Rapport de mini-projet - HAI914I
Un moteur d'évaluation de requêtes en étoile

DUCHON Damien, LAURET Nicolas

4 janvier 2022

Introduction

Dans le cadre de ce mini-projet, nous avons poursuivi deux objectifs :

— Implémenter l'approche hexastore pour l'interrogation des données RDF, ainsi que les procédures nécessaires à l'évaluation de requêtes en étoile.

— Évaluer et analyser les performances de notre système en le comparant à Jena à l'aide du benchmark WatDiv.

Le premier objectif ayant été rempli et documenté lors des précédents rapports, un bref résumé de ce qui a été effectué vous est présenté dans la section ci-dessous.

1 Dictionnaire, index et évaluation des requêtes

Afin de pouvoir mener à bien l'évaluation des requêtes en étoile, notre système récupère les données dans un dictionnaire (`HashMap<String, Integer>`) en les associant à un entier unique, lui-même utilisé pour la création des index `SPO`, `POS`, [...] (`HashMap<Integer, HashMap<Integer, HashSet<Integer>>>`).

Une fois terminé, les requêtes sont récupérées et traitées une à une afin de trouver l'élément à chercher (sujet, prédicat ou objet) et déterminer l'index le plus approprié pour la recherche.

Nous avons choisi des structures de données telles que chacun de nos accès en lecture/écriture possède une complexité en `O(1)`.

2 Préparation des bancs d'essais et métriques

Afin de compléter le deuxième objectif, nous avons dans un premier temps généré des fichiers de données de respectivement **500K**, **1M**, **2M** et **3M** de triplets. Cela nous permettra par la suite d'observer le comportement de notre approche avec des tailles de données différentes.

Nous avons ensuite créé et utilisé des *"templates"* de requêtes afin d'en générer à l'aide de WatDiv. Cet ensemble de requêtes généré a été passé à un programme annexe (disponible dans notre code final) dont le but est de filtrer et récupérer les requêtes correspondant à nos critères de sélection :

- **15-25%** de requêtes sans réponse dont **∼30%** de requêtes dupliquées.
- **>70%** de requêtes de moins de 3 conditions avec au moins une réponse dont **∼30%** de requêtes dupliquées.
- **>250 requêtes** de plus de 3 conditions avec au moins une réponse dont **∼30%** de requêtes dupliquées.
- **>25 000** requêtes récupérées au total.

Nous avons jugé important d'avoir un certain nombre de requêtes sans réponse afin de pouvoir comparer les performances des systèmes dans le cas où il n'y a aucune réponse.
De plus, avoir des requêtes avoir un nombre de conditions des prédicats/objets divers permet d'observer les comportements des approches sur des données multiples et variées.
Avoir des requêtes avec un grand nombre de conditions permet également d'analyser le temps que mettent les deux systèmes.
Enfin, les requêtes dupliquées dans chacun des cas sont utiles pour les tests *"warm"*.

Les requêtes finales respectent ainsi nos critères pour les données de taille 1M, 2M et 3M.

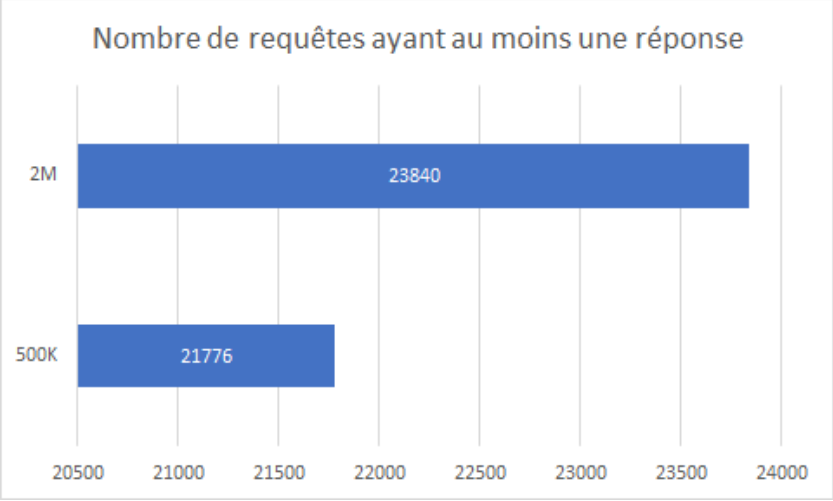

FIGURE 1 – Nombre de requêtes ayant au moins une réponse sur 34 367 requêtes

Afin d'évaluer les performances des moteurs de requêtes RDF, nous avons décidé comme métriques de comparer le temps d'exécution total (*"cold"* et *"warm"*), le temps d'exécution moyen (*"warm"*) ainsi que le nombre de réponses aux requêtes (*"warm"*) afin de vérifier que les deux systèmes récupèrent l'ensemble des réponses attendues.

3 Hardware, software et facteurs

Pour la réalisation de nos tests, nous avons utilisé la machine et la configuration suivante :

— **CPU** : Ryzen 7 1700 @ 3GHz, 20Mo cache, 8C/16T
— **RAM** : 16Go 2666CL16
— **Stockage** : SSD Crucial MX300 Sata, 500Mo/s lecture et écriture
— **Système** : Windows 10 Pro, 64 bits
— **Java** : Java 17
— **État** : Aucun processus en cours (hors processus en arrière-plan)
— **Exécution** : Fichier JAR exécuté en ligne de commande.

Cette configuration a servi de base de comparaison entre notre système et celui de Jena sur le jeu de requêtes généré.

Nous sommes convaincus que notre configuration est adaptée à l'analyse des performances de ces deux systèmes même si nous restons conscients des potentiels facteurs pouvant menacer la fiabilité de notre benchmark.

Un des facteurs primordiaux qui va entrer en jeu pendant l'évaluation du système est sa conception en elle-même. En effet selon la structure de données que nous choisissons et l'implémentation de l'algorithme, nous pouvons obtenir des résultats divergents dû à une différence de complexité.

Bien sûr, l'environnement a une place importante dans cette évaluation. L'utilisation et la capacité du processeur et de la mémoire vive vont être les deux facteurs qui vont le plus influer sur nos résultats côté *hardware*, sans oublier les tâches en arrière-plan et le temps de lecture et d'écriture de nos données et requêtes depuis le disque dur.

Enfin, nous pouvons considérer comme secondaires les facteurs liés au système d'exploitation et à la version de Java/Jena utilisée pour l'exécution des systèmes, ainsi que les possibles *logs* dans le terminal (appels bloquants).

4 Évaluation des performances

Notre protocole de réalisation des mesures consiste à exécuter **5 fois** les scénarios suivants pour chacun des systèmes (le notre et celui de Jena) sur des jeux de données de taille **500K**, **1M**, **2M** et **3M** : ensemble de requêtes **sans doublons** (test *"cold"*), ensemble de requêtes **avec uniquement les doublons** (test *"warm"*), ensemble **total** des requêtes (test *"warm"*).

De plus, nous avons réalisé des tests de performances en fonction de la taille de la mémoire allouée aux systèmes en faisant varier la mémoire à **2Go**, **4Go** et **8Go**.

Afin de vérifier la correction et la complétude de notre approche, nous comparons l'ensemble des réponses retournées avec l'ensemble des réponses retournées par Jena (considéré comme un système "oracle") afin de vérifier que les deux ensembles sont égaux.

Au cours de nos tests nous nous sommes rendus compte que les ensembles n'étaient pas égaux et que notre solution trouvait des réponses à des requêtes où Jena n'en trouvait pas. Même si cela implique que notre système n'est pas correct, nous avons trouvé important de le mentionner car cela fait partie de la démarche d'analyse et à ce jour nous n'avons pas trouvé l'origine du problème.

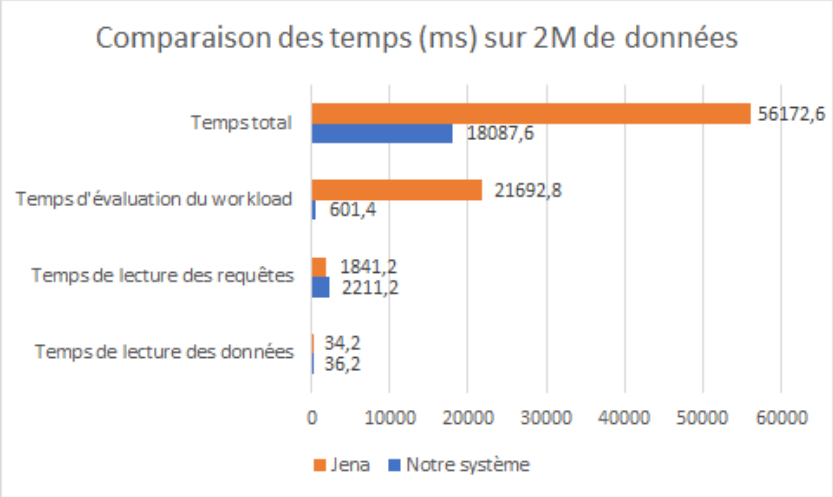

FIGURE 2 – Temps moyens (5 exécutions) sur l'ensemble des 34 367 requêtes

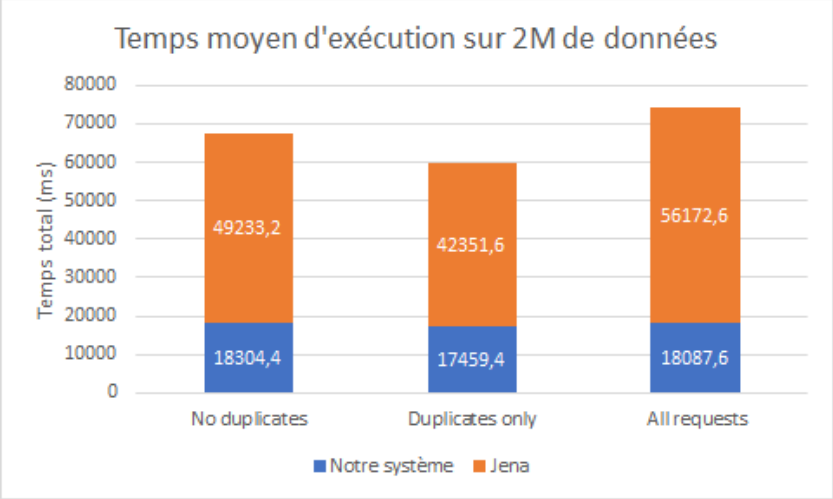

FIGURE 3 – Temps moyens d'exécution sur les différents scénarios (*"cold"* et *"warm"*)

À l'aide des résultats présentés sur les figures 2 et 3, et des données présentes en annexe, nous pouvons observer que Jena est légèrement plus rapide lors de la lecture des données

et des requêtes peu importe la taille des données, tandis que notre système est significativement plus performant pour l'évaluation du *workload* (**∼3607%** de différence) et de manière générale (**∼311%** de différence).
De plus, même si Jena semble mieux gérer les ensembles de requêtes avec uniquement des doublons, notre approche reste assez constante avec un temps bien inférieur à celui de Jena.

Afin de vérifier qu'il ne s'agit pas d'une erreur de notre part, nous avons vérifié notre implémentation de Jena avec celle d'un autre groupe afin de nous assurer d'avoir le même nombre de réponses aux requêtes et à peu près le même temps. Des comportements similaires où l'implémentation d'un groupe est plus performante que Jena a pu être observée en fonction du hardware/software utilisé.
Nous n'écartons pas la piste d'un mauvais calcul pour le temps de *workload* de notre implémentation car la différence semble énorme malgré nos accès en lecture/écriture en `O(1)`.

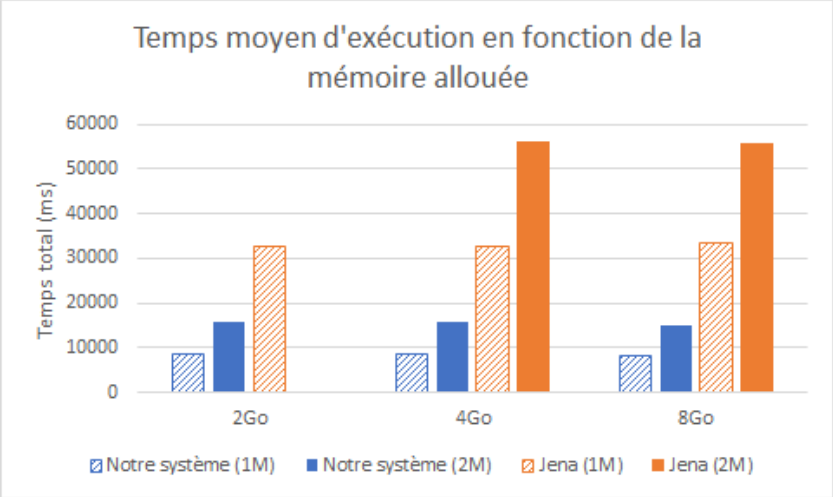

FIGURE 4 – Temps moyens d'exécution sur l'ensemble des requêtes en fonction de la mémoire

Enfin, après avoir réalisé des variations de la mémoire allouée aux deux applications, il semble assez clair que celle-ci n'a pas d'impact significatif quant à la performance du système. Il nous semble cependant important de noter que Jena ne peut traiter **2M** de données avec seulement **2Go** de RAM contrairement à notre approche.

Conclusion

Nous avons pu, au travers des divers tests effectués, extraire des données qui nous ont permis d'observer que notre implémentation est **3 fois** plus rapide que celle de Jena. Cependant, ces mêmes données mettent en évidence une incohérence quant au nombre de réponses retournées par les deux approches avec cette fois-ci un surplus de requêtes ayant au moins une réponse retournées par notre système.
Notre investigation continue à ce jour afin d'identifier la cause de cette divergence tout en vérifiant que les calculs de temps et la récupération des réponses demeure correcte.