HAI914I

Module : Gestion des données au dela de SQL : NoSQL

Évaluation et Analyse des Performances du moteur KnowledgeGraph RDF

Réalisé par :
M lle MEKHNACHE Toudherth – M2 GL

Promotion 2023/2024

Introduction :

Notre projet s'inscrit au cœur du domaine du web sémantique et des données liées, avec pour objectif principal le développement d'un moteur de requêtes en étoile dédié aux triplets RDF (Resource Description Framework). L'évolution rapide de ces technologies a engendré une demande croissante de mécanismes d'interrogation efficaces, capables de gérer des ensembles de données sémantiques de plus en plus vastes. Notre démarche repose sur l'approche hexastore, une fusion novatrice de dictionnaires et d'indexation visant à optimiser la recherche et la récupération d'informations. En conjuguant la théorie des graphes et l'informatique distribuée, notre projet vise à relever les défis actuels de l'interrogation des données RDF, tout en aspirant à des performances comparables à celles de Jena, une référence établie dans ce domaine.

Création du dictionnaire :

Le principe fondamental du dictionnaire réside dans l'association de chaque information de la base de données à un entier, simplifiant ainsi le stockage des données et améliorant les performances du programme. Notre implémentation du dictionnaire repose sur une classe appelée `Dictionary`, comprenant une `HashMap` où chaque élément est composé d'une clé de type `String` correspondant au sujet, au prédicat ou à l'objet d'une ressource RDF, et d'une valeur de type `Integer` représentant un index utilisé pour l'implémentation de l'hexastore index. De plus, une seconde `HashMap` inversée a été intégrée à cette classe, facilitant la récupération d'un élément à partir de son index lors de la restitution des résultats des requêtes. Ce dictionnaire est persistamment stocké dans un fichier CSV. La Figure 1 illustre les informations contenues dans le dictionnaire.

```
HashMap<Integer, String> reverseDico;
HashMap<String, Integer> Dico;
```

Nous stockons le dictionnaire dans un fichier csv ce qui rend les données persistantes. La figure 1 représente les informations stockées dans le dictionnaire.

id	String
0	http://db.uwaterloo.ca/~galuc/wsdbm/Offer6
1	http://schema.org/birthDate
2	1988-09-24
3	http://db.uwaterloo.ca/~galuc/wsdbm/userId
4	9764726
5	http://db.uwaterloo.ca/~galuc/wsdbm/User1
6	2536508
7	http://db.uwaterloo.ca/~galuc/wsdbm/User2
8	5196173
9	http://db.uwaterloo.ca/~galuc/wsdbm/Offer659
10	http://schema.org/eligibleRegion

FIGURE 1 – Exemple de Dictionnaire

Bien sûr, continuons avec la section sur la création des index et l'évaluation des requêtes en étoiles :

Création des index :

L'objectif de l'indexation des données RDF (Resource Description Framework) est d'optimiser la recherche et l'accès aux informations stockées. Les données RDF sont généralement représentées sous forme de triplets (sujet, prédicat, objet), constituant ainsi une structure de graphe.

Pour atteindre cet objectif, nous avons créé six arbres, à savoir SOP, SPO, PSO, POS, OSP, OPS, qui représentent différentes façons de structurer les données. Ces arbres sont implémentés sous forme de `HashMap<Integer, HashMap<Integer, ArrayList<Integer»>`, où les entiers sont associés à des chaînes de caractères à l'aide du dictionnaire pour optimiser le parcours de l'arbre. Les feuilles de ces arbres sont représentées par des `ArrayList<Integer>`.

```
HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> spo ;
HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> sop ;
HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> pso ;
HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> pos ;
HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> osp ;
HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> ops ;
```

La figure 2 represente les resultats de stockage des données sous forme des index (key) a partir de dictionnaire.

	Subject	Predicate	Object
1	0	1	2
2	0	3	4
3	32	10	31
4	33	34	35
5	33	37	38
6	36	34	35
7	36	37	38
8	5	3	6
9	7	3	8

FIGURE 2 – Les données sous forme ArrayList d'index (key)

Évaluation des requêtes en étoiles :

L'évaluation des requêtes en étoiles repose sur une fonction `getFromMap` dans l'index. Cette fonction prend en paramètre une `HashMap` correspondant à l'un des six arbres syntaxiques de l'index et parcourt cet arbre en fonction des informations fournies dans `info1` et `info2` qui sont soit des object, predicat ou subject.

```
private ArrayList<Integer> getFromMap(HashMap<Integer, HashMap<
    Integer, ArrayList<Integer>>> map, String info1, String info2
    ) {
        ArrayList<Integer> ret = new ArrayList<>() ;
        int i1 = Dictionary.getInstance().encode(info1) ;
        int i2 = Dictionary.getInstance().encode(info2) ;
        if(i1!=-1 && i2 != -1){
            if (map.get(i1) != null) {
                if (map.get(i1).get(i2)  != null) {
```

```
                    ret.addAll(map.get(i1).get(i2)) ;
                }
            }
        }
        return ret;
    }
```

Nous avons une fonction `getFromPOS(String predicat, String object)` qui prend en paramètre un object et un predicat qui va utiliser la fonction getFromMap pour parcourir l'arbre syntaxique POS.

```
public ArrayList<Integer> getFromPOS(String predicat, String
   object) {
    ArrayList<Integer> ret = new ArrayList<>(getFromMap(pos,
       predicat, object)) ;
    return ret; }
```

Cette fonction `evaluateStatementPatern` permet de récupérer les valeurs subject, predicat et object du StatementPattern et utilise la fonction getFromPOS de Index pour retourner le résultat de la requete

```
private static ArrayList<Integer> evaluateStatementPatern(
   StatementPattern sp) {
    Value p = sp.getPredicateVar().getValue();
    Value o = sp.getObjectVar().getValue();
    return new ArrayList<>(Index.getInstance().getFromPOS(p.
       toString(), o.toString()));
}
```

Cette fonction `intersect` prend en paramètre deux `ArrayList<Integer>` qui représentent deux réponses de requête, puis effectue l'intersection pour récupérer l'ensemble commun entre les deux résultats, correspondant à la requête en étoile.

```
private static ArrayList<Integer> intersect(ArrayList<Integer>
   a1, ArrayList<Integer> a2) {
        ArrayList<Integer> ret = new ArrayList<>();
        for(Integer i : a1) { if (a2.contains(i)) { ret.add(i) ;
            } }
        return ret;
    }
```

Pour la lecture des requêtes, nous avons utilisé la fonction `processAQuery(ParsedQuery query)` fournie dans le programme principal. Nous récupérons une ParsedQuery, un objet représentant une requête que nous évaluons à l'aide de notre fonction RequestResult, qui renvoie une ArrayList de chaînes de caractères correspondant aux résultats de la requête. Ensuite, nous effectuons l'intersection entre le résultat actuel (obtenu par les requêtes précédentes) et le résultat de la requête qui vient d'être exécutée.

La fonction renvoie ensuite le résultat global des requêtes. Nous affichons ce résultat en le décodant grâce à la fonction du dictionnaire.

```
private static ArrayList<Integer> evaluateStarRequest(
   ParsedQuery query) {
    ArrayList<Integer> results = null;
    List<StatementPattern> patterns = StatementPatternCollector.
       process(query.getTupleExpr());
```

```
    boolean firstPassage = true;
    for(StatementPattern sp : patterns) {
        if (results == null) {
            results = new ArrayList<>() ;
            results.addAll(evaluateStatementPatern(sp)) ;}
        else {
        results=intersect(results, evaluateStatementPatern(sp));
        }
    }
    return results ;
}
```

Résultats de lecture des requêtes :

Les résultats obtenus suite à l'exécution des requêtes sont présentés dans la Figure 3 sur quelque données qu'on a utiliser . Cette visualisation met en évidence les réponses générées pour chaque requête ainsi que le nombre total de résultats.

	C1	C2	C3	C4
1	query	nbResult	results	<unset>
2	SELECT ?v0 WHERE { ?v0	2	http://db.uwaterloo.ca/~galuc/wsdbm/Offer659	http://db.uwaterloo.ca/~galuc/wsdbm/Offer6
3	SELECT ?v0 WHERE { ?v0	2	http://db.uwaterloo.ca/~galuc/wsdbm/User21	http://db.uwaterloo.ca/~galuc/wsdbm/User73
4	SELECT ?v0 WHERE { ?v0	1	http://db.uwaterloo.ca/~galuc/wsdbm/User3	<null>

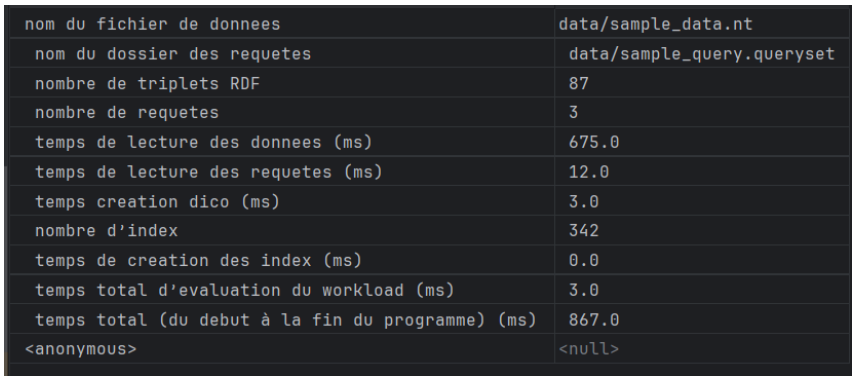
FIGURE 3 – Le resultat de l'execution des requetes

Ces résultats détaillent chaque requête, le nombre de résultats retournés, ainsi que les résultats spécifiques pour chaque requête. Ces informations exportées dans un fichier CSV offrent une trace précise des performances de système lors de l'évaluation du workload.

Mesure des performances :

Pour évaluer les performances de nos requêtes, nous avons mesurer le temps d'exécution, le nombre de résultats retournés, etc. pour cela nous avons utiliser `System.currentTimeMillis()` pour mesurer le temps d'exécution.

Notre programme est conçu pour fournir une transparence totale sur les performances en capturant et en exportant les temps d'évaluation des requêtes, ainsi que d'autres métriques cruciales. La figure 4 comment ces informations sont présentées dans un fichier CSV, conformément aux spécifications .



nom du fichier de donnees	data/sample_data.nt
nom du dossier des requetes	data/sample_query.queryset
nombre de triplets RDF	87
nombre de requetes	3
temps de lecture des donnees (ms)	675.0
temps de lecture des requetes (ms)	12.0
temps creation dico (ms)	3.0
nombre d'index	342
temps de creation des index (ms)	0.0
temps total d'evaluation du workload (ms)	3.0
temps total (du debut à la fin du programme) (ms)	867.0
<anonymous>	<null>

FIGURE 4 – Le resultat de l'execution des requetes

Cette approche garantit une visibilité complète sur les performances du système, offrant à l'utilisateur une analyse détaillée des temps d'exécution à chaque étape du processus d'évaluation du workload

Annexes :

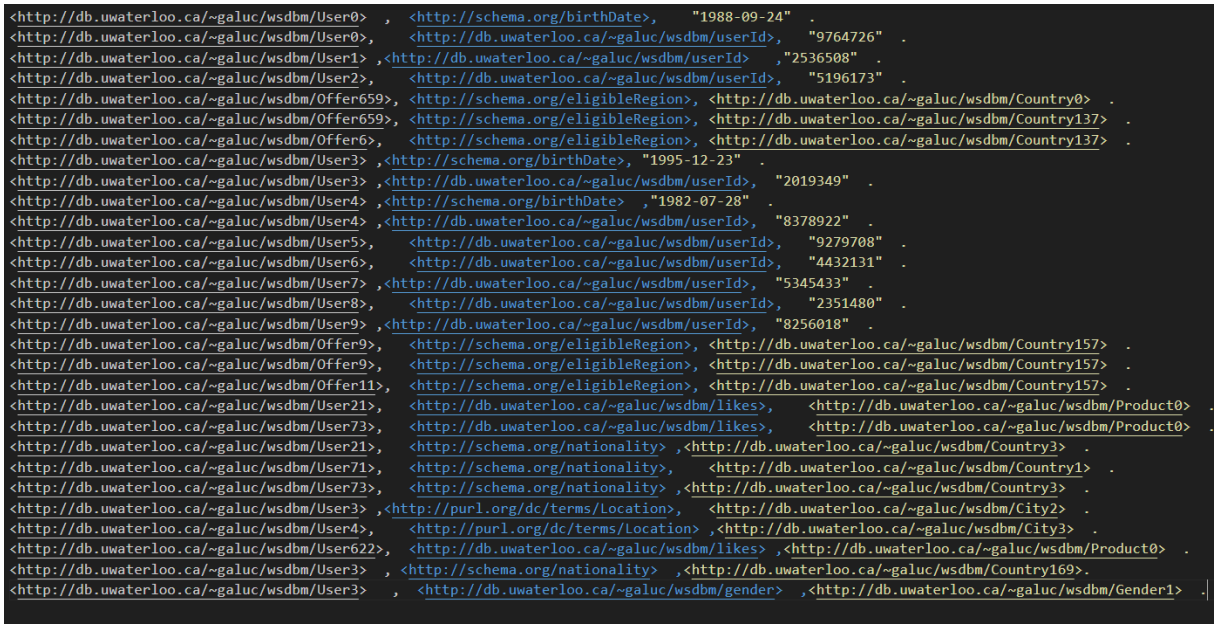


FIGURE 5 – Les données utiliser pour les testes