Université de Montpellier

Rapport de projet NoSQL
Partie II - Évaluation et Analyse des Performances

El Houiti Chakib
Kezzoul Massili

20 décembre 2021

Introduction

Ce rapport fait suite à sa première partie qui concerne l'implémentation d'un moteur de requête *SPARQL* en étoiles en utilisant l'approche *hexastore*. Dans cette partie, nous allons analyser les performances de notre implémentation par rapport à d'autres implémentations. Notamment celle de Jena.

Dans un premier temps, nous allons préparer et analyser des bancs d'essais en utilisant *WatDiv*[1]. *WatDiv* est système developpé afin de mesurer les performances d'un moteur de requête *SPARQL*. Il consiste en la génération de jeux de données ainsi que des jeux de requêtes.

Dans un second temps, nous allons définir et comparer plusieurs plans de tests afin d'en trouver un (ou plusieurs) qui donne des résultats correctes, significatifs et interprétables.

Enfin, viendra la partie concrète d'évaluation des performances. On executera les plans de tests précédemment réalisés. Suivant les résultats obtenus, nous les présenterons selon des représentations graphiques que nous allons analyser. Nous expliquerons aussi les raisons à la base de ces résultats.

1 Bancs d'essais

1.1 Préparation des bancs d'essais

La première partie est de générer les jeux de données sur lesquelles les tests vont être exécutés. Pour cela, on utilise le programme *WatDiv*.

1.1.1 Génération des données

La génération d'un *dataset* se fait en utilisant la commande suivante :

```
./watdiv -d <model-file> <scale-factor>
```

Listing 1 – Commande pour la création d'un dataset

- *model-file* est la template sur laquelle le programme se base pour créer le jeu de données ;
- Pour un $scaleFactor = 1$, on obtient environ $100K$ triplets. On peut donc augmenter le nombre de triplets en augmentant la valeur de $scaleFactor$.

Afin de répondre au besoin de l'évaluation, nous avons générer un *dataset* de $500K$ triplets ainsi qu'un autre de $1M$ de triplets.

1.1.2 Génération des requêtes

Afin de générer des requêtes, nous avons écrit un script *Shell* qui, à partir de quelques templates, génére pour chaqu'une d'elle un fichier contenant 1000 requêtes.

```
for template in ${TEMPLATE_DIR}/*.sparql-template;
do
  template_name=${template##*/}
  template_name=${template_name%.sparql-template}
  querie_file=$RESULT_DIR"/"$template_name".queryset"

  if [ $override -eq 0 ] && [ -f $querie_file ]; then
      echo -e "File '$querie_file' already exists.\nDo you want to overwrite ? (any key to continue / CTRL-C to exit)"
      read $x
      echo -e "Overwriting..."
      override=1
```

1. Waterloo SPARQL Diversity Test Suite.

```
    fi

    $WATDIV/watdiv -q $WATDIV/model/wsdbm-data-model.txt ${template} $NB_QUERIES 1 > $querie_file
    i=$((i+1))
done
```

Listing 2 – "Extrait du script qui génére les requêtes

1.2 Analyse des bancs d'essais

L'analyse des bancs d'essais se fait principalement sur la *qualité*[2] des requêtes générer.

1.2.1 Première version

Une première version du banc à été réaliser en utilisant les templates fournies de base. Quelques statistiques ont été extraites de ces templates afin de visualiser la *qualité* des ses requêtes.

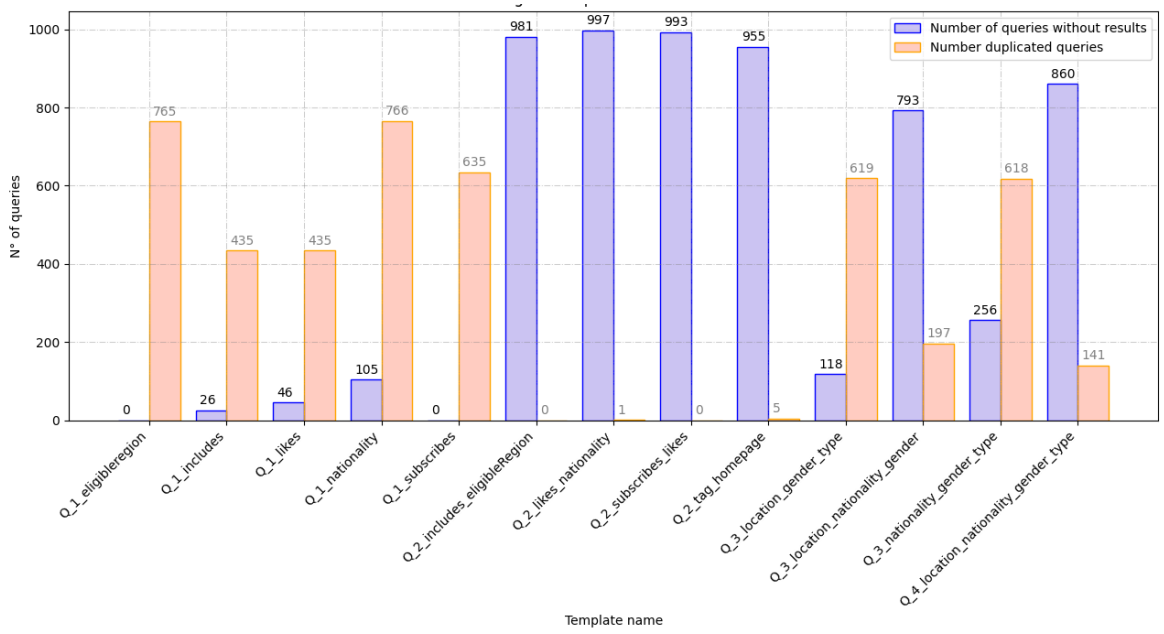

FIGURE 1 – Nombres de requêtes sans réponses et dupliquées

Sur la figure 1, on voit pour chaque template le nombre de requêtes sans réponses (en bleu) ainsi que le nombre de requêtes en doubles (en orange). Le nom des templates commence par Q_i, où i represente le nombre de *patterns* des requêtes associées.

On observe clairement sur l'histograme que trois groupes de templates se forment.

1 pattern Ces templates ont un grand nombre de requêtes dupliquées et peu de requêtes sans résultats. Cela est probablement dû au faite que plus la requête est simple (peu de pattern) et plus elle a de résultats. Par contre, cela diminue le nombre de requêtes *différentes* qu'on peut générer pour cette template.

2. On définit ici la *qualité* d'une requête par la qualité des analyses qui decoulerons de celles-ci

2 patterns On observe sur ce groupe que quasiment toutes ses requêtes sont uniques mais aussi qu'elles sont toutes sans résultats. Ce qui n'est pas souhaitable. Nous reviendrons plus tard sur ces templates afin de voir d'où viens ce problème.

3 patterns ou plus Ce groupe donne un résultat assez varié. On observe clairement ici une corrélation entre le nombre de doublons et le nombre de requêtes sans réponses. plus l'un est élevé et plus l'autre est faible.

Ces résultats ne sont pas satisfaisant car ils biaiserons fortement les résultats des performances. En effet, un nombre élevé de requêtes sans réponses affectera forcement le temps d'éxecution de notre implémenation car celle-ci s'arrête dès qu'un des patterns ne donne pas de résultat. la requête ne s'exécute pas entièrement. Cette valeur doit donc être *minimal*.

Le nombre de doublons aussi doit être de préférence proche de zero. Dans le cas contraire une requête qui apparait trop souvent influencera plus que les autres la moyenne du temps d'execution. Si par exemple, cette requête s'exécute plus rapidement alors la moyenne se verra sous-éstimé. Par contre, ce facteur reste moins important que le nombre de requête sans réponses.

1.2.2 Vers une amélioration du banc d'essai

2 Plan des tests

3 Évaluation des performances

Cette partie est en cours de réalisation.

4 Conclusion

4.1 Apport des évaluations

4.2 Perspectives d'amélioration