l'importance de ce domaine pour le monde économique.

Parmi les systèmes existants, nous pouvons distinguer les solutions qui implémentent leur propre système de stockage, désigné comme natif, et celles qui utilisent

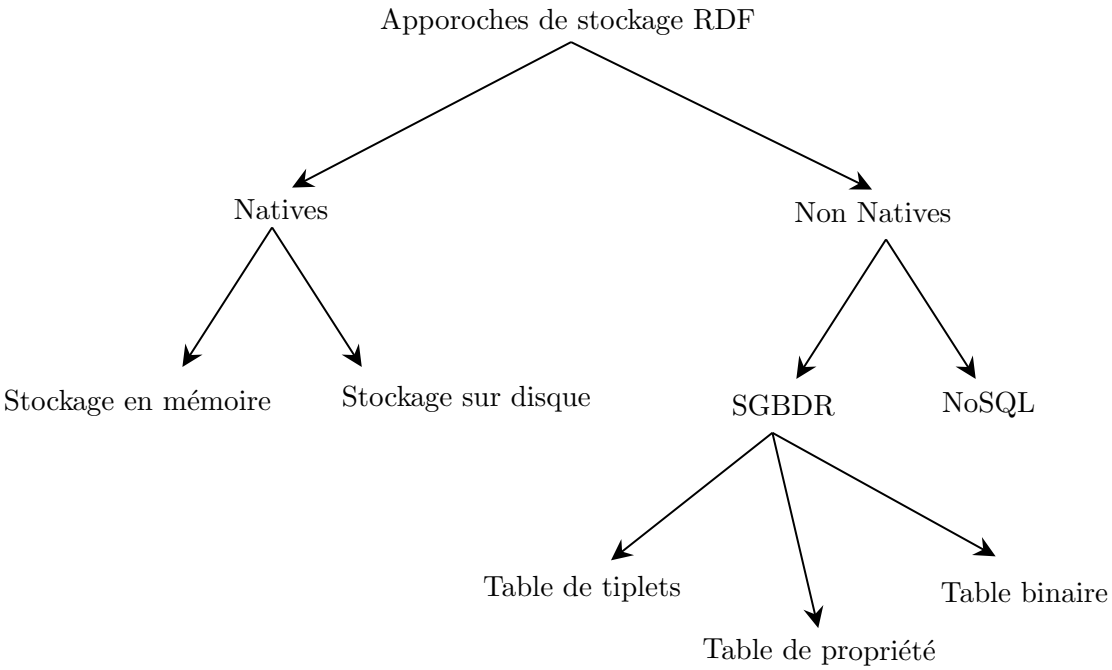


FIGURE 3.2 – Approches de stockage RDF

un système de gestion de base de données existant, désigné comme non natif. La Figure 3.2 montre une classification selon ces deux axes pour le stockage et l'accès aux données RDF. La Figure 2.3 montre un exemple de graphe RDF, qui sera utilisé dans le reste du manuscrit.

3.2.1 Traitement non-natif de RDF

Un ensemble de techniques a été proposé pour stocker les données RDF en utilisant un système existant de gestion de données relationnel. Actuellement, ce type d'approches est largement considéré comme l'approche la plus performante de stockage et de persistance de données en raison de la grande quantité de travail accompli pour le rendre efficace, extrêmement évolutif et robuste.

Le stockage efficace de données RDF a déjà été discuté dans la littérature avec différentes techniques d'organisation physique telles que la table des triplets, la table des propriétés et les approches de partitionnement vertical. La stratégie utilisée par chaque outil dépend du type de performances attendues par l'utilisateur, i.e., les performances de l'évaluation de requêtes ou les performances de la mise à jour de données RDF.

3.2.1.1 Big table

Nommée aussi "triple table", cette approche, introduite tout d'abord dans [Cyganiak (2005)], proposait de stocker et d'interroger les triplets stockés dans une seule très grande table de trois attributs. Les attributs correspondent respectivement au sujet, au prédicat et à l'objet d'un triplet. Pour accélérer les recherches dans cette table, plusieurs index peuvent être ajoutés sur chacune des colonnes. Cette approche

permet de rendre les jointures moins coûteuses.

Le nombre de jointures correspond au nombre de triplets dans la requête et pour les résoudre, il est nécessaire d'analyser la table en entière plusieurs fois, ce qui implique un coût très élevé. Cependant, comme la collection de triplets est stockée dans une seule table RDF, les requêtes peuvent être très lentes à exécuter.

De nombreux systèmes ont adoptés l'approche "triple table", les plus connus sont Oracle, RedLand, 3-Store et Virtuoso. La Figure 3.3 montre un exemple l'organisation en "triple table" pour le graphe RDF représenté dans la Figure 2.3. Les trois colonnes sont consacrées respectivement au stockage des sujets, prédicats et objets.

— **Oracle :**Oracle dans [Chong et al. (2005)] a proposé un schéma basé sur SQL pour interroger les données RDF, en introduisant une fonction spécifique nommée, SQL RDF_MATCH. L'appel à cette fonction engendre une réécriture de la requête en ce basant seulement sur des clauses SQL. Cela permet de profiter du système d'évaluation de requêtes SQL, surtout la gestion de la mémoire, et par conséquence éviter de surcharger le système. Cette approche a été intégré au sein du SGBDR Oracle en utilisant le langage proposé par Oracle pour gérer les fonctions sur des tables relationnelles. De plus, l'implémentation utilise des indexes de type arbre B, des fonctions et des vues matérialisées.

— **3stores :**3stores [Harris and Gibbins (2003)] a été développé dans le cadre du projet Advanced Knowledge Technologies. 3stores prend en charge les implémentations RDF et RDFS sur des bases de connaissances RDF relativement grandes en utilisant une base de données relationnelle pour effectuer les requêtes. 3stores utilise un analyseur RDF standard, à savoir le toolkit Free Software Raptor, pour analyser les sources RDF/XML et transmettre les triplets résultants à la bibliothèque RDFsql de 3stores pour les affirmer dans la base de connaissances. Cette bibliothèque traduit les triplets en instructions SQL de type INSERT et les transmet au SGBDR.

— **Redland :** Redland [Beckett (2002)] est l'un des premiers moteurs proposés pour le stockage et l'interrogation de données RDF. Il peut stocker les graphes RDF en utilisant plusieurs stratégies, i.e., en mémoire principale, dans des bases de données relationnelles et dans des fichiers. Redland stocke trois indexes, basés sur des tables de hachage. L'index PO2S relie une clé sur p et o avec la valeur de s, l'index SO2P relie s et o avec la valeur de p et l'index SP2O relie s et p avec o.

— **RDFKB :**RDFKB (Resource Description Framework Knowledge Base) [McGlothlin and Khan (2009)] est un système de base de données relationnelles adaptés pour les données RDF. Il supporte l'inférence et la gestion de connaissances. Il prend en charge l'inférence au moment du stockage des données plutôt qu'au moment du traitement des requêtes. Toutes les règles d'inférence connues sont appliquées à l'ensemble de données pour déterminer toutes les connaissances possibles. Le principe de la conception de RDFKB est le suivant : pour chaque triplet RDF, tous les triplets RDF supplémentaires possibles seront inférés, stockés, et mises à disposition pour l'évaluation de requêtes. RDFKB supporte l'ajout de nouveaux triplets. De plus, l'inférence est déclenchée une fois qu'un triplet est ajouté, ce qui a comme effet d'augmenter le temps requis pour ajouter et stocker de nouveaux triplets. RDFKB ne supporte pas les transactions liées à la mise à jour.

S	P	O
La loi et l'ordre	vedette	Bryan Dennehy
La loi et l'ordre	vedette	Al Pacino
La loi et l'ordre	vedette	Robert de Niro
La loi et l'ordre	genre	Drama
La loi et l'ordre	Réalisateur	Jon Avnet
La loi et l'ordre	catégorie	Films policier américain
Films policier américain	Gamme	Films de crimes
Al Pacino	a-gagné	Oscar
Al Pacino	a-gagné	Golden Globe
Al Pacino	Né_à	New York
Robert De Niro	Né_à	New York
Robert De Niro	a-gagné	Oscar
Robert De Niro	a-gagné	Golden Globe
Jon Avnet	Né_à	New York
Jon Avnet	est-nommé	Golden Globe

FIGURE 3.3 – Table de triplets

— **Jena SDB :**Le système Jena SDB [Wilkinson et al. (2003) adopte également l'approche à base de "triple table" et correspond essentiellement à un chargeur Java pour plusieurs SGBDR, tels que MySQL, PostgresSQL, Oracle, SQL Server et DB2. Les triplets ou les quadruplets peuvent être chargés respectivement dans une table de triplets ou quadruplets. Intuitivement, les triplets ou quadruplets gérés avec Jena SDB sont stockés dans des tables de trois ou quatre colonnes. Dans le premier type, une clé primaire SPO est définie et des index PO et OS supplémentaires sont créés. Dans le dernier type (c'est-à-dire, quadruplets), la clé primaire est GSPO (avec G est l'URI du graphe nommé). Cinq index supplémentaires y sont créés : GPO, GOV, SPO, OS et PO.

— **Virtuoso :**Le système Virtuoso [Erling (2012)] est un produit commercialisé par OpenLink software. Le moteur de base de données a d'abord été développé en tant qu'un SGBDR et a progressivement évolué vers les modèles de données XML et RDF. Le système Virtuoso stocke aussi des quadruplets en intégrant un identifiant de graphe à chaque triplet s, p, o. Il stocke donc conceptuellement les quadruplets dans une table de triplets avec une colonne supplémentaire. Les colonnes sont g pour l'identifiant du graphe, p pour prédicat, s pour sujet et o pour objet. Les quadruplets sont stockés en utilisant deux indexes de couverture, <g, s, p, o> et <o, g, p, s>, où les URI sont codés dans un dictionnaire. Plusieurs autres optimisations sont ajoutées, y compris l'indexation bitmap Bellatreche et al. (2007). Virtuoso propose une solution open-source sous la licence GPL, qui est disponible pour un déploiement sur une seule machine. La version commerciale permet un déploiement sur un cluster de machines.

3.2.1.2 Tables de propriétés

La technique de table de propriétés a été introduite pour améliorer l'organisation des données RDF en permettant à de multiples triplets qui référencent le même sujet d'être récupéré sans une jointure coûteuse. L'idée de base du stockage de table de propriété est que les triplets RDF, souvent accessibles ensembles, sont regroupés dans la même table relationnelle. Cette approche est très efficace pour les requêtes en forme d'étoile, car elle réduit l'utilisation de la jointure sur des sujets communs.

Cependant, cette approche a aussi de sérieuses limites car une table de propriété peut contenir un nombre significatif de valeurs nulles qui font que la table de propriété est dispersée, ce qui introduit un effort supplémentaire pour le stockage et l'interrogation de ces valeurs nulles.

Un deuxième inconvénient de la table de propriétés réside dans la gestion des attributs multivalués qui sont fréquents dans les données RDF. En effet, ce type d'attributs sont difficiles à exprimer. Dans un modèle de données sans schéma fixe comme RDF, il est courant de rechercher toutes les propriétés définies d'un sujet donné, ce qui, dans l'approche de la table de propriétés, nécessite l'analyse de toutes les tables. Cette approche a néanmoins été adoptée par des outils comme Jena, Sesame, RDFsuite et 4store.

— **Jena :**Jena [McBride (2002)] est l'un des principaux outils conçus spécialement pour le web sémantique. Jena définit deux types de tables de propriétés. Le premier type, appelé table de propriétés en cluster, regroupe les propriétés qui ont tendance à se produire dans les mêmes sujets. Jena s'adapte aux types des attributs (valeurs uniques ou multiples) en proposant des structures adaptées pour chaque type.

  Pour les propriétés à valeur unique, la table contient une colonne pour stocker le sujet et un certain nombre de colonnes pour stocker les valeurs (objets) des propriétés liées au même sujet (référencé dans la première colonne). La valeur d'une propriété donnée peut être Null s'il n'y a pas de triplet RDF, avec cette même propriété, qui est associé au sujet en cours de définition.

  Chaque ligne de la table regroupe les informations associées à plusieurs triplets RDF, dont le nombre peut être inféré en comptant les valeurs non nulles du tuple relationnel. Pour ce type de tables, le sujet est la clé primaire. Pour les propriétés à valeurs multiples, la structure de table comprend le sujet et la propriété à valeurs multiples. Chaque ligne de cette table représente un triplet RDF unique, la clé de la table est la clé composée (sujet, propriété). Jena définit également une table de classe de propriété qui regroupe les sujets avec le même type de propriété dans une table de propriété. Dans ce cas, tous les membres d'une classe sont regroupés ensemble dans une table. La colonne "Type" est la valeur de RDF :type pour chaque propriété de cette ligne. .

— **Sesame :**Sesame [Broekstra et al. (2002)], propose une architecture pour le stockage et l'interrogation des données RDF et RDFS. Sesame est développé par Aidministrator Nederland B.V.3 dans le cadre du projet européen IST-onto-knowledge [7]. Sesame est construit pour intégrer n'importe quel système de stockage puisqu'il implémente une couche de stockage et d'inférence pour interagir avec le système de stockage particulier avec lequel il est associé. Sesame

7. On-To-Knowledge (IST-1999-10132).Voir http ://www.ontoknowledge.org/

vedette, genre, Réalisateur, catégorie,				
	vedette	genre	Réalisateur	catégorie
La loi et l'ordre	$^*T_1$	Drama	Jon Avnet	Films policier américain

a-gagné, Né_à		
S	a-gagné	Né_à
Al Pacino	$^*T_2$	New York
Robert De Niro	$^*T_2$	New York

$^*T_1$	
	vedette
La loi et l'ordre	Al Pacino
La loi et l'ordre	Robert de Niro
La loi et l'ordre	Bryan Dennehy

$^*T_2$	
	a-gagné
Robert De Niro	Oscar
Robert De Niro	Golden Globe
Al Pacino	Oscar
Al Pacino	Golden Globe

est-nommé, Né_à		
S	est-nommé	Né_à
Jon Avnet	Golden Globe	New York

FIGURE 3.4 – Table de propriétés

offre un support pour le contrôle de la concurrence, l'exportation indépendante de données RDF et RDFS et un moteur de requête pour RQL.

— **4Store :**4store [Harris et al. (2009)] a été conçu à l'origine pour répondre aux besoins de données de Garlik, une société Web sémantique basée au Royaume-Uni. 4store est un système de stockage RDF et d'interrogation SPARQL open source. 4store fournit une infrastructure stable pour implémenter le raisonnement décentralisé en chaîne parce qu'il est implémenté en tant que base de données RDF distribuée. 4store distribue les données en segments. Ces segments sont identifiés par un entier et la stratégie d'allocation est basée sur un algorithme de type Round-robin sur le sujet du quadruplet. Un quadruplet est un tuple où le premier élément est l'URI du graphe RDF tant dis que le reste représente la structure du triplet RDF typique (sujet, prédicat, objet).

— **RDFSuite :** RDFSuite [Alexaki et al. (2001)] est une suite d'outils pour la gestion des métadonnées RDF. RDFSuite répond à un besoin de traitement RDF dans les applications Web (telles que les portails Web) qui visent à fournir un contenu d'informations riches composées d'un grand nombre de descriptions de ressources hétérogènes. Il comprend des mécanismes efficaces pour l'analyse et la validation des descriptions RDF, le chargement dans un SGBD ainsi que le traitement de requêtes et l'optimisation dans RQL.

L'avantage de l'approche de la table de propriétés est que les jointures dans les requêtes en étoile (c'est-à-dire les jointures sujet-sujet) deviennent des simples parcours de table unique. Par conséquent, la requête traduite a moins de jointures. Les inconvénients associés à ce type d'approches, sont liées principalement au nombre significatif de valeurs nulles dans les tables. Aussi, traiter avec propriétés multivaluées nécessite un effort particulier.

En outre, bien que les requêtes en étoile puissent être traitées efficacement, cette approche peut ne pas aider beaucoup avec d'autres types de requêtes. Enfin, lorsque l'affectation manuelle est utilisée, le regroupement de propriétés "similaires" est non trivial et de mauvaises décisions de conception implique la gestion du problème lié aux valeurs nulles.

Né_à	
S	O
Robert De Niro	New York
Jon Avnet	New York
Al Pacino	New York

genre	
S	O
La loi et l'ordre	Drama

vedette	
S	O
La loi et l'ordre	Bryan Dennehy
La loi et l'ordre	Al Pacino
La loi et l'ordre	Robert De Niro

Réalisateur	
S	O
La loi et l'ordre	Jon Avnet

est-nommé	
S	O
Jon Avnet	Golden Globe

Catégorie	
S	O
La loi et l'ordre	Films policier américain

a-gagné	
S	O
Al Pacino	Oscar
Al Pacino	Golden Globe
Robert De Niro	Oscar
Robert De Niro	Golden Globe

Gamme	
S	O
Films policier américain	Films de crimes

FIGURE 3.5 – Tables binaires

3.2.1.3 Tables binaires

L'approche de partitionnement vertical ou tables binaires, suggérée dans swStore [Abadi et al. (2007)], est une alternative à la solution basée sur la table de propriétés. Elle permet en effet d'améliorer le traitement de requêtes, tout en offrant des performances similaires et une mise en œuvre plus facile. Dans cette approche, les données RDF sont partitionnées verticalement à l'aide d'un modèle de stockage entièrement décomposé (DSM).

Chaque table de triplets est divisée en n tables à deux colonnes, où n est le nombre de propriétés uniques mentionnées dans les triplets RDF. Dans chacune de ces tables, la première et la deuxième colonne stockent respectivement les valeurs du sujet et de l'objet. La Figure 3.5 montre la représentation des données de la Figure 2.3 en utilisant des tables binaires. Les tables sont stockées, en utilisant un SGBD orienté colonne comme un ensemble de colonnes plutôt qu'un ensemble de lignes. swStore utilise aussi des vues matérialisées pour accélérer les jointures fréquentes. La colonne liée à l'objet peut également être indexée (par exemple, en utilisant un arbre de type B+). Une deuxième copie de la table peut être aussi créée en regroupant les colonnes d'objet. L'un des principaux avantages du partitionnement vertical est la prise en charge efficace des jointures sujet–sujet. Cet avantage est obtenu en triant les tables par sujet. Cette approche fonctionne très efficacement pour les requêtes impliquant un petit nombre de prédicats. Inversement, si ce nombre augmente, le système devient très inefficace en raison du coût élevé lié au traitement des jointures.

3.2.2 Le stockage natif de RDF

Les solutions de stockage natives offrent un moyen personnalisé de stockage de données RDF qui permet de garder la structure logique liée au modèle de données. Elles utilisent la nature des triplets RDF comme un atout. Ces systèmes peuvent être généralement classés comme systèmes basés sur disque, c'est–à-dire persistants, et systèmes basés sur la mémoire, c'est-à-dire volatiles.

3.2.2.1 Le stockage en mémoire

Dans cette catégorie de systèmes, la mémoire centrale est utilisée pour stocker les différentes structures permettant de répondre aux requêtes sur les données RDF. Ce type d'approches repose sur la maturité des résultats de recherche dans le domaine des bases de données et surtout sur tout ce qui est des techniques à base d'indexation multiple. L'inconvénient majeur de ce type d'approches réside dans le pré-traitement et plus particulièrement tout ce qui est chargement et analyse des fichiers RDF, ainsi que la créations d'indexes appropriés. Le temps consacré à ce type d'opérations a impacté l'adoption de ce type de systèmes par les utilisateurs lambda. De plus, un système RDF doit avoir une représentation efficace de données en mémoire afin de laisser suffisamment d'espace pour le fonctionnement des algorithmes d'évaluation. Parmi les systèmes les plus connus on peut citer entre autres, Jena [McBride (2001)], Hexastore [Weiss et al. (2008)], et Bitmat [Atre et al. (2008)].

3.2.2.2 Le stockage sur disque

Le stockage persistant sur disque est un moyen qui permet de garder en permanence les données RDF en utilisant un système de fichiers. Les systèmes entrant dans cette catégorie peuvent s'appuyer sur des structures d'indexes connues, telles que les arbres B+. Il faut noter que l'accès au disque pourrait ralentir le processus d'évaluation et le rendre inacceptable en termes de performances. On peut distinguer alors entre les représentations autonomes et les représentations intégrées. D'une part, les représentations autonomes et natives de RDF peuvent être stockées, transmises et traitées par leurs propres moyens sans se référer à une configuration matérielle spécifique. Par exemple, une représentation DOM (in-memory Document Object Model) d'un document RDF/XML peut être construite à l'aide d'un analyseur SAX. Parmi les formats célèbres utilisés dans ce contexte, on peut citer N3, n-triplets, Turtle, RDF/XML et Trix. D'autre part, les représentations natives RDF intégrées font partie d'une application/ framework spécifique et ne sont utilisables que via ce dernier. Leur représentation est définie par rapport à un contexte lié à ce cadre applicatif. D'ailleurs, les triplets sont produits en appliquant une transformation. RDFa, eRDF et SMW sont les solutions les plus connues proposées dans ce contexte.

3.2.3 Traitement natif de RDF

Les approches appartenant à cette catégorie évitent l'utilisation d'un SGBDR et se concentrent plutôt sur des techniques d'indexation spécifiques au modèle de données RDF. Elles proposent des méthodes pour réorganiser les données en mémoire ou sur disque. Le traitement de requêtes pourrait être effectué plus efficacement par rapport aux approches simples comme les "triple table". Ce type de systèmes est

motivé par le fait que l'utilisation d'un SGBDR traditionnel pour le stockage de données RDF entraîne la propagation de limites telles que le schéma rigide, qui est l'une des principales raisons de l'adoption du modèle de données RDF. Les différentes propositions visent à être plus proches du modèle de requêtes adopté par le Web sémantique.

— **RDF-3X :** RDF-3X [Neumann and Weikum (2008)] est un moteur de stockage et d'exécution de requêtes SPARQL. Il permet de manipuler de grands dépôts de triplets RDF. Il a été conçu, mis en œuvre, étendu et maintenu à l'Institut Max Planck en Allemagne. Il est considéré comme l'état de l'art dans les systèmes de stockage RDF. Il est d'ailleurs cité dans un nombre énorme d'articles scientifiques. Il est considéré aussi comme le système indispensable pour valider les approches proposées en relation avec la gestion de données RDF.

  RDF-3X propose un système personnalisé pour la gestion de données RDF. Il dépend de son propre système de stockage qui a été spécialement conçu pour les données RDF. Pour cela il se distingue des autres conceptions de systèmes de gestion de triplets par l'utilisation extensive des jointures. Tous les triplets de données sont stockés dans un arbre de type B+, groupés et compressés. Les triplets sont triés lexicographiquement dans l'arbre B+. RDF-3X utilise les six permutations possibles des S, P, O c-à-d (SPO, SOP, OSP, OPS, PSO et POS.) alors il dispose de six index différents. Pour limiter l'empreinte mémoire, la solution adopte une approche de codage de dictionnaire standard et obtient donc un taux de compression intéressant. La stratégie d'indexation de RDF-3X repose également sur neuf autres index : six stockent deux entrées sur trois d'un triple (SP, SO, PS, PO, OS et OP) et sont appelés index agrégés.

— **YARS :** Le système YARS [Harth and Decker (2005)] combine des méthodes de récupération d'informations et de bases de données pour permettre de meilleures performances lors du traitement de requêtes sur des données RDF. YARS fournit des indexes pour tous les modèles d'accès RDF avec contexte. Il utilise des indexes sur les représentations de chaînes de caractères et sur les graphes RDF. Il stocke les données RDF en permanence en utilisant six index B+. YARS stocke le sujet, le prédicat et l'objet, et aussi des informations liées au contexte associé à l'origine des données. Les six indexes construits couvrent tous les modèles d'accès possibles des quadruplets sous la forme s, p, o, c, où c'est le contexte du triplet (s,p,o). YARS sacrifie l'espace et la vitesse d'insertion pour donner priorité aux performances de l'évaluation de requêtes. En effet, pour récupérer n'importe quel modèle d'accès avec une seule recherche d'index, chaque triplet est encodé dans le dictionnaire six fois, avec un ordre de tri différent. L'inférence n'est pas prise en charge par le système YARS.

— **HEXASTORE :** [Weiss et al. (2008)] Hexastore a trouvé que la plupart des approches de stockage de triplets existantes ont des limites quand il s'agit de garantir le passage à l'échelle et/ou de traiter les requêtes avancées (Wildcard, Tri, Agrégation).

  Hexastore est basé sur l'idée de l'indexation multiple en mémoire principale des données RDF. Les données RDF sont indexées de six façons possibles, une pour chaque ordre possible des trois éléments RDF. La représentation est basée sur n'importe quel ordre d'importance des ressources et des propriétés de RDF

et peut être considérée comme une combinaison de partitionnement vertical et d'approches d'indexation multiple. Pour limiter la quantité de stockage nécessaire pour les URI, Hexastore utilise le codage de dictionnaire typique des URI et des littéraux c-à-d que chaque URI ou littéral reçoit un identifiant numérique unique. Hexastore favorise les performances des requêtes par rapport au temps d'insertion. Les opérations de mise à jour et d'insertion affectent les six indexes et peuvent donc être lentes. Juste comme YARS, Hexastore ne fournit pas de support d'inférence.

— **Kowari :** Kowari [Wood (2005)] est un projet Open Source sous licence "Mozilla Public License". Il est conçu spécialement pour le stockage, l'extraction et l'analyse de métadonnées. Kowari fournit une infrastructure de stockage évolutive et sécurisée pour les déclarations RDF et utilise iTQL [8] pour l'analyse et l'exécution des requêtes. Le système utilise une approche similaire à YARS. En effet, les instructions RDF sont également stockées sous forme de quadruplets dans lesquels les trois premiers éléments forment un triplet RDF standard et le quatrième décrit dans quel modèle l'instruction apparaît. L'approche utilise six ordres différents d'éléments quadruplet agissant comme un index composé, et contient indépendamment toutes les données RDF. Dans cet ordre, les quatre éléments du quadruplet peuvent être disposés de telle sorte que n'importe quelle collection d'un à quatre éléments peut être utilisée pour trouver n'importe quel triplet ou groupe de triplets correspondant. Cependant, Kowari utilise un arbre hybride AVL et b-trees (au lieu de B+trees) pour l'indexation multiple. Plusieurs versions du système Kowari ont été publiées lors d'une collaboration avec Tucana Technologies Inc., qui a été acheté en septembre 2005 par Northrop Grumman.

— **BitMat :** BitMat [Atre et al. (2008)] utilise une matrice de bits tridimensionnelle (sujet, prédicat, objet), dans laquelle chaque cellule est un bit représentant un triplet unique en indiquant la présence ou l'absence de ce triplet par la valeur de bit 1 ou 0. Cette matrice peut être vu comme un cube de bits de trois dimensions.

  BitMat est basée sur la mémoire principale pour représenter un grand ensemble de triplets RDF. Pour représenter le cube de bits en mémoire, il est aplati dans une matrice bidimensionnelle. Il y a six façons d'aplatir un cube de bits en BitMat. Chaque structure contribue au traitement efficace d'un ensemble de requêtes de mono-jointures. BitMat est conçu pour être principalement un système de stockage de triplet RDF en lecture seule. L'insertion ou la suppression dynamique de triplets RDF n'est pas supportée.

— **BRAHMS :** BRAHMS[Janik and Kochut (2005)] est un système de stockage RDF offrant la base nécessaire pour la mise en œuvre des algorithmes de découverte d'associations sémantiques rapides. BRAHMS n'a pas été conçu pour les modifications de données RDF, mais seulement pour les requêtes axées sur la récupération de données. Il emploie six indexes, c'est-à-dire deux par dimension. Ces indexes sont nécessaires pour une récupération rapide des nœuds voisins, ainsi que pour les fusionner pendant le processus de découverte de l'association sémantique.

8. Tucana Technologies, iTQL Commands, http ://kowari.org/ 271.htm

— **Parliament :** Parliament[Kolas et al. (2009)] est un système de gestion de donnée RDF développé par Raytheon BBN Technologies [9] et utilisé depuis 2001. Il décrit un schéma de stockage et d'indexation basés sur des listes chaînées et des fichiers en mémoire avec une structure de stockage composée de trois parties :

1. la table de ressources qui est un simple fichier d'enregistrements de longueur fixe. Les enregistrements sont numérotés séquentiellement. Cela permet un accès direct à un enregistrement en fonction de son ID. Chaque enregistrement comporte huit composants : trois champs d'ID représentant les premières instructions qui contiennent cette ressource en tant que sujet, prédicat et objet, respectivement ; trois champs de comptage contenant le nombre d'instructions utilisant cette ressource en tant que sujet, prédicat et objet, respectivement ; un décalage utilisé pour récupérer la représentation de chaîne de la ressource ; et des drapeaux de champs de bits codant divers attributs de la ressource. Les trois premières valeurs pointent vers une position dans la table d'instructions.
2. la table de déclaration qui est similaire à la table de ressources, est un fichier unique d'enregistrements de longueur fixe (avec un ID par déclaration), dont chacun représente une seule déclaration. Cette table est composée de sept champs : trois ID de ressources pour les positions sujet, prédicat et objet, trois pointeurs vers la position suivante de cette valeur respectivement au sujet, prédicat et position de l'objet, des drapeaux de champ de bits qui aident à soutenir les opérations de mise à jour
3. le dictionnaire de ressources pour fournir une correspondance bidirectionnelle entre une ressource et son ID. Parliament est conçu comme un système de gestion de données RDF intégré et ne comprend pas de logiciel SPARQL ou d'autre langage de requête.

— **iStore :** iStore [Tran et al. (2009)] est une approche pour le partitionnement de données et le traitement de jointure, qui exploite un index de structure construit automatiquement à partir des données. Cet index peut agir comme un schéma, permettant la navigation et l'interrogation efficaces de données du Web sans schéma. La structure d'index est essentiellement un graphe qui peut être calculé pour les graphes RDF en général, ce qui permet de capturer plusieurs structures de modèles liés aux données. Les sommets de ce graphe représentent des groupes d'éléments de données dont la structure est similaire. La structure fait référence à l'ensemble des connexions entrantes et sortantes. Ainsi, les propriétés du schéma sont utilisées pour effectuer un partitionnement vertical. Le traitement d'une requête donnée commence par la mise en correspondance avec l'index de structure pour identifier les groupes de données qui correspondent à la structure globale de la requête. Le partitionnement proposé par iStore résulte en un stockage contigu de triplets qui ont la même structure.

— **TripleT :** L'approche TripleT [Fletcher and Beck (2009)] adopte une technique d'indexation partielle pour limiter l'espace de stockage utilisé par la structure de données globale. La proposition accorde une attention à la localisation des

9. Voir : https ://www.raytheonintelligenceandspace.com/capabilities/advanced-tech

données pour éviter les situations où une donnée apparaît à plusieurs emplacements. Une seule structure de données de mémoire secondaire B+tree est créée et contient chaque ressource, littérale ou URI, quelle que soit la position (sujet, prédicat ou objet) dans l'ensemble de triplets. La position à laquelle une ressource se produit en triplets est prise en charge par trois compartiments distincts (chaque ressource de l'index y est associée).

— **gStore :** Le système gStore [Zou et al. (2011)] considère les ensembles de données RDF d'un point de vue graphe, plus précisément comme des ensembles d'arcs étiquetés. Ce choix est motivé par trois caractéristiques liées à la nature données RDF : les graphes RDF ont généralement une taille importante (nous parlons de plusieurs milliards d'arcs), le nombre d'occurrences d'étiquettes de sommet et d'arête est plus grand et la proportion de requêtes SPARQL pratiques prenant la forme d'une requête en étoile est importante. Le système gStore a deux objectifs supplémentaires par rapport aux autres systèmes existants : prendre en charge les opérations de mise à jour sur les triplets RDF et répondre efficacement aux requêtes SPARQL contenant des filtres à base d'expressions régulières.

  Intuitivement, le moteur de stockage attribue à chaque ressource un identifiant entier unique et une signature. La signature, stockée sous forme de chaîne de bits, contient des informations sur les triplets dans lesquels la ressource joue le rôle d'un sujet. Une structure, prenant la forme d'un arbre, dénommé arbre de signature de sommet, est conçue pour indexer les signatures de toutes les ressources. Cet index correspond à un arbre équilibré où une feuille est la signature d'une ressource et les nœuds internes sont des signatures correspondant aux bits ou aux signatures des fils. Cette structure permet de localiser efficacement, à partir de sa racine, une certaine signature de ressources. La prise en charge d'un traitement efficace des requêtes RDF avec des expressions régulières est la principale motivation derrière la création de cet index.

— **TripleBit :** TripleBit[Yuan et al. (2013)] est un système de gestion de données RDF basé sur une structure de stockage à base de matrice de bits bidimensionnelle. Dans cette matrice, les colonnes correspondent à un regroupement de prédicats, et les lignes sont des sujets et des objets. La matrice est composée de valeurs booléennes avec 1 spécifiant la présence d'une paire sujet–prédicat ou objet–prédicat. Par conséquent, deux 1 ne peuvent être stockés que par colonne.

  Le système introduit deux structures d'indexation à base de matrices de bits : ID-chunk (morceau) et ID-prédicat. Le premier supporte la recherche rapide de morceaux pertinents correspondant à un sujet ou un objet donné. Ce dernier fournit une solution de mise en correspondance d'un sujet ou d'un objet à une liste des propriétés connexes. Le système utilise une approche de dictionnaire qui utilise une méthode de compression de préfixe.

— **Allegrograph :** Allegrograph est un système commercial développé par Franz Inc. Il est parmi les premiers systèmes de gestion de données RDF. L'apparition de sa première version date de 2004. Il adopte une présentation à base de graphe et il définit le système comme une base de données de graphes NoSQL. Néanmoins, parce que le système est conçu sur sa propre implémentation, il est considéré comme une approche de stockage RDF native. Comme les

autres systèmes de base de données de cette catégorie, tels que Neo4J, la compatibilité complète des transactions ACID est implémentée. Allegrograph prend en charge sept index : SPOGI, POSGI, OSPGI, GSPOI, GPOSI, GOSPI et I, où S, P, O sont les entrées standard liées aux sujet, prédicat et d'objet d'un triplet, et G représente un graphe nommé basé sur des URI. Le I représente un identifiant de triplets unique du système de gestion de données RDF.

— **Stardog :**le système commercial Stardog [Cerans et al. (2012)] est développé par Clark & Parsia. Il est bien connu dans la communauté web sémantique pour la mise en œuvre du système de raisonnement OWL pellet [Sirin et al. (2007)]. Le système est annoncé comme une base de données graphe qui utilise des données RDF, SPARQL pour les requêtes et OWL pour le raisonnement. Deux modes d'indexation sont pris en charge dans Stardog, le premier est basé uniquement sur les triplets tant dis que le deuxième est conçu spécialement pour les quadruplets. Dans les deux cas, les indexes sont stockés sur le disque, mais un mode "mémoire" est disponible aussi.

— **GRIN :**Le système GRIN [Udrea et al. (2007)] a été le premier système d'indexation RDF à utiliser le partitionnement à base de graphes. Son objectif est de fournir une solution d'indexation adaptée aux approches d'évaluation de requêtes basées sur la recherche des appariements de graphes. L'objectif de cette structure d'indexation est de maintenir un ensemble des nœuds du graphe qui sont proches les uns des autres dans le graphe.

  Ceci est obtenu en utilisant les notions de centralité et de distance entre les nœuds d'un graphe. Un rayon est calculé pour chaque sommet central. Tous les sommets dans un rayon donné sont stockés dans la même structure que le nœud central. Néanmoins, GRIN ne fonctionne pas sur disque. La taille de données qui peuvent être stockées est relativement limitée.

3.3 Systèmes RDF distribués

Les systèmes de gestion de base de données RDF que nous avons étudié jusqu'à présent sont caractérisés comme centralisés, ce qui signifie que le stockage et le traitement de requêtes sont pris en charge en utilisant une seule machine. Ce type d'architecture n'arrive pas à s'adapter aux nouveaux besoins liés à la gestion de données RDF, surtout avec l'arrivée du Big Data qui nécessite généralement le traitement de très grands volumes de données. De plus, les données et le traitement associé, tels que les réponses aux requêtes, sont répartis sur un ensemble de machines. Par conséquent, pour les applications confrontées à des contraintes liés au volume de données, un système de gestion de base de données distribuée (DDBMS) est requis. Un DDBMS peut être défini comme un système où la gestion des données est répartie sur plusieurs machines ayant une communication qui repose sur un réseau informatique.

Faisant partie intégrante des écosystèmes Big Data, RDF est totalement concerné par les phénomènes liés au calcul massive. Les systèmes RDF distribués ont bénéficié de l'expérience et des résultats obtenus dans le contexte de DDBMS relationnels. Par conséquent, il n'est pas surprenant d'observer plus ou moins les mêmes catégories de système même si la nature spéciale de RDF impose quelques particularités. La Figure 3.6 présente une taxonomie des systèmes RDF distribués.

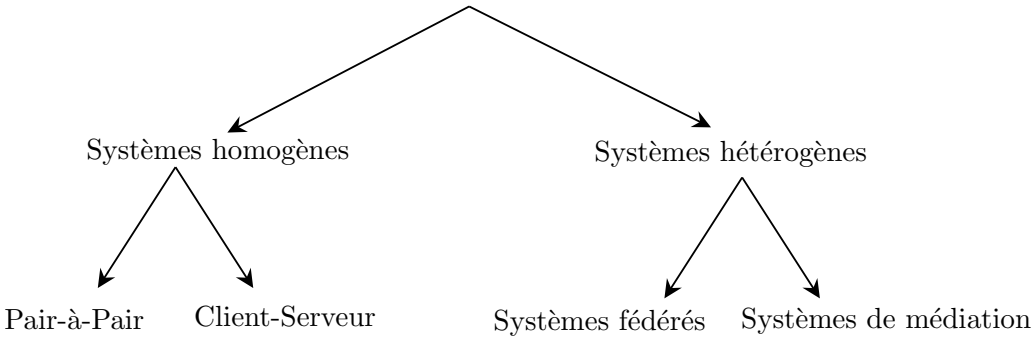


FIGURE 3.6 – Taxonomie des systèmes RDF distribués

Dans les DDBMS homogènes, les nœuds qui gèrent les bases de données distribuées correspondent à des systèmes similaires. Les principales architectures de DDBMSs homogènes sont basées sur des approches P2P ou client-serveur. En revanche, les nœuds d'un système hétérogène sont censés exploiter différents types de SGBD. Nous pouvons identifier deux types principaux de systèmes distribués hétérogènes RDF en fonction de la présence d'un schéma global de médiation par rapport aux différentes bases de données. L'approche basée sur le médiateur correspond aux systèmes OBDA. Les solutions qui ne nécessitent pas de schéma global sont désignées comme des systèmes RDF fédérés.

3.3.1 Propriétés des systèmes distribués

— Un aspect commun entre approches homogènes et hétérogènes est celui lié à la notion de transparence, qui peut prendre différentes formes. Le plus important du point de vue de l'utilisateur final ou du développeur d'applications est la transparence de la requête. Ceci est attendu des systèmes RDF distribués homogènes et à base de médiateur. Intuitivement, on écrit des requêtes comme si la base de données était centralisée c.-à-d. que l'utilisateur final ou le développeur n'a pas besoin de savoir que la base de données est distribuée, sur quels nœuds du cluster certaines données sont stockées, etc. Dans les systèmes homogènes et à base de médiateur, la présence d'un schéma global permet de profiter de cette propriété.Le module de traitement de requête est ainsi chargé des tâches de la traduction de la requête originale en un ensemble de requêtes, qui seront envoyées à certains des nœuds de cluster. Cette forme de transparence ne s'applique pas à une approche fédérée, qui est caractérisé par l'absence d'un schéma global. En effet, l'utilisateur final doit spécifier le point de d'accès SPARQL qui peut répondre à certains modèles de triplets.

— Une autre forme de transparence est liée au schéma c.-à-d. que les systèmes distribués décident de manière autonome quel nœud du cluster sera utilisé pour placer certaines données. Cela concerne principalement la catégorie des systèmes homogènes car, en général, on ne contrôle pas vraiment cet aspect avec des systèmes hétérogènes. Cette propriété donne à l'administrateur de base de données (DBA) l'impression qu'il administre une base de données centralisée car la plupart des aspects de la distribution sont pris en charge par le logiciel. Il est généralement préférable, pour des raisons de performance, de

donner au DBA un certain contrôle sur la méthode de distribution.

— D'autres formes de transparence ont fait l'objet de nombreux travaux en relation avec le modèle relationnel mais ne concernent pas, du moins pour le moment, les systèmes RDF. Prenons le cas de la transparence des mises à jour, qui permet de spécifier les mises à jour de la base de données sans assumer les aspects de distribution. Très peu des systèmes RDF distribués abordent efficacement cette transparence. Par conséquent, on peut comprendre que la transparence des transactions n'a pas fait l'objet de projets de recherche pour les systèmes RDF. Cette forme de transaction vise à garantir que toutes les transactions distribuées maintiennent l'intégrité et la cohérence de la base de données.

3.3.2 Avantages des systèmes distribués

De nombreuses améliorations d'utilisation peuvent être obtenues en se basant sur la fiabilité des systèmes de base de données distribuée, la haute disponibilité et les performances, le passage à l'échelle et l'intégration des données. Nous avons souligné que la distribution des données va souvent de pair avec la réplication. Les avantages que l'on peut obtenir avec une telle approche est une fiabilité améliorée du système. Une conséquence de la fiabilité accrue est une plus grande disponibilité, en particulier par rapport à un système centralisé. En cas de panne, ce dernier est incapable de fonctionner alors qu'un système distribué, peut encore être en mesure d'effectuer certaines ou toutes les opérations. Cet aspect fait partie de la conception d'un système distribué.

Une haute performance du système est obtenue pour les raisons suivantes : 1) avec une distribution de données intelligente, on peut s'assurer que les données nécessaires à certaines requêtes sont stockées à des nœuds géographiquement plus proches de l'émetteur de ces requêtes. Cela garantira une réponse plus rapide à la requête en raison de la faible communications réseau. 2) Certaines requêtes peuvent également être exécutées en parallèle, c'est-à-dire que la charge de travail de requête peut être distribuée sur plusieurs nœuds, chacun exécutant une partie des opérations totales. Enfin 3) les systèmes de bases de données distribuées effectuent un équilibrage de charge efficace par rapport aux requêtes à exécuter.

Le support du passage à l'échelle est une qualité importante d'un système distribué. On sait qu'il est plus facile d'étendre une architecture distribuée en ajoutant de nouveaux nœuds (c.-à-d., scale out) que d'étendre un serveur donné avec certains disques, CPU ou mémoire (c.-à-d., scale up). Il est également financièrement avantageux de passer à l'échelle avec du matériel de base que de passer à l'échelle avec un serveur initialement coûteux qui va contraindre le genre d'extensions possibles au matériel coûteux propriétaire.

Enfin, l'intégration des données est le point fort des bases de données distribuées. Il est particulièrement facile d'intégrer de nouvelles données dans le cas de bases de données hétérogènes, car il faut enrichir les assertions de correspondances dans l'approche à base de médiateur et écrire des requêtes qui abordent le bon point de d'accès SPARQL dans l'approche fédérée. L'effort est plus important pour les systèmes homogènes si les données à intégrer ne sont pas conformes au modèle de données cible mais, compte tenu des avantages précédemment cités, c'est encore plus

facile que dans la plupart des systèmes centralisés.

3.3.3 Limites des systèmes distribués

Du côté de l'utilisateur final et du DBA, la principale limitation est l'effort nécessaire à la conception de base de données afin de traiter l'aspect distribution et réplication. Autrement dit, décider comment distribuer et répliquer les données sur un cluster de nœuds qui pourraient être géographiquement éloignés nécessite une très bonne compréhension des applications qui invoqueront les bases de données et aussi le jeu de requêtes. Du côté du développeur du DDBMS, les avantages qui viennent d'être présentés nécessitent des efforts majeurs. Cette complexité implique des problèmes de sécurité et de cohérence lorsque la réplication des données est utilisée.

Enfin, les avantages financiers doivent être mis en contraste avec le coût de la gestion de la complexité de la mise en œuvre et de la maintenance, ainsi que le coût lié à l'embauche de développeurs expérimentés et DBA. Cela justifie que certains systèmes en production (e.g., Virtuoso) proposent une version gratuite centralisée et une version commerciale distribuée.

3.3.4 Les systèmes homogènes

Dans un système distribué homogène, toutes les machines du cluster utilisent le même SGBD. En fait, un DDBMS homogène peut être considéré comme une base de données centralisée où un ensemble de machines interconnectées gérerait de manière transparente le stockage des données ainsi que les tâches de traitement. Cette caractéristique rend la conception et la gestion de tels systèmes beaucoup plus faciles par rapport aux systèmes hétérogènes. Cette qualification a les conséquences suivantes : il est moins exigeant de garantir le passage à l'échelle en ajoutant de nouveaux nœuds qui exécutent le même système ; et on peut améliorer les performances globales du système en bénéficiant des capacités de traitement parallèle.

Dans un système homogène, le traitement distribué est généralement géré par le DDBMS. Il faut donc se concentrer sur la distribution des données. Une bonne conception de cette distribution a un impact majeur sur les performances globales du système. Cette conception est influencée par deux facteurs principaux : la fragmentation et la répartition (ou l'allocation). Différentes architectures de système peuvent également être adoptées pour un SGBD homogène. Dans cette section, nous nous concentrons sur les architectures P2P et client-serveur (lorsque plusieurs clients et serveurs peuvent être utilisés), qui sont particulièrement populaires dans les systèmes RDF distribués.

3.3.4.1 Pair à pair (P2P)

Dans les systèmes P2P, il n'y a pas de contrôle centralisé ou d'organisation hiérarchique. Par conséquent, les systèmes fonctionnant à chaque nœud sont tous équivalents. Par rapport à un système client-serveur, nous pouvons considérer que chaque nœud P2P agit à la fois comme un client et un serveur car il est capable d'envoyer une requête à un autre nœud et de répondre à une requête. En général, un réseau P2P implémente une sorte de réseau de superposition virtuel sur un réseau

physique. Via une connexion de superposition logique, les pairs sont capables de communiquer entre eux, et les échanges de données réelles sont effectuées sur le réseau physique sous-jacent.

Il existe plusieurs formes de réseaux P2P qui sont différenciés selon le type possible du lien entre pairs et la méthode utilisée pour indexer et localiser les ressources. Dans les réseaux P2P non structurés, tels que Napster, Gnutella et Gossip [Lui and Kwok (2002)], aucune structure particulière n'est imposée au réseau de superposition. Cela nous permet de construire et de maintenir facilement de tels réseaux, mais il est également livré avec de mauvaises performances liées au traitement de requêtes car les requêtes doivent inonder le réseau pour rechercher une ressource donnée. Les réseaux P2P structurés, tels que Chord [Binzenhöfer et al. (2005)] et Pastry [Hoßfeld et al. (2006)], imposent une topologie spécifique au réseau de superposition et garantissent une recherche efficace des ressources. L'approche la plus courante des réseaux P2P structurés consiste à implémenter une table de hachage distribuée (DHT) pour identifier et localiser les nœuds et les ressources. Cette approche prend en charge le traitement efficace de requêtes, mais se fait au détriment de la maintenance du DHT, la découverte et l'annonce des ressources, et aussi de l'équilibre de charge efficace.

Compte tenu de cette définition, nous pouvons affirmer que dans un système de gestion de données RDF P2P, les triplets RDF et les indexes sont distribués et maintenus sur un réseau P2P. Une coopération entre les pairs est nécessaire pour construire un index distribué. L'efficacité de cette coopération a un impact majeur sur la performance globale du système. Le système vise également à équilibrer la charge à la fois pour le stockage des données et pour les tâches de traitement des requêtes.

De nombreux systèmes RDF utilisent l'architecture P2P :

— **Edutella** [Nejdl et al. (2002)] est basé sur Gnutella. Le système ne permet pas d'utiliser un index centralisé et ne possède pas de méthode pour localiser les triplets RDF sur le réseau. Par conséquent, sa seule solution pour évaluer les requêtes consiste à envoyer une requête donnée (exprimée en RDF-QEL dans le cas de ce système) à tous les nœuds. Chaque nœud traite localement la requête, éventuellement sans résultat, et le jeu de réponses final est calculé à partir du résultat de chaque nœud. Il a été démontré que cette inondation de requête importante empêche Edutella de passer à l'échelle en termes de nombre de nœuds.

— Dans[**Nejdl et al. (2003)**], les auteurs ont proposé un successeur à Edutella qui surmonte certains problèmes de passage à l'échelle. De meilleures performances sont obtenues en introduisant le stockage de métadonnées sur le contenu présent à chaque nœud. Cette approche est désignée comme un réseau P2P basé sur un schéma, car les métadonnées sont utilisées à des fins de routage. Pour éviter l'envoi de requêtes à des pairs inutiles et une consommation excessive du réseau, une topologie de super pairs pour les réseaux basés sur des schémas est proposée. Dans ce cas, chaque pair se connecte à un seul super Pair, et ils sont tous interdépendants pour créer un réseau de super pairs. Ainsi, ces super-pairs déterminent les pairs et les super-pairs qui recevront des requêtes. Deux limites sont constatées : la nécessité d'une définition initiale des schémas et l'identification des super pairs.

— **RDFPeers** [Cai and Frank (2004)] est le premier système RDF distribué qui

utilise un réseau P2P structuré. Il fournit le stockage, l'indexation et l'interrogation à L'aide de RDQL puisque SPARQL n'était considéré qu'un brouillon au niveau de la W3C. Ce système stocke chaque triplet à trois endroits différents dans un réseau adressable multi-attributs (MAAN) en utilisant plusieurs fonctions de hachage globalement connues pour récupérer respectivement les entrées du sujet, du prédicat et de l'objet des triplets RDF. RDFPeers a plusieurs limites : l'équilibrage de charge n'est pas garanti car les nœuds stockant des entrées populaires seront rapidement dépassés ; chaque triplet est répliqué trois fois en raison de l'approche d'indexation à trois voies ; et les requêtes avec une faible sélectivité peuvent subir des performances non-optimales.

— **RDFCube** [Matono et al. (2006)] est un référentiel RDF distribué qui permet de rechercher efficacement les triplets RDF. Chaque triplet dans RDFCube est stocké en spécifiant son sujet, prédicat ou objet comme clé. Le schéma de stockage de RDFCube se compose d'un ensemble de cubes de même taille appelés cellules. Chacune de ces cellules contient un bit appelé indicateur d'existence, qui indique la présence ou l'absence de triplets correspondant à la cellule. Pendant le traitement d'une requête, en vérifiant les indicateurs d'existence des cellules dans lesquelles des triplets de réponse candidats sont liés, il est possible de connaître l'existence des triplets avant d'accéder réellement aux nœuds distants où les triplets de réponse candidats sont stockés. Ces informations permettent de réduire la quantité de données transférées entre les nœuds lors du traitement d'une requête de jointure, car il est possible de réduire les triplets candidats en utilisant l'opérateur "ET Logique" entre les bits d'indicateurs d'existence et de transférer uniquement les triplets candidats présents. Cependant, l'utilisation d'une table de hachage distribuée (DHT) pour l'indexation peut causer certains problèmes, tels que la fraîcheur des données et la sécurité. L'insertion ou la suppression dynamique de triplets RDF n'est pas prise en charge à l'heure actuelle.

— **MIDAS-RDF** [Tsatsanifos et al. (2011)] est un système RDF P2P récent qui essaye de combler certains inconvénients de Edutella et RDFPeers et prend en charge certaines fonctionnalités nouvelles. Par exemple, il ne stocke pas les doublons de triplets RDF et fournit une solution de routage efficace. Grâce à une structure d'index multidimensionnelle distribuée, il est capable de prendre en charge une recherche de plage efficace sur le réseau de superposition. Il met également en œuvre des services de raisonnement pour RDFS.

3.3.5 Les systèmes hétérogènes

3.3.5.1 Systèmes fédérés

Certaines requêtes ne peuvent être répondues qu'en récupérant l'intégralité des résultats stockés dans différentes sources de données. Cela nécessite une nouvelle approche fédérée où l'évaluateur de requêtes nous permet d'accéder aux données stockées sur des SGBD hétérogènes. Dans les systèmes RDF, différents types de traitements fédérés des requêtes peuvent être définis selon le mécanisme d'accès aux sources.

Un système de requêtes fédéré fait référence à tout système fournissant une interface unique pour accéder à plusieurs sources de données. SPARQL est considéré

comme un langage de requête multidatabase qui nécessite des sources de données explicitement spécifiées. Au niveau de la requête, des techniques permettant d'obtenir un calcul de requête efficace dans un cadre distribué doivent être développées. Dans SPARQL 1.1, la fusion des requêtes peut être effectuée via les mots-clés SERVICE et VALUES. Le premier mot-clé, i.e., SERVICE, permet d'appeler un point de d'accès SPARQL spécifique pour une sous-requête, tant dis que le mot-clé VALUES permet de transférer le résultat d'une sous-requête vers une requête externe. Même si la recommandation SPARQL ne spécifie pas comment, le traitement fédéré de requêtes est nécessaire pour l'intégration virtuelle des différents jeux de données RDF.

Un système fédéré est basé généralement sur les composants suivants : un langage de requête déclarative, un catalogue de données, un optimiseur de requête et un protocole de transfert de données. Le langage de requête déclarative (SPARQL), permet de formuler des requêtes complexes de manière concise en fournissant un moyen flexible d'exprimer des contraintes sur les entités de données et les relations entre elles. Le Catalogue de données vise à faire correspondre les requêtes à des sources de données. L'optimiseur de requête est nécessaire pour minimiser les coûts de traitement et de communication lors de la récupération de données provenant de différentes sources. Enfin, le protocole de transfert de données définit comment les informations (requêtes et résultats) sont échangées entre les sources.

Plusieurs approches fédérées ont été proposé dans la littérature, les plus connues sont :

— **Splendid** [Görlitz and Staab (2011)] aborde les points d'accès SPARQL avec des requêtes de type ASK. Il s'appuie sur un index, ce qui peut être considéré comme une approche hybride, pour fournir une sélection de sources optimales et une efficacité de routage des requêtes. La sélection des sources et l'optimisation des requêtes sont achevées automatiquement et reposent sur des informations statistiques fournies par les descriptions VoID (vocabulaire des ensembles de données interconnectés). VoID prend la forme d'un schéma RDF qui est en charge de la description des métadonnées sur les ensembles de données RDF. Il cible des applications visant la découverte, l'archivage et le catalogage d'ensembles de données RDF, et peut ainsi aider les utilisateurs finaux désireux de concevoir de telles applications.

— **FedX :** est un framework développé par FluidOps pour un accès transparent aux sources fédérées. Il est disponible en tant que logiciel open-source. FedX [Schwarte et al. (2011)] étend l'API Sesame SAIL (la couche de stockage et d'inférence de Sesame) avec une couche de traitement fédéré de requêtes, ce qui permet un traitement efficace des requêtes sur les sources distribuées. FedX adopte une approche sans index. Il analyse en effet la requête globale et la divise en sous-requêtes locales auxquelles peuvent répondre des sources de données individuelles. Cette requête globale est optimisée en tenant compte les différents aspects liés à la distribution de données. Plus précisément, il propose des sous-requêtes locales auxquelles les points d'accès SPARQL identifiés répondront. Les résultats obtenus sont ensuite fusionnés par le système FedX et renvoyés sous forme agrégée.

— **DARQ :** est un moteur de traitement fédéré de requêtes complet qui est assisté par index et maintient son propre ensemble de statistiques. Il ne repose sur aucun mécanisme en ligne pour sélectionner les sources. Les tâches de sélection

de source, de décomposition de requête et d'optimisation sont prises en charge par des descriptions de services. Ils sont définis à l'aide du formalisme RDF et vise à déclarer les données disponibles aux points d'accès sous la forme de capacités. Ces capacités spécifient quel type de motifs de triplets peut être répondu avec une source donnée. Ils ne sont définis qu'en termes de prédicats et de contraintes sur les sujets et les objets. Les contraintes correspondent aux expressions de filtre SPARQL et aident à sélectionner les sources avec plus de précision. Les descriptions de services permettent également de définir les droits d'accès aux données.

3.3.6 Traitement massivement parallèle pour RDF

Un système de traitement massivement parallèle (MPP) s'appuie sur un nombre important de nœuds de traitement homogènes interconnectés via un réseau à grande vitesse. L'architecture la plus utilisé pour construire une telle plateforme s'appelle "Shared nothing". En effet, les nœuds de traitement dans ce type de plateforme sont indépendants et généralement ne partagent pas de ressources matérielles (i.e., mémoire, processeur et disque). Chaque nœud peut exécuter sa propre instance du système d'exploitation. Les plateformes MPP fournissent aussi des applications de contrôleur systémiques hébergées sur des nœuds dits maîtres et qui gèrent la répartition des tachés assignées aux nœuds de traitement (appelés aussi esclaves).

Les machines d'une plateforme MPP peuvent également être connectées directement à leurs propres périphériques d'E/S, ce qui favorise l'échange de données dans l'ensemble du système via des interconnexions à haut débit. La Communication entre les nœuds est susceptible de se produire de manière coordonnée, lorsque tous les nœuds cessent leurs traitements et participent à un échange de données à travers le réseau, ou de manière non coordonnée via des messages ciblés pour des nœuds spécifiques.

Une machine qui opère dans une plateforme MPP est parfaitement adaptée aux traitements parallèles de données. Tous les nœuds exécutent d'ailleurs le même programme sur différents flux de données.

Plusieurs systèmes récents adoptent le framework MapReduce comme cadre applicatif MPP. Le maître correspond à ce qu'on appelle le traqueur de travail, qui s'exécute sur une seule machine. Les nœuds de traitement sont des traqueurs de tâches qui s'exécutent sur les nœuds restants du cluster. Lorsqu'un travail est soumis au framework MapReduce, le maître le divise en plusieurs tâches qui sont exécutées par les clients.

Initialement, les plateformes MPP ont été développées et popularisées par les géants du Web (Google, Facebook, Yahoo, ...)., mais aujourd'hui, même les petites entreprises en profitent et ce grâce aux cloud publiques. La variété de domaines d'utilisation ne cesse d'augmenter, ce qui a permis de construire une nouvelle économie basée sur le traitement massive de données. Les platefomes MPP ont aussi contribué à l'évolution des systèmes de gestion de données RDF en offrant la possibilité de traiter des volumes importants de données. Les systèmes les plus connus qui ont adopté une approache MPP sont : SHARD, HadoopRDF, gStore-D et CliqueSquare.

— **SHARD :** [Rohloff and Schantz (2011)] est un système de gestion de données RDF basé sur le cadre applicatif MPP Hadoop. Il s'appuie sur ce cadre pour

les aspects liés à la persistance de données et de traitement des requêtes. Par rapport au stockage de données, les triplets RDF sont regroupés dans des fichiers distribués dits plats qui sont gérés par le système de fichiers HDFS.

Chaque ligne de ces fichiers plats est organisée en une seule paire clé-valeur, où la clé est un sujet du triplet RDF et la valeur est un ensemble de paires (prédicat, objet) associées à ce sujet.

Cette approche de stockage de données est similaire à celle proposée par HBase. Cette organisation de fichiers plats est adaptée aux traitements MapReduce, qui attend des paires clé-valeur comme entrées. Néanmoins, il est inefficace en termes de redondance des données, et il y a un index unique sur le sujet. Ce deuxième aspect est censé être pris en charge par le traitement de requête parallèle, qui utilise le framework Hadoop. Intuitivement, chaque motif de triplets d'une requête SPARQL est géré par une itération d'un job MapReduce. Les correspondances de variables obtenues dans une itération sont assignées aux motifs de triplets SPARQL qui n'ont pas encore été exécutés. Un job MapReduce final est effectué pour filtrer les variables distinctes (celles présentes dans la clause SELECT) et pour supprimer les doublons.

— **HadoopRDF** [Huang et al. (2011)] combine l'utilisation du cadre applicatif distribué Hadoop et le système RDF centralisé RDF-3X [Neumann and Weikum (2008)], pour interroger les bases de données RDF à large échelle. Il propose un module de partitionnement qui permet de créer des sous-graphes disjoints. Un algorithme de partitionnement à base de nœuds du graphe RDF est utilisé afin d'assurer cette tâche. Cela permet de garder les triplets qui partagent les mêmes sujets dans la même machine. HadoopRDF décompose automatiquement la requête en sous-requêtes qui peuvent être évaluées indépendamment avec une communication nulle entre les partitions et utilise le framework Hadoop pour combiner les résultats partiels obtenus. Les triplets voisins liés à un sujet sont dupliqués afin d'éviter les jointures distribuées, ce qui permet de réduire le coût lié à la communication réseau lors de l'évaluation des requêtes. Le module de réplication de données décide quels triplets sont à la limite de chaque partition et les réplique en fonction des garanties n-hop spécifiées. HadoopRDF s'appuie également sur HadoopDB [Abouzeid et al. (2009)] pour gérer le partitionnement de l'évaluation des requêtes entre les systèmes de gestion de données RDF (i.e., RDF-3X) et le framework Hadoop.

— **CliqueSquare** [Djahandideh et al. (2015)] est une autre plateforme de gestion de données RDF basée sur Hadoop à la fois pour le stockage et le traitement des requêtes. Avec l'objectif principale de limiter le nombre de tâches MapReduce et le transfert de données entre les nœuds pendant l'évaluation des requêtes, CliqueSquare exploite le mécanisme de réplication de données intégré dans le système de fichiers HDFS (Hadoop Distributed File System). Chaque partition a trois copies par défaut, de qui permet de partitionner l'ensemble de données RDF de différentes manières. En particulier, pour la première copie, CliqueSquare partitionne les triplets en fonction des valeurs de leurs sujets, de propriétés et objets. Pour la seconde copie, CliqueSquare stocke toutes les partitions sujet, propriété et objet de la même valeur dans le même nœud. Enfin, pour la troisième copie, CliqueSquare regroupe toutes les partitions de sujet dans un nœud par rapport à la valeur de la propriété dans leurs triplets.

De même, il regroupe toutes les partitions d'objets en fonction de leurs valeurs de propriété. En outre, CliqueSquare implémente un traitement spécial pour les triplets dont la propriété est "rdf :type", en les traduisant dans une grande partition de propriété. CliqueSquare divise ensuite la partition de propriété de rdf :type en plusieurs partitions plus petites en fonction de leur valeur d'objet. Pour le traitement des requêtes SPARQL, CliqueSquare s'appuie sur un algorithme à base de cliques, qui produit des plans de requête qui minimisent le nombre d'étapes MapReduce. L'algorithme est basé sur le graphe des variables d'une requête et sa décomposition en sous-graphes à base de cliques. L'algorithme fonctionne de manière itérative pour identifier les cliques et les réduire en évaluant les jointures sur les variables communes de chaque clique. Le processus se termine lorsque le graphe des variables est constitué d'un seul nœud. Puisque les triplets liés à une ressource particulière sont colocalisés sur un nœud, CliqueSquare peut effectuer toutes les jointures de premier niveau dans les requêtes RDF (SS, SP, SO, PP, PS, PO, etc.) localement sur chaque nœud et de réduire le transfert de données à travers le réseau. En particulier, il permet de traiter les requêtes composées de motifs de graphe à 1 saut avec un seul travail MapReduce, ce qui permet un avantage concurrentiel significatif en termes de performances.

— **gStore-D** [Peng et al. (2016)] est un système distribué de gestion de données RDF qui dépend d'une version centralisé révisée de gStore[Zou et al. (2011)]. Ce système s'appuie sur Metis pour partitionner les données en se basant sur les propriétés graphes de RDF. Il offre d'ailleurs un cadre d'évaluation et d'assemblage partiels. En effet, l'évaluation d'une requête peut impliquer l'accès à plusieurs fragments (c'est-à-dire que les correspondances de sous-graphes sont évaluées localement). Il ne décompose pas la requête qui est envoyée en l'état à tous les nœuds de traitement. gStoreD démarre l'évaluation de la requête en calculant les correspondances locales partielles au niveau de chaque nœud de traitement. Ensuite, gStoreD assemble les correspondances partielles pour générer les résultats de la partition croisée (ceux-ci sont appelés des correspondances de croisement). gStoreD permet deux modes d'assemblage : centralisé et distribué. Le mode d'assemblage centralisé envoie les résultats partiels à un site centralisé, tandis que le mode distribué les assemble en plusieurs sites en parallèle.

— **SparkRDF** [Chen et al. (2015)] est un moteur RDF basé sur Spark qui partitionne le graphe RDF en MESGs (Multi-layer Elastic SubGraphs) selon les prédicats (P) et les classes (C) en construisant cinq types d'indices (C, P, CP, PC, CPC) avec différentes granularités pour supporter une évaluation efficace pour les différents motifs de triplets de requêtes. SparkRDF crée un fichier d'index pour chaque structure d'index et stocke ces fichiers directement dans le HDFS, qui sont la seule représentation des triplets utilisés pour l'exécution de la requête. Ces indices sont modélisés sous forme de sous-graphes discrétisés résilients (Rdsg), une collection de sous-graphes en mémoire partitionnés sur des nœuds. Les requêtes SPARQL sont évaluées sur ces indices à l'aide d'une série d'opérateurs de base (par exemple, filter, join). Tous les résultats intermédiaires sont représentés en tant que des Rdsgs et maintenus dans la mémoire distribuée pour prendre en charge des opérations de jointure plus ra-

pides. SparkRDF utilise un algorithme gourmand basé sur la sélectivité pour construire un plan de requête avec un ordre d'exécution optimal basé sur les motifs de triplets de requêtes qui vise à réduire efficacement la taille des résultats intermédiaires. Il utilise aussi une stratégie de pré-partitionnement sans emplacement qui évite le coût de brassage.

— **S2RDF** [Schätzle et al. (2016)] (SPARQL on Spark for RDF) a introduit un schéma de partitionnement relationnel, appelé ExtVP (Extended Vertical Partitioning), pour encoder les données RDF et qui étend le schéma de partitionnement vertical (VP) introduit par Abadi et al. [Abadi et al. (2007)]. S2RDF utilise un prétraitement basé sur la semi-jointure pour minimiser la taille des données en entrée de la requête et ce en prenant en compte les corrélations de jointure possibles entre les tables de codage sous-jacentes des données RDF. En particulier, ExtVP précalcule les relations de jointure possibles entre les partitions (c'est-à-dire les tables) de VP. L'objectif principal De ExtVP est de réduire les E/S inutiles, les comparaisons et la consommation de mémoire pendant l'exécution des opérations de jointure tout en évitant les tuples dans les tables en entrée, c'est-à-dire les tuples qui ne peuvent pas être complétés par la jointure. ExtVP n'utilise pas de précalcules exhaustifs pour toutes les opérations de jointure possibles. Au lieu de cela, un seuil de sélectivité facultatif pour ExtVP peut être spécifié pour matérialiser uniquement les tables où la réduction des tables VP d'origine est suffisamment grande. Ce mécanisme permet de contrôler et de réduire les coûts liés à la taille tout en préservant la plupart de ses avantages coté performance. Par conséquent, lors de l'exécution de la requête, S2RDF peut utiliser les tables de semi-jointures précalculées, si elles existent, ou utilise alternativement les tables de codage de base. S2RDF est construit sur Spark, un système à usage général de traitement de données en mémoire distribué, et exécute des requêtes SPARQL en les traduisant en requêtes SQL qui sont évaluées à l'aide de SparkSQL, un processeur de requêtes SQL basé sur Spark, sur l'encodage ExtVP. S2RDF utilise le format de stockage en colonnes pour stocker les données RDF et s'appuie sur le système de fichiers distribué Hadoop (HDFS).

— **S2X**[Schätzle et al. (2015)] (SPARQL to GraphX) exploite la structure graphe héritée de RDF pour traiter les requêtes SPARQL en utilisant des opérations à base de graphes au-dessus de GraphX. Il utilise le modèle parallèle centré sur les sommets pour trouver les appariements BGP de SPARQL tandis que les autres opérateurs, tels que OPTIONAL et FILTER, sont traités par les opérateurs sur des RDD Spark. L'appariement BGP commence par distribuer tous les motifs de triplets à tous les sommets du graphe. Chaque sommet fait correspondre ses étiquettes d'arcs au prédicat du triplet.

  Les sommets de graphe valident en coopération leur résultats potentiels avec leurs voisins directs en échangeant des messages. Ensuite, les résultats partiels sont collectés et fusionnés progressivement. S2X utilise deux types de codage de chaînes de caractères : hash et count. L'encodage basé sur le hachage utilise une fonction de hachage à base de 64 bits pour encoder les sujets et les objets, tandis que l'encodage basé sur le comptage leur attribue des valeurs numériques uniques. S2X n'a pas de "partitionneur" RDF spécial, il utilise le hachage GraphX 2D, qui hache sur le sujet codé puis sur l'objet codé, pour

partitionner le graphe en entrée entre les nœuds de traitement.

— **D-SPARQ :** Le système D-SPARQ [Mutharaju et al. (2013)] utilise Hadoop pour distribuer et traiter les triplets sur un cluster de machines. Certains triplets à la limite des fragments sont répliqués. L'approche de fragmentation est basée sur les valeurs des sujets, ce qui permet à tous les triplets liés à un sujet donnée d'être stockés dans le même fragment et par conséquence sur la même machine. Cela permet un traitement très efficace des requêtes étoiles où le sujet représente le nœud central. Pour prendre en charge efficacement les autres modèles de requête, des indexes composés sont créés sur les paires sujet-prédicat et prédicat-objet. Comme tous les autres systèmes, l'optimisation des requêtes se concentre sur l'ordre de jointure des motifs de triplets. Intuitivement, après avoir analysé la requête SPARQL, S2X essaie d'identifier des motifs de triplets, e.g., étoiles, chaines. Compte tenu des motifs identifiés, D-SPARQ essaie de paralléliser la récupération des données à partir des différents nœuds en utilisant des stratégies spécifiques pour chaque motif, un ordre de jointure y est établi afin d'optimiser les performances.

— **YARS2 :** [Harth et al. (2007)] est basé sur le système Yars. Il fournit des méthodes d'indexation distribuées et des méthodes d'évaluation de requête parallèles. Basé sur le fait qu'il stocke les quadruplets, le système propose trois types d'indexes : 1) Un index de mots-clés qui utilise Apache Lucene comme un index inversé pour faire correspondre des termes se produisant dans les objets RDF aux sujets. 2) Six index de quadruplets (motivés par l'utilisation de recherches à base de préfixes) basés sur des index éparpillés (les index sont en mémoire et six fichiers triés et compressé par l'algorithme de Huffman sont stockés sur disque). 3) Des indexes de jointure pour accélérer les requêtes contenant des combinaisons de valeurs ou de chemins dans le graphe. La méthode de partitionnement est basée sur le hachage. L'évaluateur de requêtes distribuées utilise des méthodes de recherche effectuées en parallèle.

3.3.7 Fragmentation des données

Intuitivement, la fragmentation des données concerne les méthodes utilisées pour séparer les données. Dans le modèle relationnel, des fragmentations horizontales et verticales peuvent être utilisées. Dans le premier, la division est effectuée au niveau des tuples (c'est-à-dire qu'un ensemble de n tuples est stocké sur chaque machine), tandis que dans le second, la division est effectuée au niveau des colonnes (c'est-à-dire qu'un ensemble de m colonnes est stocké sur chaque machine). Certains systèmes permettent également une approche de fragmentation hybride qui mélange à la fois les fragmentations horizontales et verticales c-à-d que pour un sous-ensemble de tuples, seul un sous-ensemble des colonnes est stocké sur un nœud du cluster. Bien sûr, pour définir une stratégie de fragmentation efficace, il faut posséder une compréhension précise du domaine de la base de données et de l'application (et de l'ensemble de requêtes) qui est envisagée. De toute évidence, lorsque plusieurs applications accèdent à la base de données, la fragmentation devient une tâche complexe. Nous pouvons considérer que plus la fragmentation est fine, plus la compréhension de ces paramètres est précise. Évidemment, la manière dont on décide de regrouper les tuples ou les colonnes a un fort impact sur la performance globale du système.

Il est généralement supposé qu'une solution de fragmentation est sans perte c-à-d qu'aucune donnée ne devrait être perdue pendant le processus de décomposition et qu'une reconstruction des données d'origine devrait être autorisée.

Dans le contexte du modèle de données RDF sans schéma, nous pouvons facilement démontrer que la fragmentation verticale, et conjointement la fragmentation hybride, n'est pas une option efficace. La forme la plus intuitive de fragmentation verticale stockerait les sujets, les prédicats et les objets d'un ensemble de triplets à des nœuds distincts. Nous pouvons également penser à différentes permutations, telles que des sujets et des prédicats sur une machine et des objets sur une autre. De même, une fragmentation hybride impliquerait de stocker des sous-ensembles indépendants des sujets (respectivement des prédicats et des objets) sur des nœuds distincts. De telles fragmentations sont impraticables car elles nécessiteraient des jointures dans un motif de triplets ainsi que des jointures entre des motifs de triplets. Toutes ces jointures nécessiteraient une communication entre les nœuds du cluster, ce que nous avons déjà souligné est un goulot d'étranglement majeur des performances des systèmes distribués.

3.3.8 Allocation des fragments

Le partitionnement des données est responsable du placement des fragments à différents nœuds du cluster. Évidemment, on peut s'attendre à un partitionnement optimal en termes de performances du système. Les données partitionnées peuvent être répliquées à différents nœuds pour améliorer la fiabilité et les performances des requêtes en lecture seule, mais elles se feront au prix d'écritures de requête lentes ou de compromis en termes de cohérence.

Dans le contexte d'un système RDF distribué, le partitionnement des données peut être effectuée en utilisant l'une des méthodes suivantes :

— **Partitionnement aléatoire :** Chaque fragment est stocké aléatoirement sur l'une des machines de cluster. En termes de traitement des requêtes, cette approche impose d'envoyer des requêtes à toutes les machines et implique un coût de communication élevé pour calculer le résultat final des requêtes. Évidemment, c'est loin d'être idéal car le traitement des requêtes est très inefficace en termes de charge de travail de chaque nœud et de communication réseau.

— **Partitionnement basée sur le hachage :** Une fonction de hachage identifie la machine sur laquelle le fragment doit être stocké. La fonction de hachage fonctionne sur une clé de partage, qui est dans de nombreux cas l'objet du triplet RDF ; cela garantit que tous les triplets avec un sujet donné sont stockés sur la même machine. Cette approche permet la recherche directe vers les machines où les triplets ou quadruplets sont stockés avec une fonction de hachage connue. Il est reconnu comme efficace pour les requêtes en étoiles mais pas pour les requêtes complexes, car trop de communications réseau sont nécessaires. Cette approche est utilisée dans des systèmes tels que YARS2, SHARD et Virtuoso.

— **Partitionnement basée sur le rang :** Ce type d'allocation stocke les données en entrée dont les clés de partage appartiennent à un rang donné sur une machine particulière. Par exemple, la clé de partage pourrait être les sujets des triplets et on peut décider de stocker tous les URI commençant par les

lettres de A à M sur la machine M1 et ceux de N à Z sur la machine M2. Cette approche prend en charge les recherches directes via une structure de données globale, mais généralement elle est associée à un biais de données car on peut se retrouver avec beaucoup plus de triplets sur une machine que sur une autre.

— **Partitionnement des graphes :** Cette méthode utilise une fonction graphe pour diviser le graphe en sous-graphes et les stocker sur différentes machines. L'idée principale est de garder les triplets qui sont proches les uns des autres sur la même machine. Cela devrait limiter, au moins pour certaines classes de requête, la communication réseau entre les nœuds au moment de l'évaluation de requêtes. Le partitionnement graphe est connu pour être un problème difficile qui nécessite généralement des algorithmes à base d'heuristiques et d'approximation.

— **Partitionnement basée sur le jeu de requêtes :** Ce type de répartition considère qu'on a un ensemble de requêtes typiques sur la base de données distribuée à modéliser. Étant donné un ensemble de requêtes, une répartition optimale peut être défini en conservant les triplets qui vont être consulté aux mêmes nœuds pour s'assurer que les jointures sont exécutées localement. Cette méthode soulève plusieurs problèmes, tels que la possibilité de mettre à jour efficacement la stratégie de répartition lorsque de nouvelles requêtes apparaissent. De plus, on peut se demander si les systèmes RDF sont vraiment le genre de SGBD pour lequel la charge de travail de requête est précisément identifiée au préalable.

3.4 Évaluation des systèmes de gestion des LOD

Les systèmes de gestion de données RDF sont nombreux, et il est nécessaire d'évaluer ces systèmes avec des données et de requêtes avec diverses caractéristiques. Un certain nombre de benchmarks RDF ont été développés pour évaluer les performances des systèmes RDF. Les benchmarks RDF se concentrent principalement sur les performances du système RDF en termes de temps de chargement des données, d'évolutivité du système (c-à-d. le nombre de triplets pris en charge), de temps de réponse aux requêtes et de capacités de raisonnement. Les benchmarks RDF sont classés en deux catégories : ceux qui se basent sur des données synthétiques et ceux qui utilisent des données réelles. Dans ce qui suit nous présentons une brève analyse de certains benchmarks populaires.

— **Le Lehigh University Benchmark (LUBM) :** [Guo et al. (2005)] est un benchmark créé pour faciliter l'évaluation des systemes de gestion de données RDF de manière standard et systématique. Le benchmark est basé sur une ontologie liée au domaine universitaire et peut générer des données synthétiques de taille arbitraire et fournit quatorze requêtes qui représentent une variété de propriétés de graphe RDF et plusieurs mesures de performance.

Le benchmark LUBM, selon ses créateurs, a été conçu avec les objectifs suivants :

— Prise en charge des requêtes extensives plutôt d'intensives : les créateurs du benchmark conjecturent que la majorité des applications web sémantiques utiliseront des requêtes sur les instances de données, des requêtes

extensives plutôt que des requêtes sur les classes et les propriétés, des requêtes intensives. Le benchmark se concentre donc sur les requêtes d'extension.

- Prise en charge de l'évolutivité des données : les créateurs de LUBM prédisent qu'à long terme, les données seront plus nombreuses que les ontologies et que le benchmark doit donc être évolutif.
- Prise en charge d'une taille modérée et d'une ontologie complexe : les benchmarks tentent de trouver un équilibre entre les requêtes qui manipulent des ontologies grandes et complexes et les requêtes qui manipulent des ontologies basées sur OWL Lite.

LUBM fournit quatorze requêtes de référence qui varient dans leur taille, la sélectivité, la complexité, la classe et la hiérarchie des propriétés, ainsi que le degré d'inférence logique qui est nécessaire pour répondre à la requête.

LUBM mesure la taille de données en entrée comme la proportion des instances de classe impliquées dans une requête par rapport au nombre total d'instances de classe dans les données de référence. La taille de données est considérée comme grande si la proportion est supérieure à 5%. La sélectivité, en revanche, mesure le rapport entre les instances de classe de nombre impliquées dans une requête et le nombre d'instances de classe qui satisfont la requête. Une requête est dite très complexe si cette proportion est inférieure à 10%. Les créateurs du benchmark prétendent que la complexité de la requête peut être implicitement mesurée par le nombre de classes et de propriétés impliquées dans la requête ainsi que par la profondeur et la largeur des hiérarchies de classes impliquées dans la requête. Les quatorze requêtes de référence utilisées par LUBM ont différents niveaux de taille, de sélectivité et de complexité.

L'ensemble de données généré par LUBM se compose de quinze à vingt-cinq départements par Université, chacun décrit dans un fichier OWL séparé. Le générateur de données LUBM génère un ensemble de données basé sur le nombre d'universités que les utilisateurs spécifient.

- **Berlin SPARQL Benchmark (BSBM)** était présenté pour la première fois en 2009. Il est conçu à partir d'un scénario de commerce électronique dans lequel un ensemble de produits est fourni par divers fournisseurs et les consommateurs publient des avis sur les produits. Ce benchmark permet l'utilisation de trois représentations de données suivant une sémantique similaire : les triplets RDF, les Modèles de données de graphes nommés et les données relationnelles. BSBM fournit des requêtes SPARQL via un SPARQL Endpoint (Endpoint est un service web permettant l'exécution de requêtes SPARQL) pour évaluer les systèmes RDF. BSBM peut également évaluer les performances à différentes grandeurs de données et aussi la pression d'accès, en modifiant la taille de l'ensemble de données et le nombre de clients simulés. Cependant, les données et les requêtes BSBM sont synthétiques et le schéma de données est très homogène et ressemble à une base de données relationnelle. Ceci est raisonnable pour comparer les performances des systèmes de gestion de données RDF avec le SGBDR, mais ne donne pas beaucoup d'informations sur les spécificités de la gestion des données RDF.
- **YAGO :** est une vaste base de connaissances qui intègre des déclarations

de Wikipedia, Wordnet et GeoNames. Il décrit plus de dix millions d'entités (qui comprennent des personnes, des organisations et d'autres). L'ensemble de données d'origine n'est pas accompagné d'un ensemble de requêtes. Cependant, Neumann and Weikum (2008) a fourni huit requêtes pour une version antérieure de l'ontologie YAGO, dans le cadre de l'évaluation du système RDF-3X. Ces requêtes sont principalement des requêtes de recherche et de jointure qui utilisent les opérateurs SPARQL filter, distinct et count ; les requêtes de jointure sont des requêtes de types chemin ou étoile, la plus complexe contenant jusqu'à dix motifs de triplets.

La dernière version de YAGO (YAGO3), contient 17 millions d'entités et 150 millions de faits sur ces types. Le schéma comprend environ 488 000 types et 77 relations.

— **WatDiv :** est un générateur de données qui permet aux utilisateurs de contrôler les caractéristiques diverses des données générées telles que les entités incluses, la structure de **la collecte**, la manière dont les entités sont associées et la probabilité d'association et la cardinalité pour différents types d'entités. Cela permet d'obtenir une hétérogénéité et une structure différentes **d'une collecte** de données. Le benchmark est livré avec les scénarios d'utilisation suivants : 1) Test de base : 20 requêtes de complexité diverse, 2) Extensions aux tests de base avec deux cas d'utilisation,3) Les tests linéaires incrémentiels testent les performances des requêtes avec un nombre croissant de motifs de triplets, 4) Le test linéaire mixte teste la performance pour les requêtes de différents nombres (pas nécessairement croissants) de motifs triplets. Le nombre de motifs de triplets varie entre 5 et 10, 5) Test de résistance qui offre un test de complet des systèmes.

— **SP2Bench** est un autre benchmark pour les systèmes de gestion de données RDF, introduit en 2009. Ses données RDF sont basées sur le Projet Digital Bibliography & Library (DBLP) et comprennent des informations sur les publications et leurs auteurs. Il utilise le Générateur SP2Bench pour générer ses données de test synthétiques, qui est dans son hétérogénéité de schéma encore plus limitée que LUBM. Le principal avantage de SP2Bench par rapport à LUBM est que ses requêtes de test incluent une variété de fonctionnalités SPARQL (telles que OPTIONAL, UNION, DISTINCT et FILTER). Le benchmark contient onze requêtes SELECT et trois requêtes ASK. La principale différence entre le benchmark **DBpedia et SP2Bench** est que les données de test et les requêtes sont synthétiques dans SP2Bench. De plus, SP2Bench n'a publié que des résultats pour des triplets allant JUSQU'à 25M, ce qui est relativement faible en ce qui concerne les ensembles de données tels que DBpedia et LinkedGeoData.

— Un autre benchmark décrit dans [Owens et al. (2008)] compare les performances de BigOWLIM et AllegroGraph. La taille de son ensemble de données synthétiques sous-jacents est de 235 millions de triplets, ce qui est suffisamment grand. Le benchmark mesure les performances d'une variété de constructions SPARQL pour les deux systèmes lors de l'exécution en mode simple et en mode multithread. Il mesure également les performances de l'ajout de données, à la fois en utilisant l'ajout en bloc et l'ajout partitionné. L'inconvénient de ce benchmark est qu'il compare les performances de seulement deux systèmes de

gestion de données RDF. De plus, les performances de chaque système ne sont pas évaluées pour différentes tailles d'ensembles de données, ce qui empêche les comparaisons d'évolution.

En général, les efforts existants de benchmarking SPARQL tels que LUBM, BSBM et SP2Bench ressemblent à des repères de base de données relationnelles. En particulier, les structures de données sous-jacentes à ces benchmarks sont essentiellement des structures de données relationnelles, avec des classes relativement peu nombreuses et structurées de manière homogène. Cependant, les bases de connaissances RDF sont de plus en plus hétérogènes. Ainsi, ils ne ressemblent pas à des structures relationnelles et ne sont pas facilement représentables en tant que tels. Des exemples de telles bases de connaissances sont des ontologies biomédicales organisées telles que celles contenues dans Bio2RDF ainsi que des bases de connaissances extraites de sources non structurées ou semi-structurées telles que DBpedia ou LinkedGeoData. Par exemple, DBpedia contient des milliers de classes et de propriétés. DBpedia (version 3.6) par exemple contient 289 016 classes dont 275 classes appartiennent à L'ontologie DBpedia. De plus, il contient 42 016 propriétés, dont 1335 sont dans L'ontologie DBpedia. En outre, divers types de données et références d'objets de différents types sont utilisés dans les valeurs de propriété. De telles bases de connaissances ne peuvent pas être facilement représentées selon le modèle de données relationnelles et, par conséquent, les caractéristiques de performance pour le chargement, l'interrogation et la mise à jour de ces bases de connaissances pourraient potentiellement être fondamentalement différentes des bases de connaissances ressemblant à des structures de données relationnelles.

3.5 Évaluation expérimentale des systèmes existants

Pour appuyer notre analyse théorique une étude expérimentale a été mené afin d'analyser la capacité des systèmes existants à supporter le passage à l'échelle. Cette étude a été mené en utilisant quartes benchmarks (constitués de données réels et synthétiques) connus : Watdiv, LUBM, Yago et DBLP. Nous nous sommes limités à des jeux de données avec maximum 1 milliard de triplets. Nous avons choisi trois systèmes, considérés comme l'état de l'art en matière de gestion de données RDF : une approche relationnelle (Virtuoso), une approche à base de graphes (gStore) et une approche d'indexation intensive (RDF-3X). Dans la première série d'expérimentations, nous avons testé la capacité de ses systèmes de supporter le chargement des jeux de données fournis par les benchmarks. Le résultats sont présentés dans Tableau 3.1. Il est clair qu'aucun système n'est en mesure de charger toutes les données. L'évolution du volume du jeu de données pourrait impacter le fonctionnement du système de gestion de données RDF. La deuxième série de tests concernent la possibilité de traiter des requêtes SPARQL avec des fonctionnalités avancées (BGP, agrégations, Tri, filtrage à base d'expressions régulières). Nous avons le même constat : aucun système testé n'est en mesure de garantir le traitement de ses requêtes.

Les expérimentations que nous avons faites ont montré la limite des approches existantes et la nécessité d'une approche qui garantit le passage à l'échelle et la performance d'évaluation des requêtes.

	Chargement de données			Wildcards			Order-By & Group-By		
	gStore	3X	Virt.	gStore	3X	Virt.	gStore	3X	Virt.
Watdiv100M	✓	✓	✓	✓	✘	✓	✓	✘	✓
Watdiv1B	✘	✓	✓	✘	✘	✓	✘	✘	✓
LUBM500M	✘	✓	✓	✘	✘	✓	✘	✘	✓
LUBM1B	✘	✘	✘	✘	✘	✘	✘	✘	✘
Yago	✘	✓	✓	✘	✘	✓	✘	✘	✓
DBLP	✘	✓	✓	✘	✘	✓	✘	✘	✓

3X : RDF-3X, Virt. : Virtuoso

TABLEAU 3.1 – Résultats expérimentaux de comparaison

3.6 Conclusion

Dans ce chapitre, nous avons donné un aperçu de l'état de l'art en matière de gestion de données RDF. Nous avons catégorisé ses approches selon l'approche suivie dans le développement et plus particulièrement le fait qu'ils s'appuient sur un système existant ou non. Nous avons analysé différents aspects liés au stockage de données RDF et l'évaluation de requêtes SPARQL. Cette démarche nous a permis de détecter les limites des solutions existantes, qui sont liées principalement au passage à l'échelle et aux performances. Une étude expérimentale a été également intégrée afin d'analyser la capacité de ces systèmes à prendre en charge des données variées avec une taille importante et aussi les fonctionnalités avancées (tri, agrégation et filtrage à base d'expressions régulières) de SPARQL. Nous avons pu par ailleurs confirmer nos indicateurs que nous avons cités dans l'Introduction Générale autour des limites des approches existantes et le besoin de proposer un nouveau système qui permet de garantir un bon compromis entre passage à l'échelle et performances.

Deuxième partie

L'approche RDF_QDAG

Chapitre 4

L'approche RDF QDAG : bases théoriques, mise en œuvre et optimisation

Sommaire

4.1 Bases théorique de l'évaluation de requêtes 94
4.1.1 Représentation logique de données 94
4.1.2 Représentation logique de requêtes 96
4.1.3 Évaluation de requêtes 99
4.2 Mise en oeuvre et optimisation 103
4.2.1 Stockage de graphes RDF 103
4.2.2 Opérations d'évaluation de requêtes 107
4.2.3 Adaptation du modèle Volcano 113
4.2.4 Traitement des requêtes wildcard 117
4.2.5 Modules complémentaires 118
4.2.6 Exemple d'exécution 119
4.3 Conclusion 125

Dans le chapitre précédent, à l'aide de notre méthode "Tester, Analyser, Proposer", nous avons pu identifier d'une manière expérimentale les limites des systèmes actuels et motivé la nécessité d'en proposer un nouveau pour assurer un compromis entre la performance des requêtes SPARQL et le passage à l'échelle des données RDF. Une analyse fine de ces systèmes et indépendamment de leurs résultats expérimentaux nous a permis de dégager deux types d'optimisations utilisés par ces derniers : (i) une optimisation physique à l'aide des structures accélératrices de traitement comme les index souvent définis sur les composantes d'un triplet (sujet> < prédicat> <objet>), des structures de stockage des triplets (par ex. la liste adjacente), et la fragmentation de graphe RDF. (ii) Une optimisation dirigée par les services offerts par les infrastructures de déploiement tels que le traitement parallèle et distribué.

La vision d'optimisation utilisée par ces systèmes ne suit pas malheureusement celle adoptée par des SGBDs relationnels qui règnent sur le marché des données depuis plus de 40 ans. En plus des optimisations physiques, les SGBDs relationnels

offrent un autre type d'optimisations connues sous le nom des optimisations logiques. Ces dernières ne nécessitent pas une compréhension approfondie des structures d'optimisation physiques. Elles demandent juste une compréhension des structures logiques des données et l'algèbre de requêtes associée à ces données. Dans cette thèse, nous proposons de réutiliser un des facteurs des succès des SGBDs relationnels à savoir la vision logique des données RDF et des requêtes SPARQL. Cette réflexion nous a permis de remettre en cause la représentation traditionnelle des graphes RDF sous forme d'un ensemble de (motif de) triplets en proposant une nouvelle représentation logique qui permet de définir des mécanismes d'optimisation au niveau logique. Deux mécanismes sont proposés dans cette thèse : la fragmentation de données et l'exploration de graphes. Nous proposons aussi plusieurs mécanismes de mises en œuvre afin de garantir les deux propriétés que nous avons fixées précédemment, à savoir le passage à l'échelle en termes de données et les performances.

Ce chapitre est composé de 3 sections : La Section 4.1 donne les bases théoriques de l'évaluation de requêtes en utilisant la fragmentation et l'exploration de graphes. Nous nous basons sur une nouvelle représentation logique des données RDF, appelée étoile de données qui joue le rôle d'une unité de base de données RDF. Nous décrivons aussi la représentation logique de requêtes SPARQL en utilisant une structure similaire, appelée étoile de requêtes. Enfin, dans cette même section, nous détaillons le processus d'évaluation de requêtes SPARQL en utilisant les structures que nous avons identifiées. Dans la section 4.2, nous présentons les détails techniques de mise en œuvre ainsi que les optimisations liées à notre nouvelle approche. Nous présentons notre modèle de stockage et d'évaluation de requêtes. Ensuite, nous détaillons les composants principaux de notre système et nous expliquons en grande partie la stratégie, basée sur le modèle d'exécution Volcano, utilisée pour gérer la mémoire principale lors de l'exécution des requêtes. Finalement, la Section 4.3 conclut ce chapitre.

4.1 Bases théorique de l'évaluation de requêtes

4.1.1 Représentation logique de données

Dans cette section, nous présentons de nouvelles structures logiques permettant de mieux représenter les données. En effet, la représentation des graphes RDF sous forme d'un ensemble de triplets engendre une perte d'informations précieuses que nous pouvons exploiter lors de l'évaluation de requêtes. Traiter un triplet RDF séparément ne permet pas de considérer les triplets voisins, ceux qui partagent le même sujet ou le même objet. Ce niveau de granularité nous prive de plusieurs mécanismes d'élagage.

Dans notre travail, nous nous appuyions sur l'exploration, qui est basée sur les arcs entrants et sortants des nœuds du graphe. L'ensemble des arcs (entrants ou sortants) d'un nœud permet d'identifier d'une manière implicite le type (ou la classe) du nœud. La structure principale que nous utilisons est basée sur l'étoile de données, qui est formellement détaillée dans la définition 4.1.1.

Les étoiles de données permettent d'identifier les triplets liés à une instance (nœud) spécifique en regroupant les données (les triplets) par sujet (ou objet).

Definition 4.1.1. (Étoile de données) Nous appelons étoile de données (ED) l'en-

TABLEAU 4.1 – Exemple : Étoiles de données

Étoiles de données	Triplets
$ED_{1f}(La_loi_et_l'ordre)$	{(La_loi_et_l'ordre,genre,Drama), (La_loi_et_l'ordre,Vedette,Al_Pacino), (La_loi_et_l'ordre,Vedette,Robert De Niro)}
$ED_{2f}(Film_policier_américain)$	{(Al_Pacino,a_gagné,Oscar), (Al_pacino,né_à,New_york)}
$ED_{3f}(Al_Pacino)$	{(Robert De Niro,né_à,New_york)}
$ED_{1b}(Oscar)$	{(Al_Pacino,a_gagné,Oscar)}
$ED_{2b}(New_York)$	{(Al_Pacino,né_à,New_york), (Robert De Niro,,né_à,New_york)}
$ED_{...}(...)$	{(...,...,...),(...,...,...),...}

semble des arcs liées à un nœud spécifique dans le graphe de données. Si les arcs sont sortants, nous appelons cette étoile de données "étoile de données sortante". Nous utilisons le symbole : $ED_f(x)$, où $x \in V_c$, pour désigner l'étoile de données sortante obtenue à partir de x. Dans ce cas, x est appelé la tête de l'étoile de données. Si les arcs sont entrants, nous appelons cette étoile de données "étoile de données entrante". Nous utilisons le symbole : $ED_b(x)$. Pour simplifier, nous utilisons $\overrightarrow{x}$ (respectivement $\overleftarrow{x}$) pour désigner les étoiles sortantes (respectivement entrantes) obtenues à partir du nœud x.

Les étoiles de données sortantes étendent la notion d'enregistrement dans les SGBDR. En effet, dans un modèle relationnel, nous ne pouvons pas associer plus d'une valeur pour un attribut. La clé primaire est utilisée comme tête de l'étoile de données dans le cas de l'existence d'un seul attribut en tant que clé. Dans le cas contraire, nous utilisons l'identifiant de l'enregistrement.

Afin de clarifier cette définition, nous utilisons l'exemple du Tableau 4.1 qui montre quelques étoiles de données extraites à partir du graphe de la Figure 2.3. De cet exemple, nous pouvons facilement apercevoir qu'il y a des étoiles de données qui partagent le même ensemble de prédicats. Cet ensemble s'appelle ensemble des caractéristiques [Pham et al. (2015); Neumann and Moerkotte (2011)] et peut refléter le type implicite des nœuds utilisés comme des têtes. Pour notre exemple, nous avons "Al_Pacino" et "Robert_De_Niro" qui partagent le même ensemble {a-gagné,Né-à}.

Chaque sujet s dans le graphe RDF G a un ensemble de caractéristiques sortants qui est défini comme suit :

Definition 4.1.2. (Ensemble de caractéristiques sortants)
$\overrightarrow{ec}(s) = \{p | \exists_o : (s, p, o) \in G\}$.

De même pour les objets, nous définissons l'ensemble de caractéristiques entrants.

Definition 4.1.3. (Ensemble de caractéristiques entrants)
$\overleftarrow{ec}(o) = \{p | \exists_s : (s, p, o) \in G\}$.

Dans notre travail, nous regroupons les étoiles de données avec le même ensemble de caractéristiques. Cela permet de répondre facilement lorsque nous demandons

toutes les étoiles de données qui ont un ensemble de caractéristiques donné. Nous appelons les groupes créés à l'issue de ce processus "fragments de graphe". Une formalisation de la notion de fragment est donnée dans la définition 4.1.4.

Definition 4.1.4. (fragment de graphe) Un fragment de graphe ($\mathcal{G}f$) est un ensemble d'étoiles de données qui partagent le même ensemble de caractéristiques ec : $\overrightarrow{\mathcal{G}f} = \{ED(x)|ec(x) = ec\}$. ec et $\mathcal{G}f$ sont tous les deux entrants ou sortants.

Certaines approches existantes ont tenté aussi de regrouper les triplets RDF en utilisant des ontologies. Ces dernières permettent de fournir un schéma global pour les données représentées avec un graphe RDF. Cependant, plusieurs études montrent qu'il existe encore une utilisation très partielle des classes d'ontologie et que parfois les sujets partagent des triplets avec des propriétés provenant de différentes sources ontologiques [Pham et al. (2015)]. Contrairement aux approches (e.g., Jena[McBride (2002)] et DB2RDF[Bornea et al. (2013)]) qui utilisent ce type explicite, nous avons décidé d'utiliser un type implicite qui est identifié en fonction des prédicats associés à une entité à l'aide des ensembles de caractéristiques.

Cette organisation correspond à la notion d'ensembles de caractéristiques [Neumann and Moerkotte (2011)], sauf que les ensembles de caractéristiques sont des structures logiques utilisées uniquement pour collecter des statistiques sur des données. Ces ensembles n'ont pas d'impact sur l'organisation des triplets et sur l'évaluation des requêtes. Nos données sont physiquement organisées en fragments. La Figure 4.1 montre les différents fragments créés à partir du graphe de la Figure 2.3.

Pour trouver efficacement les fragments pertinents, nous indexons les étiquettes des fragments [1]) sous la forme d'un treillis [Aït-Kaci et al. (1989)]. L'opérateur de satisfaction du treillis permet de trouver un fragment pertinent par rapport à l'ensemble des prédicats demandés, chaque nœud du treillis est lié à un ensemble de prédicats et une liste des fragments satisfaisant ces prédicats. Les étiquettes des fragments du graphe dans la Figure 2.3 sont organisées dans les treillis montré dans la Figure 4.2. Les données sont en effet organisées en 4 fragments SPO (avec les étiquettes 1 2 4 8, 5 6, 5 7 et 3) et 8 fragments OPS (avec les étiquettes 1, 2, 3, 5, 7, 8, 6 7. Par rapport à OPS, lorsque nous sommes à la recherche de nœuds qui ont un arc entrant avec l'étiquète "à-gagné", seulement deux fragments (associés aux nœuds du treillis ayant les étiquètes 6 7 et 7) sont considérés.

4.1.2 Représentation logique de requêtes

Dans cette section, nous détaillons les structures de base que nous utilisons pour représenter les requêtes SPARQL. Nous nous concentrons seulement sur la représentation des motifs basiques de graphes qui permettent de capturer la nature structurelle des requêtes SPARQL. Nous expliquons dans le chapitre suivant comment traiter d'autres opérateurs (i.e., FILTER, GROUP BY et ORDER BY).

Les motifs basiques de graphes sont naturellement représentables par un graphe. Nous donnons une formalisation de cette présentation dans la définition 2.4.6.

Considérons la requête de la Figure 2.9 avec l'opérateur Order By :

1. L'ensemble des prédicats liés au sujet (dans le cas d'un fragment SPO) ou aux objets (dans le cas d'un fragment OPS

Vedette, genre, Réalisateur, Catégorie		
S	P	O
La loi et l'ordre	Vedette	Bryan Dennehy
La loi et l'ordre	Vedette	Al Pacino
La loi et l'ordre	Vedette	Robert de Niro
La loi et l'ordre	genre	Drama
La loi et l'ordre	Réalisateur	Jon Avnet
La loi et l'ordre	sujet	Film policier américain

a_gagné, né_à		
S	P	O
Robert De Niro	né_à	New York
Robert De Niro	a_gagné	Oscar
Robert De Niro	a_gagné	Golden Globe
Al Pacino	a_gagné	Oscar
Al Pacino	a_gagné	Golden Globe
Al Pacino	né_à	New York

Gamme		
S	P	O
Film policier américain	Gamme	Films de crime

né_à, est_nommé		
S	P	O
Jon Avnet	né_à	New York
Jon Avnet	est_nommé	Golden Globe

(a) Fragmentation de la structure SPO

a_gagné, est_nommé		
O	P	S
Golden Globe	est_nommé	Jon Avnet
Golden Globe	a_gagné	Al Pacino
Golden Globe	a_gagné	Robert De Niro

né_à		
O	P	S
New York	né_à	Jon Avnet
New York	né_à	Al Pacino
New York	né_à	Robert De Niro

a_gagné		
O	P	S
Oscar	a_gagné	Al Pacino
Oscar	a_gagné	Robert De Niro

Vedette		
O	P	S
Bryan Dennehy	Vedette	La loi et l'ordre
Al Pacino	Vedette	La loi et l'ordre
Robert De Niro	Vedette	La loi et l'ordre

genre		
O	P	S
Drama	genre	La loi et l'ordre

Réalisateur		
O	P	S
Jon Avnet	Réalisateur	La loi et l'ordre

Catégorie		
O	P	S
Film policier américain	Catégorie	La loi et l'ordre

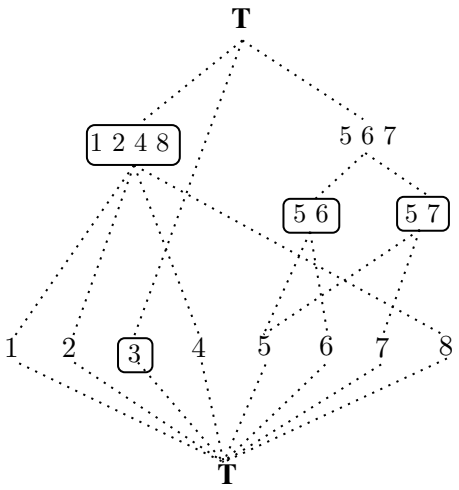
Gamme		
O	P	S
Films de crime	Gamme	Film policier américain

(b) Fragmentation de la structure OPS

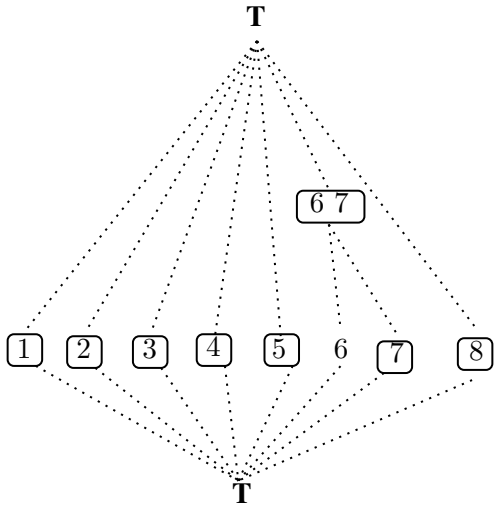
FIGURE 4.1 – Exemple de découpage de données

```
SELECT ?ville, ?acteur1, ?acteur2
WHERE {
?film  genre Drama          #TP1
```

Predicate	ID
Vedette	1
Catégorie	2
Gamme	3
Réalisateur	4
né_à	5
est_nommé	6
a_gagné	7
Genre	8



Figure(a) : Exemple de treillis SPO



Figure(b) : Exemple de treillis OPS

FIGURE 4.2 – Exemple de treillis

```
?film Vedette  ?acteur1     #TP2
?film Vedette ?acteur2      #TP3
?acteur1 a_gagné      Oscar #TP4
?acteur1 né_à  ?ville       #TP5
?acteur2 né_à  ?ville       #TP6
}
ORDER By ?ville
```

Le motif basique de graphes est montré dans la Figure 2.9. Il est constitué de six motifs de triplets TP1, TP2, TP3, TP4, TP5, TP6. Cette requête permet de trouver les paires d'acteurs qui sont nés dans la même ville et aussi ont gagné l'Oscar. Les résultats de cette requête sont montré dans le Tableau 4.3.

Afin de traiter cette requête, nous proposons d'utiliser des structures similaires à celles que nous avons utilisées pour représenter les données. En effet, chaque requête est décomposée en étoiles. Nous détaillons ce concept dans définition 4.1.5.

TABLEAU 4.2 – Exemple : Etoiles de requêtes

ER	Stars
$ER_{1f}(?film)$	{(?film,genre,Drama), (?film,Vedette, ?acteur1), (?film,Vedette, ?acteur2)}
$ER_{2f}(?acteur1)$	{(?acteur1,a_gagné,Oscar), (?acteur1,né_à, ?ville)}
$ER_{3f}(?acteur2)$	{(?acteur2,né_à, ?ville)}
$ER_{1b}(Oscar)$	{(?acteur1,a_gagné,Oscar)}
$ER_{2b}(?ville)$	{(?acteur1,né_à, ?ville),(?acteur2,,né_à, ?ville)}
$ER_{3b}(?acteur1)$	{(?film,Vedette, ?acteur1)}
$ER_{4b}(?acteur2)$	{(?film,Vedette, ?acteur2)}
$ER_{5b}(Drama)$	{(?film,genre,Drama)}

?ville	?acteur1	?acteur2
New York	Al Pacino	Robert De Niro

TABLEAU 4.3 – Résultats de la requêtes

Definition 4.1.5. (Étoile de requête) Nous appliquons le même principe que les étoiles de données (voir la définition 4.1.1) pour distinguer les étoiles dans une requête. Nous utilisons une étoile de requête sortante ($ER_f(x)$) et une étoile de requête entrante ($ER_b(x)$), où $x \in V_c \cup V_p$, pour désigner les étoiles obtenues à partir des triplets sortants et des triplets entrants respectivement.

Pour notre requête exemple, les étoiles de requêtes sont montré dans le Tableau 4.2. La première colonne présente la tête de l'étoile tant dis que la deuxième représente les motifs de triplets associés.

En utilisant ces différentes définitions, nous sommes en mesure maintenant d'identifier les fragments pertinents pour chaque étoile de requêtes.

Proposition 4.1.1. *Soit un ensemble de prédicats dans un nœud du treillis S et un ensemble de prédicats associés à une étoile ER, S satisfait ($\models$) ER si prédicats (ER) $\subseteq$ prédicats(S)*

Afin de répondre à l'étoile de requêtes $ER_{1f}(?film)$, les fragments identifiés par les étiquettes [1 2 4 8] dans la Figure 4.2 (a) doivent être considérés.

4.1.3 Évaluation de requêtes

Dans cette section, nous présentons les principaux concepts et définitions liés à notre approche d'évaluation de requêtes. Nous formalisons la partie logique de notre technique par rapport aux structures décrites précédemment (e.g., fragments, étoiles). Nous proposons d'étendre le mécanisme de d'évaluation proposé par Pérez et al. (2006) et qui est basé sur l'évaluation des motifs de triplets d'une requête. Nous nous nous appuyions sur les structures en étoiles que nous avons définies.

Nous expliquons maintenant comment évaluer une requête en se basant sur les étoiles de requêtes. Dans la définition 4.1.6, nous présentons la notion de correspondance qui permet de d'associer un candidat pour une variable.

Definition 4.1.6. (Correspondance)[Pérez et al. (2006)] Une correspondance est une fonction partielle $\mu : V_p \rightarrow V_c$ d'un sous-ensemble de variables V_p aux nœuds constants V_c. Le domaine de correspondances μ, écrit DOM(μ), est défini comme le sous-ensemble de V_p pour lequel μ est défini. Par M nous dénotons l'univers de toutes les correspondances.

A titre d'exemple et par rapport aux résultats de la requête (Tableau 4.3), (?ville, "New_york") est une correspondance.

Nous définissons ensuite la notion de compatibilité entre les correspondances. De façon informelle, deux correspondances sont compatibles s'ils ne contiennent pas de liaisons de variables contradictoires, *i.e.* si les variables partagées sont liées toujours à la même valeur dans les deux correspondances.

Definition 4.1.7. (Compatibilité des correspondances) Étant donné deux correspondances $\mu 1$, $\mu 2$, nous dirons que $\mu 1$ est compatible avec $\mu 2$ si $\mu 1($?x$) = \mu 2($?x$)$ pour tout ?x $\in$ dom($\mu 1$) $\cap$ dom($\mu 2$). Nous écrivons $\mu 1 \sim \mu 2$ si $\mu 1$ et $\mu 2$ sont compatibles, et $\mu 1 \not\sim \mu 2$ sinon.

Nous notons par $vars$ la fonction qui retourne des variables de l'élément passé comme paramètre. Par exemple, nous notons par $vars(t)$ toutes les variables du motif de triplets t, tandis que $vars(ER_f(x))$ dénote toutes les variables de l'étoile de requêtes $ER_f(x)$. Par rapport à notre requête d'exemple, $vars(ER_{1f}(?film))$ renvoie ?film et ?acteur tandis que vars(TP1) renvoie ?film.

Nous écrivons μ(t) pour indiquer le motif de triplets obtenu lors du remplacement de toutes les variables ?x $\in vars(t)$ dans t par $\mu($?x$)$.

Dans ce qui suit, nous utilisons $[\![x]\!]_G$ [2] pour désigner le processus permettant de trouver les correspondances pertinents de x par rapport à G.

Definition 4.1.8. (Évaluation d'un motif de triplet) Nous appelons $[\![t]\!]_G$ le processus permettant de trouver les correspondances liées à un motif de triplet t par rapport à un graphe G. $[\![t]\!]_G$ est formellement défini comme suit : $[\![t]\!]_G := \{\mu | dom(\mu) = vars(t)$ et $\mu(t) \in G\}$

De façon informelle, nous essayons de trouver des correspondances de telle sorte que lorsque nous remplacerons les nœuds variables par des constantes correspondantes, le triplet obtenu se trouve dans le graphe de données.

Dans ce qui suit, nous utilisons une évaluation de triplet pour définir une évaluation d'une étoile de requête.

Definition 4.1.9. (Évaluation de l'étoile de requêtes) De façon informelle, l'évaluation d'une étoile de requêtes permet de trouver des correspondances pour les variables dans l'étoile de requête par rapport au graphe de données. Pour chaque triplet t dans l'étoile, nous essayons de trouver l'ensemble des correspondances satisfaisant t. Nous construisons ensuite les correspondances de l'étoile de requête en joignant des correspondances liées aux triplets de l'étoile de requêtes.

Formellement, l'évaluation d'une requête $ER(x)$ par rapport au graphe de données G est définie comme suit :

$[\![ER(x)]\!]_G := \{[\![t_1]\!]_G \bowtie [\![t_2]\!]_G \bowtie ... \bowtie [\![t_n]\!]_G | n = card(ER(x))\}$

où :

$[\![t_i]\!]_G \bowtie [\![t_j]\!]_G = \{\mu_l \cup \mu_r | \mu_l \in [\![t_i]\!]_G$ et $\mu_r \in [\![t_j]\!]_G$, $\mu_l \sim \mu_r$ et $\mu_l(t_i) \neq \mu_r(t_j)\}$

2. x peut être un motif de triplets ou une étoile de requêtes

A partir de cette définition, nous définissons l'évaluation de l'opérateur de jointure de deux étoiles de requêtes. En effet, la jointure sera utilisée comme opérateur de base de l'évaluation de la requête.

Definition 4.1.10. (Jointure des étoiles) De façon informelle, la jointure de deux étoiles de requête permet d'assembler les correspondances obtenus en évaluant deux étoiles de requêtes. Formellement, l'évaluation d'une jointure entre deux étoiles de requêtes est définie comme suit :

$[\![ER_i]\!]_G \bowtie [\![ER_j]\!]_G = \{\mu_l \cup \mu_r | \mu_l \in [\![ER_i]\!]_G, \mu_r \in [\![ER_j]\!]_G \text{ et } \mu_l \sim \mu_r\}$

Nous prenons une correspondance obtenue à partir de l'étoile de requête de gauche et une autre de l'étoile de requête de droite, nous vérifions si les deux correspondances sont compatibles, l'union de ces correspondances est une correspondance valide pour la jointure des deux étoiles de requête.

Avant de présenter l'évaluation des requêtes à l'aide des étoiles de requêtes, nous allons définir le concept de couverture. En effet, nous utilisons ce concept pour garantir qu'un ensemble donné des étoiles de requêtes permette une évaluation correcte de la requête.

Definition 4.1.11. (Couverture d'étoile)

Nous notons $Cover_q(ER)$ l'ensemble des motifs de triplets de requête partagés avec l'étoile de requête.

$Cover_q(ER) = \{t | t \in triplets(q) \cap triplets(ER)\}$

La fonction triplets renvoie les motifs de triplets associés à une requête ou une étoile de requêtes.

A partir des définitions précédentes, nous pouvons définir l'évaluation des requêtes à l'aide d'un ensemble d'étoile de requête, comme suit :

Proposition 4.1.2. *Étant donné un ensemble d'étoiles* $\{ER_1, ER_2, ..., ER_n\}$ *qui couvrent la requête, i.e.,* $Cover_q(ER_1) \cup Cover_q(ER_2) \cup ... \cup Cover_q(ER_n) = triplets(q)$*, l'évaluation de q en utilisant l'ensemble des étoiles de requête est définie comme suit :*

$[\![q]\!]_G = [\![ER_1]\!]_G \bowtie [\![ER_2]\!]_G \bowtie ... \bowtie [\![ER_n]\!]_G$ *par rapport aux fragments, nous pouvons définir l'évaluation de la requête comme suit :*

$[\![q]\!]_G = \bigcup_{fG \models ER_1} [\![ER_1]\!]_{fG} \bowtie \bigcup_{fG \models ER_2} [\![ER_2]\!]_{fG} \bowtie ... \bowtie \bigcup_{fG \models ER_n} [\![ER_n]\!]_{fG}$

Il est à noter que les résultats de l'évaluation d'un motif de triplet, d'une étoile de requêtes ou d'une requête est un ensemble de correspondances. Pour le reste de ce manuscrit, nous notons par ω cet ensemble de correspondance. Un ensemble de correspondance, qui représente tous les résultats obtenus en évaluant un motif de triplet, une étoile de requêtes ou une requête, est noté Ω (*i.e.*, $\Omega = \{\omega\}$)

Étant donné une requête et les étoiles obtenues à partir des arcs entrants sortants, nous pouvons facilement montrer que l'ensemble des étoiles de requête permettant d'évaluer la requête n'est pas unique. A titre d'exemple, la requête de la Figure 2.9 peut être évalué à la fois par les étoiles de requêtes $[ER_{1f}, ER_{2f} et ER_{3f}]$ et les étoiles $[ER_{1f}, ER_{2f} et ER_{2b}]$.

Nous utilisons le mot "Plan" pour faire référence à un ensemble d'étoiles de requête permettant d'évaluer une requête donnée. Formellement, un plan est défini comme suit :

Proposition 4.1.3. *Un plan est une fonction d'ordre sur un ensemble d'étoile de requête permettant d'évaluer des requêtes. Nous notons par* $p = [ER_1, ER_2, ..., ER_n]$ *le plan formé en exécutant* ER_1*, puis* ER_2*,..., jusqu'à* ER_n

Afin refléter notre stratégie d'exploration de graphe, nous ajoutons au plan quelques contraintes. Cela permet aussi d'éviter l'utilisation du produit Cartésien qui est très coûteux en termes de calcul. Nous appelons le nouveau plan "Plan d'exécution Valide ($\mathcal{PV}$)", s'il respecte les conditions suivantes :

1. *Couverture :* Tous les nœuds et prédicats de la requête sont *couverts* par l'ensemble des étoiles de requêtes du plan. Par exemple, pour la requête de la Figure 2.9, le plan d'exécution $[\overrightarrow{ER}(?film), \overrightarrow{ER}(?acteur1)]$ n'est pas valide puisque l'arc `né_à` qui relie $?acteur1$ à $?ville$ n'est pas couvert par le plan.

2. *Instantiation des têtes :* Cette condition garantit que pour un plan $\mathcal{P} = [ER_1, ... , ER_n]$, $\forall_{i>1} ER_i$, la tête de ER_i doit être déjà instantiée. Nous utilisons cette condition pour éviter l'utilisation du produit Cartésien lorsque des correspondances sont échangées entre deux étoiles de requêtes.

   Par exemple, le plan $[\overrightarrow{ER}(?film), \overleftarrow{ER}(?ville)]$ n'est pas valide puisque la tête de la deuxième étoile de requêtes ($\overleftarrow{ER}(?ville)$ n'a pas été instantiée avant de chercher les correspondances pour cette étoile de requêtes. Pour le plan $[\overrightarrow{ER}(?film), \overrightarrow{ER}(?acteur1), \overleftarrow{ER}(?ville)]$, la tête a été instantiée.

La définition formelle d'un plan valide est donnée dans la proposition 4.1.4.

Proposition 4.1.4. *Plan Valide ($\mathcal{PV}$) Considérons q comme une requête donnée,* $\overrightarrow{ER}$ *et* $\overleftarrow{ER}$ *en tant que ensembles d'étoile de requêtes entrants et sortants respectivement, T comme ensemble de motifs de triplets et les fonctions suivantes :*

— *Tr :* $q \cup \overrightarrow{ER} \cup \overleftarrow{ER} \rightarrow T$ *Elle renvoie l'ensemble des motifs de triples d'un graphe.*

— *Nd :* $q \cup \overrightarrow{ER} \cup \overleftarrow{ER} \rightarrow V$ *Elle renvoie les nœuds d'un graphe.*

— *Head :* $\overrightarrow{ER} \cup \overleftarrow{ER} \rightarrow V$ *une fonction qui renvoie la tête d'une étoile de requêtes.*

Un ***Plan Valide*** $\mathcal{PV}$ *est un couple* $< X, f >$ *où* $X \subset \overrightarrow{ER} \cup \overleftarrow{ER}$ *and* $f : X \rightarrow \{1...|X|\}$ *est la fonction d'ordre des étoiles de requêtes telle que :*

1. $\bigcup_{ER \in X} Tr(ER) = Tr(q)$
2. $\forall i \in \{2...|X|\}, head(f^{-1}(i)) \in \bigcup_{j=1}^{i-1} Nd(f^{-1}(j))$

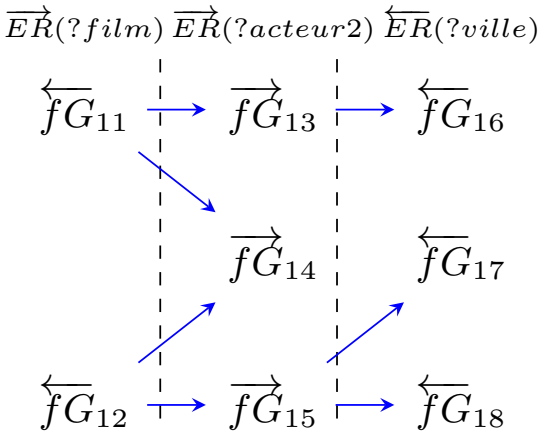


FIGURE 4.3 – Flux d'exécution

D'un point de vu flux de données, les correspondances circulent d'un fragment à un autre. Chaque fragment valide et enrichie les correspondances qu'il a reçues. Un aperçu d'un exemple de la circulation des correspondances entre fragments est montré dans la Figure 4.3. L'une des stratégies d'optimisation que nous proposons consiste à regrouper les flux pour minimiser les accès disques.

4.2 Mise en oeuvre et optimisation

Dans cette section, nous détaillons nos choix d'implémentation, qui couvrent principalement notre stratégie de stockage, nos mécanismes d'évaluation de requêtes et pour finir nos techniques d'optimisation.

4.2.1 Stockage de graphes RDF

Dans cette section, nous détaillons notre approche de stockage de données RDF. En effet, le schéma de stockage influe considérablement le modèle d'exécution car le moteur d'évaluation de requêtes est conçu pour une configuration de stockage spécifique. Comme l'a été mentionné précédemment, plusieurs modèles de stockage de données RDF ont été proposé. Toutefois, d'un point de vue représentation à base de graphe, les approches existantes matérialisent les arcs entrants, les arcs sortants ou les deux. Par exemple, RDF-3X [Neumann and Weikum (2008)], HexaStore [Weiss et al. (2008)] et Virtuoso [Erling (2012)] proposent de matérialiser les triplets RDF sous la forme de SPO (Sujet, Prédicat, Objet) et de OPS (Objet, Prédicat, Sujet). Dans ce type d'approches, il s'agit de matérialiser à la fois les arcs entrants et les arcs sortants du graphe construit à partir des triplets. De l'autre côté, les systèmes stockant les données comme des listes d'adjacence et non dans des tables SPO ou OPS (e.g., gStore [Zou et al. (2011)]) matérialisent les arcs sortants de ce même graphe. Selon l'approche de matérialisation, nous pouvons déduire facilement quels types d'exploration sont efficaces sur ce type de graphes. Par exemple, dans le premier type d'approches, l'exploration est possible dans les deux sens (entrant et sortant) tandis que pour le deuxième type, nous pouvons nous appuyer seulement sur l'exploration en utilisant les arcs sortants. Les deux structures SPO et OPS représentent la

même information mais avec un ordre diffèrent. Cet ordre permet en effet d'accélérer l'exploration dans un sens ou dans un autre. Généralement, une seule structure physique (fichier, index, ...) est utilisé pour matérialiser SPO et/ou OPS. La recherche dans cette structure devient pour les jeux de données avec plusieurs milliards de triplets inefficace même si nous nous appuyons sur les indexes.

SPO et OPS stockent des informations hétérogènes. Par exemple, on peut trouver dans le même graphe des informations sur les films et aussi des informations sur les transactions boursières. Cependant, les requêtes manipulent rarement des informations non liées. Une requête se concentre sur un sous-ensemble du jeu de données caractérisé principalement par ses prédicats. Dans notre travail, nous considérons une représentation de données basée sur un graphe utilisant à la fois des structures SPO et OPS. Nous n'avons pas envisagé d'autres combinaisons, comme par exemple POS et SOP, car notre technique de d'évaluation n'en pas besoin. SPO et OPS sont généralement stockées sous forme d'un arbre de type B+ groupé, ce qui permet d'effectuer une recherche de triplets avec un coût d'accès disque de l'ordre de $log(n)$ (n est le nombre de triplets dans la base de données). Une lecture complète des structures physiques associés n'est pas nécessaire pour les requêtes sélectives. Comme mentionné précédemment, au lieu de matérialiser SPO et OPS avec une seule structure physique, nous proposons de les fragmenter selon les ensembles de caractéristiques [Neumann and Moerkotte (2011)], qui à l'origine ont été conçu uniquement pour collecter des statistiques sur des données. Ces ensembles n'ont pas d'impact sur l'organisation des triplets et sur l'évaluation des requêtes. Dans notre cas, les données sont physiquement stockées sous forme de fragments de graphe. Dans le cas d'une requête (non sélective), où nous devons effectuer une lecture complète, seuls les fragments pertinents (qui répondent à la requête) seront pris en compte. Lors de l'exécution de la requête, on commence par l'identification du ou des ensembles de caractéristiques en fonction des prédicats de la requête. Nous proposons aussi d'indexer séparément les fragments en utilisant un arbre de type B+ groupé. Cela permet d'accélérer la recherche au sein de ses structure. Un nouveau mécanisme est aussi proposé pour accéder aux données lors que le même fragment est sollicité plusieurs fois.

Notre approche d'évaluation de requêtes permet d'explorer les fragments du graphe en utilisant les voisins d'un nœud (sujet ou objet du triplet RDF) en cours de traitement. Afin de passer à un autre nœud, nous aurons besoin d'identifier le fragment qui regroupe les arcs (entrants ou sortants) de ce nœud. Nous proposons donc d'étendre chaque triplet pour intégrer des informations liées aux fragments qui hébergent les arcs entrants et sortants des nœuds mentionnés dans ce triplet :

$< Node_1, \mathcal{G}f$,Predicat,$Node_2, \mathcal{G}f_{in}, \mathcal{G}f_{out} >$

où :

— $Node_1$ représente le sujet en cas d'un triplet SPO ou l'objet en cas d'un triplet OPS.

— $\mathcal{G}f$ représente l'identifiant du fragment regroupant les arcs sortants dans le cas où $node_1$ est un sujet ou des arcs entrants dans le cas d'un objet.

— $Node_2$: représente l'objet en cas de triplet SPO ou le sujet en cas de triplet OPS.

— $\mathcal{G}f_{in}$ représente l'identifiant du fragment qui regroupe les arcs entrants du $node_2$.

— $\mathcal{G}f_{out}$ représente l'identifiant su fragment qui regroupe les arcs sortants du $node_2$.

$node_1$ et $node_2$ sont représentés en utilisant 64 bits (8 octets). En effet, ce choix permet à notre système de gérer plusieurs types de données numériques, à savoir Short (2 octets), Integer (4 octets), Long (8 octets), Float (4 octets) et Double (8 octets). Ainsi, nous intégrons également des opérations de filtrage natives sur ces types de données en utilisant la structure arbre B+.

D'autre part, $\mathcal{G}f$, $\mathcal{G}f_{in}$ and $\mathcal{G}f_{out}$ sont représentés en utilisant 32 bits (4 octets). Notre système peut donc gérer jusqu'à 4294967296 (2^{32}) différents fragments. Stocker toutes ces données directement sur le disque pourrait entraîner un coût de stockage très important. Nous proposons une technique de compression adaptée qui sera détaillée dans la section suivante.

Les arbres B+ permettent de stocker les triplets encodés avec des valeurs numérique. Nous proposons d'ailleurs de créer une sorte de dictionnaire bidirectionnel qui permet d'associer à chaque latéral et URI un identifiant numérique. Ce dictionnaire offre deux opérations, la première permet de chercher un identifiant pour une chaîne de caractères tandis que la deuxième permet de trouver une chaîne de caractères à partir d'un identifiant donné. Ce dictionnaire permet aussi de trouver des nœuds (identifiants) en utilisant des expressions régulières. L'exemple de la Figure 4.4 montre l'encodage des triplets en utilisant des identifiants.

4.2.1.1 Compression

Dans cette section, nous expliquons comment les données sont physiquement représentées sur le disque. En effet, comme expliqué précédemment, les données sont regroupées en fragments. Chaque fragment regroupe des étoiles de données avec le même ensemble de caractéristiques. Les étoiles de données sont triées en fonction de la valeur de leurs têtes et sont stockées sous forme d'un arbre B+ groupé. De plus, comme notre méthode d'évaluation (basée sur l'exploration de graphe) doit localiser le fragment hébergeant les arcs entrants et sortants des nœuds du graphe, nous proposons de matérialiser cette information directement lors du stockage de triplets liés à une étoile de données.

Indicateur	**Prédicat**
17 bits	1 bit

$Node_1$	**fragment**	$Node_2$	**fragment (in)**	**fragment (out)**
0-8 bytes	0-4 bytes	0-8 bytes	0-4 bytes	0-4 bytes

Tableau 4.4 – Diagramme de Compression

Tout d'abord, nous stockons le schéma (liste des prédicats et leurs types) de chaque fragment de graphe. Nous trions les prédicats par rapport à leurs identifiants. Ensuite, nous consacrons un seul bit (0 ou 1 comme valeur) pour indiquer si le triplet en cours d'encodage a le même prédicat comme le triplet précédent.

Ceci est possible puisque :

— Nous regroupons les étoiles de données avec le même ensemble de caractéristiques dans le même fragment.

Dictionnaire de nœuds	
ID	String
1	la loi et l'ordre
2	Brian Dennehy
3	Jon Avnet
4	Al Pacino
5	Robert De Niro
6	Film policier américain
7	Films de crimes
8	Oscar
9	Golden globe
10	Drama
11	New York

Dictionnaire de prédicats	
ID	String
1	Vedette
2	Catégorie
3	Gamme
4	Réalisateur
5	né_à
6	est_nommé
7	a_gagné
8	Genre

(a) Dictionnaire des IDs

Fg.1 $\models 1,2,4,8$		
S	P	O
1	1	2
1	1	4
1	1	5
1	3	6
1	4	3
1	8	10

Fg.2 $\models 5,7$		
S	P	O
4	5	11
4	7	8
4	7	9
5	5	11
5	7	8
5	7	9

Fg.3 $\models 3$		
S	P	O
6	3	7

Fg.4 $\models 5,6$		
S	P	O
3	5	11
3	6	9

(b) Fragmentation de la structure SPO sur disque

Fg.5 $\models 6,7$		
O	P	S
9	6	3
9	7	4
9	7	5

Fg.6 $\models 5$		
O	P	S
11	5	3
11	5	4
11	5	5

Fg.7 $\models 7$		
O	P	S
8	7	4
8	7	5

Fg.8 $\models 1$		
O	P	S
2	1	1
4	1	1
5	1	1

Fg.9 $\models 8$		
O	P	S
10	8	1

Fg.10 $\models 4$		
O	P	S
3	4	1

Fg.11 $\models 2$		
O	P	S
6	2	1

Fg.12 $\models 3$		
O	P	S
7	3	6

(c) Fragmentation de la structure OPS sur disque

FIGURE 4.4 – Exemple de stockage sur disque

— l'ensemble des triplets est trié par têtes (sujet ou objet du triplet associé)

— chaque étoile de données est associée à un seul ensemble de caractéristiques.

Le premier bit est égal à 1 pour le premier triplet d'une étoile de donnée. Si le triplet suivant appartient à la même étoile de données et qu'il a le même prédicat que le triplet précédent, la valeur de son bit est 0. Si le prédicat n'est pas le même, alors les valeurs de bit est égal à 1.

Pour compresser les autres éléments du triplet étendu, nous avons été inspirés par RDF3X [Neumann and Weikum (2008)]. Comme pour RDF-3X, nous utilisons le concept d'indicateur. Nous avons choisi d'allouer un nombre variable d'octets pour chaque élément du triplet. Ce nombre ne représente que ce qui est nécessaire pour représenter la valeur de l'élément du triple. En revanche, dans notre cas, comme nous avons étendu la notion de triplet pour ajouter des informations sur les fragments liés aux nœuds (sujet et objet) du triple, l'indicateur est également calculé pour ces fragments. Un total de 17 bits est alloué à chaque triplet.

Pour le $Node_1$, nous devons représenter 9 États : Etat = 0 pour indiquer que le $Node_1$ ne change pas par rapport au triplet précédent. État $\in$ [1..8] pour indiquer le nombre d'octets que nous avons utilisé pour coder le nouveau $Node_1$. Nous avons besoin de 4 bits pour représenter les 9 États. Par exemple, avec l'État = 1 et pour représenter la valeur de nœud 1, nous avons besoin d'au plus 1 octet.

Pour $\mathcal{G}f$, $\mathcal{G}f_{in}$ et $\mathcal{G}f_{out}$: nous avons besoin de 5 états : 0 pour indiquer que cette information ne change pas ou NULL. i $\in$ [1..4] pour indiquer le numéro que nous avons utilisé pour encoder le nouveau identifiant du fragment. Nous avons donc besoin de 3 bits pour représenter les 5 états.

Enfin, Pour $node_2$, nous utilisons i $\in$ [1..8] pour indiquer le numéro que nous avons utilisé pour encoder le nouveau $node_2$. Nous avons donc besoin de 4 bits pour représenter les 8 états.

Comme illustré dans la Figure 4.4, il faut au total 18 bits. Ces bits représentent l'indicateur du nombre d'octets utilisés pour encoder chaque triplet. Il est à noter que seules les pages feuilles de l'arbre B+ groupé sont compressées.

La Figure 4.5 illustre notre stratégie d'encodage de données pour le fragment fg.2. La première partie montre comment intégrer les informations liées aux fragments associés au nœuds mentionnés par le triplet. Nous montrons aussi comment remplacer les id des prédicats par 0 ou 1. Dans la deuxième partie, nous montrons comment calculer les indicateurs pour les triplets t1 et t2 de ce même fragment.

4.2.2 Opérations d'évaluation de requêtes

Nous présentons dans cette section notre mécanisme d'évaluation de requêtes. Nous commençons par présenter nos différents opérateurs d'évaluation. Notre système est basé sur cinq opérateurs :

1. Extracteur_Etoiles_Données (EED) : Cet opérateur permet d'extraire les étoiles de données à partir des fragments.
2. Extracteur_Correspondances (EC) : Cet opérateur permet de trouver les correspondances pour une étoile de requêtes en fonction des correspondances déjà extraites et des étoiles de données.
3. Mappeur_Dictionnaire (MD) : Cet opérateur permet d'interagir avec le dictionnaire pour encoder et décoder les chaînes de caractères.
4. Agrégateur_Données (AD) : Cet opérateur permet d'agréger les données dans le cas de requêtes avec "Group By". Nous supportons plusieurs fonctions d'agrégation (e.g., sum, count, avg).
5. Trieur_Données (TD) : Cet opérateur permet de trier les données en fonction des variables qui sont mentionnées dans la clause "Order By" de la requête.

S	P	O
4	5	11
4	7	8
4	7	9
5	5	11
5	7	8
5	7	9

(a) Fg2

Intégration des fragments

T	n1	Fg	Pr	n2	Fg(in)	Fg(out)
t1	4	∅	5	11	6	∅
t2	4	∅	7	8	7	∅
t3	4	∅	7	9	5	∅
t4	5	∅	5	11	6	∅
t5	5	∅	7	8	7	∅
t6	5	∅	7	9	5	∅

Compression (prédicat)

T	n1	Fg	Pr	n2	Fg(in)	Fg(out)
t1	4	∅	1	11	6	∅
t2	4	∅	1	8	7	∅
t3	4	∅	0	9	5	∅
t4	5	∅	1	11	6	∅
t5	5	∅	1	8	7	∅
t6	5	∅	0	9	5	∅

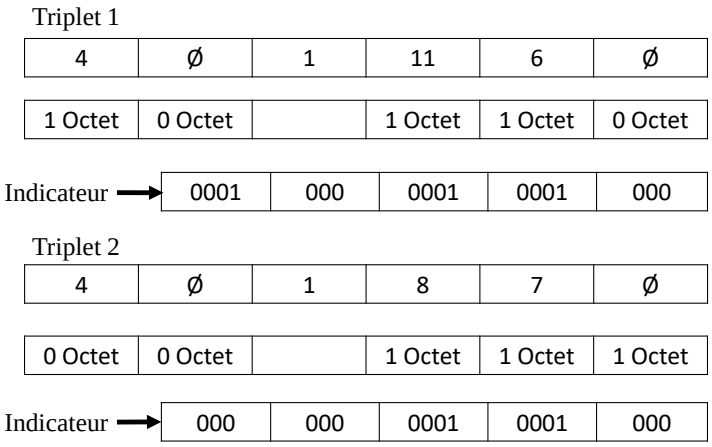


FIGURE 4.5 – Illustration de la méthode de compression

6. Collecteur_Résultats (CR) : Cet opérateur intègre une stratégie de traitement de résultats finaux selon les besoins de l'utilisateur. Cela pourrait de l'affichage sur écran, l'écriture dans un fichier ou même la transmission de résultats via le réseau.

Nous détaillons dans la suite de cette section le fonctionnement de quelques opérateurs.

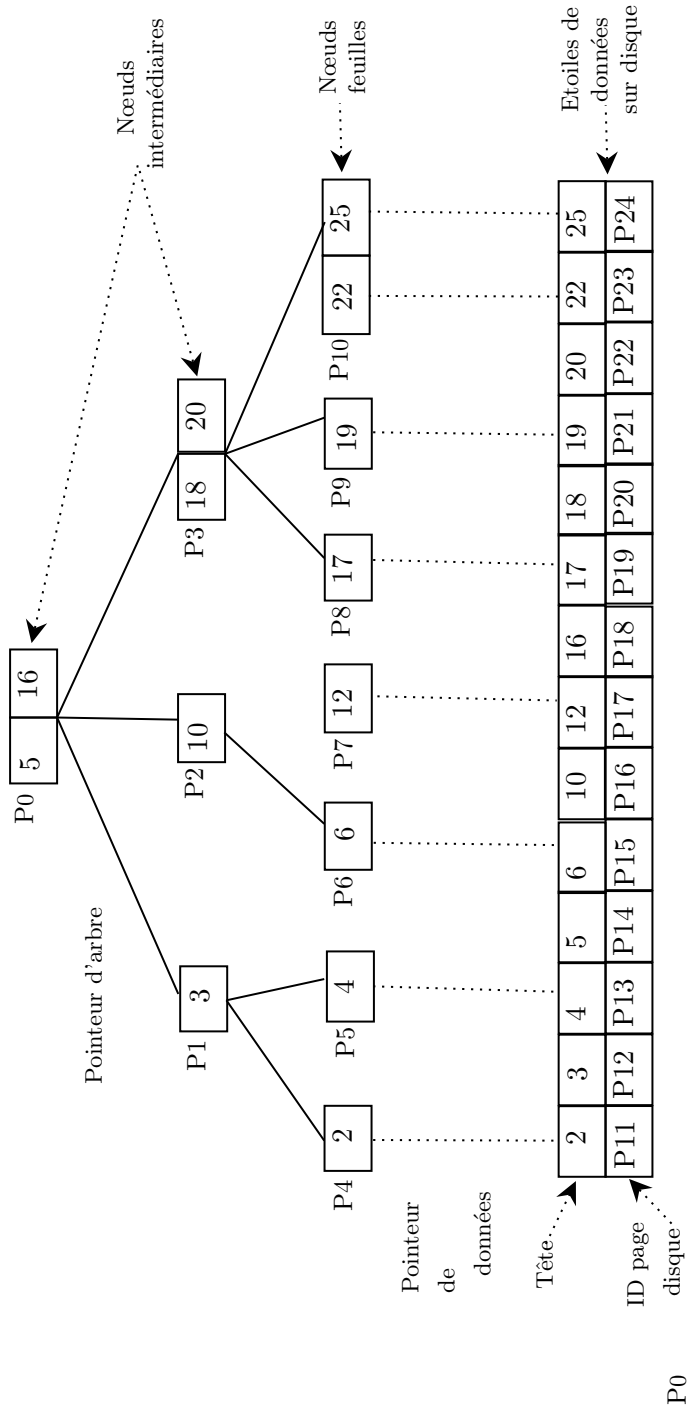


FIGURE 4.6 – Apperçu d'un arbre B+.

4.2.2.1 Extracteur d'Etoiles de Données (EED)

L'étoile de données est l'une des structures de données principale. Toute demande de lecture à partir du disque génère un ou plusieurs étoiles de données. Le module EED offre plusieurs types d'appel pour un fragment donné :

- Extraction de la totalité des étoiles d'un fragment en fonction d'un ensemble de prédicat. Généralement, cette stratégie est utilisée lorsque nous voulons traiter une étoile de requêtes avec une tête variable.
- Extraction d'une étoile de données liée à une valeur donnée. Cette stratégie est utilisée lorsque la tête de l'étoile de requêtes est une constante. Seulement les arcs mentionnés par un prédicat dans l'étoile de requêtes sont pris en compte.
- Extraction de plusieurs étoiles de données en amont. Cette stratégie, permet de récupérer les étoiles de données liées à une liste de têtes passées comme paramètre. Nous détaillons cette stratégie dans la section suivante. Cette stratégie est utile pour les étoiles de requêtes qui n'ont pas la première position dans le plan d'exécution.
- Extraction de plusieurs étoiles avec une tête qui satisfait une condition booléenne (e.g., $>, <, between$). Cette stratégie est utilisée pour le traitement de la première étoile de requêtes qui est mentionnée dans la clause "FILTER".

4.2.2.2 Accès aux données via l'arbre B+

Dans cette section, nous discutons notre stratégie de recherche en amont d'arcs entrants (ou sortants) d'un nœud à l'aide d'un arbre B+. Nos opérateurs d'évaluation utilisent la structure de l'arbre B+ pour trouver les arcs pertinents. L'algorithme naïf consiste à effectuer une recherche sur l'arbre B+ pour chaque nœud dont nous avons besoin. Si nous avons n nœuds, nous avons besoin de $n * log(n)$ pour trouver tous les arcs nécessaires. Prenons comme exemple l'arbre B+ montré dans la Figure 4.6. Pour trouver les étoiles de données avec les têtes [5,12,16,18,25], nous avons besoin de lire les pages suivantes :

- Pour 5 : $[P_0, P_1, P_5, P_{13}, P_{14}]$
- Pour 12 : $[P_0, P_2, P_7, P_{17}]$
- Pour 16 : $[P_0, P_2, P_7, P_{17}, P_{18}]$
- Pour 18 : $[P_0, P_3, P_8, P_{19}, P_{20}]$
- Pour 25 : $[P_0, P_3, P_{10}, P_{24}]$

Au total, ce processus implique la lecture de 23 pages disques.

Nous proposons une autre stratégie afin de minimiser le nombre d'accès lié à la lecture des clés intermédiaires. Nous factorisons en effet les traversées de L'arbre B+. Nous commençons par trouver la première étoile de données. Au cours de cette opération, nous mémorisons les pages intermédiaires que nous avons utilisées. Nous les passons aux prochaines étoiles de données. Nous comparons avec la clé précédemment mémorisée, nous déclenchons une découverte partielle si nous violons l'une des clés mémorisées.

L'entrée de notre algorithme est un vecteur de têtes d'étoiles de données et la sortie est un ensemble d'étoiles de données.

$BTreePath$ est une liste temporaire de nœuds d'arbre B+. Chaque nœud est composé de trois éléments :

1. page disque
2. condition
3. position

Algorithm 1: Extraction_Etoiles_Btree(LT,$BTreePath$,ED)

```
   Result: ED : L'ensemble des étoiles de données demandées
 1 if LT est vide then
 2 |   return ED;
 3 else
 4 |   Clé ← la valeur minimal dans LT;
 5 |   for N ∈ BTreePath do
 6 |   |   if Clé ne respecte pas N.condition then
 7 |   |   |   Supprimer N et ses successeurs de BTreePath
 8 |   end
 9 |   N ← dernier nœud dans BTreePath;
10 |   Chercher_Etoile(Clé,N,ED,BTreePath);
11 end
12 Extraction_Etoiles_Btree(LT,BTreePath)
```

Notre procédure principale "Extraction_Etoiles_Btree"(algo. 1) prend en entrée une liste de têtes d'étoiles de données à chercher ("LT") et une trace des nœuds de l'arbre B+ déjà chargé en mémoire ("BTreePath") et renvoie une liste d'étoiles de données ("ED"). Cette procédure commence par vérifier s'il reste des têtes à traiter. Si c'est affirmatif, nous commençons par extraire la tête avec la valeur minimale. Nous examinons ensuite la trace de l'arbre pour garder que les nœuds valides (qui respectent les conditions). Les autre nœuds sont supprimés. A partir du dernier nœud valide, nous déclenchons la recherche de nouvelles étoiles de données en utilisant la procédure "Chercher_Etoile". Nous passons ensuite aux autres têtes dès la fin de l'exécution de cette procédure.

La procédure "Chercher_Etoile"(algo. 2) prend en entrée la clé à chercher (la tête de l'étoile), le nœud l'arbre que nous voulons explorer, ainsi que la trace de l'exploration de l'arbre B+. Si la page que nous volons explorer est une page intermédiaire, nous commençons par chercher la clé de cette page qui limite la valeur de la tête de l'étoile cible. Nous créons ensuite un nouveau nœud de trace pour le fils qui satisfait cette clé et nous déclenchons la recherche récursive pour ce nouveau nœud. Dans le cas d'une page feuille, nous examinons l'existence de l'étoile dans cette page. Dans le cas positif, cette étoile sera ajoutée à la liste des résultats. Sinon, nous considérons les pages voisines. Nous arrêtons lorsque nous trouvons l'étoile recherchée.

Avec notre approche, seulement 15 (vs. 23 pour l'approche standard) pages ont été chargé du disque. Les pages que nous considérons sont les suivantes :
$[P_0, P_1, P_5, P_{13}, P_{14}, P_2, P_7, P_{17}, P_{18}, P_3, P_8, P_{19}, P_{20}, P_{10}, P_{24}]$

Il est à noter que les algorithmes que nous proposons sont adaptés à notre contexte et ne remplaceront dans aucun cas l'algorithme standard de recherche.

Algorithm 2: Chercher_Etoile($Clé$,N,ED,$BTreePath$)

```
1  if N.page est une page intermédiaire then
2  |  i ← N.position;
3  |  while i < N.page.nb_clés && Clé > N.page.Clés do
4  |  |  i++;
5  |  end
6  |  N.condition ← "<= Clé" ;
7  |  Créer un nouveau nœud N' ;
8  |  N'.page ← fils[i] de N.page ;
9  |  N'.position ← 0 ;
10 |  Chercher_Etoile(Clé,N',ED,BTreePath);
11 else
12 |  P ← N.page ;
13 |  if P contient l'étoile ed avec Clé comme tête then
14 |  |  ajouter ed à ED;
15 |  else
16 |  |  P ← P.successeur ;
17 |  |  N.page ← P ;
18 |  |  Chercher_Etoile(Clé,N,ED,BTreePath);
19 |  end
20 end
```

4.2.2.3 Extracteur de Correspondances

Cet opérateur permet d'extraire de nouvelles correspondances liées à une étoile de requêtes. Nous considérons deux cas selon la position de l'étoile de requêtes dans le plan d'exécution :

— Lorsque l'étoile est la première du plan, nous n'avons pas de correspondances existantes qu'on doit enrichir. La tête de l'étoile pourrait être variable ou constante. Une tête variable peut être aussi mentionné dans un filtre. Selon ces types, nous déclenchons les méthodes d'extractions adéquates. Une extraction de correspondances est effectuée. Si nous avons des variables (autre que la tête) mentionnées dans la clause FILTRE avec une expression régulière, nous faisons appel au dictionnaire pour décoder la valeur de la correspondance. La correspondance de l'étoile de requête est gardée seulement si la valeur satisfait l'expression régulière.

— Lorsque l'étoile n'est pas la première, la génération d'une correspondance doit respecter celles qui sont déjà créés. Dans ce cas, nous cherchons que les étoiles de données mentionnées par une correspondance. Nous faisons appel à la méthode d'extraction que nous avons développé. Après la génération de correspondances, nous faisons un filtrage si c'est nécessaire.

4.2.2.4 Mappeur de Dictionnaire

Le dictionnaire joue un rôle très important dans notre système. En effet, il est sollicité à plusieurs reprises pendant l'évaluation d'une requête :

— Lors du pré-traitement de la requête, nous sollicitons le dictionnaire pour décoder les constantes. Cela permet de trouver l'identifiant et les fragments hébergeant les étoiles de données liées à cette constante. Nous pouvons facilement charger les étoiles pertinentes.

— Pendant l'évaluation de la requête, nous sollicitions le dictionnaire pour vérifier les valeurs qui sont associés à des variables mentionnées dans la clause FILTER.

— Après l'exécution de la requête, nous sollicitions le dictionnaire pour associer des chaînes de caractères aux identifiants qui contribuent à la construction des résultats finaux. Ces derniers sont en effet envoyés à l'utilisateur final.

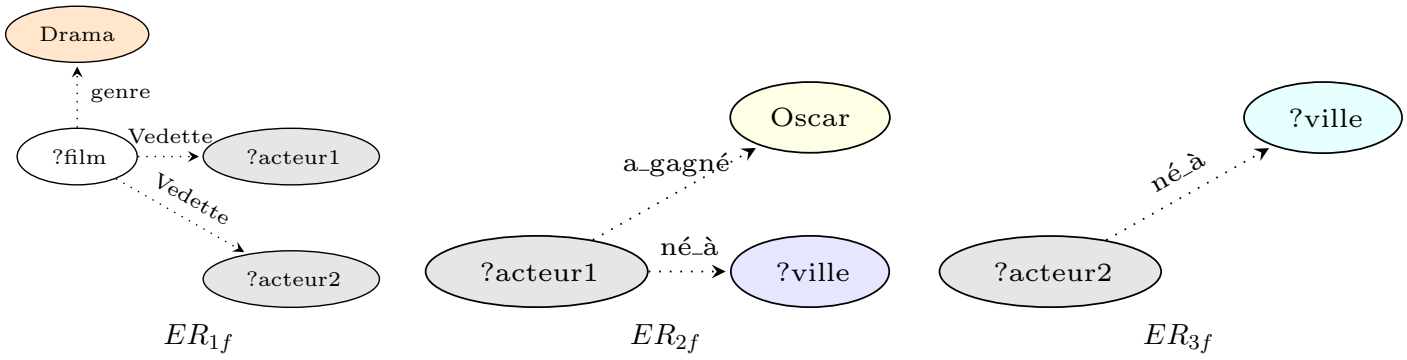
4.2.2.5 Collecteur de Résultats

Cet opérateur permet d'interagir avec l'utilisateur. Il assure principalement la transmission des résultats. Nous proposons plusieurs implémentations de ces modules. La première est la plus simple. Elle affiche les résultats sur l'écran. Elle est utilisée pour des besoin volatiles. La deuxième implémentation consiste à stocker les résultats sur disque, ce qui permet de persister ces résultats. Nous pouvons ensuite les réutiliser. La dernière stratégie permet de transmettre les résultats via le réseau informatique. En effet, un client pourrait être installé sur une machine distante. Des interfaces de programmation (e.g., JDBC, ODBC) seront intégrées lors de la prochaine version du système.

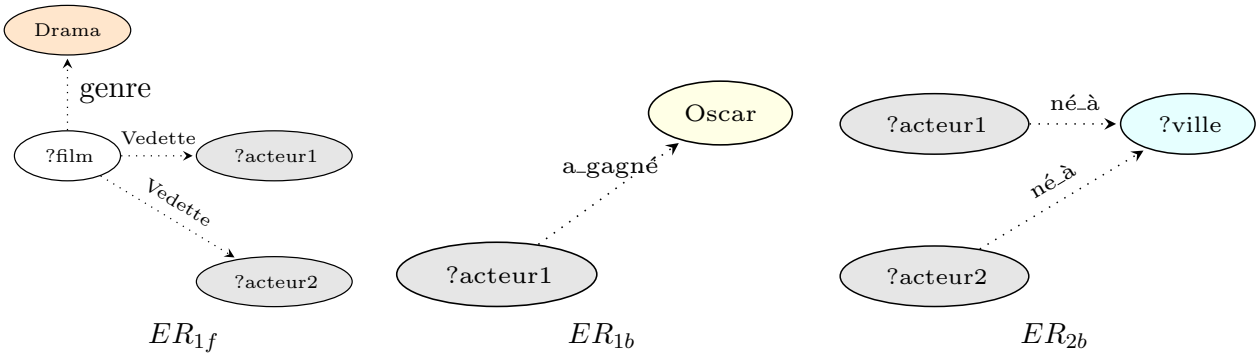
4.2.3 Adaptation du modèle Volcano

La première caractéristique de "RDF_QDAG" est le passage à l'échelle. En effet, notre objectif était de supporter le traitement plusieurs milliards de triplets RDF. Afin d'atteindre un tel objectif, il est nécessaire d'avoir un contrôle total sur la mémoire centrale. Concrètement, il faut être en mesure de contrôler le nombre d'étoiles de données et de correspondances qui sont stockées en mémoire. Le noyau d'exécution du système est basé sur le modèle Volcano [Graefe (1994)], qui permet de contrôler la quantité de données chargées dans la mémoire principale en évitant tout débordement. Nous avons conçu "RDF_QDAG" pour garantir une telle propriété en introduisant plusieurs types de mémoires tampon, qui interagissent avec les différents opérateurs :

1. La mémoire tampon pour étoiles de données (MTD) permet de stocker les étoiles de données. Elle est remplie par l'extracteur d'Etoiles de Données (EED). Son contenu est consommé par un extracteur de correspondances (EC).
2. La mémoire tampon pour les correspondances (MTC) permet de stocker les correspondances partielles. Elle est remplie par un EC. Son contenu est consommé par un autre EC. Afin de minimiser les accès multiples au même fragment, nous proposons de regrouper les correspondances en fonctions du fragment données et à l'étoile demandée. $[[fG_1, [t_1, [w_{11}, .., w_n]], [t_2, [w'_{12}, .., w'_n]], .., [fG_k, [..]]$. Pour traiter $[w_{11}, .., w_n]$, nous chargeons une seule fois l'étoile de données avec la tête t_1 à partir du fragment fG_1.
3. La mémoire tampon pour l'agrégateur (MTA) permet de faire une agrégation partielle des données. Elle est remplie par un EC. Son contenu est consommé par agrégateur de données. Dans le cas où le contenu de la mémoire dépasse



(a) Plan 1 : $\{ER_{1f}, ER_{2f}, ER_{3f}\}$



(b) Plan 2 : $\{ER_{1f}, ER_{1b}, ER_{2b}\}$

FIGURE 4.7 – Exemples de plan d'exécution de requête

la limite possible, une partie des données est stocké sur disque. L'agrégateur nécessite en effet la totalité de données pour garantir des résultats corrects. L'agrégateur se base sur cette mémoire pour faire une agrégation globale.

4. La mémoire tampon pour le trieur (MTT) permet de faire un tri partiel. Elle a le même fonctionnement que la MTA. Elle pourrait être remplie par un EC ou un agrégateur (AD).
5. La mémoire tampon pour le collecteur de résultats (MTR) permet de stocker une partie des résultats finaux avant transmission à l'utilisateur. Ces résultats peuvent être produits par un EC, un agrégateur (AD) ou un trieur (TD).

Chaque mémoire tampon a une limite définie par l'administrateur du système. Nous autorisons le dépassement seulement pour la MTA et la MTT.

Nous intégrons aussi un opérateur spécial qui permet d'orchestrer l'exécution d'un plan. Ce composant prend en entrée un plan d'exécution et instancie les différents opérateurs et mémoires tampons.

L'algorithme 3 permet d'instancier les différents composants pour un plan donné. Nous instancions pour chaque ER un EC. Pour chaque EC nous créons et y associons une MTD et une MTC. Pour le dernier EC, nous ne crions pas de MTC. Pour chaque EC, nous associons la MTC crée par l'EC précèdent comme MT d'entrée. Si la requête contient une clause "Group By", nous créons un AD et une MTA. La

MTA sera considérée comme une sortie du dernier EC. Si la requête contient une clause "Order By", nous créons un TD et une MTT. La MTA sera considéré comme une sortie du AD si une clause "Group By" est exprimée sinon du dernier EC. Pour finir, nous créons un CR et une MTR. La MTR sera utilisée comme sortie du TD si la clause "Order By" est utilisée, comme sortie du AD s'il y a que la clause "Group By" qui est utilisée ou comme sotie du dernier EC s'il n'y a pas de clauses de type "Group By" et "Order By" dans la requête.

L'exemple dans la Figure 4.9 montre les différents composants créés pour le plan 4.7a. Nous avons commencé par créer un ED et un mappeur de dictionnaire. Ce plan est composé de trois étoiles de données. Nous avons créé donc trois EC et trois MTD. Pour les MTC, nous avons créé que deux, la première est utilisée comme sortie pour le premier EC et aussi comme entrée du deuxième EC. Le deuxième MTC est utilisé comme sortie pour le deuxième EC et entrée du troisième EC. La requête comporte aussi une clause Order By, nous avons donc créé un TD et une MTT qui sera utilisée comme sortie du dernier EC et entrée du TD. Pour finir, nous avons créé un RC et une MTR. La MTT est utilisé pour stocker les résultats du TD avant transmission à l'utilisateur finale.

L'orchestrateur n'assure pas que la création des composants nécessaires pour l'évaluation des plans, mais aussi la gestion de l'exécution des différents composants. Il est en mesure de décider à quel moment il faut exécuter un composant donné. L'orchestrateur se base sur les états (i.e., vide, pleine) des différentes mémoires tampons. Nous avons modélisé les différentes décisions sous forme d'arbre qui est présenté dans la Figure 4.8.

L'ensemble de décisions suivent quelques règles :

— Avant l'exécution d'un EC, nous essayons de charger le maximum d'étoiles de données en mémoire. Pour cela l'ED est sollicité pour remplir la MTD liée à ce EC.

— Un EC s'arrête de produire des correspondances si la mémoire en entrée est vide, la mémoire de données est vide ou la mémoire en sortie est remplie.

— Lorsque la mémoire tampon d'un EC est remplie, nous exécutons l'opérateur suivant sauf dans le cas d'un AD ou TD, qui nécessitent l'ensemble des données pour fonctionner.

— Si la mémoire de sortie d'un EC est assez remplie et ce EC n'a pas assez de donnée pour fonctionner, nous pouvons déclencher l'opérateur suivant.

— Si un EC n'a pas assez de données pour fonctionner et la mémoire de sortie n'est pas assez remplie, on laisse un EC précèdent s'exécuter.

— Un EC est marqué comme fini, si son MT en entrée est vide et que tout les EC précédents ont aussi fini.

Lorsque le dernier EC ne peut pas faire plus, nous examinons le type de l'opérateur suivant comme montré dans la Figure4.10. Dans le cas où il n'y a pas de AD et de TD, nous exécutons directement le RC. Nous donnons ensuite la main aux différents EC pour continuer. Les AD et TD sont déclenchés seulement si tous les EC ont fini.

L'algorithme 4 montre le traitement assuré par un EC, dont l'exécution dépend de l'emplacement de l'étoile de requêtes. S'il s'agit du premier EC, nous commençons par extraire une étoile de données à partir de la MTD liée à ce EC. Cette

Algorithm 3: Instatiation_Operateurs(Q,$Plan$)

Result: EC : liste d'EC, ED : extracteur de données, MTD : liste de MTD, MTC : liste de MTC, AD, MTA, TD, CR, MTR

```
1  ED ← Créer un nouveau ED ;
2  i=0 ;
3  for ER ∈ Plan do
4  |  EC[i] ← Créer un nouveau EC ;
5  |  MTD[i] ← Créer un nouveau MTD ;
6  |  EC[i].SetMTD( MTD[i]);
7  |  if i < Plan.nbEtoiles()-1 then
8  |  |  MTC[i] ← Créer un nouveau MTC ;
9  |  |  EC[i].SetMTCSortie(MTC[i]);
10 |  if i > 0 then
11 |  |  EC[i].SetMTCEntree(MTC[i-1]);
12 |  i++ ;
13 end
14 if Q.a_une_Clause_Groupe_By() then
15 |  AD ← Créer un nouveau AD ;
16 |  MTA ← Créer un nouveau MTA ;
17 |  AD.SetMTA(MTA) ; EC[i].SetMTSortie(MTA) ;
18 if Q.a_une_Clause_Order_By() then
19 |  TD ← Créer un nouveau TD ;
20 |  MTT ← Créer un nouveau MTT ;
21 |  TD.SetMTT(MTT);
22 |  if AD n'est pas instancié then
23 |  |  EC[i].SetMTSortie(MTT);
24 |  else
25 |  |  AD.SetMTSortie(MTT);
26 |  end
27 |  CR ← Créer un nouveau CR ;
28 |  MTR ← Créer un nouveau MTR ;
29 |  CR.SetMTR(MTR);
30 |  if TD est instancié then
31 |  |  TD.SetMT(MTR);
32 |  else
33 |  |  if AD est instancié then
34 |  |  |  AD.SetMT(MTR);
35 |  |  else
36 |  |  |  EC[i].SetMTSortie(MTR);
37 |  |  end
38 |  end
```

Algorithm 4: *EC[i].exec()*

```
1  if i == 0 then
2      while MTD[i] n'est pas vide et EC[i].MTSortie n'est pas remplie do
3          ed ← MTD[i].extract_ed_suivant() ;
4          W ← génère_Correspondnaces(ER[i],ed);
5          W ← Verifier_filtres(ER[i],W) ;
6          EC[i].MTSortie.Ajouter(W) ;
7      end
8  else
9      while MTD[i] et EC[i].MTEntree ne sont pas vides et EC[i].MTSortie
         n'est pas remplie do
10         ed ← MTD[i].extract_ed_suivant() ;
11         W ← EC[i].MTEntree.extract_Cor(ed.tête) ;
12         while w ∈ W do
13             W′ ← génère_Correspondnaces(ER[i],ed,w) ;
14             W′ ← Verifier_filtres(ER[i],W′) ;
15             EC[i].MTSortie.Ajouter(W′) ;
16         end
17     end
18 end
```

étoile de données est utilisée pour générer ensuite les correspondances par rapport à l'étoile de requêtes. Nous vérifions ensuite les correspondances par rapport à la clause FILTER. Seules les variables mentionnées dans cette clause sont considérées. Les correspondances valides sont ajoutées à la mémoire tampon en sortie. Pour les autres EC, nous considérons les correspondances existantes que nous récupérons à partir MTC en entrée. Nous générons seulement les correspondances compatibles avec les correspondances existantes. Comme pour le premier EC, nous gardons que les correspondances qui respectent la clause FILTER.

L'ED de son côté quand il est sollicité pour remplir la MTD d'un EC outre que le premier se base sur les correspondances qui entrent dans la MTC de ce EC. Lors de son exécution, il fait appel à l'algorithme 1. Dans le cas d'un premier EC avec une tête variable, il est nécessaire de faire une lecture complète des fragments compatible avec l'ER associée. En revanche, l'algorithme classique de l'arbre B+ est utilisé lorsqu'il s'agit de traiter une ER avec une tête constante.

4.2.4 Traitement des requêtes wildcard

Jusqu'ici, nous n'avons traité que des requêtes SPARQL avec des motifs basiques de graphes. Dans cette section, nous décrivons notre stratégie de traitement des requêtes avec la clause *FILTER*, utile pour exprimer des requêtes plus souples dans les applications réelles. Considérons la requête :

```
SELECT ?acteur WHERE
{ ?acteur <a_gagné>       <Oscar> .
  ?acteur <birthDate> ?date
```

```
FILTER(regex(str(?date),"1943") }
```

Dans ce cas, même si la date de naissance exacte de l'auteur recherché est inconnue, l'utilisateur est en mesure d'indiquer que l'auteur recherché est né en 1943.

Dans notre système, nous distinguons deux cas pour traiter cette clause sur chaque requête d'étoile individuelle. Tout d'abord, nous considérons une requête dans laquelle l'expression filter se trouve sur la *tête* de l'étoile de requêtes comme indiqué dans la Figure 4.12a. Le filtre est traité à la volée avant de charger les étoiles de données, comme expliqué dans la section précédente. À cette fin, le dictionnaire et l'arbre B + servent à filtrer les expressions régulières et les valeurs numériques, respectivement. Cela élimine un grand nombre de résultats intermédiaires à envoyer au prochain *EC* du plan.

Par contre, lorsque le filtre ne figure pas en tête de la requête en étoile (comme illustré dans la Figure 4.12b, les étoiles de données candidates sont toutes chargées et le filtrage s'applique par conséquent à l'ensemble des étoiles de données candidats.

4.2.5 Modules complémentaires

Après avoir présenté notre stratégie d'évaluation, nous discutons dans cette section des modules complémentaires que nous utilisons pour charger les données et optimiser le traitement. Notre flux de traitement est illustré dans la Figure 4.13.

"RDF_QDAG" comprend trois modules principaux : le chargeur de données, l'optimiseur de requêtes et le module d'évaluation. Dans cette section, nous donnons quelques détails sur les deux composants chargeur de données et optimiseur.

4.2.5.1 Chargement de données

Ce module est en charge du pré-traitement et de l'indexation des données. Il reçoit en entrée un fichier RDF et il est en charge de le fragmenter, le compresser et l'indexer. Ce module est composé de trois unités principales :

— *fragmenteur* : Ce composant est chargé du pré-traitement du fichier RDF d'entrée. Il crée d'abord les étoiles de données, puis les regroupe et les stocke dans des fichiers de fragments distincts. Les chaînes de caractères dans le fichier d'entrée sont indexées et les données sont compressées avec le format de stockage graphe que nous avons défini précédemment.

— *Indexeur* : Ce module reçoit en entrée les données pré-traitées envoyées par le fragmenteur. Ce composant crée les fichiers binaires codés et génère des indexes sous forme d'arbres B+.

— *Constructeur de dictionnaires* : Ce composant gère la création de l'index pour les chaînes de caractères.

4.2.5.2 Optimisation de requêtes

Le plan d'exécution de la requête est généré dans ce module. Ses principaux composants transforment la requête SPARQL en un plan physique d'exécution, qui sera envoyé à la couche d'exécution du système.

— *Parser* : ce sous-module analyse la requête SPARQL initiale et la transforme en une liste d'étoiles de requête (ER_f or ER_b).

— *Générateur de plan* : En utilisant des méthodes heuristiques, des plans d'exécution valides exprimés sous forme d'une séquence de requêtes en étoile sortante ou entrante (ER) sont générés. Deux plans d'exécution pour une requête spécifique sont illustrés dans la Figure 4.7.

— *Sélectionneur de plan* : Cette tâche est basée sur l'approche heuristique présentée dans [Tsialiamanis et al. (2012)]. Le sélecteur de plan envoie ensuite ce plan à la couche d'exécution du système. Nous nous basons sur un ensemble de règles pour ordonner les étoiles de requêtes. Après avoir généré toutes les étoiles de requêtes possibles pour un plan, nous les classons selon :

  — *Constantes* : Une étoile de requêtes avec des valeurs limitées doit être exécutée avant les étoiles qui contiennent que des nœuds variables. En outre, si la valeur limitée est la tête de l'étoile de requêtes, elle doit être exécutée en premier.

  — *Literals/URIs* : Une étoile de requêtes avec plus de littéraux/URI est plus sélective qu'une étoile avec moins de littéraux.

  — *Nombre d'arcs* : Une étoile de requêtes avec plus d'arêtes est plus sélective qu'une étoile de requêtes avec moins d'arêtes.

  — *Prioriser les étoile avec des arcs sortants* : En se basant sur nos propres expérimentations, ils ont moins de valeurs à explorer.

4.2.6 Exemple d'exécution

Dans cette section, nous développons un exemple du début à la fin en utilisant le graphe et la requête présentée dans la Figure 2.9. Supposons que l'utilisateur envoie une requête qui est d'abord analysée par le composant *Parser* du module optimiseur de requêtes. Ensuite, une série de plans d'exécution est générée par l'optimiseur de requêtes. Les requêtes étoiles sortantes et entrantes de chaque nœud sont présentées dans le Tableau 4.2. Deux plans d'exécution sont illustrés dans les Figures 4.7a et 4.7b. Le sélectionneur de plan sélectionne le meilleur plan à l'aide de l'heuristique que nous avons présentée. Une fois le meilleur plan sélectionné, il est envoyé au module d'exécution.

Le module d'exécution sélectionne un ensemble fragments de graphe *candidats* correspondant aux prédicats de l'étoile de requêtes. Pour cela, les prédicats sont examinés dans le treillis SPO et OPS (voir la Figure 4.2). En supposant que le plan 1 de la Figure 4.7a ait été sélectionné, nous illustrerons les fragments candidats pour chaque ER de la Figure 4.11. Les résultats de ER_{1f} se trouvent dans les fragments sortants du graphe dont les prédicats sont `genre` et `Vedette`. Par conséquent, ER_{1f} n'a qu'un fragment avec l'ID 1. ER_{2f} et ER_{3f} ont respectivement 1 et 2 graphe de fragments candidats. Le composant EC du système recherche les triplets correspondants pour chaque requête en étoile et les transfère comme illustré dans la Figure 4.11. Le transfert est effectué avec le modèle Volcano décrit dans la section précédente. Si la requête avait impliqué des agrégations, les résultats auraient été envoyés aux composants *Aggregator* et *Sorter* de l'exécuteur. Ensuite, les chaînes de caractères sont décodées dans le *Dictionary Matcher* puisque tout le processus de correspondance a été effectué avec les données compressées. Enfin, les résultats sont envoyés au *Results Writer* qui formate et renvoie à l'utilisateur la réponse à la requête dans la sortie souhaitée (par exemple, console, fichier).

Tout le processus d'exécution décrit précédemment est effectué à l'aide des données compressées et des chaînes codées. Toutefois, si la requête comporte des filtres ou si la tête du ER est connue, les chaînes sont décodées en supprimant les résultats intermédiaires non valides. Par exemple, le nœud *Drama* est examiné dans l'index du dictionnaire avant de trouver les correspondances de l'étoile de requêtes.

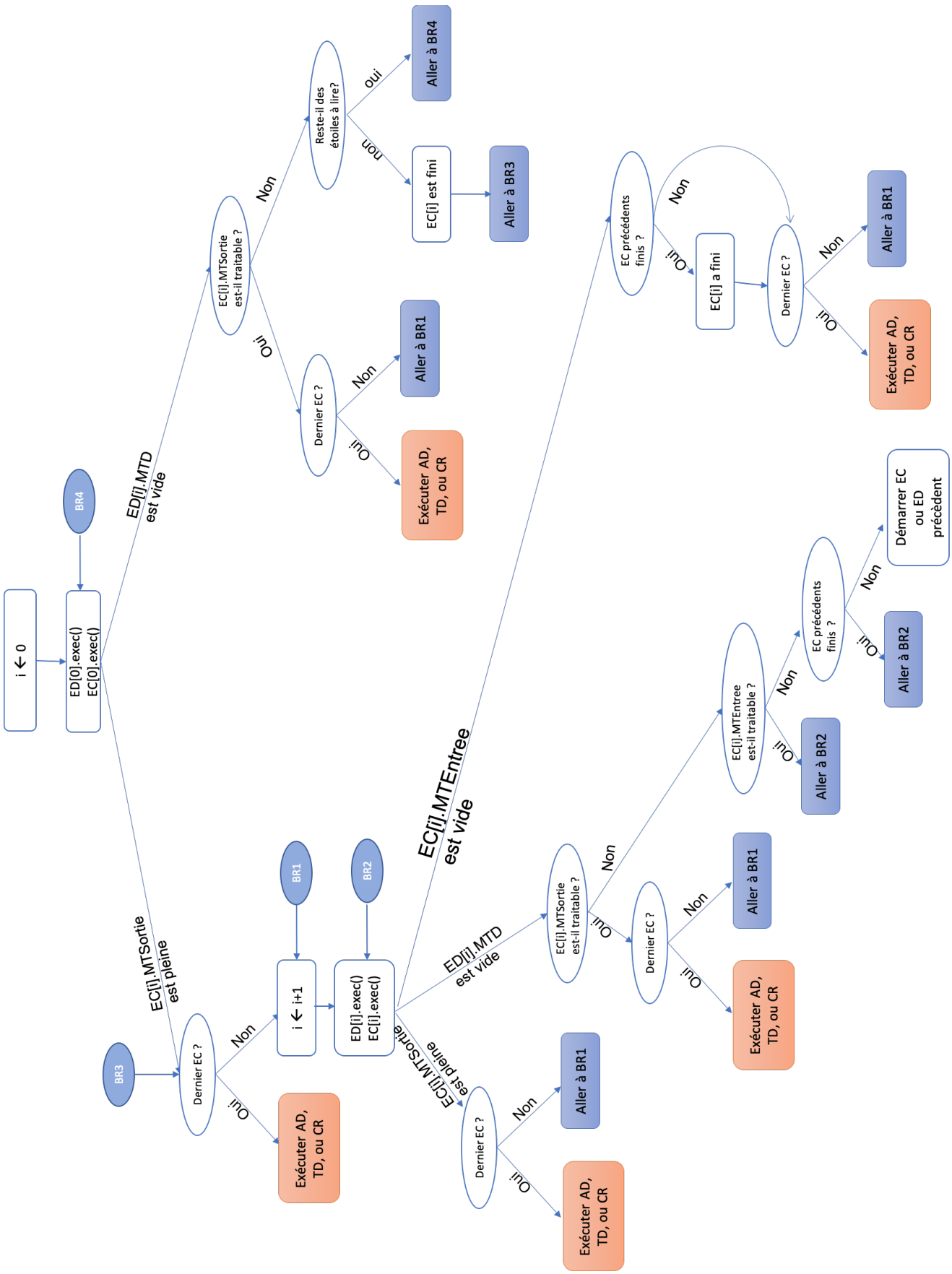


FIGURE 4.8 – Arbre de décisions

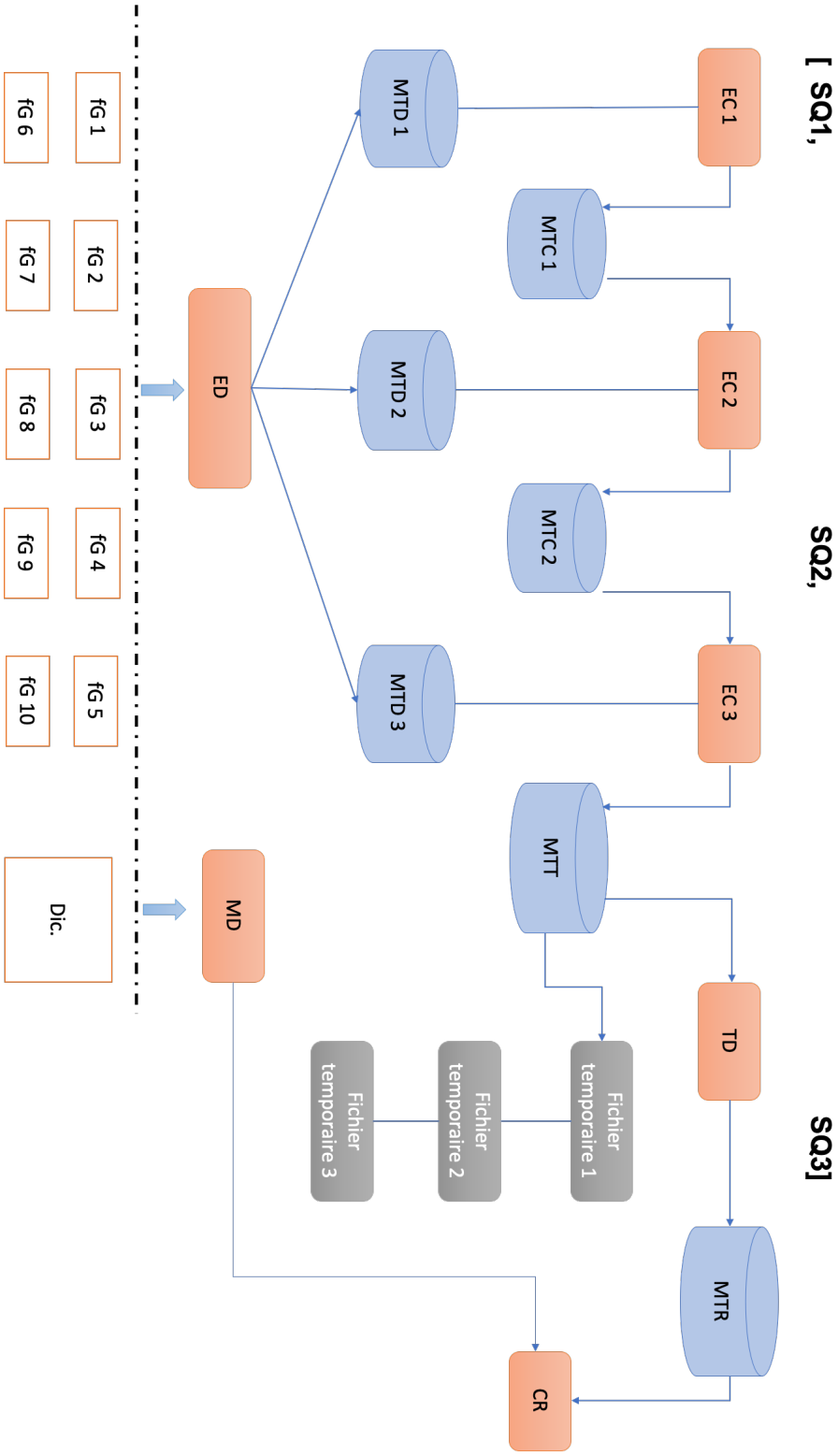


FIGURE 4.9 – Exemple d'un plan physique d'exécution

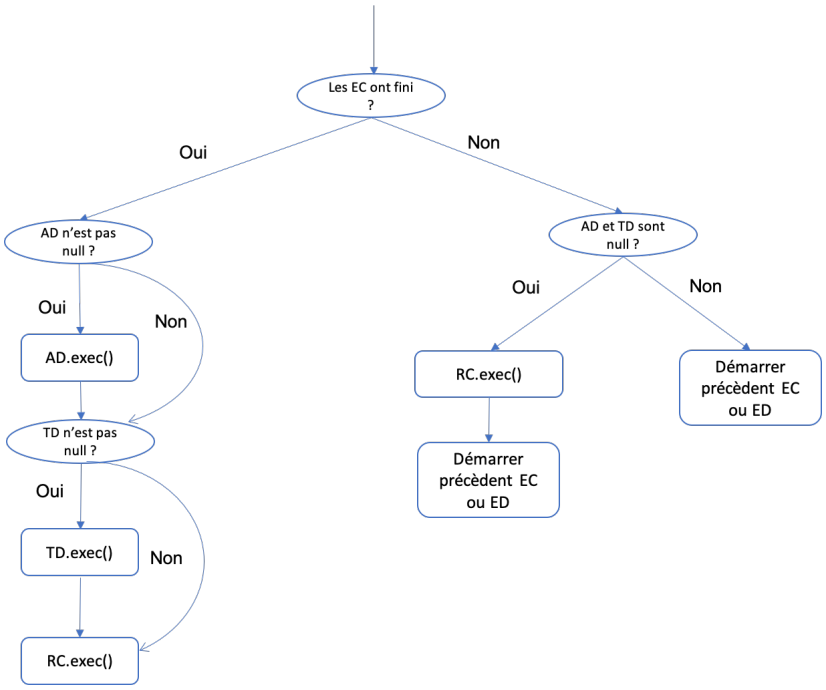


FIGURE 4.10 – Arbre de décisions (suite)

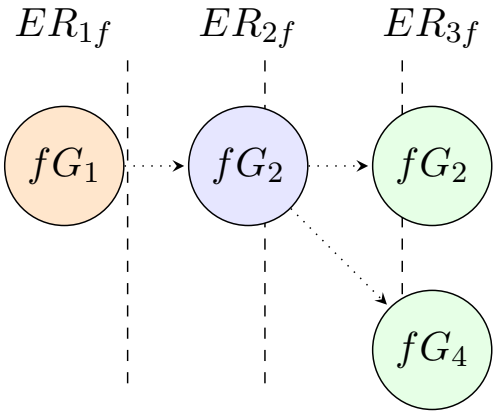
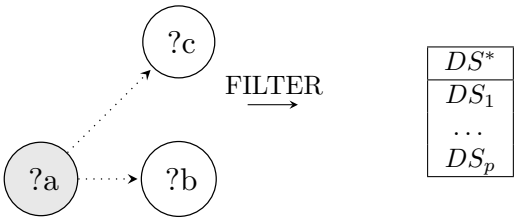
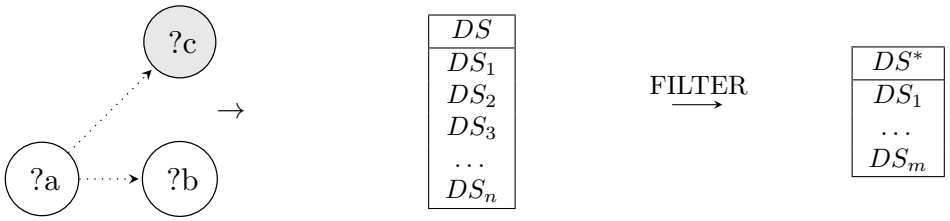


FIGURE 4.11 – Schéma d'exécution



(a) Filter sur la tête de l'Étoile en requêtes



(b) Filter pas sur la tête de l'Étoile en requêtes

FIGURE 4.12 – Traitement des filtres

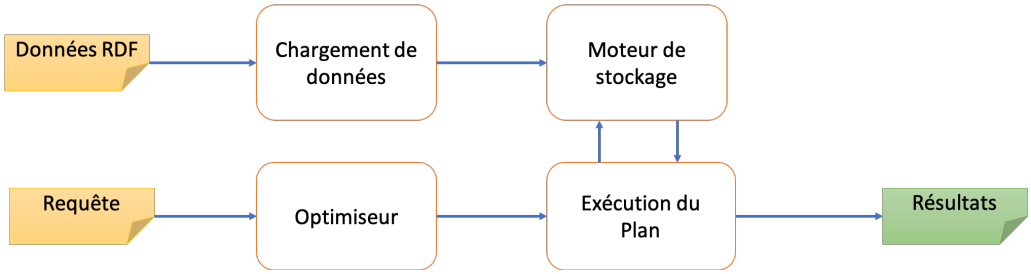


FIGURE 4.13 – Flux de traitement de requêtes et de données

4.3 Conclusion

Dans ce chapitre, nous avons commencé par présenter une cadre théorique d'évaluation de requêtes basé sur la fragmentation et l'exploration de graphes. Nous avons montré comment construire formellement des résultats d'une requêtes en se basant sur le mécanisme d'évaluation que nous proposons. Notre approche étend le mécanisme de Pérez et al. (2006), qui est basé sur les motifs de triplets, et permet d'exploiter la structure logique de graphes. Nous avons ensuite présenté les différentes techniques utilisées dans la mise en œuvre de "RDF_QDAG", notre nouveau système de gestion de données RDF. Notre objectif était de garantir non seulement le passage à l'échelle, qui est un problème crucial dans les systèmes existants lors du traitement de plusieurs milliards de triplets, mais également les performances (principalement le temps d'exécution) lorsqu' il s'agit de l'évaluation de requêtes en utilisant des configurations matérielles avec une mémoire principale limitée. Le chapitre suivant est consacré à la validation expérimentale de nos proposions par rapport à l'état de l'art en termes de gestion de données RDF.

Chapitre 5

L'approche RDF QDAG : Validation expérimentale

Sommaire

5.1 Introduction 129
5.2 Environnement d'expérimentations 129
5.3 Évaluation du pré-traitement 130
5.4 Performances des requêtes 130
5.4.1 Basic Graph Pattern queries 131
5.4.2 Aggregations 133
5.4.3 Requêtes Wildcard 133
5.5 RDF_QDAG vs. les systèmes parallèles 135
5.6 Conclusion 139

Après avoir détaillé le cadre formel de nos unités de travail représentant les données et les requêtes ainsi que les composantes de notre système, nous avons alors tous les ingrédients pour présenter nos résultats expérimentaux en utilisant des bancs d'essai existants. Pour évaluer la satisfaction des critères que nous avons imposés qui sont la performance des requêtes et le passage à l'échelle, notre système est comparé avec plusieurs systèmes existants, qui sont aussi discutés dans notre état de l'art. Ces résultats montrent également que notre démarche est effective.

Ce chapitre est composé de six sections : La Section 5.1 présente brièvement notre démarche expérimentale. La Section 5.2 décrit en détail l'environnement global de nos expérimentations qui intègre les logiciels utilisés, les systèmes utilisés pour la comparaison, les jeux de données, et les requêtes. La Section 5.3 présente le processus de pré-traitement lié aux chargements de données. La Section 5.4 présente nos résultats d'évaluation de l'ensemble de requêtes. La Section 5.5 compare notre système RDF_QDAG avec certains systèmes parallèles. Finalement la Section 5.6 conclut ce Chapitre.

5.1 Introduction

Nous avons évalué les performances de notre système sur des ensembles de données synthétiques (WatDiv et LUBM) et réelles (Yago et DBLP). Nous avons comparé les temps de chargement et d'exécution des requêtes SPARQL avec différentes configurations. Nous avons comparé notre système avec Virtuoso [Erling (2012)] (un système relationnel), RDF-3X [Neumann and Weikum (2008)] (un système d'indexation intensive) et gStore [Zou et al. (2014)] (un système à base de graphes). Nous n'avons pas comparé notre système avec des systèmes populaires basés sur des graphes comme Neo4J puisque même si les deux représentent des données sous forme de graphe, Neo4J ne supporte pas SPARQL. Nous avons également tenté de charger les données dans la version communautaire de GraphDB [1], mais il n'a pas été possible de charger des ensembles de données de plusieurs millions de triplets avec notre configuration matérielle.

Nous commençons à décrire notre environnement d'expérimentations et nos jeux de données. Ensuite, nous comparons les capacités du pré-traitement pour tous les systèmes. Enfin, nous évaluons les performances de tous les systèmes à l'aide de requêtes BGP (Basic Graph Pattern), d'agrégations et de requêtes Wildcard.

5.2 Environnement d'expérimentations

Matériel : Dans le cadre de nos expérimentations, nous avons effectué tous les tests sur une machine dotée d'un processeur Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz CPU, 1 Disque dur de 1 To et 32GB de mémoire. Ubuntu server 16.04 LTS est utilisé comme système d'exploitation.

Logiciels : Les principaux composants de notre approche nommé "RDF_QDAG" (*i.e.*, modules de fragmentation, d'allocation et d'indexation) sont implémentés en Java. L'extraction des étoiles de données est codée avec le langage C++.

Systèmes comparés : Notre système a été comparé à cinq approches de pointe qui appliquent différents paradigmes d'exécution. Tout d'abord un système d'exécution basé sur les graphe, nous avons effectué des tests sur la version centralisée gStore [2] et la version distribuée du système gStoreD, un système basé sur un SGBDR (Virtuoso) [3], une approche d'indexation intensive (RDF-3X), et un système basé sur le cloud fragmentant les données par prédicats (Cliquesquare).

Jeux de données : Nous évaluons et comparons les performances des systèmes à l'aide des jeux de données synthétiques (WatDiv et LUBM) et réelles (DBLP et Yago). Pour WatDiv, nous avons généré un jeu de données de 100 millions et de 1 milliard de triplets. Pour LUBM, nous avons généré 500 millions et 1 milliard de triplets, et ce afin de comparer les performances de tous les systèmes à grande échelle. Les détails de tous les jeux de données se trouvent dans le Tableau 5.1.

Requêtes : Nous comparons le temps d'exécution pour répondre aux requêtes que nous avons classées en trois groupes : BGP, Aggregation et Wildcard. Le premier considère le BGP avec quatre configurations (linéaire, Étoile, Flocon De Neige et complexe). Ce groupe de requêtes a été évalué dans tous les systèmes considérés.

1. `http://graphdb.ontotext.com/`
2. `https://github.com/pkumod/gStore`
3. `https://github.com/openlink/virtuoso-opensource`

Tableau 5.1 – Jeux de données expérimentaux

Jeu de données	Triplets (M)	Taille (GB)	# S (M)	# P (M)	# O (M)
WatDiv100M	109	15	5.21	86	9.76
WatDiv1B	1000	149	52.12	86	179.09
LUBM1B	1367	224	222.21	18	165.29
LUBM500M	500	83	81.38	17	60.54
Yago	284	42	10.12	98	52.37
DBLP	207	32	6.84	27	35.52

M : Millions, GB : Gigabytes

Le deuxième groupe de requêtes comprend des requêtes avec des fonctions d'agrégation et de tri. Enfin, le troisième groupe comprend des requêtes génériques (avec l'opérateur `FILTER`). Pour ces deux derniers groupes, nous avons comparé notre système avec Virtuoso uniquement puisque RDF-3X ne supporte pas ces types de requêtes.

La liste des requêtes se trouve en annexes de la thèse.

5.3 Évaluation du pré-traitement

Nous avons d'abord testé la capacité des systèmes à pré-traiter et à charger des jeux de données RDF bruts (au format N-Triples). "RDF_QDAG", Virtuoso et RDF-3X ont été en mesure de charger tous les jeux de données du benchmark WatDiv. gStoreD n'a pas pu charger les jeux de données dont la taille était supérieure à 15 Go, ce qui représente plus de 50% de la mémoire principale disponible. LUBM a été traitée avec succès par tous les systèmes à l'exception du milliard de triplés, qui n'a été pris en charge que par "RDF_QDAG". Les résultats pour chaque ensemble de données sont résumés dans le Tableau 5.2.

Tableau 5.2 – Jeux de données testés par les systèmes

Jeu de données	gStore	Rdf-3X	Virtuso	gStoreD	CS	QDAG
WatDiv100M	✓	▲	✓	▲	▲	✓
WatDiv1B	✘	▲	✓	▲	▲	✓
LUBM1B	✘	✘	✘	▲	▲	✓
LUBM500M	✘	▲	✓	▲	▲	✓
Yago	✘	▲	✓	▲	✘	✓
DBLP	✘	▲	✓	▲	▲	✓

✓ : Supporte toutes les requêtes, ▲ : Chargé avec succès mais ne supporte pas toutes les requêtes, ✘ : Incapable de charger, CS : Cliquesquare

5.4 Performances des requêtes

Dans cette section, nous montrons les résultats des évaluations de requête pour tous les ensembles de données et les systèmes. Nous organisons cette section comme

Tableau 5.3 – Résultats des requêtes BGP en secondes sur WatDiv100M

	C1	C2	C3	F1	F2	F3	F4	F5
gS	93.9	92.1	96.4	91.3	93.1	97.8	88.9	91.9
3X	12.9	**0.7**	66.1	**0.1**	0.7	0.8	1.5	0.01
V	13.3	2.2	69.3	0.4	**0.7**	**0.3**	**0.2**	0.3
QD	**1.4**	1.7	**11.1**	0.3	1.2	0.6	0.6	**0.01**

	L1	L2	L3	L4	L5	S1	S2	S6	S7
gS	93.8	94.4	92.3	93.3	95.2	91.2	89.2	85.0	93.8
3X	**0.1**	1.1	0.03	0.8	0.6	**2.0**	0.3	0.9	**14.8**
V	0.3	**0.1**	0.01	**0.3**	**0.1**	✘	**0.1**	**0.2**	20.3
QD	0.1	1.1	**0.01**	1.1	0.4	3.0	0.7	1.5	1.9

gS : gStore, 3X : RDF-3X, V : Virtuoso, QD : QDAG, ✘ : Error

suit. Tout d'abord, nous présentons les résultats pour les requêtes BGP pour tous les ensembles de données. Ensuite, nous montrons les résultats lorsque nous traitons des agrégations et finalement nous détaillons les performances de requête Wildcard pour les systèmes qui prennent en charge ce type de requêtes.

5.4.1 Basic Graph Pattern queries

Dans cette section, nous présentons les résultats de la requête par jeu de données. Les requêtes BGP sont supportées par tous les systèmes testés. Toutefois, lorsque les requêtes sont exécutées dans les bases de données de ces requêtes, elles sont exécutées dans des jeux de données représentant au moins la moitié de la mémoire principale du système, comme indiqué dans le Tableau 5.2. Tous les systèmes ne peuvent pas les traiter.

WatDiv : Nous avons utilisé cet ensemble de données pour évaluer les performances des requêtes pour les types linéaires (L), étoiles (S), Flocons de neige (F) et complexes (C). Les résultats pour les jeux de données de 100 millions et de 1 milliard sont présentés respectivement dans les tableaux 5.3 et 5.4. Pour WatDiv100M, RDF-3X, Virtuoso et "RDF_QDAG" ont des performances très similaires, bien meilleures que celles de gStore. Nous attribuons cette différence à la configuration de notre système dans laquelle la mémoire principale est très limitée. Notre système brille dans la résolution des requêtes pour le jeu de données d'un milliard de triplets, en particulier dans la solution de requêtes complexes (C) dans lesquelles notre système présente une amélioration des performances jusqu'à 7 fois supérieure (par exemple, C3 dans le Tableau 5.4). Ceci est prouvé avec les requêtes complexes et celles avec le type flocons de neige dans lesquelles notre système est en moyenne 1,6 fois plus rapide. Le comportement des mêmes requêtes dans un ensemble de données de 100 millions est très similaire, "RDF_QDAG" est capable de résoudre des requêtes beaucoup plus complexes qui sont à peine transformées en SQL avec une performance raisonnable.

LUBM : Comme pour WatDiv, nous avons évalué la capacité des systèmes à traiter des requêtes de différentes formes. gStore a été incapable de charger l'ensemble de données de 500 millions de triplés car il est supérieur à la mémoire principale

TABLEAU 5.4 – Résultats des requêtes BGP en secondes sur WatDiv1B

	C1	C2	C3	F1	F2	F3	F4	F5
3X	6818.2	16.1	694.2	**0.4**	14.0	12.7	10.3	1.3
V	152.4	111.9	921.8	4.7	56.2	10.4	8.0	4.6
Q	**21.8**	**13.4**	**94.8**	1.0	**2.0**	**3.7**	**1.4**	**0.3**

	L1	L2	L3	L4	L5	S1	S2	S6	S7
3X	0.4	56.1	0.5	21.0	**19.1**	173.7	6.0	11.6	267.0
V	33.7	**2.3**	30.8	33.1	37.6	E	**1.5**	1.8	697.5
QD	**0.2**	1652.0	**0.3**	**6.2**	36.0	**89.0**	16.7	**2.1**	**0.2**

gS : gStore, 3X : RDF-3X, V : Virtuoso, QD : QDAG, ✘ : Error

TABLEAU 5.5 – Résultats des requêtes BGP en secondes sur LUBM500M

	LB1	LB2	LB3	LB4	LB5	LB6	LB7
3X	0.1	388.4	0.1	0.1	0.14	553.4	0.1
QD	**0.01**	**151.9**	**0.01**	**0.03**	**0.10**	**550.0**	**0.03**
V	0.05	214.9	0.1	0.1	2.8	683.3	0.1

	LB8	LB9	LB10	LB11	LB12	LB13	LB14
3X	**0.2**	72.7	0.1	**0.4**	**0.2**	250.6	**533.1**
QD	4.0	**13.9**	**0.01**	3.6	3.4	**248.0**	550.0
V	6.7	27.0	0.7	3.8	4.6	326.6	796.9

3X : RDF-3X, V : Virtuoso, QD : QDAG

disponible. Les performances de requête de RDF-3X, de Virtuoso et de notre système sont présentées dans le Tableau 5.5. En moyenne, notre système a fonctionné jusqu'à 6 fois plus vite que Virtouoso et 3 fois plus vite que RDF-3X.

DBLP : Nous avons conçu des requêtes avec différentes formes comme il a été fait pour le jeu de données WatDiv (complexe, linéaire et étoile). Les résultats de cet ensemble de données sont présentés dans le Tableau 5.6. Pour ce jeu de données, les résultats variaient en fonction de la sélectivité de la requête. Pour les requêtes très sélectives, les trois systèmes ont eu une performance très similaire. Cependant, lorsque la requête n'était pas très sélective (comme c'est le cas dans de nombreuses applications réelles), "RDF_QDAG" surpasse les deux systèmes.

	DB_L1	DB_L2	DB_L3	DB_L4
3x	4226.4	0.4	**0.1**	**0.1**
V	✘	0.05	0.5	0.13
QD	**414.3**	**0.01**	45.9	0.01

	DB_S1	DB_S2	DB_S3	DB_S4	DB_S5
3x	2687.5	0.2	155.4	2.7	439.4
V	✘	2.3	210.8	88.3	405.9
QD	**165.7**	**0.007**	**139.5**	**1.2**	**354.5**

3x : RDF-3X, V : Virtuoso, QD : QDAG, ✘ : Error

TABLEAU 5.6 – Résultats des requêtes BGP en secondes (DBLP)

Yago2 : Les performances de RDF-3X, Virtuoso et "RDF_QDAG" sont illustrées dans la Figure 5.1. Nous montrons pour des raisons de lisibilité les temps logarithmiques pour chaque requête. "RDF_QDAG" a surpassé les trois systèmes pour la plupart des requêtes, en particulier les plus complexes qui ne sont pas très sélectifs.

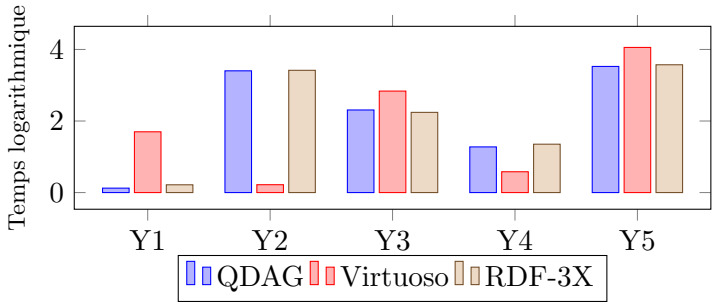


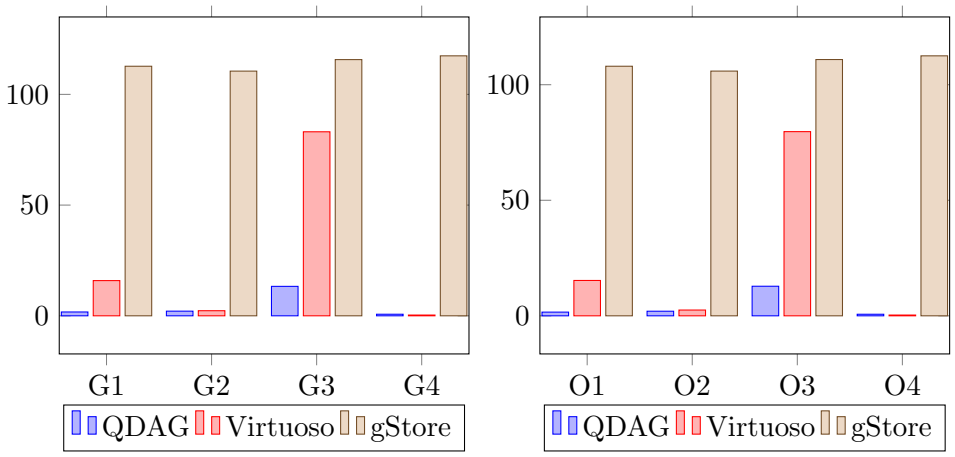
FIGURE 5.1 – Requêtes BGP sur Yago2 (Temps logarithmique)

5.4.2 Aggregations

WatDiv : Les résultats des requêtes d'agrégation (GroupBy, OrderBy) définis pour les jeux de données de 100 millions et d'un milliard sont présentés respectivement dans les figures5.2 et 5.3. RDF-3X n'a pas pu traiter ces types de requêtes. "RDF_QDAG" a surpassé Virtuoso pour toutes les requêtes de test. Le mécanisme de filtrage expliqué précédemment élimine efficacement les résultats intermédiaires et permet à notre système de résoudre ces requêtes en moyenne 4 fois plus rapidement que Virtuoso.

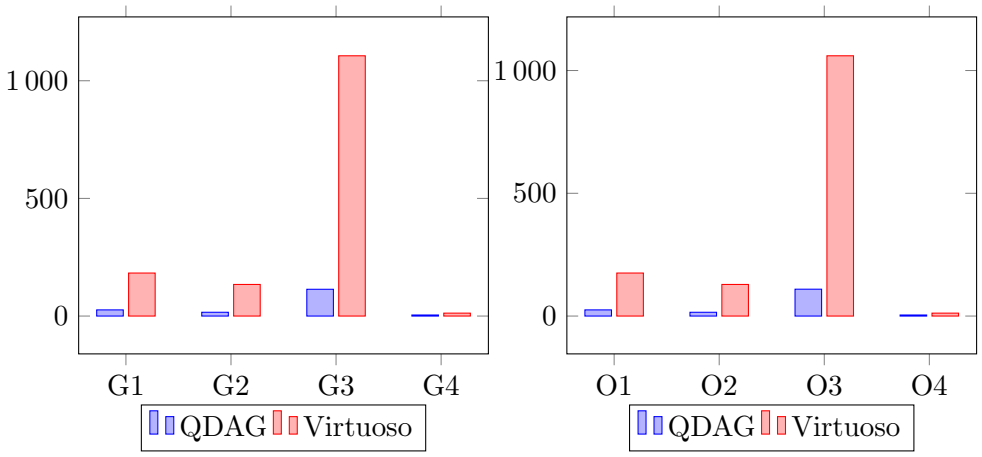
5.4.3 Requêtes Wildcard

WatDiv : Nous avons conçu 14 requêtes avec plusieurs expressions régulières. Nous avons réussi à créer des requêtes à la fois sélectives et non sélectives en utilisant le benchmark WatDiv (100 millions et 1 milliard de triplés). Les résultats de nos expériences sont présentés dans les tableaux 5.7 et 5.8. Virtuoso et "RDF_QDAG" ont eu une performance très similaire pour le jeu de données le plus petit. "RDF_-QDAG" a résolu plus efficacement les requêtes dans lesquelles le filtre n'est pas extrêmement sélectif, par exemple quand ce dernier porte sur une date comme c'est le cas dans les requêtes WC5 et WC9.



(a)Group By (b) Order By

FIGURE 5.2 – Résultat aggrégations - WatDiv100M



(a) Group By (b) Order By

FIGURE 5.3 – Résultat aggrégations - WatDiv1B

Tableau 5.7 – Requêtes Wildcards sur WatDiv100M (résultats en secondes)

	WC1	WC2	WC3	WC4	WC5	WC6	WC7
gS	92.3	92.6	97.1	88.1	92.5	92.3	78.4
V	**0.1**	**0.2**	**0.3**	**0.03**	0.003	**0.02**	**0.01**
QD	0.5	0.9	0.6	0.4	**0.001**	1.3	0.1

	WC8	WC9	WC10	WC11	WC12	WC13	WC14
gS	93.9	88.3	91.4	86.4	87.1	106.6	97.7
V	**0.01**	0.1	**0.2**	**0.001**	0.8	**0.2**	**0.4**
QD	1.1	**0.001**	1.6	0.1	**0.8**	1.5	0.6

gS : gStore, V : Virtuoso, QD : QDAG.

Tableau 5.8 – Requêtes Wildcards sur WatDiv1B (résultats en secondes)

	WC1	WC2	WC3	WC4	WC5	WC6	WC7
V	3.18	15.18	6.96	2.21	3.13	1.20	0.97
QD	**1.87**	**2.18**	**2.32**	**1.98**	**0.28**	**0.18**	**0.11**

	WC8	WC9	WC10	WC11	WC12	WC13	WC14
V	25.73	3.75	**0.61**	**0.04**	12.58	187.45	33.13
QD	**2.53**	**0.13**	2.08	0.14	**11.45**	**14.76**	**8.98**

V : Virtuoso, QD : QDAG.

5.5 RDF_QDAG vs. les systèmes parallèles

Pour voir le comportement de notre système vis à vis les systèmes parallèles, nous avons comparé notre système avec deux des systèmes les plus connus 1) la version distribuée de gStore (gStoreD) et un système basé sur le cloud fragmentant les données par prédicats (Cliquesquare version centralisée et distribuée).

LUBM : Comme vu précédemment la version centralisée de gStore n'a pas pu charger le jeu de données de 100 millions de triplets, nous avons généré un jeu de données de 20 millions de triplets pour comparer tous les systèmes avec ce benchmark. Les résultats des deux ensembles de données sont présentés dans la Figure 5.4 et la Figure 5.5 respectivement. Le système "RDF_QDAG" a surpassé tous les autres systèmes testés (versions centralisées ou distribuées). Même si Cliquesquare a pu évoluer les requêtes, il a souffert des frais généraux de MapReduce et du coût de démarrage du moteur MapReduce pour résoudre chaque requête.

WatDiv : WatDiv Les résultats présentés dans la Figure 5.6 montrent un comportement similaire à LUBM. Pour la plupart des requêtes, le temps d'exécution de "RDF_QDAG" est jusqu'à 60x moins que tous les autres systèmes. Dans les requêtes complexes avec de nombreux résultats (par exemple C2), gStoreD est efficace mais comme il est montré lors du traitement des ensembles de données réels, il ne s'adapte pas lorsque l'ensemble de données ne se trouve pas dans la mémoire principale.

Yago : Les résultats sont présentés à la Figure 5.7. Cliquesquare n'a pas pu traiter les requêtes testées. "RDF_QDAG" a surperformé gStoreD par presque 1000x.

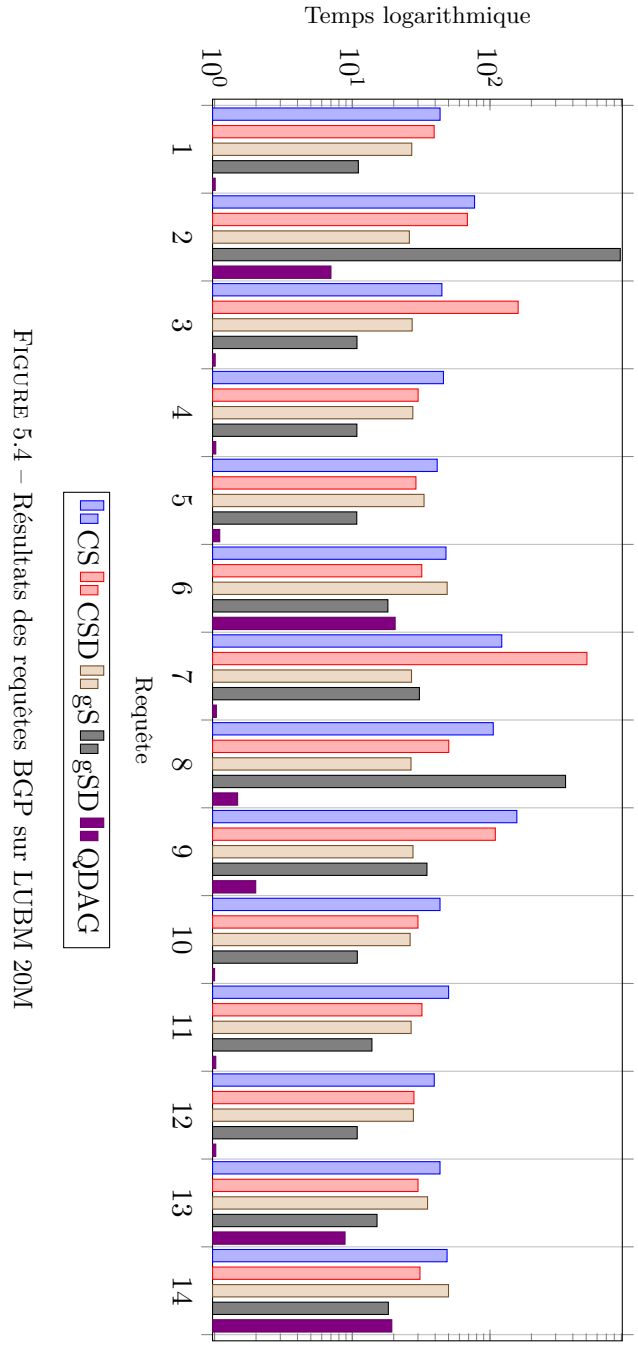


FIGURE 5.4 – Résultats des requêtes BGP sur LUBM 20M

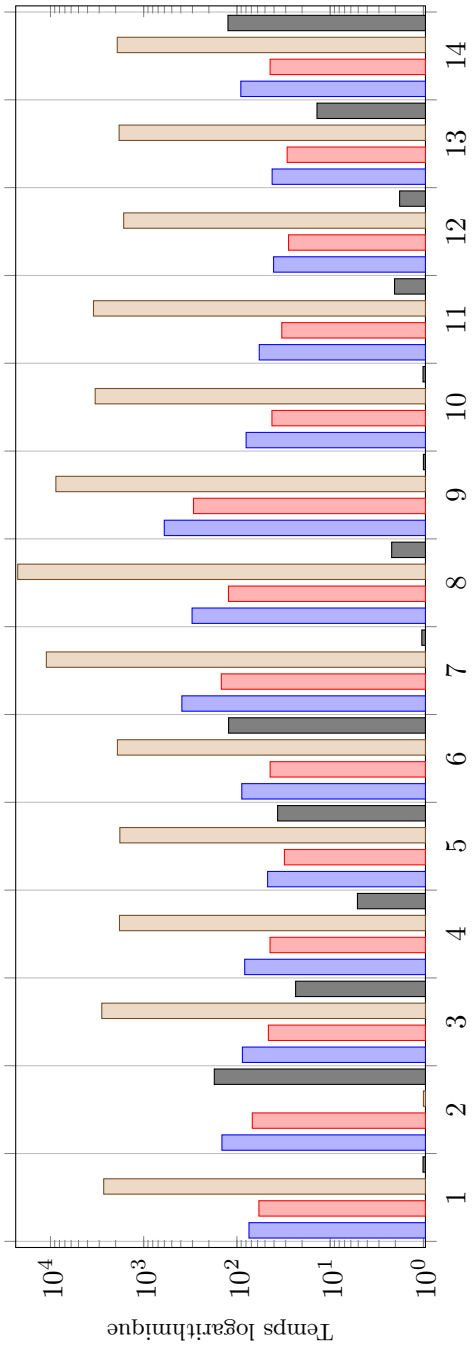


FIGURE 5.5 – Résultats des requêtes BGP sur LUBM100M

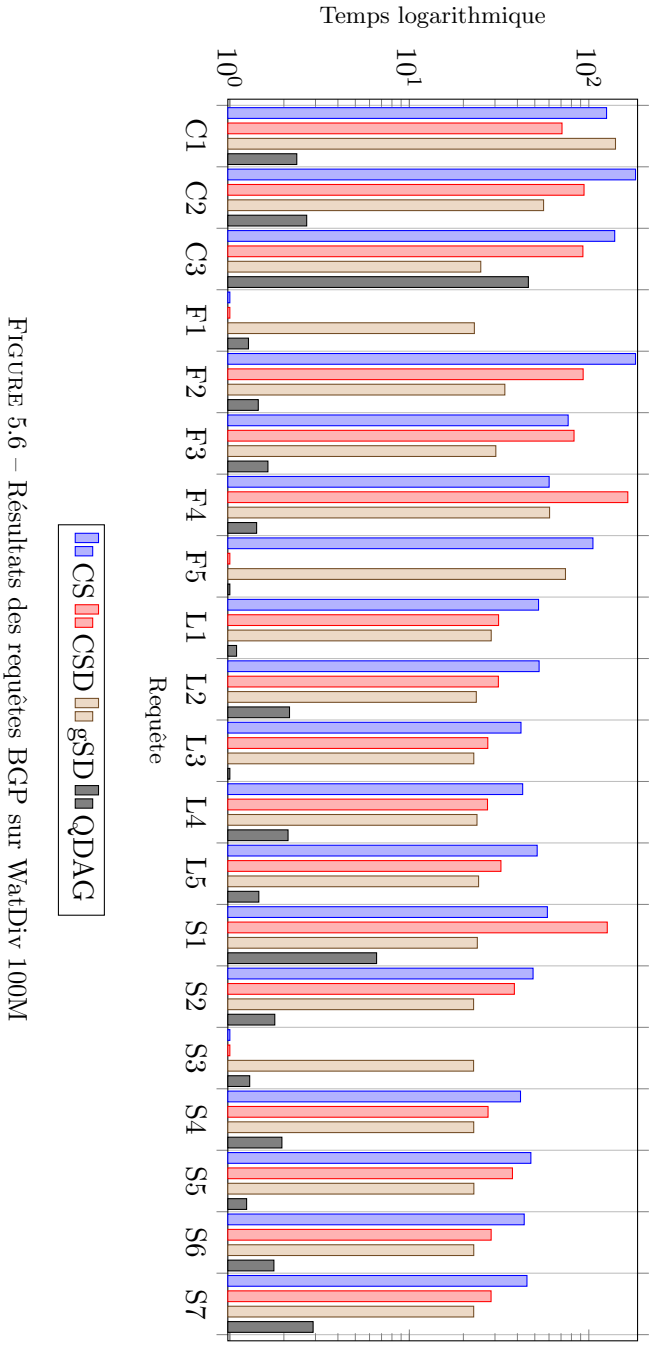


FIGURE 5.6 – Résultats des requêtes BGP sur WatDiv 100M

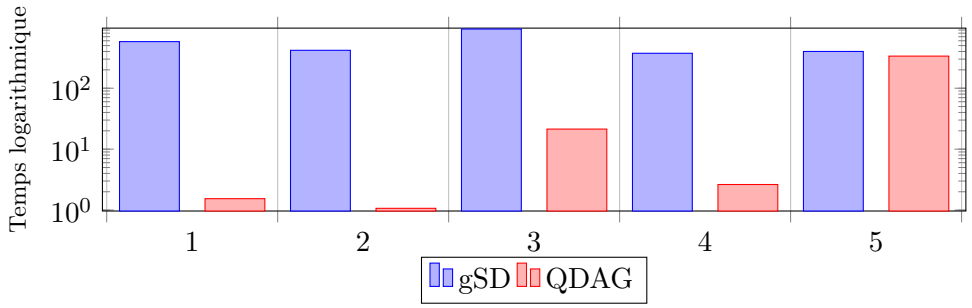


FIGURE 5.7 – Résultats de requêtes BGP sur YAGO

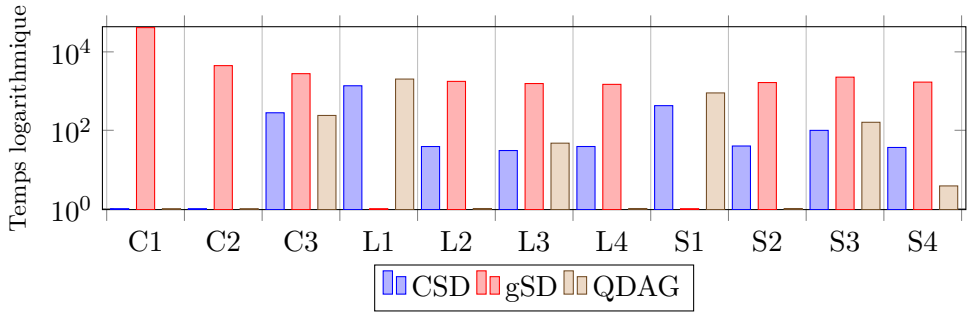


FIGURE 5.8 – Résultats des requêtes BGP (DBLP) avec systèmes distribués

DBLP : La version distribuée de gStore et Cliquesquare ainsi que "RDF_QDAG" ont pu traiter les requêtes sur cet ensemble de données. "RDF_QDAG" a réalisé en moyenne 500x mieux que tous les systèmes testés comme montre la Figure 5.8.

Dans tous les jeux de données testés, notre système "RDF_QDAG" a surpassé les systèmes distribuant les données avec une fonction de hachage et par des prédicats.

5.6 Conclusion

Dans la phase de validation expérimentale, nous avons considéré quatre types de requêtes couvrant la majorité du vocabulaire du langage SPARQL : BGP, filtres à base d'expressions régulières, agrégation et tri. Nous avons comparé notre approche avec cinq autres systèmes à la pointe de la technologie : l'approche relationnelle (Virtuoso), une approche orientée graphe (g-Store), une approche d'indexation intensive (RDF-3X) et deux systèmes parallèles (gStore-D et CliqueSquare). "RDF_QDAG" est le seul système qui a réussi à charger toutes les bases de données de test et à exécuter toutes les requêtes. Notre système surpasse les systèmes existants, en particulier lors de l'évaluation des requêtes dans des bases de données à grande échelle.

Troisième partie

Conclusion et perspectives

Chapitre 6

Conclusion & Perspectives

6.1 Conclusion

Dans cette thèse nous avons mis l'accent sur l'explosion spectaculaire des *Knowledge Graphs* et des Données Ouvertes Liées. Nous pouvons facilement annoncer que tout domaine traditionnel (la médecine, la banque, la biologie, l'agriculteur, l'énergie) a son *Knowledge Graph* ou ses données liées. D'autres domaines tels que l'industrie du *Luxe* et des *Produits de Beauté* ont affiché leur intérêt à ces sources de données pour promouvoir leurs produits (e.g., Zalando [1]). Récemment, l'Industrie 4.0 (I40) s'intéresse de plus en plus à ces sources de données pour optimiser les processus complexes manipulant d'énormes quantités de données provenant de différents fournisseurs, systèmes et plateformes. Les services offerts par les *Knowledge Graphs* et les Données Ouvertes Liées en termes de partage, de publication, et de collaboration ont motivé les organismes de standardisation de constituer des réseaux de normes et de standards existants. Ces derniers sont souvent intégrés dans les processus de fabrication dans la chaîne industrielle. Nous pouvons citer l'initiative I40KG (Standards Knowledge Graph) [2].

Cette situation a fait naître deux acteurs principaux : les *producteurs* et les *consommateurs* de ces données et connaissances. Tout producteur doit se soucier de la qualité de la base de données/connaissances qu'il produit, et assurer un accès facile et efficace aux utilisateurs. Deux types de consommateurs existent : (i) des utilisateurs lambda qui interrogent ces sources à l'aide des requêtes ou via des systèmes de questions réponses et (ii) des utilisateurs qui exploitent ces données pour enrichir des solutions (tels que les entrepôts de données, les systèmes de recommandation, etc.). Quel que soit le type des acteurs, le besoin de stockage et d'interrogation de ces données/connaissances devient un enjeu important.

Le langage RDF avec ses facilités est devenu l'un des outils les plus adaptés pour représenter les données. D'ailleurs, plusieurs formats ont été proposés pour échanger les données RDF. Les plus simples sont basés sur la notion de triplets tandis que les plus avancées s'appuient sur la théorie des graphes. Le niveau de maturité de SPARQL lui a permis de devenir le langage d'interrogation de ces données (SPARQL Endpoints).

Le stockage des données RDF n'a pas échappé à la loi imposée par le modèle

1. `https://www.youtube.com/watch?v=QkgAFKL26Vg`
2. `https://github.com/i40-Tools/I40KG`

relationnel. L'infrastructure de ce dernier a été quasiment réutilisée pour supporter les modèles de données qui sont nés après le modèle relationnel. Nous pouvons citer le cas des données Objets, XML, multidimensionnelles, etc. Les premières approches de gestion de données RDF se sont basées sur les systèmes de gestion de données relationnelles. Une table est utilisée pour stocker les triplets RDF et les requêtes SPARQL sont traduites en SQL. Ce type de systèmes, catégorisé comme non-natifs, engendre une dégradation de performances à cause de l'inadéquation de la représentation relationnelle aux données RDF. Pour remédier aux limites de ces systèmes, des nouvelles approches ont été proposées qui ont donné lieu aux systèmes natives qui prennent en compte les spécificités du modèle de graphe RDF. Les différentes propositions ont contribué aussi à l'évolution rapide des données RDF et ont permis de créer plusieurs jeux de données qui regroupent plusieurs milliards de triplets. RDF est devenu d'ailleurs la victime de son succès, où le besoin de gérer plusieurs milliards de triplets est devenu une demande classique des producteurs et consommateurs de ces données. En conséquence, nous avons rapidement constaté des difficultés des systèmes natifs et non natifs pour répondre à ce besoin.

La satisfaction de ce besoin (le passage à l'échelle des systèmes de stockage des données RDF) a été assurée par les développements des systèmes parallèles et plus particulièrement ceux basés sur l'architecture MPP. En adoptant une telle approche, les performances sont sacrifiées. En se basant sur ces constats, nous avons privilégié la piste de développement d'un système centralisé. Après la maîtrise de ces composantes, ses fonctionnalités, ses fondements mathématiques il serait envisageable par la suite de fournir une version parallèle. Cette vision est largement adaptée par les éditeurs traditionnels de systèmes de gestion et de stockage de données comme PostgreSQL.

Lorsque nous avons fixés ces objectifs, nous avons rapidement réalisé l'ampleur des systèmes de stockage des données RDF dans la littérature. Devant cette situation, il est indispensable d'analyser ces systèmes afin de confirmer nos indicateurs initialement définis. Nous avons alors suivi la méthode : *Tester, Analyser, Proposer* qui consiste à identifier les systèmes de stockage existants les plus utilisés et les tester en utilisant les mêmes jeux de données déployés sur les mêmes plateformes. Les résultats de ces campagnes de test sont alors analysés, et critiqués. Un point important que nous souhaitons souligner est les efforts considérables que nous avons déployés pour réaliser cette méthode. Sur la base de ces résultats, nous avons proposé dans cette thèse, le système "RDF_QDAG", un nouveau système pour gérer les données RDF à large échelle. Ce système revisite les composants de base liés au stockage et à l'évaluation de requêtes. "RDF_QDAG" est basé sur la fragmentation des graphes. Les triplets de graphes avec les mêmes caractéristiques sont regroupés ensemble, ce qui permet d'éviter certains parcours inutiles et la reconstruction de certains motifs de graphes en se basant sur des jointures. Nous gardons au mieux la structure logique du graphe, et ce en se basant sur des références entre les fragments. Cela nous permettra d'explorer l'intégralité des données en passant d'un fragment à un autre, et par conséquence profiter de la représentation des données sous forme de graphe. Nous avons deux types de fragments qui représentent respectivement les arcs entrants et les arcs sortant du graphe RDF. L'exploration peut donc se faire dans les deux sens.

Chaque fragment est indexé avec un arbre B+. Les pages feuilles de l'arbre sont

compressées en utilisant les caractéristiques des fragments. Malgré le fait que les triplets RDF soient enrichis par quelques informations liées à la navigation inter-fragments et aussi aux deux types d'arcs entrants et sortants, cette technique de compression permet de maîtriser l'espace de stockage. Nous avons aussi proposé une nouvelle stratégie pour accéder aux arbres B+, qui permet de factoriser les accès quand l'index est sollicité plusieurs fois durant l'évaluation de la requête. Cette technique permet de minimiser le coût de transfert Disque/Mémoire centrale.

Notre stratégie d'évaluation est basée sur l'exploration, guidée par la requête, du graphe de données. Chaque requête est décomposée en sous-requêtes en fonction des arcs entrant et sortant de chaque nœud. Chaque sous-requête pourrait être satisfaite par un ou plusieurs fragments. Nous proposons donc d'élaguer les fragments qui ne contribuent pas à la construction des résultats de la requête. Avec cette technique nous avons pu constater qu'une requête peut être évaluée de plusieurs manières en fonction des sous-requêtes choisies et leurs ordres. Nos expérimentations, ont montré que le choix de la stratégie d'évaluation pourrait impacter considérablement les performances. Nous avons défini la notion de plan d'exécution qui permet de capturer à la fois les sous-requêtes et leur ordre. Une requête pourrait par conséquence avoir plusieurs plans possibles. Nous proposons une approche permettant de choisir au mieux le plan d'exécution. Notre approche d'évaluation est basée sur le modèle Volcano, ce qui permet d'éviter de matérialiser sur disque les résultats intermédiaires liés au traitement de la partie BGP de la requête. Quand il s'agit de traiter des requêtes avec des opérateurs de tri et/ou de regroupement, nous sommes obligés dans certains cas de recourir à la matérialisation des données sur le disque. En effet, certains opérateurs nécessitent l'intégralité de données pour fonctionner. Quand il s'agit de traiter les requêtes avec des filtres, notre moteur d'évaluation s'adapte pour interroger le dictionnaire au fur et mesure afin de garantir la validité du résultat partiel obtenu.

Pour conclure nous avons proposé de revisiter plusieurs composants d'un système de gestion de données. Cela a permis d'améliorer plusieurs points mais dans l'ensemble, "RDF_QDAG" est le seul système centralisé permettant de garantir à la fois le passage à l'échelle et les performances. Nous avons d'ailleurs, mené plusieurs expérimentations à large échelle qui ont confirmé sa supériorité. "RDF_QDAG" a été confronté avec non seulement des systèmes centralisés mais aussi avec des systèmes parallèles. Nous avons utilisé plusieurs jeux de données réels et synthétiques, avec des jeux de données qui peuvent regrouper plusieurs milliards de triplets. Nous avons mesuré dans un premier temps la capacité de "RDF_QDAG" à supporter le chargement de données. Par rapport aux systèmes centralisés, notre système est le seul en mesure de charger tous les jeux de données. Il surpasse même certains systèmes parallèles comme gStore-D. Le deuxième point de comparaison est lié aux performances. Nous avons confirmé la supériorité de notre approche. Dès que les requêtes sont exécutées sur des bases avec un nombre important de triplets, "RDF_QDAG" devient le meilleur système à répondre au besoin lié aux performances. Nous avons également comparé "RDF_QDAG" avec quelques systèmes parallèles (GStore-D et CliqueSquare) qui nécessitent plus de ressources. Les résultats obtenus ont montré que notre système "RDF_QDAG" garantit un bon compromis entre le passage à l'échelle et la performance et ce pour plusieurs types de requêtes.

6.2 Perspectives

Les travaux présentés dans ce manuscrit laissent envisager de nombreuses perspectives. Certaines pistes sont en cours de développement tandis que d'autres sont envisagées à moyen termes.

— Actuellement, nous étudions de nouvelles stratégies afin d'aborder le problème lié au choix du plan d'exécution et ce en proposant des modèles de coûts adéquats. Nous travaillons sur des techniques d'estimations basée sur les fragments et leurs interactions. Deux coûts se dégagent, le premier est lié à la structure interne du fragment, ce qui permet d'estimer le coût d'entrées/sorties. Le défi majeur derrière ce problème consiste à trouver une bonne heuristique qui permet de trouver rapidement un plan d'exécution avec de bonnes performances. Une technique de type programmation dynamique combinée à des propriétés liées la structure graphe des requêtes permettrait d'élaguer certains plans que nous jugeons inutiles à évaluer. Vu la complexité du problème, il serait peut-être judicieux d'envisager une solution basée sur l'apprentissage automatique. Cette dernière serait basée sur l'analyse les traces d'exécutions des requêtes. Une technique de type *Deep Learning* serait envisageable si nous exploitions les traces des utilisateurs (consommateurs) disponibles sur "RDF_QDAG".

— Nous avons aussi pu constater que le nombre de fragments générés pouvait être très grand. On se retrouve quelques fois avec des fragments avec moins de 10 triplets surtout pour les fragments qui stockent les arcs entrants. Nous prévoyions donc de regrouper certains fragments qui partagent des caractéristiques communes. Nous pensons aussi à casser certains fragments qui ont une taille excessive. Cette stratégie serait combinée avec une technique de *clustering* afin de regrouper les fragments avec une forte connexion. Une telle organisation nous permettra de choisir les fragments en fonctions de plusieurs sous requêtes au lieu de considérer une indépendance entre les sous requêtes.

— L'inférence doit également être pris en compte. L'intégration des ontologies RDFS et OWL permettrait d'apporter des connaissances aux données RDF. Dans notre cas, nous pensons adapter le moteur d'évaluation pour réécrire les requêtes en fonctions des ontologies.

— Nous prévoyons aussi d'étendre "RDF_QDAG" pour supporter les données RDF spatio-temporelles. Cela nécessite d'intégrer de nouveaux opérateurs liés aux traitements spatio-temporelles. Nous devons également intégrer des indexes multi dimensionnelle (e.g. R-Tree) afin d'accélérer la recherche de données spatio-temporelles. Lors du choix du plan d'exécution, il sera important de considérer aussi ce type de données. L'exploration, considérée jusqu'à maintenant comme structurelle, doit évoluer pour devenir à la fois structurelle et spatio-temporelle.

— "RDF_QDAG" pourrait naturellement évoluer vers une version parallèle. En effet, sa nature qui est basée sur la fragmentation lui permettrait d'être facilement parallélisable. Nous travaillons actuellement sur de nouvelles techniques d'allocation de fragments qui permettrait de prendre en compte plusieurs contraintes liées aux ressources disponibles et la connectivité entre les fragments. Le système devrait aussi supporter la tolérance aux pannes. Nous

pensons assurer cette tâche au niveau fragment. Des stratégies de duplication devrait être aussi intégrées. Il serait d'ailleurs intéressant d'intégrer les contraintes sur la réplication lors de l'allocation. "RDF_QDAG" devrait intégrer une stratégie de gestion dynamique, et ce en s'adaptant aux requêtes exécutées. Nous réfléchissons à des stratégies de réallocation de fragments. L'idée est de donner au système la possibilité de réorganiser dynamiquement ses fragments sur plusieurs stratégies : dupliquer, déplacer, splitter et regrouper. Les problèmes de réallocations sont des problèmes combinatoires et nécessitent un effort dans la conception des heuristiques adaptées. Il sera aussi nécessaire d'intégrer une stratégie de transfert de résultats intermédiaires entre les machines. Le transfert réseau pourrait être un facteur important impactant sérieusement les performances. Cela est vrai quand plusieurs machines sollicitent le transfert de données intermédiaires en même temps. Nous réfléchissons donc à une stratégie pour ordonnancer les transferts tout en maximisant l'utilisation de la bande passante. L'optimiseur doit aussi s'adapter au coût réseau. Nous pouvons nous appuyer sur la connectivité des fragments. Le coût total de traitement comporte donc deux parties : une partie qui capture les accès disque et une autre qui capture le nombre de résultats transférés entre les fragments qui se situent sur des machines différentes.

Bibliographie

Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. (2007). Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd international conference on Very large data bases*, pages 411–422. VLDB Endowment. (Cité en page 20), (Cité en page 67), (Cité en page 83)

Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D. J., Rasin, A., and Silberschatz, A. (2009). Hadoopdb : An architectural hybrid of mapreduce and DBMS technologies for analytical workloads. *Proc. VLDB Endow.*, 2(1) :922–933. (Cité en page 81)

Agrawal, R. (1988). Alpha : An extension of relational algebra to express a class of recursive queries. *IEEE Trans. Software Eng.*, 14(7) :879–885. (Cité en page 50)

Aho, A. V. and Ullman, J. D. (1979). The universality of data retrieval languages. In *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979*, pages 110–120. (Cité en page 50)

Aït-Kaci, H., Boyer, R., Lincoln, P., and Nasr, R. (1989). Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(1) :115–146. (Cité en page 96)

Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., and Tolle, K. (2001). The ICS-FORTH rdfsuite : Managing voluminous RDF description bases. In Decker, S., Fensel, D. A., Sheth, A. P., and Staab, S., editors, *Proceedings of the Second International Workshop on the Semantic Web - SemWeb'2001, Hongkong, China, May 1, 2001*, volume 40 of *CEUR Workshop Proceedings*. CEUR-WS.org. (Cité en page 66)

Alkhateeb, F., Baget, J.-F., and Euzenat, J. (2007). Rdf with regular expressions. (Cité en page 50)

Atre, M., Srinivasan, J., and Hendler, J. A. (2008). Bitmat : A main-memory bit matrix of RDF triples for conjunctive triple pattern queries. In *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008), Karlsruhe, Germany, October 28,*. (Cité en page 22), (Cité en page 68), (Cité en page 70)

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). *Dbpedia : A nucleus for a web of open data*. Springer. (Cité en page 17)

Auer, S. and Lehmann, J. (2007). What have innsbruck and leipzig in common ? extracting semantics from wiki content. In Franconi, E., Kifer, M., and May, W., editors, *The Semantic Web : Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings*, volume 4519 of *Lecture Notes in Computer Science*, pages 503–517. Springer. (Cité en page 37)

Beckett, D. (2002). The design and implementation of the redland RDF application framework. *Computer Networks*, 39(5) :577–588. :journals/cn/Beckett02

Bellatreche, L., Missaoui, R., Necir, H., and Drias, H. (2007). Selection and pruning algorithms for bitmap index selection problem using data mining. In *9th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, pages 221–230. (Cité en page 64)

Bellomarini, L., Gottlob, G., Pieris, A., and Sallinger, E. (2017). Swift logic for big data and knowledge graphs. In Sierra, C., editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 2–10. ijcai.org. (Cité en page 17), (Cité en page 19)

Berkani, N., Bellatreche, L., Khouri, S., and Ordonez, C. (2019). Value-driven approach for designing extended data warehouses. In *Proceedings of the 21st International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT Joint Conference, DOLAP@EDBT/ICDT*. (Cité en page 19)

Berners-Lee, T., Hendler, J., and Lassila, O. (2001a). The semantic web. *Scientific american*, 284(5) :34–43. (Cité en page 18)

Berners-Lee, T., Hendler, J., and Lassila, O. (2001b). The semantic web : A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *ScientificAmerican.com*. (Cité en page 33)

Binzenhöfer, A., Staehle, D., and Henjes, R. (2005). On the stability of chord-based P2P systems. In *Proceedings of the Global Telecommunications Conference, 2005. GLOBECOM '05, St. Louis, Missouri, USA, 28 November - 2 December 2005*, page 5. IEEE. (Cité en page 77)

Bizer, C., Heath, T., and Berners-Lee, T. (2011). Linked data : The story so far. In *Semantic services, interoperability and web applications : emerging concepts*, pages 205–227. IGI Global. (Cité en page 18)

Bollacker, K. D., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase : a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250. (Cité en page 17)

Bornea, M. A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., and Bhattacharjee, B. (2013). Building an efficient rdf store over a relational

database. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 121–132. (Cité en page 96)

Brickley, D. (2004). Rdf vocabulary description language 1.0 : Rdf schema. *http ://www. w3. org/TR/rdf-schema/.* (Cité en page 18)

Bröcheler, M., Pugliese, A., and Subrahmanian, V. S. (2009). Dogma : A disk-oriented graph matching algorithm for rdf databases. In *International Semantic Web Conference*, pages 97–113. Springer. (Cité en page 21)

Broekstra, J. and Kampman, A. (2004). Serql : An rdf query and transformation language. (Cité en page 50)

Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame : A generic architecture for storing and querying RDF and RDF schema. In *The Semantic Web - ISWC,First International Semantic Web Conference, Italy, June 9-12*, pages 54–68. (Cité en page 20), (Cité en page 65)

Cai, M. and Frank, M. R. (2004). Rdfpeers : a scalable distributed RDF repository based on a structured peer-to-peer network. In Feldman, S. I., Uretsky, M., Najork, M., and Wills, C. E., editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 650–657. ACM. (Cité en page 22), (Cité en page 77)

Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., and Rosati, R. (2018). Ontology-based data access and integration. In *Encyclopedia of Database Systems, Second Edition.* (Cité en page 22)

Cerans, K., Barzdins, G., Liepins, R., Ovcinnikova, J., Rikacovs, S., and Sprogis, A. (2012). Graphical schema editing for stardog owl/rdf databases using owlgred/s. In *Proceedings of the Workshop on OWL : Experiences and Directions (OWLED).* :2012

Chamberlin, D. D., Astrahan, M. M., Blasgen, M. W., Gray, J. N., King, W. F., Lindsay, B. G., Lorie, R., Mehl, J. W., Price, T. G., Putzolu, F., et al. (1981). A history and evaluation of system r. *Communications of the ACM*, 24(10) :632–646. (Cité en page 59)

Chamberlin, D. D. and Boyce, R. F. (1974). Sequel : A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pages 249–264. ACM. (Cité en page 59)

Chaudhuri, S. (1998). An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43. ACM. (Cité en page 60)

Chen, A., Kao, Y.-F., Pong, M., Shak, D., Sharma, S., Vaishnav, J., and Zeller, H. (1993). Query processing in nonstop sql. *IEEE Data Eng. Bull.*, 16(4) :29–41. (Cité en page 59)

Chen, X., Chen, H., Zhang, N., and Zhang, S. (2015). Sparkrdf : Elastic discreted RDF graph processing engine with distributed memory. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2015, Singapore, December 6-9, 2015 - Volume I*, pages 292–300. IEEE Computer Society. (Cité en page 23), (Cité en page 82)

Chong, E. I., Das, S., Eadon, G., and Srinivasan, J. (2005). An efficient sql-based RDF querying scheme. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 1216–1227. :conf/vldb/ChongDES05

Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6) :377–387. (Cité en page 50), (Cité en page 59)

Consortium, W. W. W. et al. (2014). Rdf 1.1 concepts and abstract syntax. (Cité en page 39)

Cyganiak, R. (2005). A relational algebra for sparql. *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170*, page 35. (Cité en page 62)

Deppisch, U. (1986). S-tree : a dynamic balanced signature index for office retrieval. In *Proceedings of the 9th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 77–87. ACM. (Cité en page 21)

Djahandideh, B., Goasdoué, F., Kaoudi, Z., Manolescu, I., Quiané-Ruiz, J., and Zampetakis, S. (2015). Cliquesquare in action : Flat plans for massively parallel RDF queries. In Gehrke, J., Lehner, W., Shim, K., Cha, S. K., and Lohman, G. M., editors, *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1432–1435. IEEE Computer Society. (Cité en page 23), (Cité en page 81)

Djebri, A. E. A., Tettamanzi, A. G. B., and Gandon, F. (2019). Publishing uncertainty on the semantic web : Blurring the LOD bubbles. In *4th International Conference on Conceptual Structures (ICCS)*, pages 42–56. (Cité en page 34)

Erling, O. (2012). Virtuoso, a hybrid rdbms/graph column store. *IEEE Data Eng. Bull.*, 35(1) :3–8. (Cité en page 20), (Cité en page 64), (Cité en page 103), (Cité en page 129)

Fletcher, G. H. L. and Beck, P. W. (2009). Scalable indexing of RDF graphs for efficient join processing. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 1513–1516. (Cité en page 21), (Cité en page 71)

Foster, C. C. (1973). A generalization of avl trees. *Communications of the ACM*, 16(8) :513–517. (Cité en page 21)

Ghazouani, S., Mezni, H., and Slimani, Y. (2020). Bringing semantics to multicloud service compositions. *Softw. Pract. Exp.*, 50(4) :447–469. (Cité en page 19)

Goasdoué, F., Kaoudi, Z., Manolescu, I., Quiané-Ruiz, J., and Zampetakis, S. (2013). Cliquesquare : efficient hadoop-based rdf query processing. In *BDA'13-Journées de Bases de Données Avancées*. (Cité en page 25)

Goeta, S. (2016). *Instantiate data, instantiate publics : a sociological inquiry in the backrooms of open data*. Theses, Télécom ParisTech. (Cité en page 17)

Görlitz, O. and Staab, S. (2011). SPLENDID : SPARQL endpoint federation exploiting VOID descriptions. In *Proceedings of the Second International Workshop on Consuming Linked Data (COLD2011), Bonn, Germany, October 23, 2011*. (Cité en page 22), (Cité en page 79)

Graefe, G. (1994). Volcano - an extensible and parallel query evaluation system. *IEEE Trans. Knowl. Data Eng.*, 6(1) :120–135. :journals/Tkde/Graefe94

Graefe, G. et al. (2011). Foundations and trends® in databases. *Foundations and Trends® in Databases*, 3(4) :203–402. (Cité en page 21)

Gray, J. (1981). The transaction concept : virtues and limitations. In *Proceedings of the seventh international conference on Very Large Data Bases-Volume 7*, pages 144–154. VLDB Endowment. (Cité en page 59)

Guo, Y., Pan, Z., and Heflin, J. (2005). Lubm : A benchmark for owl knowledge base systems. *Web Semantics : Science, Services and Agents on the World Wide Web*, 3(2) :158–182. (Cité en page 86)

Haase, P., Broekstra, J., Eberhart, A., and Volz, R. (2004). A comparison of RDF query languages. In *The Semantic Web - ISWC 2004 : Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, pages 502–517. (Cité en page 50)

Harris, S. and Gibbins, N. (2003). 3store : Efficient bulk RDF storage. In *PSSS1 - Practical and Scalable Semantic Systems, Proceedings of the First International Workshop on Practical and Scalable Semantic Systems, Sanibel Island, Florida, USA, October 20, 2003*. :conf/psss/HarrisG03

Harris, S., Lamb, N., Shadbolt, N., et al. (2009). 4store : The design and implementation of a clustered rdf store. In *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pages 94–109. (Cité en page 18), (Cité en page 66)

Harris, S. and Shadbolt, N. (2005). SPARQL query processing with conventional relational database systems. In *Web Information Systems Engineering - WISE 2005 Workshops, WISE 2005 International Workshops, New York, NY, USA, November 20-22, 2005, Proceedings*, pages 235–244. (Cité en page 50)

Harth, A. and Decker, S. (2005). Optimized index structures for querying RDF from the web. In *Third Latin American Web Congress (LA-Web 2005), 1 October - 2 November 2005, Buenos Aires, Argentina*, pages 71–80. (Cité en page 21), (Cité en page 69)

Harth, A., Umbrich, J., Hogan, A., and Decker, S. (2007). YARS2 : A federated repository for querying graph structured data from the web. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, pages 211–224. (Cité en page 22), (Cité en page 84)

Heath, T. and Bizer, C. (2011). Linked data : Evolving the web into a global data space. *Synthesis lectures on the semantic web : theory and technology*, 1(1) :1–136. (Cité en page 18)

Hickson, I. (2013). Html microdata. *W3C Working Group Note*. (Cité en page 18)

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., Rudolph, S., et al. (2009). Owl 2 web ontology language primer. *W3C recommendation*, 27(1) :123. (Cité en page 18)

Hoßfeld, T., Oechsner, S., Tutschku, K., and Andersen, F. (2006). Evaluation of a pastry-based P2P overlay for supporting vertical handover. In *IEEE Wireless Communications and Networking Conference, WCNC 2006, 3-6 April 2006, Las Vegas, Nevada, USA*, pages 2291–2296. IEEE. (Cité en page 77)

Huang, J., Abadi, D. J., and Ren, K. (2011). Scalable SPARQL querying of large RDF graphs. *Proc. VLDB Endow.*, 4(11) :1123–1134. (Cité en page 23), (Cité en page 81)

Ibragimov, D., Hose, K., Pedersen, T. B., and Zimányi, E. (2015). Processing aggregate queries in a federation of SPARQL endpoints. In *12th European Semantic Web Conference (ESWC)*, pages 269–285. (Cité en page 22)

Ibragimov, D., Hose, K., Pedersen, T. B., and Zimányi, E. (2016). Optimizing aggregate SPARQL queries using materialized RDF views. In *15th International Semantic Web Conference (ISWC)*, pages 341–359. (Cité en page 23)

Jagadish, H. V. (1989). Incorporating hierarchy in a relational model of data. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, USA, May 31 - June 2, 1989*, pages 78–87. (Cité en page 50)

Janik, M. and Kochut, K. (2005). BRAHMS : A workbench RDF store and high performance memory system for semantic association discovery. In *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC, Galway, Ireland, November 6-10, 2005, Proceedings*, pages 431–445. (Cité en page 22), (Cité en page 70)

Janowicz, K., Hitzler, P., Adams, B., Kolas, D., and Vardeman, C. (2014). Five stars of linked data vocabulary use. *Semantic Web*, 5(3) :173–176. (Cité en page 16)

Karanikola, L. and Karali, I. (2018). Towards a dempster-shafer fuzzy description logic - handling imprecision in the semantic web. *IEEE Transactions on Fuzzy Systems*, 26(5) :3016–3026. (Cité en page 34)

Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., and Scholl, M. (2002). RQL : a declarative query language for RDF. In *Proceedings of the Eleventh International World Wide Web Conference, WWW 2002, May 7-11, 2002, Honolulu, Hawaii, USA*, pages 592–603. (Cité en page 50)

Kedar, S. (2007). *Database Management Systems*. Technical Publications. (Cité en page 59)

Knuth, D. (1997). *The Art of Computer Programming, volume 3 :* ■ *Sorting and Searching* ■. Addison-Wesley. (Cité en page 21)

Kolas, D., Emmons, I., and Dean, M. (2009). Efficient linked-list rdf indexing in parliament. (Cité en page 22), (Cité en page 71)

Lanasri, D., Khouri, S., Saidoune, R., Boudoukha, K., and Bellatreche, L. (2019). Crumbs4cube : Turning breadcrumbs into smart enriched data cubes. In *Proceedings of the ER Forum and Poster & Demos Session on Publishing Papers with CEUR-WS co-located with 38th International Conference on Conceptual Modeling (ER*, pages 128–132. (Cité en page 19)

Lassila, O. (2002). Taking the RDF model theory out for a spin. In *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, pages 307–317. (Cité en page 50)

Lassila, O., Swick, R. R., et al. (1998). Resource description framework (rdf) model and syntax specification. (Cité en page 18)

Lui, S. M. and Kwok, S. H. (2002). Interoperability of peer-to-peer file sharing protocols. *SIGecom Exch.*, 3(3) :25–33. (Cité en page 77)

Matono, A., Amagasa, T., Yoshikawa, M., and Uemura, S. (2005). A path-based relational RDF database. In *Database Technologies 2005, Proceedings of the Sixteenth Australasian Database Conference, ADC 2005, Newcastle, Australia, January 31st - February 3rd 2005*, pages 95–103. (Cité en page 50)

Matono, A., Pahlevi, S. M., and Kojima, I. (2006). Rdfcube : A p2p-based three-dimensional index for structural joins on distributed triple stores. In *Databases, Information Systems, and Peer-to-Peer Computing, International Workshops, DBISP2P 2005/2006, Trondheim, Norway, August 28-29, 2005, Seoul, Korea, September 11, 2006, Revised Selected Papers*, pages 323–330. (Cité en page 21), (Cité en page 78)

McBride, B. (2001). Jena : Implementing the rdf model and syntax specification. In *Proceedings of the Second International Conference on Semantic Web-Volume 40*, pages 23–28. CEUR-WS. org. (Cité en page 68)

McBride, B. (2002). Jena : A semantic web toolkit. *IEEE Internet computing*, (6) :55–59. (Cité en page 20), (Cité en page 65), (Cité en page 96)

McGlothlin, J. P. and Khan, L. R. (2009). RDFKB : efficient support for RDF inference queries and knowledge management. In Desai, B. C., Saccà, D., and Greco, S., editors, *International Database Engineering and Applications Symposium (IDEAS 2009), September 16-18, 2009, Cetraro, Calabria, Italy*, ACM International Conference Proceeding Series, pages 259–266. ACM. (Cité en page 20), (Cité en page 63)

McGoveran, D. (1999). *Guide to SYBASE and SQL Server*. Addison-Wesley Longman Publishing Co., Inc. (Cité en page 59)

Mutharaju, R., Sakr, S., Sala, A., and Hitzler, P. (2013). D-SPARQ : distributed, scalable and efficient RDF query engine. In Blomqvist, E. and Groza, T., editors, *Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013*, volume 1035 of *CEUR Workshop Proceedings*, pages 261–264. CEUR-WS.org. (Cité en page 84)

Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., and Risch, T. (2002). EDUTELLA : a P2P networking infrastructure based on RDF. In *Proceedings of the Eleventh International World Wide Web Conference, WWW 2002, May 7-11, 2002, Honolulu, Hawaii, USA*, pages 604–615. (Cité en page 77)

Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M. T., Brunkhorst, I., and Löser, A. (2003). Super-peer-based routing and clustering strategies for rdf-based peer-to-peer networks. In Hencsey, G., White, B., Chen, Y. R., Kovács, L., and Lawrence, S., editors, *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, pages 536–543. ACM. (Cité en page 77)

Neumann, T. and Moerkotte, G. (2011). Characteristic sets : Accurate cardinality estimation for rdf queries with multiple joins. In *Data Engineering (ICDE)*, pages 984–994. (Cité en page 95), (Cité en page 96), (Cité en page 104)

Neumann, T. and Weikum, G. (2008). Rdf-3x : a risc-style engine for rdf. *Proceedings of the VLDB Endowment*, 1(1) :647–659. (Cité en page 21), (Cité en page 69), (Cité en page 81), (Cité en page 88), (Cité en page 103), (Cité en page 107), (Cité en page 129)

Owens, A., Gibbins, N., and mc schraefel (2008). Effective benchmarking for rdf stores using synthetic data. Project report. Event Dates : 26/10/2008. (Cité en page 88)

Peng, P., Zou, L., Özsu, M. T., Chen, L., and Zhao, D. (2016). Processing SPARQL queries over distributed RDF graphs. *VLDB J.*, 25(2) :243–268. (Cité en page 82)

Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics and complexity of sparql. In *International semantic web conference*, volume 4273, pages 30–43. Springer. (Cité en page 99), (Cité en page 100), (Cité en page 125)

Pham, M., Passing, L., Erling, O., and Boncz, P. A. (2015). Deriving an emergent relational schema from RDF data. In Gangemi, A., Leonardi, S., and Panconesi, A., editors, *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 864–874. ACM. (Cité en page 95), (Cité en page 96)

Prud'hommeaux, E., Seaborne, A., et al. (2017). Sparql query language for rdf. w3c recommendation (2008). (Cité en page 18), (Cité en page 50)

Ramakrishnan, R. and Gehrke, J. (2000). Database management systems. (Cité en page 60)

Rohloff, K. and Schantz, R. E. (2011). Clause-iteration with mapreduce to scalably query datagraphs in the SHARD graph-store. In Kosar, T., editor, *DIDC'11, Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing, San Jose, CA, USA, June 8, 2011*, pages 35–44. ACM. (Cité en page 23), (Cité en page 80)

Schätzle, A., Przyjaciel-Zablocki, M., Berberich, T., and Lausen, G. (2015). S2X : graph-parallel querying of RDF with graphx. In Wang, F., Luo, G., Weng, C., Khan, A., Mitra, P., and Yu, C., editors, *Biomedical Data Management and Graph Online Querying - VLDB 2015 Workshops, Big-O(Q) and DMAH, Waikoloa, HI, USA, August 31 - September 4, 2015, Revised Selected Papers*, volume 9579 of *Lecture Notes in Computer Science*, pages 155–168. Springer. (Cité en page 83)

Schätzle, A., Przyjaciel-Zablocki, M., Skilevic, S., and Lausen, G. (2016). S2RDF : RDF querying with SPARQL on spark. *Proc. VLDB Endow.*, 9(10) :804–815. (Cité en page 23), (Cité en page 83)

Schwarte, A., Haase, P., Hose, K., Schenkel, R., and Schmidt, M. (2011). Fedx : Optimization techniques for federated query processing on linked data. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pages 601–616. (Cité en page 22), (Cité en page 79)

Seaborne, A. (2004). Rdql-a query language for rdf. *http ://www. w3. org/Submission/RDQL/.* (Cité en page 22), (Cité en page 50)

Singhal, A. (2012). Introducing the knowledge graph : things, not strings. *Official google blog*, 5. (Cité en page 17)

Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., and Katz, Y. (2007). Pellet : a practical owl-dl reasoner. *Web Semantics : Science, Services and Agents on the World Wide Web*, 5 :51–53. (Cité en page 73)

Stonebraker, M., Held, G., Wong, E., and Kreps, P. (1976). The design and implementation of ingres. *ACM Transactions on Database Systems (TODS)*, 1(3) :189–222. (Cité en page 59)

Stonebraker, M., Rowe, L. A., and Hirohama, M. (1990). The implementation of postgres. *IEEE Transactions on Knowledge & Data Engineering*, (1) :125–142. (Cité en page 59)

Sumathi, S. and Esakkirajan, S. (2007). *Fundamentals of relational database management systems*, volume 47. Springer. (Cité en page 59)

Thorsen, H. K. and Pattuelli, M. C. (2013). Designing a crowdsourcing tool to analyze relationships among jazz musicians : The case of linked jazz 52nd street. In *Human Computation and Crowdsourcing : Works in Progress and Demonstration Abstracts, An Adjunct to the Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing, November 7-9, 2013, Palm Springs, CA, USA*, volume WS-13-18 of *AAAI Workshops*. AAAI. (Cité en page 18)

Tran, T., Ladwig, G., and Rudolph, S. (2009). Istore : efficient rdf data management using structure indexes for general graph structured data. *Institute AIFB, Karlsruhe Institute of Technology.* (Cité en page 21), (Cité en page 71)

Tsatsanifos, G., Sacharidis, D., and Sellis, T. K. (2011). On enhancing scalability for distributed RDF/S stores. In Ailamaki, A., Amer-Yahia, S., Patel, J. M., Risch, T., Senellart, P., and Stoyanovich, J., editors, *EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, Proceedings*, pages 141–152. ACM. (Cité en page 22), (Cité en page 78)

Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., and Boncz, P. A. (2012). Heuristics-based query optimisation for SPARQL. In *15th EDBT, Berlin, Germany, March 27-30*, pages 324–335. :conf/edbt/TsialiamanisSFCB12

Udrea, O., Pugliese, A., and Subrahmanian, V. S. (2007). GRIN : A graph based RDF index. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 1465–1470. (Cité en page 21), (Cité en page 73)

Watson, M. (2009). *Scripting intelligence : Web 3.0 information gathering and processing.* Apress. (Cité en page 21)

Weiss, C., Karras, P., and Bernstein, A. (2008). Hexastore : sextuple indexing for semantic web data management. *Proceedings of VLDB*, 1(1) :1008–1019. (Cité en page 22), (Cité en page 68), (Cité en page 69), (Cité en page 103)

Wilkinson, K., Sayers, C., Kuno, H. A., and Reynolds, D. (2003). Efficient RDF storage and retrieval in jena2. In *Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Humboldt-Universität, Berlin, Germany, September 7-8*, pages 131–150. (Cité en page 20), (Cité en page 64)

Wood, D. (2005). Kowari : A platform for semantic web storage and analysis. In *In XTech 2005 Conference*, pages 05–0402. (Cité en page 21), (Cité en page 70)

Yoose, B. and Perkins, J. (2013). The linked open data landscape in libraries and beyond. *Journal of Library Metadata*, 13(2-3) :197–211. (Cité en page 19)

Yuan, P., Liu, P., Wu, B., Jin, H., Zhang, W., and Liu, L. (2013). Triplebit : a fast and compact system for large scale RDF data. *PVLDB*, 6(7) :517–528. (Cité en page 22), (Cité en page 72)

Zhang, X. and Yoshikawa, M. (2008). Dcbdquery - query language for rdf using dynamic concise bounded description. (Cité en page 50)

Zou, L., Mo, J., Chen, L., Özsu, M. T., and Zhao, D. (2011). gstore : answering sparql queries via subgraph matching. *Proceedings of the VLDB Endowment*, 4(8) :482–493. (Cité en page 21), (Cité en page 54), (Cité en page 72), (Cité en page 82), (Cité en page 103)

Zou, L., Özsu, M. T., Chen, L., Shen, X., Huang, R., and Zhao, D. (2014). gstore : a graph-based SPARQL query engine. *VLDB J.*, 23(4) :565–590. :journals/vldb/-ZouOCSHZ14

Annexe : Requêtes expérimentales

Requêtes Watdiv

C1.

```
SELECT ?v0 ?v4 ?v6 ?v7 WHERE
{
?v0 <http://schema.org/caption> ?v1 .
?v0 <http://schema.org/text> ?v2 .
?v0 <http://schema.org/contentRating> ?v3 .
?v0 <http://purl.org/stuff/rev#hasReview> ?v4 .
?v4 <http://purl.org/stuff/rev#title> ?v5 .
?v4 <http://purl.org/stuff/rev#reviewer> ?v6 .
?v7 <http://schema.org/actor> ?v6 .
?v7 <http://schema.org/language> ?v8 .
}
```

C2.

```
SELECT ?v0 ?v3 ?v4 ?v8 WHERE
{
?v0 <http://schema.org/legalName> ?v1 .
?v0 <http://purl.org/goodrelations/offers> ?v2 .
?v2<http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbm/Country5> .
?v2 <http://purl.org/goodrelations/includes> ?v3 .
?v4 <http://schema.org/jobTitle> ?v5 .
?v4 <http://xmlns.com/foaf/homepage> ?v6 .
?v4 <http://db.uwaterloo.ca/~galuc/wsdbm/makesPurchase> ?v7 .
?v7 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseFor> ?v3 .
?v3 <http://purl.org/stuff/rev#hasReview> ?v8 .
?v8 <http://purl.org/stuff/rev#totalVotes> ?v9 .
}
```

C3.

```
SELECT ?v0 WHERE
{
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v1 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/friendOf> ?v2 .
?v0 <http://purl.org/dc/terms/Location> ?v3 .
?v0 <http://xmlns.com/foaf/age> ?v4 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/gender> ?v5 .
?v0 <http://xmlns.com/foaf/givenName> ?v6 .
}
```

F1.

```
SELECT ?v0 ?v2 ?v3 ?v4 ?v5 WHERE
{
?v0 <http://ogp.me/ns#tag> <http://db.uwaterloo.ca/~galuc/wsdbm/Topic172> .
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v2 .
?v3 <http://schema.org/trailer> ?v4 .
?v3 <http://schema.org/keywords> ?v5 .
?v3 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre> ?v0 .
?v3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                <http://db.uwaterloo.ca/~galuc/wsdbm/ProductCategory2> .}
```

F2.

```
SELECT ?v0 ?v1 ?v2 ?v4 ?v5 ?v6 ?v7 WHERE
{
?v0 <http://xmlns.com/foaf/homepage> ?v1 .
?v0 <http://ogp.me/ns#title> ?v2 .
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v3 .
?v0 <http://schema.org/caption> ?v4 .
?v0 <http://schema.org/description> ?v5 .
?v1 <http://schema.org/url> ?v6 .
?v1 <http://db.uwaterloo.ca/~galuc/wsdbm/hits> ?v7 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre>
             <http://db.uwaterloo.ca/~galuc/wsdbm/SubGenre55> .}
```

F3.

```
SELECT ?v0 ?v1 ?v2 ?v4 ?v5 ?v6 WHERE
{
?v0 <http://schema.org/contentRating> ?v1 .
?v0 <http://schema.org/contentSize> ?v2 .
?v0<http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre>
           <http://db.uwaterloo.ca/~galuc/wsdbm/SubGenre42> .
?v4 <http://db.uwaterloo.ca/~galuc/wsdbm/makesPurchase> ?v5 .
?v5 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseDate> ?v6 .
?v5 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseFor> ?v0 .
}
```

F4.

```
SELECT ?v0 ?v1 ?v2 ?v4 ?v5 ?v6 ?v7 ?v8 WHERE
{
?v0 <http://xmlns.com/foaf/homepage> ?v1 .
?v2 <http://purl.org/goodrelations/includes> ?v0 .
?v0 <http://ogp.me/ns#tag> <http://db.uwaterloo.ca/~galuc/wsdbm/Topic71> .
?v0 <http://schema.org/description> ?v4 .
?v0 <http://schema.org/contentSize> ?v8 .
?v1 <http://schema.org/url> ?v5 .
?v1 <http://db.uwaterloo.ca/~galuc/wsdbm/hits> ?v6 .
?v1<http://schema.org/language> <http://db.uwaterloo.ca/~galuc/wsdbm/Language0> .
?v7 <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v0 .
}
```

F5.

```
SELECT ?v0 ?v1 ?v3 ?v4 ?v5 ?v6 WHERE
{
?v0 <http://purl.org/goodrelations/includes> ?v1 .
<http://db.uwaterloo.ca/~galuc/wsdbm/Retailer28001>
                <http://purl.org/goodrelations/offers> ?v0 .
?v0 <http://purl.org/goodrelations/price> ?v3 .
?v0 <http://purl.org/goodrelations/validThrough> ?v4 .
?v1 <http://ogp.me/ns#title> ?v5 .
?v1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v6 .
}
```

L1.

```
SELECT ?v0 ?v2 ?v3 WHERE
{
?v0<http://db.uwaterloo.ca/~galuc/wsdbm/subscribes>
            <http://db.uwaterloo.ca/~galuc/wsdbm/Website18627> .
?v2 <http://schema.org/caption> ?v3 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v2 .
}
```

L2.

```
SELECT ?v1 ?v2 WHERE
{
<http://db.uwaterloo.ca/~galuc/wsdbm/City107>
                    <http://www.geonames.org/ontology#parentCountry> ?v1 .
?v2<http://db.uwaterloo.ca/~galuc/wsdbm/likes>
                    <http://db.uwaterloo.ca/~galuc/wsdbm/Product0> .
?v2 <http://schema.org/nationality> ?v1 .
}
```

L3.

```
SELECT ?v0 ?v1 WHERE
{
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v1 .
?v0<http://db.uwaterloo.ca/~galuc/wsdbm/subscribes>
                <http://db.uwaterloo.ca/~galuc/wsdbm/Website50122> .
}
```

L4.

```
SELECT ?v0 ?v2 WHERE
{
?v0 <http://ogp.me/ns#tag> <http://db.uwaterloo.ca/~galuc/wsdbm/Topic132> .
?v0 <http://schema.org/caption> ?v2 .
}
```

L5.

```
SELECT ?v0 ?v1 ?v3 WHERE
{
?v0 <http://schema.org/jobTitle> ?v1 .
<http://db.uwaterloo.ca/~galuc/wsdbm/City24>
                <http://www.geonames.org/ontology#parentCountry> ?v3 .
?v0 <http://schema.org/nationality> ?v3 .
}
```

S1.

```
SELECT ?v0 ?v1 ?v3 ?v4 ?v5 ?v6 ?v7 ?v8 ?v9 WHERE
{
?v0 <http://purl.org/goodrelations/includes> ?v1 .
<http://db.uwaterloo.ca/~galuc/wsdbm/Retailer52462>
                    <http://purl.org/goodrelations/offers> ?v0 .
?v0 <http://purl.org/goodrelations/price> ?v3 .
?v0 <http://purl.org/goodrelations/serialNumber> ?v4 .
?v0 <http://purl.org/goodrelations/validFrom> ?v5 .
```

```
?v0 <http://purl.org/goodrelations/validThrough> ?v6 .
?v0 <http://schema.org/eligibleQuantity> ?v7 .
?v0 <http://schema.org/eligibleRegion> ?v8 .
?v0 <http://schema.org/priceValidUntil> ?v9 .
}
```

S2.

```
SELECT ?v0 ?v1 ?v3 WHERE
{
?v0 <http://purl.org/dc/terms/Location> ?v1 .
?v0 <http://schema.org/nationality>
               <http://db.uwaterloo.ca/~galuc/wsdbm/Country14> .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/gender> ?v3 .
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                           <http://db.uwaterloo.ca/~galuc/wsdbm/Role2> .
}
```

S3.

```
SELECT ?v0 ?v2 ?v3 ?v4 WHERE
{
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                        <http://db.uwaterloo.ca/~galuc/wsdbm/ProductCategory14> .
?v0 <http://schema.org/caption> ?v2 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre> ?v3 .
?v0 <http://schema.org/publisher> ?v4 .
}
```

S4.

```
SELECT ?v0 ?v2 ?v3 WHERE
{
?v0 <http://xmlns.com/foaf/age> <http://db.uwaterloo.ca/~galuc/wsdbm/AgeGroup1> .
?v0 <http://xmlns.com/foaf/familyName> ?v2 .
?v3 <http://purl.org/ontology/mo/artist> ?v0 .
?v0 <http://schema.org/nationality> <http://db.uwaterloo.ca/~galuc/wsdbm/Country1> .
}
```

S5.

```
SELECT ?v0 ?v2 ?v3 WHERE
{
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                    <http://db.uwaterloo.ca/~galuc/wsdbm/ProductCategory7> .
?v0 <http://schema.org/description> ?v2 .
?v0 <http://schema.org/keywords> ?v3 .
?v0 <http://schema.org/language> <http://db.uwaterloo.ca/~galuc/wsdbm/Language0> .
}
```

S6.

```
SELECT ?v0 ?v1 ?v2 WHERE
{
?v0 <http://purl.org/ontology/mo/conductor> ?v1 .
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v2 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre>
                    <http://db.uwaterloo.ca/~galuc/wsdbm/SubGenre83> .
}
```

S7.

```
SELECT ?v0 ?v1 ?v2 WHERE
{
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v1 .
?v0 <http://schema.org/text> ?v2 .
<http://db.uwaterloo.ca/~galuc/wsdbm/User145869>
                    <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v0 .
}
```

Requêtes LUBM

LB1.

```
SELECT ?x WHERE
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent>.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#takesCourse>
                        <http://www.Department0.University0.edu/GraduateCourse0>.
}
```

LB2.

```
SELECT ?x ?y where
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent>.
?y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                <http://swat.cse.lehigh.edu/onto/univ-bench.owl#University>.
?z <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Department>.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#memberOf> ?z.
?z <http://swat.cse.lehigh.edu/onto/univ-bench.owl#subOrganizationOf> ?y.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#undergraduateDegreeFrom> ?y
}
```

LB3.

```
SELECT ?x WHERE
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Publication>.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#publicationAuthor>
                <http://www.Department0.University0.edu/AssistantProfessor0>.
}
```

LB4.

```
SELECT ?x ?y1 ?y3 WHERE {
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#worksFor>
                            <http://www.Department0.University0.edu>.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#name> ?y1.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#emailAddress> ?y2.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#telephone> ?y3.
}
```

LB5.

```
SELECT ?x WHERE
{
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#memberOf>
                <http://www.Department0.University0.edu>.
}
```

LB6.

```
SELECT ?x WHERE
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#UndergraduateStudent>.
}
```

LB7.

```
SELECT ?x ?y WHERE
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#UndergraduateStudent>.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#takesCourse> ?y.
?y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                    <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Course>.
<http://www.Department0.University0.edu/AssociateProfessor0>
                    <http://swat.cse.lehigh.edu/onto/univ-bench.owl#teacherOf> ?y.
}
```

LB8.

```
SELECT ?x ?y WHERE
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#UndergraduateStudent>.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#memberOf> ?y.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#emailAddress> ?z.
?y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Department>.
?y <http://swat.cse.lehigh.edu/onto/univ-bench.owl#subOrganizationOf>
            <http://www.University0.edu>.
}
```

LB9.

```
SELECT ?x ?y WHERE
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#UndergraduateStudent>.
?y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#FullProfessor>.
?z <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Course>.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#advisor> ?y.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#takesCourse> ?z.
?y <http://swat.cse.lehigh.edu/onto/univ-bench.owl#teacherOf> ?z.
}
```

LB10.

```
SELECT ?x WHERE
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#GraduateStudent>.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#takesCourse>
            <http://www.Department0.University0.edu/GraduateCourse0>.
}
```

LB11.

```
SELECT ?x ?y WHERE
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#ResearchGroup>.
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#subOrganizationOf> ?y.
?y <http://swat.cse.lehigh.edu/onto/univ-bench.owl#subOrganizationOf>
                    <http://www.University0.edu>.
}
```

LB12.

```
SELECT ?x ?y WHERE
{
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#headOf> ?y.
?y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#Department>.
?y <http://swat.cse.lehigh.edu/onto/univ-bench.owl#subOrganizationOf>
            <http://www.University0.edu>.
}
```

LB13.

```
SELECT ?x WHERE
{
?x <http://swat.cse.lehigh.edu/onto/univ-bench.owl#undergraduateDegreeFrom>
                <http://www.University0.edu>.
}
```

LB14.

```
SELECT ?x WHERE
{
?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
            <http://swat.cse.lehigh.edu/onto/univ-bench.owl#UndergraduateStudent>.
}
```

Requêtes DBLP

DBL1.

```
SELECT ?v0 ?v1 ?v2 ?v3 WHERE
{
?v0 <http://purl.org/dc/elements/1.1/creator> ?v1 .
?v0 <http://xmlns.com/foaf/0.1/maker> ?v2 .
?v0 <http://xmlns.com/foaf/0.1/homepage> ?v3 .
}
```

DBL2.

```
SELECT ?v0 ?v1 WHERE
{
?v0 <http://purl.org/dc/terms/tableOfContent> ?v1 .
?v0 <http://purl.org/dc/terms/bibliographicCitation>
            <http://dblp.uni-trier.de/rec/bibtex/series/cogtech/Helbig2006> .
}
```

DBL3.

```
SELECT ?v0 ?v1 ?v2 ?v3 WHERE
{
?v0 <http://swrc.ontoware.org/ontology#month> ?v1 .
?v2 <http://purl.org/dc/elements/1.1/subject> ?v1 .
?v3 <http://swrc.ontoware.org/ontology#isbn> ?v1 .
}
```

DBL4.

```
SELECT ?v0 ?v1 WHERE
{
?v0 <http://www.w3.org/2000/01/rdf-schema#seeAlso>
            <http://dblp.l3s.de/d2r/resource/publications/books/ph/Shasha92> .
?v0 <http://purl.org/dc/terms/tableOfContent> ?v1 .
}
```

DBS1.

```
SELECT ?v0 ?v1 ?v2 ?v5 WHERE
{
?v0 <http://purl.org/dc/elements/1.1/creator> ?v1 .
?v0 <http://xmlns.com/foaf/0.1/maker> ?v2 .
?v0 <http://swrc.ontoware.org/ontology#journal> ?v3 .
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v4 .
?v0 <http://www.w3.org/2000/01/rdf-schema#label> ?v5 .
}
```

DBS2.

```
SELECT ?v0 ?v4 ?v6 WHERE
{
?v0 <http://purl.org/dc/terms/tableOfContent> ?v1 .
?v0 <http://swrc.ontoware.org/ontology#editor>
            <http://dblp.l3s.de/d2r/resource/authors/Rodney_W._Topor> .
?v0 <http://swrc.ontoware.org/ontology#number> ?v3 .
?v0 <http://www.w3.org/2000/01/rdf-schema#label> ?v4 .
?v0 <http://purl.org/dc/terms/issued> ?v5 .
?v0 <http://purl.org/dc/terms/bibliographicCitation> ?v6 .
}
```

DBS3.

```
SELECT ?v0 ?v3 ?v1 WHERE
{
?v0 <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?v1 .
?v3 <http://xmlns.com/foaf/0.1/homepage> ?v1 .
}
```

DBS4.

```
SELECT ?v0 ?v1 ?v3 WHERE
{
?v0 <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?v1 .
?v3 <http://xmlns.com/foaf/0.1/page> ?v1 .
}
```

DBS5.

```
SELECT ?v0 ?v1 ?v2 WHERE
{
?v0 <http://purl.org/dc/elements/1.1/title> ?v1 .
?v0 <http://www.w3.org/2002/07/owl#sameAs> ?v3 .
?v0 <http://purl.org/dc/terms/bibliographicCitation> ?v2 .
}
```

Requêtes Grouping and Sorting

G1.

```
SELECT ?v6 COUNT(?v0) WHERE
{
?v0 <http://schema.org/caption> ?v1 .
?v0 <http://schema.org/text> ?v2 .
?v0 <http://schema.org/contentRating> ?v3 .
?v0 <http://purl.org/stuff/rev#hasReview> ?v4 .
?v4 <http://purl.org/stuff/rev#title> ?v5 .
?v4 <http://purl.org/stuff/rev#reviewer> ?v6 .
?v7 <http://schema.org/actor> ?v6 .
?v7 <http://schema.org/language> ?v8 .
}
GROUP BY ?v6;
```

G2.

```
SELECT ?v0 AVG(?v8) WHERE
{
?v0 <http://schema.org/legalName> ?v1 .
?v0 <http://purl.org/goodrelations/offers> ?v2 .
?v2 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbm/Country5> .
?v2 <http://purl.org/goodrelations/includes> ?v3 .
?v4 <http://schema.org/jobTitle> ?v5 .
?v4 <http://xmlns.com/foaf/homepage> ?v6 .
?v4 <http://db.uwaterloo.ca/~galuc/wsdbm/makesPurchase> ?v7 .
?v7 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseFor> ?v3 .
?v3 <http://purl.org/stuff/rev#hasReview> ?v8 .
?v8 <http://purl.org/stuff/rev#totalVotes> ?v9 .
}
GROUP BY ?v0
```

G3.

```
SELECT ?v3 COUNT(?v0) WHERE
{
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v1 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/friendOf> ?v2 .
?v0 <http://purl.org/dc/terms/Location> ?v3 .
?v0 <http://xmlns.com/foaf/age> ?v4 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/gender> ?v5 .
?v0 <http://xmlns.com/foaf/givenName> ?v6 .
}
GROUP BY ?v3
```

G4.

```
SELECT ?v6 COUNT(?v5) WHERE
 {
?v0 <http://schema.org/contentRating> ?v1 .
?v0 <http://schema.org/contentSize> ?v2 .
```

```
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre>
                <http://db.uwaterloo.ca/~galuc/wsdbm/SubGenre42> .
?v4 <http://db.uwaterloo.ca/~galuc/wsdbm/makesPurchase> ?v5 .
?v5 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseDate> ?v6 .
?v5 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseFor> ?v0 .
}
GROUP BY ?v6
```

O1.

```
SELECT ?v0 ?v4 ?v6 ?v7 ?v3 WHERE
{
?v0 <http://schema.org/caption> ?v1 .
?v0 <http://schema.org/text> ?v2 .
?v0 <http://schema.org/contentRating> ?v3 .
?v0 <http://purl.org/stuff/rev#hasReview> ?v4 .
?v4 <http://purl.org/stuff/rev#title> ?v5 .
?v4 <http://purl.org/stuff/rev#reviewer> ?v6 .
?v7 <http://schema.org/actor> ?v6 .
?v7 <http://schema.org/language> ?v8 .
}
ORDER BY ?v3
```

O2.

```
SELECT ?v0 ?v3 ?v4 ?v8 ?v9 WHERE
{
?v0 <http://schema.org/legalName> ?v1 .
?v0 <http://purl.org/goodrelations/offers> ?v2 .
?v2 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbm/Country5> .
?v2 <http://purl.org/goodrelations/includes> ?v3 .
?v4 <http://schema.org/jobTitle> ?v5 .
?v4 <http://xmlns.com/foaf/homepage> ?v6 .
?v4 <http://db.uwaterloo.ca/~galuc/wsdbm/makesPurchase> ?v7 .
?v7 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseFor> ?v3 .
?v3 <http://purl.org/stuff/rev#hasReview> ?v8 .
?v8 <http://purl.org/stuff/rev#totalVotes> ?v9 .
}
ORDER BY ?v8 ?v9
```

O3.

```
SELECT ?v0 WHERE
{
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v1 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/friendOf> ?v2 .
?v0 <http://purl.org/dc/terms/Location> ?v3 .
?v0 <http://xmlns.com/foaf/age> ?v4 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/gender> ?v5 .
?v0 <http://xmlns.com/foaf/givenName> ?v6 .
}
ORDER BY ?v6
```

O4.

```
SELECT ?v0 ?v1 ?v2 ?v4 ?v5 ?v6 WHERE
 {
?v0 <http://schema.org/contentRating> ?v1 .
?v0 <http://schema.org/contentSize> ?v2 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre>
                <http://db.uwaterloo.ca/~galuc/wsdbm/SubGenre42> .
?v4 <http://db.uwaterloo.ca/~galuc/wsdbm/makesPurchase> ?v5 .
?v5 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseDate> ?v6 .
```

```
?v5 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseFor> ?v0 .
}
ORDER BY ?v6
```

Requêtes Yago

Q1.

```
SELECT ?GivenName ?FamilyName WHERE
{
    ?p <http://yago-knowledge.org/resource/hasGivenName> ?GivenName .
    ?p <http://yago-knowledge.org/resource/hasFamilyName> ?FamilyName .
    ?p <http://yago-knowledge.org/resource/wasBornIn> ?city .
    ?p <http://yago-knowledge.org/resource/hasAcademicAdvisor> ?a .
    ?a <http://yago-knowledge.org/resource/wasBornIn> ?city .
}
```

Q2.

```
SELECT ?GivenName ?FamilyName WHERE
{
    ?p <http://yago-knowledge.org/resource/hasGivenName> ?GivenName .
    ?p <http://yago-knowledge.org/resource/hasFamilyName> ?FamilyName .
    ?p <http://yago-knowledge.org/resource/wasBornIn> ?city .
    ?p <http://yago-knowledge.org/resource/hasAcademicAdvisor> ?a .
    ?a <http://yago-knowledge.org/resource/wasBornIn> ?city .
    ?p <http://yago-knowledge.org/resource/isMarriedTo> ?p2 .
    ?p2 <http://yago-knowledge.org/resource/wasBornIn> ?city .
}
```

Q3.

```
SELECT ?name1 ?name2 WHERE
{
    ?a1 <http://yago-knowledge.org/resource/hasPreferredName> ?name1 .
    ?a2 <http://yago-knowledge.org/resource/hasPreferredName> ?name2 .
    ?a1 <http://yago-knowledge.org/resource/actedIn> ?movie .
    ?a2 <http://yago-knowledge.org/resource/actedIn> ?movie .
}
```

Q4.

```
SELECT ?name1 ?name2 WHERE
{
    ?p1 <http://yago-knowledge.org/resource/hasPreferredName> ?name1 .
    ?p2 <http://yago-knowledge.org/resource/hasPreferredName> ?name2 .
    ?p1 <http://yago-knowledge.org/resource/isMarriedTo> ?p2 .
    ?p1 <http://yago-knowledge.org/resource/wasBornIn> ?city .
    ?p2 <http://yago-knowledge.org/resource/wasBornIn> ?city .
}
```

Q5.

```
SELECT ?c1 WHERE
{
?c1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?c3 .
?c1 <http://www.w3.org/2000/01/rdf-schema#label> ?label .
?c1 <http://yago-knowledge.org/resource/hasInternalWikipediaLinkTo> ?c2 .
?c2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?c4 .
?c2 <http://www.w3.org/2000/01/rdf-schema#label> ?label .
}
```

Requêtes Wildcard

WC1.

```
SELECT ?v0 ?v2 ?v3 ?v4 ?v5 WHERE
{
?v0 <http://ogp.me/ns#tag> <http://db.uwaterloo.ca/~galuc/wsdbm/Topic172> .
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v2 .
?v3 <http://schema.org/trailer> ?v4 .
?v3 <http://schema.org/keywords> ?v5 .
?v3 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre> ?v0 .
?v3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                    <http://db.uwaterloo.ca/~galuc/wsdbm/ProductCategory2> .
FILTER(regex(str(?v4),"Website"))
}
```

WC2.

```
SELECT ?v0 ?v1 ?v2 ?v4 ?v5 ?v6 ?v7 WHERE
{
?v0 <http://xmlns.com/foaf/homepage> ?v1 .
?v0 <http://ogp.me/ns#title> ?v2 .
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v3 .
?v0 <http://schema.org/caption> ?v4 .
?v0 <http://schema.org/description> ?v5 .
?v1 <http://schema.org/url> ?v6 .
?v1 <http://db.uwaterloo.ca/~galuc/wsdbm/hits> ?v7 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre>
        <http://db.uwaterloo.ca/~galuc/wsdbm/SubGenre55> .
FILTER(regex(str(?v6),"elementary"))
}
```

WC3.

```
SELECT ?v0 ?v1 ?v2 ?v4 ?v5 ?v6 WHERE
{
?v0 <http://schema.org/contentRating> ?v1 .
?v0 <http://schema.org/contentSize> ?v2 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre>
            <http://db.uwaterloo.ca/~galuc/wsdbm/SubGenre42> .
?v4 <http://db.uwaterloo.ca/~galuc/wsdbm/makesPurchase> ?v5 .
?v5 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseDate> ?v6 .
?v5 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseFor> ?v0 .
FILTER(regex(str(?v6),"2012-04"))
}
```

WC4.

```
SELECT ?v0 ?v1 ?v2 ?v4 ?v5 ?v6 ?v7 ?v8 WHERE
{
?v0 <http://xmlns.com/foaf/homepage> ?v1 .
?v2 <http://purl.org/goodrelations/includes> ?v0 .
```

```
?v0 <http://ogp.me/ns#tag> <http://db.uwaterloo.ca/~galuc/wsdbm/Topic71> .
?v0 <http://schema.org/description> ?v4 .
?v0 <http://schema.org/contentSize> ?v8 .
?v1 <http://schema.org/url> ?v5 .
?v1 <http://db.uwaterloo.ca/~galuc/wsdbm/hits> ?v6 .
?v1 <http://schema.org/language> <http://db.uwaterloo.ca/~galuc/wsdbm/Language0> .
?v7 <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v0 .
FILTER(regex(str(?v5),"Product21"))
}
```

WC5.

```
SELECT ?v0 ?v1 ?v3 ?v4 ?v5 ?v6 WHERE
{
?v0 <http://purl.org/goodrelations/includes> ?v1 .
<http://db.uwaterloo.ca/~galuc/wsdbm/Retailer28001>
                <http://purl.org/goodrelations/offers> ?v0 .
?v0 <http://purl.org/goodrelations/price> ?v3 .
?v0 <http://purl.org/goodrelations/validThrough> ?v4 .
?v1 <http://ogp.me/ns#title> ?v5 .
?v1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v6 .
FILTER(regex(str(?v4),"2013"))
}
```

WC6.

```
SELECT ?v0 ?v2 ?v3 WHERE
{
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/subscribes>
                <http://db.uwaterloo.ca/~galuc/wsdbm/Website18627> .
?v2 <http://schema.org/caption> ?v3 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v2 .
FILTER(regex(str(?v2),"Product6"))
}
```

WC7.

```
SELECT ?v1 ?v2 WHERE
 {
<http://db.uwaterloo.ca/~galuc/wsdbm/City107>
                    <http://www.geonames.org/ontology#parentCountry> ?v1 .
?v2 <http://db.uwaterloo.ca/~galuc/wsdbm/likes>
                    <http://db.uwaterloo.ca/~galuc/wsdbm/Product0> .
?v2 <http://schema.org/nationality> ?v1 .
FILTER(regex(str(?v1),"Country2"))
}
```

WC8.

```
SELECT ?v0 ?v2 WHERE
 {
?v0 <http://ogp.me/ns#tag> <http://db.uwaterloo.ca/~galuc/wsdbm/Topic132> .
?v0 <http://schema.org/caption> ?v2 .
FILTER(regex(str(?v2),"Profiles"))
}
```

WC9.

```
SELECT ?v0 ?v1 ?v3 ?v4 ?v5 ?v6 ?v7 ?v8 ?v9 WHERE
{
?v0 <http://purl.org/goodrelations/includes> ?v1 .
<http://db.uwaterloo.ca/~galuc/wsdbm/Retailer52462>
                    <http://purl.org/goodrelations/offers> ?v0 .
?v0 <http://purl.org/goodrelations/price> ?v3 .
?v0 <http://purl.org/goodrelations/serialNumber> ?v4 .
?v0 <http://purl.org/goodrelations/validFrom> ?v5 .
?v0 <http://purl.org/goodrelations/validThrough> ?v6 .
?v0 <http://schema.org/eligibleQuantity> ?v7 .
?v0 <http://schema.org/eligibleRegion> ?v8 .
?v0 <http://schema.org/priceValidUntil> ?v9 .
FILTER(regex(str(?v6),"2013-10"))
}
```

WC10.

```
SELECT ?v0 ?v1 ?v2 WHERE
{
?v0 <http://purl.org/ontology/mo/conductor> ?v1 .
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v2 .
?v0 <http://db.uwaterloo.ca/~galuc/wsdbm/hasGenre>
                        <http://db.uwaterloo.ca/~galuc/wsdbm/SubGenre83> .
FILTER(regex(str(?v1),"User4"))
}
```

WC11.

```
SELECT ?v0 ?v1 ?v2 WHERE
{
?v0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?v1 .
?v0 <http://schema.org/text> ?v2 .
<http://db.uwaterloo.ca/~galuc/wsdbm/User145869>
                        <http://db.uwaterloo.ca/~galuc/wsdbm/likes> ?v0 .
FILTER(regex(str(?v0),"Product5"))
}
```

WC12.

```
SELECT ?v6 WHERE
{
?v0 <http://purl.org/goodrelations/includes> ?v1 .
?v0 <http://purl.org/goodrelations/price> ?v3 .
?v0 <http://purl.org/goodrelations/serialNumber> ?v4 .
?v0 <http://purl.org/goodrelations/validFrom> ?v5 .
?v0 <http://purl.org/goodrelations/validThrough> ?v6 .
?v0 <http://schema.org/eligibleQuantity> ?v7 .
?v0 <http://schema.org/eligibleRegion> ?v8 .
?v0 <http://schema.org/priceValidUntil> ?v9 .
FILTER(regex(str(?v6),"2013-10"))
}
```

WC13.

```
SELECT ?v0 ?v4 ?v6 ?v7 WHERE
{
```

```
?v0 <http://schema.org/caption> ?v1 .
?v0 <http://schema.org/text> ?v2 .
?v0 <http://schema.org/contentRating> ?v3 .
?v0 <http://purl.org/stuff/rev#hasReview> ?v4 .
?v4 <http://purl.org/stuff/rev#title> ?v5 .
?v4 <http://purl.org/stuff/rev#reviewer> ?v6 .
?v7 <http://schema.org/actor> ?v6 .
?v7 <http://schema.org/language> ?v8 .
FILTER(regex(str(?v8),"Language2"))
}
```

WC14.

```
SELECT ?v0 ?v3 ?v4 ?v8 WHERE
{
?v0 <http://schema.org/legalName> ?v1 .
?v0 <http://purl.org/goodrelations/offers> ?v2 .
?v2 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbm/Country5> .
?v2 <http://purl.org/goodrelations/includes> ?v3 .
?v4 <http://schema.org/jobTitle> ?v5 .
?v4 <http://xmlns.com/foaf/homepage> ?v6 .
?v4 <http://db.uwaterloo.ca/~galuc/wsdbm/makesPurchase> ?v7 .
?v7 <http://db.uwaterloo.ca/~galuc/wsdbm/purchaseFor> ?v3 .
?v3 <http://purl.org/stuff/rev#hasReview> ?v8 .
?v8 <http://purl.org/stuff/rev#totalVotes> ?v9 .
FILTER(regex(str(?v6),"Website60"))
}
```

ملخص

تمثل معالجة البيانات الضخمة تحديًا كبيرا ليس فقط في المجال الاجتماعي والاقتصادي وانما كذلك للبحث العلمي.، كما تمت الإشارة إليه في العديد من المقالات العلمية والتقارير الإستراتيجية، تواجه تطبيقات الحاسوب الحديثة مشاكل وقضايا جديدة تتعلق بشكل رئيسي بتخزين واستغلال البيانات الناتجة عن أدوات المراقبة والمحاكاة الحديثة. تمثل إدارة هذه البيانات اختناقًا حقيقيًا يؤدي إلى إبطاء استغلال البيانات المختلفة التي تم جمعها ليس فقط في إطار البرامج العلمية الدولية ولكن أيضًا من قبل الشركات، والتي تعتمد بشكل متزايد على تحليل البيانات واسعة النطاق. يتم نشر الكثير من هذه البيانات اليوم على الويب. في الواقع ، نشهد تطورًا للويب التقليدي ، المصمم أساسًا لإدارة المستندات ، إلى شبكة من البيانات التي تسمح بتقديم آليات للاستعلام عن المعلومات الدلالية. تم اقتراح العديد من نماذج البيانات لتمثيل هذه المعلومات على الويب. والأهم هو (RDF) الذي يوفر تمثيلًا بسيطًا ومجرّدًا لمعرفة الموارد على الويب. يمكن ترميز كل حقيقة ويب دلالات باستخدام ثلاثيات الـ RDF.

من أجل استكشاف واستعلام المعلومات المنظمة المعبر عنها في RDF ، تم اقتراح العديد من لغات الاستعلام على مر السنين. في عام 2008، أصبحت SPARQL لغة التوصية الرسمية للاستعلام عن بيانات RDF.

أدت الحاجة إلى إدارة بيانات RDF والاستعلام عنها بكفاءة إلى تطوير أنظمة جديدة مصممة خصيصًا لمعالجة تنسيق هذه البيانات. يمكن تصنيف هذه الأساليب على أنها مركزية تعتمد على جهاز واحد لإدارة بيانات RDF وتوزيعها التي يمكنها الجمع بين أجهزة متعددة متصلة بشبكة الحاسوب. تعتمد بعض هذه الأساليب على نظام إدارة بيانات موجود مثل Virtuoso و Jena، ويعتمد البعض الآخر على نهج مصمم خصيصًا لإدارة ثلاثيات الـ RDF مثل GRIN و RDF-3X و gStore. مع تطور مجموعات بيانات الـ RDF (مثل DBPedia) و Sparql ، أصبحت معظم الأنظمة قديمة و / أو غير فعالة. على سبيل المثال ، لا يستطيع أي من النظم المركزية الموجودة إدارة المليار ثلاثية المقدمة تحت معيار WatDiv. ستسمح الأنظمة الموزعة في ظل ظروف معينة بتحسين هذه النقطة ولكنها تؤدي بالتالي إلى تدهور الأداء.

في أطروحة الدكتوراه هذه، نقترح النظام المركزي "RDF_QDAG" الذي يسمح بإيجاد حل وسط جيد بين قابلية التوسع والأداء. نقترح الجمع بين تجزئة البيانات المادية واستكشاف الرسم البياني للبيانات. يدعم "RDF_QDAG" أنواعًا متعددة من الاستعلامات التي تستند ليس فقط على أنماط الشبكية الأساسية ولكن أيضًا تتضمن فلاتر استنادًا إلى التعبيرات العادية وكذلك وظائف التجميع والفرز. يعتمد "RDF_QDAG" على نموذج تنفيذ Volcano ، الذي يسمح بالتحكم في الذاكرة الرئيسية ، وتجنب أي تجاوز حتى إذا كان قدرة الأجهزة محدودة. على حد علمنا، "RDF_QDAG" هو النظام المركزي الوحيد الذي يقدم أداءً جيدًا عند إدارة عدة مليارات من الثلاثيات. لقد قمنا بمقارنة هذا النظام مع أنظمة أخرى تمثل أحدث ما توصلت إليه إدارة بيانات الـ RDF: كـ (Virtuoso)، (g-Store) و (RDF-3X) وطريقتين متوازيتين (CliqueSquare و g-Store-D). يتجاوز "RDF_QDAG" الأنظمة الموجودة عندما يتعلق الأمر بضمان قابلية التوسع والأداء.

كلمات الدلالة: RDF ، استكشاف الرسم البياني ، التجزئة ، قابلية التوسع ، الأداء

Résumé

Le Big Data représente un défi non seulement pour le monde socio-économique mais aussi pour la recherche scientifique. En effet, comme il a été souligné dans plusieurs articles scientifiques et rapports stratégiques, les applications informatiques modernes sont confrontées à de nouveaux problèmes qui sont liés essentiellement au stockage et à l'exploitation de données générées par les instruments d'observation et de simulation. La gestion de telles données représente un véritable goulot d'étranglement qui a pour effet de ralentir la valorisation des différentes données collectées non seulement dans le cadre de programmes scientifiques internationaux mais aussi par des entreprises, ces dernières s'appuyant de plus en plus sur l'analyse de données massives. Une bonne partie de ces données sont publié aujourd'hui sur le WEB. Nous assistons en effet à une évolution du Web classique permettant de gérer les documents vers un Web de données qui permet d'offrir des mécanismes d'interrogation des informations sémantiques. Plusieurs modèles de données ont été proposés pour représenter ces informations sur le Web. Le plus important est le Resource Description Framework (RDF) qui fournit une représentation des connaissances simple et abstraite pour les ressources sur le Web. Chaque fait du Web sémantique peut être codé avec un triplet RDF. Afin de pouvoir explorer et interroger les informations structurées exprimées en RDF, plusieurs langages de requête ont été proposés au fil des années. En 2008, SPARQL est devenu le langage de recommandation officiel du W3C pour l'interrogation des données RDF. La nécessité de gérer et interroger efficacement les données RDF a conduit au développement de nouveaux systèmes conçus spécialement pour traiter ce format de données. Ces approches peuvent être catégorisées en étant centralisées qui s'appuient sur une seule machine pour gérer les données RDF et distribuées qui peuvent combiner plusieurs machines connectées avec un réseau informatique. Certaines de ces approches s'appuient sur un système de gestion de données existant tels que Virtuoso et Jena, d'autres approches sont basées sur une approche spécialement conçue pour la gestion des triplets RDF comme GRIN, RDF3X et gStore. Avec l'évolution des jeux de données RDF (e.g. DBPedia) et du langage Sparql, la plupart des systèmes sont devenus obsolètes et/ou inefficaces. A titre d'exemple, aucun système centralisé existant n'est en mesure de gérer 1 Milliard de triplets fournies dans le cadre du benchmark WatDiv. Les systèmes distribués permettraient sous certaines conditions d'améliorer ce point mais une perte de performances conséquente est induite.

Dans cette thèse, nous proposons le système centralisé "RDF_QDAG" qui permet de trouver un bon compromis entre passage à l'échelle et performances. Nous proposons de combiner la fragmentation physique de données et l'exploration du graphe de données. "RDF_QDAG" permet de support plusieurs types de requêtes basées non seulement sur les motifs basiques de graphes mais aussi qui intègrent des filtres à base d'expressions régulières et aussi des fonctions d'agrégation et de tri. "RDF_QDAG" se base sur le modèle d'exécution Volcano, ce qui permet de contrôler la mémoire principale, en évitant tout débordement pour garantir les performances même si la configuration matérielle est limitée. A notre connaissance, "RDF_QDAG" est le seul système centralisé capable de gérer plusieurs milliards de triplets tout en garantissant de bonnes performances. Nous avons comparé ce système avec d'autres systèmes qui représentent l'état de l'art en matière de gestion de données RDF : une approche relationnelle (Virtuoso), une approche à base de graphes (g-Store), une approche d'indexation intensive (RDF-3X) et une approche MPP (CliqueSquare). "RDF_QDAG" surpasse les systèmes existants lorsqu'il s'agit de garantir à la fois le passage à l'échelle et les performances.

Mots-clés : Bases de données–Interrogation, Données massives, Explorations de graphes, Gestion des données (systèmes d'information), Resource Description Framework (informatique), SPARQL (langage de programmation), Fragmentation, Évolutivité, Performance

Abstract

Big Data represents a challenge not only for the socio-economic world but also for scientific research. Indeed, as has been pointed out in several scientific articles and strategic reports, modern computer applications are facing new problems and issues that are mainly related to the storage and the exploitation of data generated by modern observation and simulation instruments. The management of such data represents a real bottleneck which has the effect of slowing down the exploitation of the various data collected not only in the framework of international scientific programs but also by companies, the latter relying increasingly on the analysis of large-scale data. Much of this data is published today on the WEB. Indeed, we are witnessing an evolution of the traditional web, designed basically to manage documents, to a web of data that allows to offer mechanisms for querying semantic information. Several data models have been proposed to represent this information on the Web. The most important is the Resource Description Framework (RDF) which provides a simple and abstract representation of knowledge for resources on the Web. Each semantic Web fact can be encoded with an RDF triple. In order to explore and query structured information expressed in RDF, several query languages have been proposed over the years. In 2008, SPARQL became the official W3C Recommendation language for querying RDF data.

The need to efficiently manage and query RDF data has led to the development of new systems specifically designed to process this data format. These approaches can be categorized as centralized that rely on a single machine to manage RDF data and distributed that can combine multiple machines connected with a computer network. Some of these approaches are based on an existing data management system such as Virtuoso and Jena, others relies on an approach specifically designed for the management of RDF triples such as GRIN, RDF3X and gStore. With the evolution of RDF datasets (e.g. DBPedia) and Sparql, most systems have become obsolete and/or inefficient. For example, no one of existing centralized system is able to manage 1 billion triples provided under the WatDiv benchmark. Distributed systems would allow under certain conditions to improve this point but consequently leads a performance degradation.

In this Phd thesis, we propose the centralized system "RDF_QDAG" that allows to find a good compromise between scalability and performance. We propose to combine physical data fragmentation and data graph exploration. "RDF_QDAG" supports multiple types of queries based not only on basic graph patterns but also that incorporate filters based on regular expressions and also aggregation and sorting functions. "RDF_QDAG" relies on the Volcano execution model, which allows controlling the main memory, avoiding any overflow even if the hardware configuration is limited. To the best of our knowledge, "RDF_QDAG" is the only centralized system that good performance when manage several billion triples. We compared this system with other systems that represent the state of the art in RDF data management : a relational approach (Virtuoso), a graph-based approach (g-Store), an intensive indexing approach (RDF-3X) and two parallel approaches (CliqueSquare and g-Store-D). "RDF_QDAG" surpasses existing systems when it comes to ensuring both scalability and performance.

Keywords : Database searching, Big data, Graph exploration, Data management, RDF (Document markup language), SPARQL (Computer program language)

Secteur de recherche : Informatique et applications

Laboratoire d'Informatique et d'Automatique pour les Systèmes
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Téléport 2 – 1 avenue Clément Ader – BP 40109 – 86961 FUTUROSCOPE CHASSENEUIL CEDEX
Tél : 05.49.49.80.63 – Fax : 05.49.49.80.64