rapport2

Doha CHEMAOU

January 2022

1 Introduction

Avant de passer à la deuxième partie, je tiens à préciser que j'avais un code qui n'était pas optimal. J'ai dû le revisiter pour l'optimiser. Donc je vous explique l'idée derrière comment les reponses aux requetes sont calculées .
Lors de l'indexation, je crée une HashMap que j'appelle "po_s" qui a comme clé "predicate_object" et comme valeur une liste de sujets. Autrement dit, je range les sujets par predicat et objet.
Pour chaque requete, je récupères ce que j'appelle "branches" dans mon code où l'on va trouver predicat de la première condition , puis object de la première condition, puis predicat de la seconde condition, puis object de la seconde condition , ... ainsi de suite . Je parcours cette liste deux par deux; Quand il s'agit de la première condition, je regarde si po_s a un élément dont la clé sera predicat_object de cette première condition, si j'en trouve, je récupère la liste des sujets (cette liste sera la réponse aux requetes à une condition) . Je stockes ces sujets dans une liste que je nomme "subject". Passons à la condition 2 . Comme pour la première condition , je recupere la liste des sujets dont le predicat et l'objet sont ceux de la condition. Je stocke ces sujets dans une liste que je nomme "subject_". Je parcours la liste "subject". Si "subject_" contient l'élément lu en cours depuis "subject" je la rajoute à une troisième liste que je nomme "subject_common".
A la fin du parcours de la liste "subject" je fais cette affectation :

subject = subject_common;
subject_common=new ArrayList<>();

Ceci dit, il y a le cas ou "subject_" va etre null , c'est a dire que je ne trouve pas d'élément avec la clé voulue. Dans ce cas là, je fais cette affectation :

subject = new ArrayList<>();

Avant de passer à la condition suivante, je verifie si la liste "subject" est vide ; si c'est le cas, alors la requete n'a pas de reponse, j'arrete la boucle qui se fait sur els conditions, sinon , j'avance à la prochaine condition.
C'est ainsi que j'ai pu optimiser le calcul qui me prenais beaucoup de temps avant. Je vous invite à regarder la courbe de temps d'exécution du calcul des requetes par taille des données et par taille des requetes juste après.

2 Histogrammes

2.1 nombre des requêtes ayant au moins une réponse

2.2 requetes 1K

– 13 000 requetes au total

– 5000 requetes à 1 condition
– 4000 requetes à 2 conditions
– 3000 requetes à 3 conditions
– 1000 requetes à 4 conditions

– doublons sur les requetes à 1 condition : 3026 / 5000
– doublons sur les requetes à 2 condition : 784 / 4000
– doublons sur les requetes à 3 condition : 1725 / 3000
– doublons sur les requetes à 4 condition : 198 / 1000

————————————————————500K

requetes n'ayant pas de reponses : 4881 / 13 000
requetes ayant des reponses : 8119
–> 4 secondes au total pour repondre à toutes les requetes

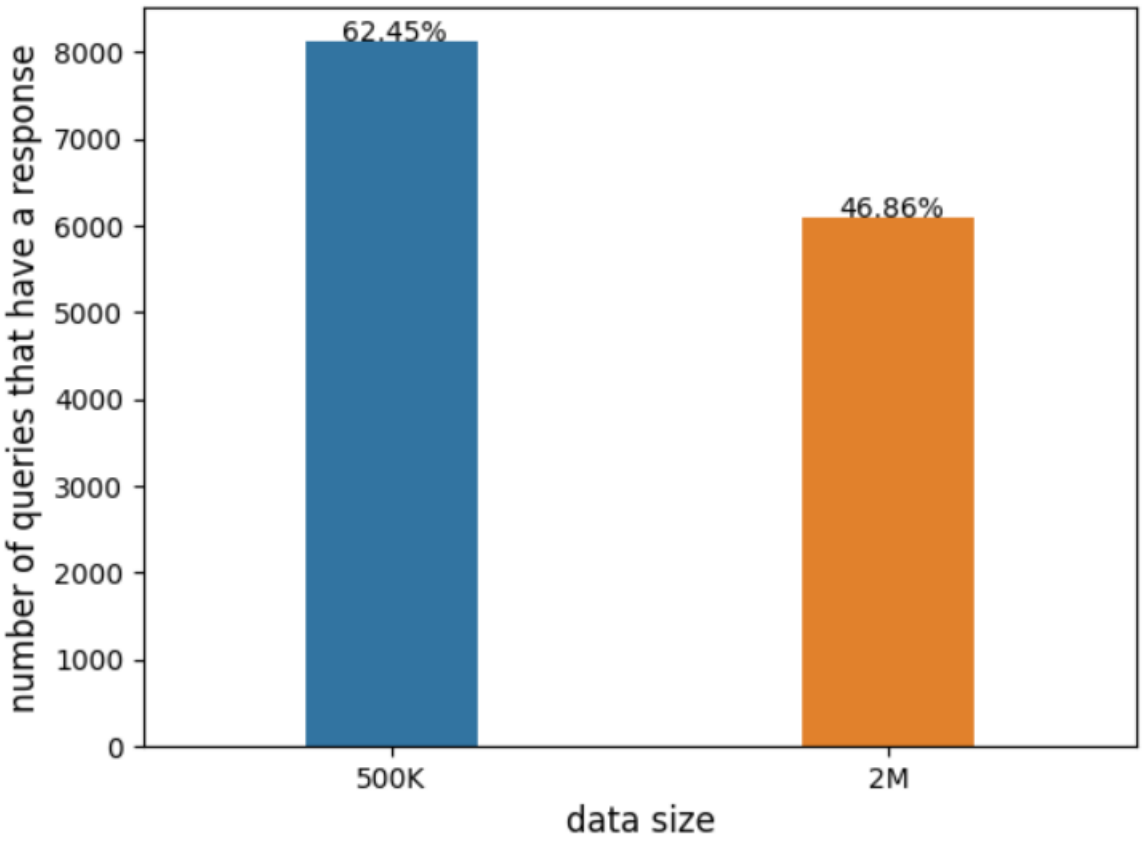
————————————————————2M

requetes n'ayant pas de reponses : 6908 / 13 000
requetes ayant des reponses : 6092
-> 9 secondes au total pour repondre à toutes les requetes

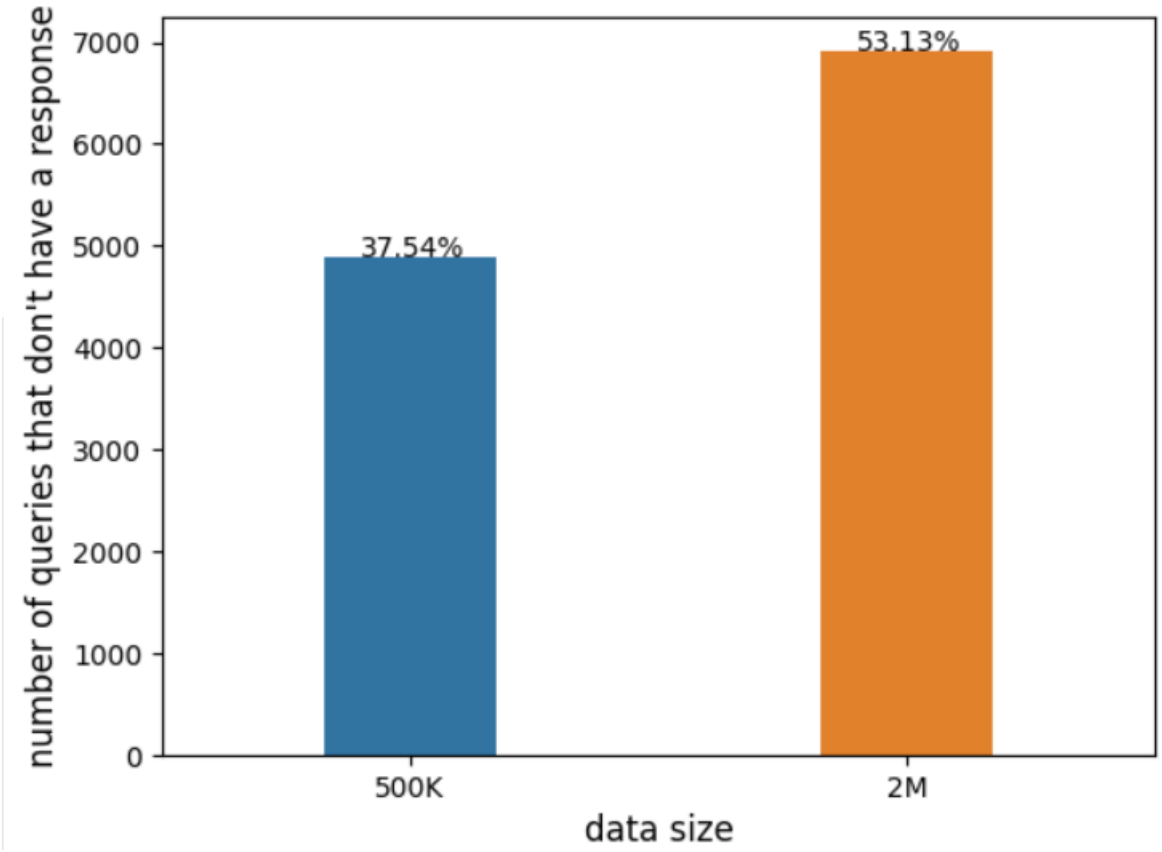
Avant de passer aux histogrammes, je tiens à préciser que le pourcentage sur les bars c'est pour montrer sur le nombre de requêtes qu'on a, à quel pourcent

s'élève le nombre de requêtes qui ont des réponses , qui n'ont pas de réponses , ... J'ai réalisé les histogrammes avec python .

2.2.1 nombre de requetes ayant au moins une reponse



2.2.2 nombre de requetes n'ayant pas de reponse



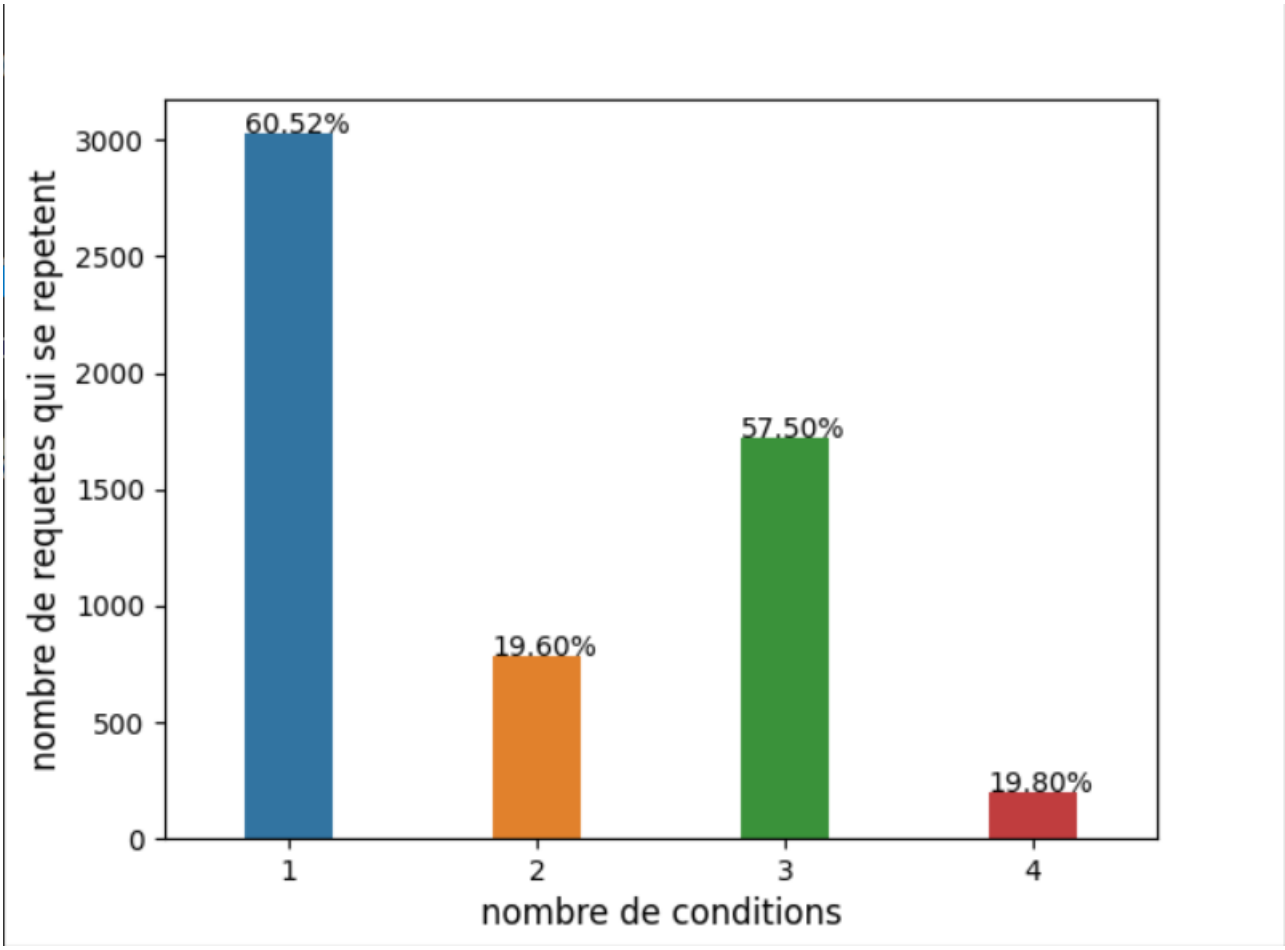
– 62.45% des 13 000 requetes ont au moins une réponse pour les données 500K .

– 37.54% des 13 000 requetes n'ont pas de réponse pour les données 500K .

– 46.86% des 13 000 requetes ont au moins une réponse pour les données 2M .

– 53.13% des 13 000 requetes n'ont pas de réponse pour les données 2M .

2.2.3 requêtes qui se répètent (classées selon le nombre de conditions)



– 62.52% des 5 000 requetes (ayant 1 condition) sont des doublons .

– 19.60% des 4 000 requetes (ayant 2 conditions) sont des doublons .

– 57.50% des 3 000 requetes (ayant 3 conditions) sont des doublons .

– 19.80% des 1 000 requetes (ayant 4 conditions) sont des doublons .

2.3 requetes 10K

– 130 000 requetes au total

– 50 000 requetes à 1 condition
– 40 000 requetes à 2 conditions
– 30 000 requetes à 3 conditions
– 10 000 requetes à 4 conditions

– doublons sur les requetes à 1 condition : 44 405 / 50 000
– doublons sur les requetes à 2 condition : 10 330 / 40 000
– doublons sur les requetes à 3 condition : 23 647 / 30 000
– doublons sur les requetes à 4 condition : 4718 / 10 000

————————————————————500K

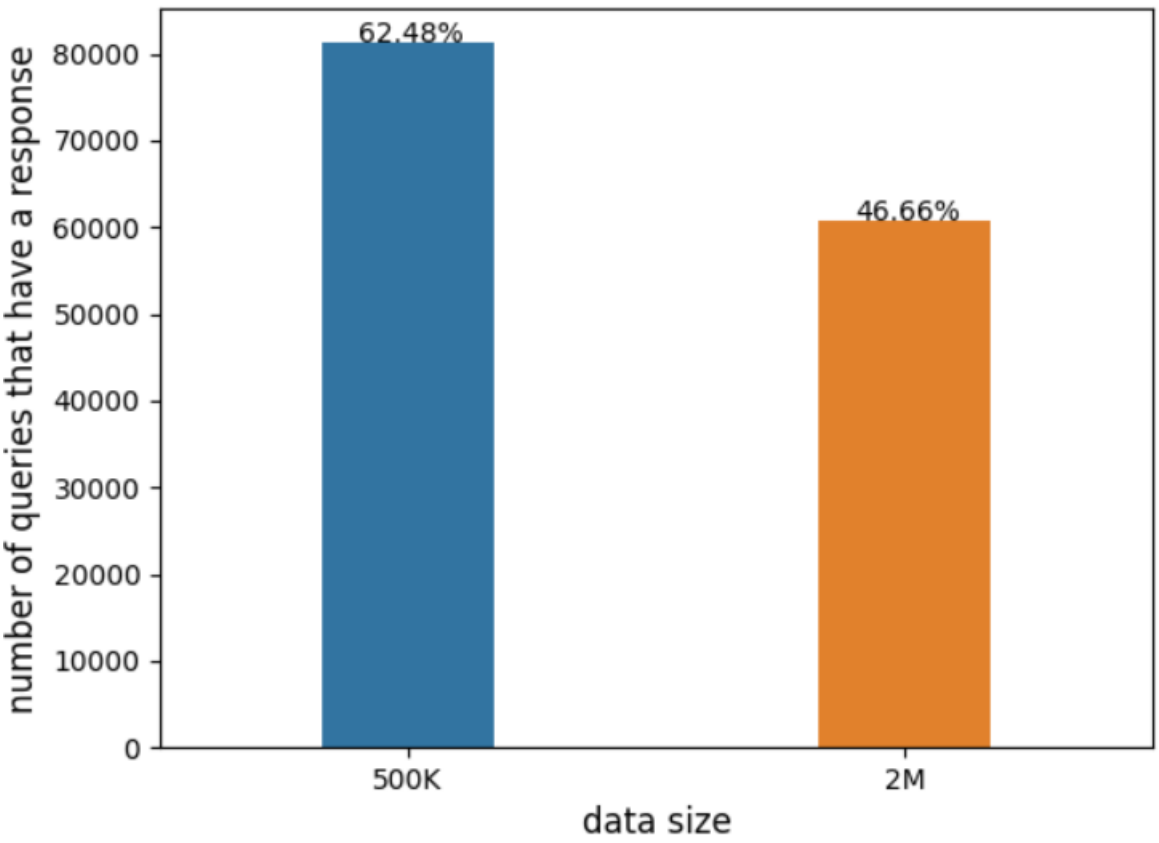
requetes n'ayant pas de reponses : 48 774 / 13 000

requetes ayant des reponses : 85 595
–> 82 secondes au total pour repondre à toutes les requetes

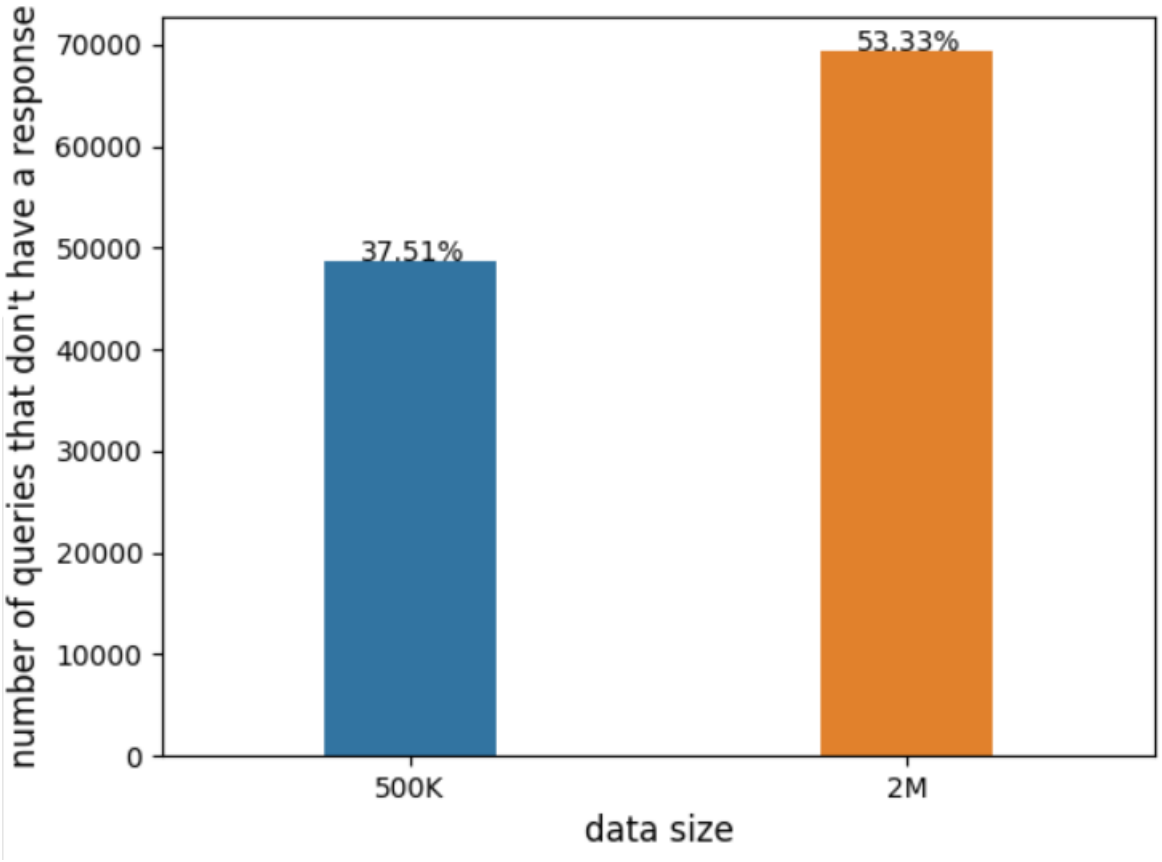
————————————————————2M

requetes n'ayant pas de reponses : 69 337 / 13 000
requetes ayant des reponses : 60 663
-> 109 secondes au total pour repondre à toutes les requetes

2.3.1 nombre de requetes qui ont au moins une reponse



2.3.2 nombre de requetes qui n'ont pas de reponse



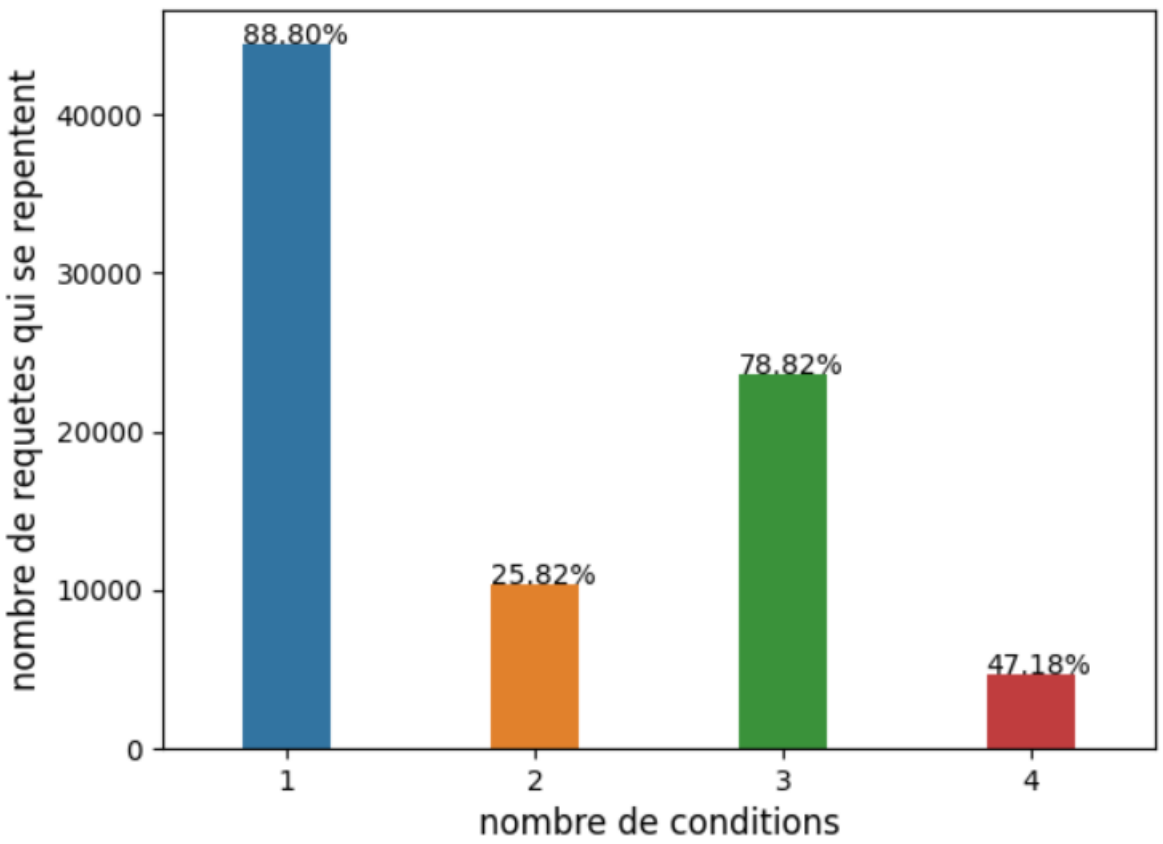
– 62.48% des 13 000 requetes ont au moins une réponse pour les données 500K .

– 37.51% des 13 000 requetes n'ont pas de réponse pour les données 500K .

– 46.66% des 13 000 requetes ont au moins une réponse pour les données 2M .

– 53.33% des 13 000 requetes n'ont pas de réponse pour les données 2M .

2.3.3 requêtes qui se répètent (classées selon le nombre de conditions)



– 88.80% des 50 000 requetes (ayant 1 condition) sont des doublons .

– 25.82% des 40 000 requetes (ayant 2 conditions) sont des doublons .

– 78.82% des 30 000 requetes (ayant 3 conditions) sont des doublons .

– 47.18% des 10 000 requetes (ayant 4 conditions) sont des doublons .

2.4 petite conclusion

– Les pourcentages sont presque les memes dans les deux cas.

– je remarque que les données 500K sont mieux placées comparé aux 2M (62% pour 500K contre 46% pour 2M qui ont au moins une reponse aux requetes .

– Je trouve quand meme que ce n'est pas bon pour le benchmark à peine la moitié des requetes ont une réponse pour les 500K alors que les 2M sont loins de la moitié.

– Beaucoup de doublons sur les requetes, ce qui laisse le moteur de recherche tourner sur les mêmes requetes.

– sur l'ensemble des requetes 10K, 63.92 des requetes sont des doublons.

– sur l'ensemble des requetes 1000, 44.10 des requetes sont des doublons.

– pour les deux cas, le nombre des doublons est plutot élevé, ce qui ralentit le moteur de recherche.

3 Quel type de hardware et de software allez vous utiliser pour vos tests ?

J'utilise un ordinateur portable HUAWEI avec 16 Go de RAM, 4 coeurs, 2.30 Ghz. J'utilise VisualStudio code sous Windows pour effectuer les tests.
—> Il est clair que plus l'ordinateur est puissant, plus il va être plus adapté pour analyser les performances du système. Aussi, un IDE joue un grand rôle au niveau de la détection d'erreurs. Il facilite aussi l'écriture du code et fait gagner du temps.
En ce qui concerne la machine sur laquelle l'analyse a été effectuée, je ne dirai pas qu'elle est adaptée à 100%, mais reste quand même acceptable pour effectuer l'analyse.
En ce qui concerne l'IDE, VisualStudio code n'est pas un ide reconnu pour la programmation en Java, mais il m'a été suffisant pour faire l'analyse.

4 Métriques, facteurs et niveaux

– Le temps de réponse : Le temps qu'il faut pour avoir une réponse à une requête.
– La qualité des réponses : le pourcentage des réponses justes et des réponses fausses
– Le débit : le nombre de requêtes qui peuvent être exécutées dans n'importe quel laps de temps .
– usage/nécessité : les ressources consommées par le système.
– Extensibilité :

* maintenance d'un bon temps d'exécution avec plus de données/utilisateurs.
* temps d'exécution réduit avec plus de ressources.

Les facteurs principaux sont le temps de réponse, la qualité des réponses ainsi que le débit.
Les facteurs secondaires sont usage/nécessité et extensibilité.