AUBRY QUENTIN
SOULIGNAC LORIANNE

UNIVERSITÉ TOULOUSE Jean Jaurès

# BENCHMARK

## Projet movie recommender

Années universitaire 2020/2021

# Sommaire

# Introduction

Dans le cadre du module portant sur les bases de données noSQL, nous avons dû réaliser un projet de base de données pour une start up victime de son succès et dont la base de données relationnelle ne convenait plus. Le projet consiste à comparer plusieurs solutions de système de base de données. Le CTO (Chief Technical Officer) a confié à une de ses équipes de développeurs l'installation, la migration et la comparaison (a l'aide d'un benchmark) de l'application à succès Movie Recommender.

Cette application est un haut lieu pour les amateurs du septième art. En effet, cette application permet d'accéder à une liste non exhaustive de films depuis les années 70. De plus, chaque utilisateur peut noter chacun de ces films afin de pouvoir lui recommander par la suite d'autres films du genre. Cette notation permet aussi de recommander des films aux autres utilisateurs de l'application et ainsi pouvoir recommander au mieux les films pour chaque utilisateur.

Cette application est encore en cours de développement. Mais elle est basée sur le framework Java orienté backend, Spring, ce dernier intègre de nombreuses fonctionnalités, notamment de sécurité informatique (comme la protection contre les attaques de type XSS ou bien encore d'injection SQL).

Le projet Movie Recommander est exécuté à l'aide d'un serveur tomcat 7 qui lui, permet de gérer l'ensemble des utilisateurs qui se connectent à l'application de manière parfaitement autonome.

Afin de pouvoir stocker et traiter les données des utilisateurs de l'application, Movie Recommender utilise une base de données relationnelle MySQL. Afin que Movie Recommender puisse accéder au contenu de cette base de données, l'application utilise un module de liaison JDBC.

Malgré son succès, de nombreux utilisateurs réguliers de l'application, le CTO,  ont fait part du manque de réactivité de l'application, et la difficulté à gérer cette infrastructure (notamment en termes de rapidité d'exécution). C'est pour cela que la totalité de l'application Movie Recommender doit être révisée afin de pouvoir utiliser le système de gestion de données orienté graphes, Neo4j. En parallèle sera développée la version de Movie Recommender utilisant le système de gestion de données orienté documents MongoDB.

Ces deux versions d'applications permettent donc de mettre en lumière les divers points positifs ou négatifs des deux solutions de réadaptation de Movie Recommender a l'aide d'un benchmark réalisé avec l'application Apache JMETER. Cet outil permet notamment de simuler des connexions au service de l'application et de pouvoir réaliser des rapports (notamment sur les erreurs de connexions, les temps de latence en fonction de la charge, …) sur les deux solutions de réadaptation et pouvoir ainsi prendre un choix optimal pour la suite.

# Méthodes utilisé concernant le benchmark

Comme énoncé précédemment, nous avons choisi la solution de benchmarking Apache JMETER. Cette solution nous permet d'émuler des connexions aux services backend de l'application Movie Recommender. Afin de pouvoir au mieux se conformer à la réalité de l'utilisation de cette application, nous avons choisi de séparer notre benchmark en deux point :

- Le premier est de simuler une utilisation "normale" de 1000 utilisateurs qui font les mêmes actions afin de faire monter en charge de l'application (c'est-à-dire sans simuler une connexion multiple d'un ou de plusieurs utilisateurs).
- La deuxième est de simuler en plus de la première, une connexion multiple d'un ou de plusieurs utilisateurs à l'application (c'est à dire de simuler un utilisateurs qui utilise plus l'application que d'autres)

## Mise en place dans JMETER

Notre processus JMETER est découpé en 3 actions qui sont :

- Récupérer les valeurs d'un fichier de données contenant les identifiants des utilisateurs (fichier disponible via internet : users.csv).
- Pour chaque valeur de ce fichier récupéré, l'identifiant d'un utilisateur est comparé à des identifiants pré-définis dans l'action JMETER et celui-ci est utilisé afin de simuler une connexion supplémentaire.
- Simuler une montée en charge de l'application avec des utilisateurs générés aléatoirement par JMETER.

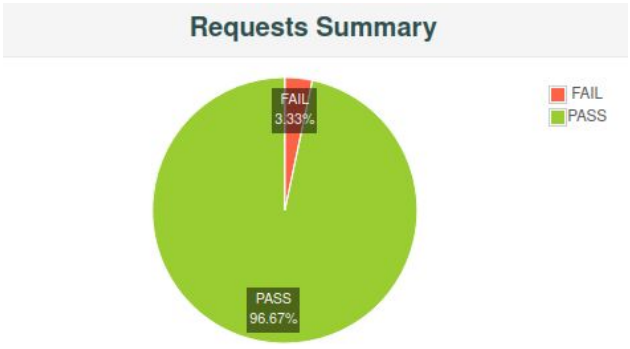
## Méthodes non prises en charges concernant le benchmark

Dans notre processus de benchmark, nous n'avons pas pris en compte les points de test ci-dessous:

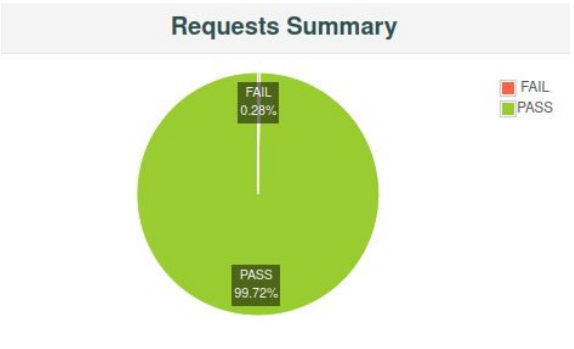
- Le suivi des utilisateurs de leurs recommandations, car malgré les diverses aides externes que l'on as pu avoir, nous n'avons pas su implémenter correctement la méthode de recommandations des films dans la solution Movie Recommender. Par conséquent nous avons décidé de réaliser le benchmark sur la fonctionnement retournant l'ensemble des films à l'utilisateur.
- Nous n'avons pas pris en charge la création, suppression ou modification d'éléments durant notre benchmark avec JMETER car nous n'avons pas implémenté cette fonctionnalité dans notre solution.

# Résultat du benchmark

## Résultats des requêtes http




neo4j




mongoDB

On peut voir sur ces graphiques que l'utilisation de neo4j entraîne plus souvent des échecs sur les requêtes réalisées par des utilisateurs lors de fortes charges. En effet, il y a 3,33% d'échec pour neo4j, alors qu'avec mongoDB il y moins de 1% de personnes qui n'ont pas pu accéder à leur informations. Cependant, la différence n'est pas vraiment significative car un taux de 3,33% d'erreur peut être négligeable au même titre qu'un taux d'erreur de 1%.

## Performance de l'application

APDEX (Application Performance Index)

| Apdex | T (Toleration threshold) | F (Frustration threshold) | Label |
|---|---|---|---|
| 0.065 | 500 ms | 1 sec 500 ms | Total |
| 0.065 | 500 ms | 1 sec 500 ms | HTTP Request |

neo4j

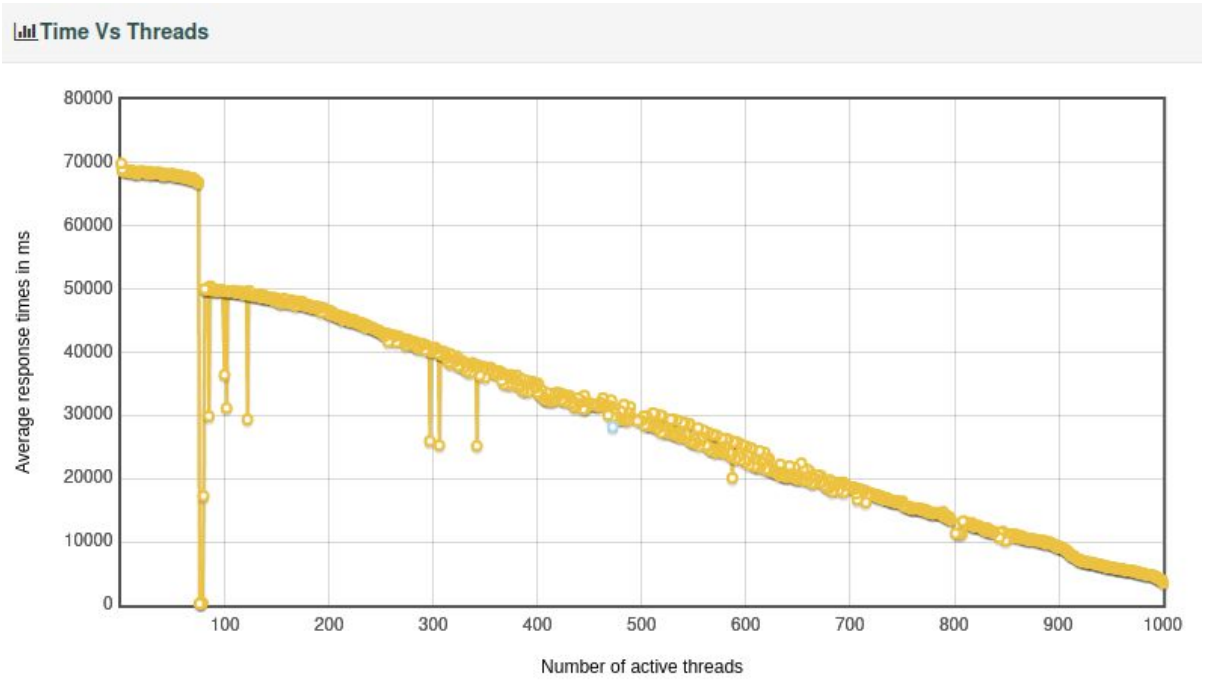
APDEX (Application Performance Index)

| Apdex | T (Toleration threshold) | F (Frustration threshold) | Label |
|---|---|---|---|
| 0.065 | 500 ms | 1 sec 500 ms | Total |
| 0.065 | 500 ms | 1 sec 500 ms | HTTP Request |

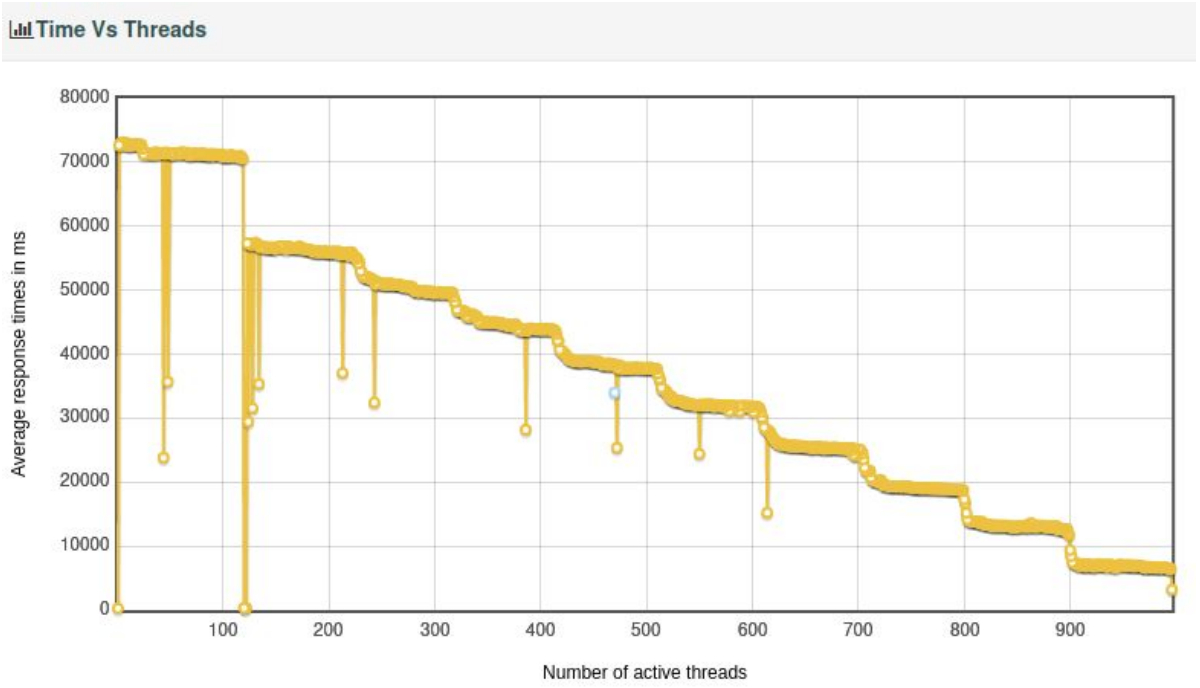
mongoDB

On peut voir sur ces graphiques que l'utilisation de neo4j ou bien mongoDB n'impacte en aucun cas les performances de l'application. Chaque application a un ratio d'"APDEX" (Application Performance Index) de 0,065. Toutes les valeurs sont exactement les mêmes donc il n'y a pas de résultats significatifs.

## Temps de réponses en fonction des utilisateurs connectés




neo4j




mongoDB

On peut voir sur ces graphiques, pour Neo4j, lorsque le nombre d'utilisateurs est peu élevé (~70 utilisateurs), ce dernier met jusqu'à 70000 ms a retourné la valeur mais plus le nombre d'utilisateurs connectés est élevé plus les données sont retournées rapidement. L'ensemble des retours des données est globalement descendant mais on peut constater la présence de paliershomogène descendante. Concernant mongoDB, lorsque le nombre d'utilisateurs est peu élevé (~ 70 utilisateurs), ce dernier met plus de 70000 ms a retourné une valeur. Plus le nombre d'utilisateurs évolue, plus les requêtes sont traitées rapidement. L'ensemble de ses données suit un mode une courbe en escalier qui stagne entre chaque paliers.

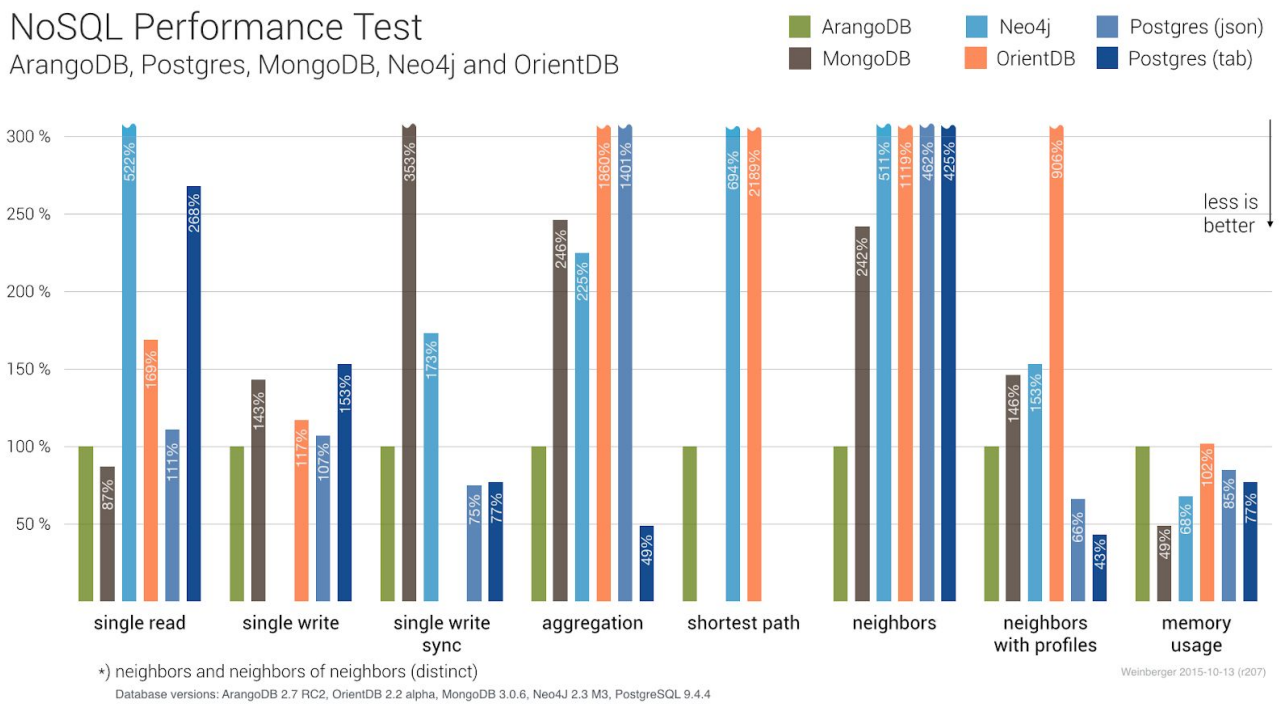
# Questions récurrentes posé par le CTO

## Comment garantir une comparaison des performances égales ?

Lors de notre benchmark nous avons mis en place un outil qui permet de ne pas affecter les capacités externes. Cet outil se prénomme JMETER et a été explicité plus haut. Nous avons décidé d'utiliser cet outil afin de ne pas biaiser les comparaisons avec une erreur humaine. L'outil teste les ressources de la même manière car il simule un nombre de requêtes envoyées, elle monte en charge l'application. Cette montée en charge va aussi faire monter en charge notre système de gestion de données. L'outil va effectuer le même test pour les deux systèmes. L'outil a été configuré de la même manière pour les tests de noe4j et mongoDB, cela assure une entière égalité dans les tests. De plus les deux tests ont été réalisés sur la même machine avec une connexion égale, cela aussi assure l'égalité des tests de performances.

## Dans quelle mesure cela peut-il fausser les résultats ?

Actuellement, Movie Recommender n'intègre pas la possibilité de création, d'édition, et de suppression d'éléments (utilisateurs, films). Ces fonctionnalités peuvent impacter le benchmark, car les systèmes de gestion de données (graphe et document ), ont leur propre manière de fonctionner, c'est-à-dire, que l'accès aux informations, la suppression, modification peut varier entre les système de données .




Comparaison de performance entre plusieurs systèmes de données

# Choix de la solutions retenue

Suite aux analyses des différents graphiques que nous avons pu récolter, il n'y a pas de différence réellement significative qui nous permettrait de prendre une réelle décision, mais après analyse du graphique ci-dessus(Comparaison de performance entre plusieurs systèmes de données), notre décision se tournerait plus vers neo4j car il semblerait qu'elle est en moyenne de meilleures performances que mongoDB sur plusieurs cas de figure. Cependant, nous avons également essayé de rechercher des avis d'utilisateurs expérimentés sur différents forum afin de trouver des témoignages sur le sujet. Suite à cette recherche, notre choix de neo4j a été appuyé car il semblerait, selon plusieurs avis, que pour réaliser des recommandations , la technologie neo4j est réellement performante.

# Idées d'évolution concernant les performances

- Modifier la structure des données afin de faciliter et améliorer la rapidité lors des requêtes d'accès.
- Mettre en place une architecture qui s'adapte aux besoins. Par exemple, si il y a une montée en charge l'architecture permettra d'augmenter les performances et fera l'inverse si les charges sont moindres.
- Utiliser un ORM afin de limiter les risques liés à l'accès des données de façon manuelle. Par exemple, si le développeur envoie deux requêtes au lieu d'une seule nécessaire.