NF26 : Rapport projet Cassandra

Virgile Vançon

Université de Technologie de Compiègne

Table des matières

1 Introduction

Ce rapport présente le travail effectué au cours du projet de NF26 avec la plateforme Apache Cassandra, qui est une plateforme de gestion de base de données NoSQL. Ce projet a consisté en l'analyse d'un ensemble de données recueillies auprès d'une entreprise de taxis au Portugal, ainsi que son insertion dans une base de données Cassandra. Les informations regroupent les descriptions des trajets réalisés par 442 taxis au sein de la ville de Porto. Les données datent du 11 juillet 2015 et comptent 1710671 instances.

Ce compte-rendu commencera par décrire les données mises à notre disposition par l'entreprise de taxis, puis présentera les différentes techniques misent en place pour le traitement de ces données. Dans un troisième temps, les résultats analysés des traitements précédents seront présentés.

L'objectif de ces traitements est de fournir à l'utilisateur des bases de faits possédant différents indicateurs utiles à la gestion de l'entreprise de taxis. Ils pourront par exemple être utilisés afin d'améliorer le service de la compagnie en plaçant davantage de taxis dans les lieux stratégiques, qui auront étés déterminés par analyse des données fournies. La finalité est donc de pouvoir extraire des connaissances utiles à partir des résultats obtenus.

Le processus mis en oeuvre est composé d'une modélisation de l'entrepôt de données puis de la conception et de l'alimentation de ce dernier avec la plateforme Cassandra. Les langages utilisés seront le Python ainsi que le CQL. Enfin, nous tenterons d'extraire des indicateurs en utilisant des méthodes de reporting telles que la méthode des k_means, ou la régression afin de pouvoir réaliser de l'informatique décisionnelle.

Remarque : Ce projet a été réalisé en binôme avec Laura BROS. Nous avons donc effectué le même travail, en se basant sur la même modélisation. Les scipts fournis en annexes sont donc identiques, et nos rapports semblables.

2 Description des données

Les données à notre disposition décrivent les trajets effectués par 442 taxis au cours d'une année civile, et est stocké dans un fichier de type CSV. Les courses réalisées par ces taxis sont séparées en trois catégories :

- A : la course a été dispatchée depuis la centrale de taxis. Dans ce cas, un id anonyme est stocké s'il a pu être récupéré pendant l'appel du client.
- B : la course a été demandée directement par un client dans les stands réservés aux taxis.
- C : la course a débutée dans une rue quelconque à la demande d'un client.

Une course est décrite par 9 caractéristiques, c'est-à-dire 9 colonnes dans le fichier CSV :

- TRIP_ID (string) : identifiant unique pour chaque trajet de type chaîne de caractères.
- CALL_TYPE : caractère qui défini le type de trajet (A, B, ou C expliqués ci-dessus)
- ORIGIN_CALL : identifiant unique de type entier qui stocke le numéro de téléphone utilisé pour demander un trajet. Dans le cas où le type de trajet est 'A', le numéro de téléphone se réfère au numéro du client qui a appelé. Dans les deux autres cas, ce paramètre prend la valeur NULL.
- ORIGIN_STAND : identifiant unique de type entier qui permet de localiser le stand d'origine du trajet, dans le cas où le type de trajet est 'B' bien-sûr. Dans les deux autres cas, cet identifiant prend la valeur nulle.
- TAXI_ID : entier contenant un identifiant unique pour le conducteur du taxi qui réalise le trajet.
- TIMESTAMP : entier qui contient la date de début du trajet (en secondes).
- DAYTYPE : caractère qui identifie le type de jour du trajet. Ce caractère peut prendre les valeurs 'A' si le trajet a été réalisé un jour "normal", 'B' si il a été effectué pendant les vacances ou pendant un jour férié, et 'C' si ce trajet a eu lieu la veille d'un jour spécial de type 'B'.
- MISSING_DATA : indicateur booléen permettant de savoir si il manque des données dans le champ POLYLINE.
- POLYLINE : chaîne de caractères contenant la liste des coordonnées GPS relevées toutes les 15 secondes pendant le trajet. Cette chaîne de caractère contient une liste de couples [Longitude, Latitude] entre crochets, qui indiquent les différentes positions prises par le taxi durant le trajet. Cette liste commence par la position de départ et se termine par celle d'arrivée.

3 Traitement des données

3.1 Constitution de l'entrepôt de données

Le processus à modéliser est le suivi des trajets effectués par des taxis dans la ville de Porto au cours d'une année civile. Le niveau de granularité choisi est un trajet. Pour la modélisation, plusieurs dimensions ont étés définies :

- une dimension TEMPS : cette dimension correspond à l'identification d'une date décrite par l'année, le mois, le jour, l'heure, les minutes et les secondes.

— une dimension LIEU : permet de localiser un point grâce à sa latitude et à sa longitude.
— Dimension Taxi_ID : permet de référencer les identifiants des taxis.

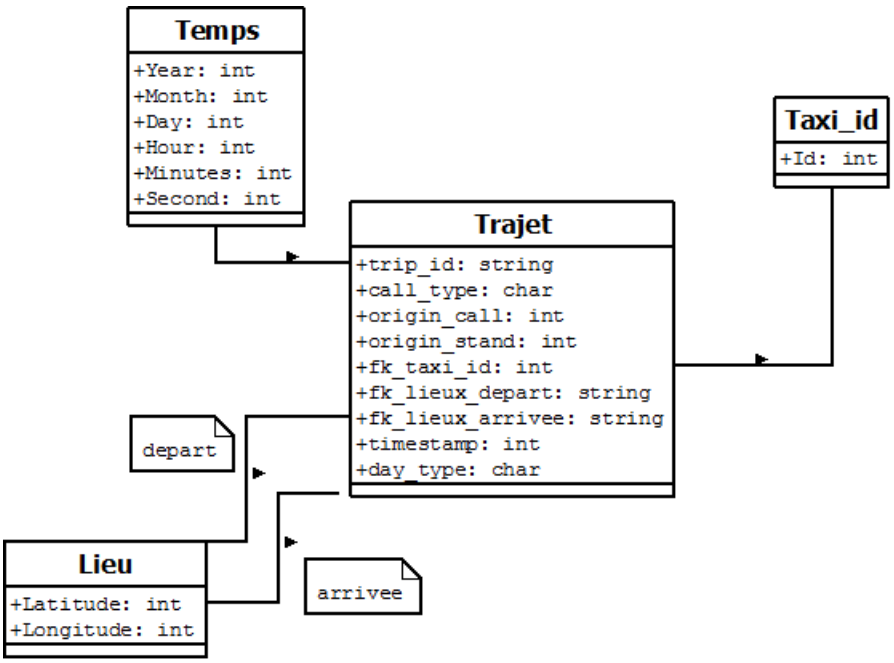

Figure 1. Modélisation de la table de faits et de ses dimensions

La table centrale est la table des faits. Elle regroupe l'ensemble des faits et des indicateurs utiles à l'utilisateur. Au niveau de la modélisation et à ce niveau seulement, cette table possède des clés étrangères vers les différentes dimensions, afin de présenter un modèle normalisé. La suite de ce rapport présente la *dénormalisation* de ce modèle, afin d'être en accord avec les problématiques soulevées par la manipulation de gros volumes de données.

3.2 Alimentation de l'entrepôt de données

Les dimensions présentées dans la partie précédente ne seront pas matérialisées dans la base de données Cassandra. En effet, la taille jeu de données est trop importante pour permettre l'utilisation de différentes dimensions, différentes tables et clés étrangères qui imposeraient des jointures coûteuses en temps et en ressources. Nous avons donc décidé de dénormaliser le schéma obtenu ci-dessus dans le but de n'obtenir que des tables de faits et donc un système complètement centralisé. Nous avons réintégré les différentes dimensions présentées auparavant aux différentes tables de faits. Ces différentes tables de faits possèdent quasiment toutes les mêmes attributs, mais diffèrent par leurs clés de partitionnement et de clustering.

Dans nos tables de faits, chaque trajet possède un point de départ, un point d'arrivée, et un indicateur de la distance du trajet. Les points de départ de tous les trajets nous ont permis de créer des pavés dans la ville de Porto, correspondant à des zones géographiques. Nous avons utilisés pour cela la méthode des K-Means, qui est une méthode de classification non-supervisée. Cette méthode permet, à partir de l'ensemble des points de départ, de créer un ensemble de K classes (K étant fixé par l'utilisateur, 5 dans notre cas) caractérisées par un *centroïde*. Chaque point de départ est alors rattaché au centroïde le plus proche. L'ensemble des classes et de leurs centroïdes sont stockés dans une table annexe. Il est important de noter que le raisonnement que nous avons appliqués pour les points de départ est parfaitement valide pour les points d'arrivées, mais le temps nous a manqué pour faire les 2, et il nous a paru plus judicieux de privilégier les points de départ. Ce raisonnement aurait également pu être appliqué aux vecteurs des trajets, afin d'obtenir des trajets types dans la ville.

Les tables de faits Les tables de faits possèdent les attributs suivants :

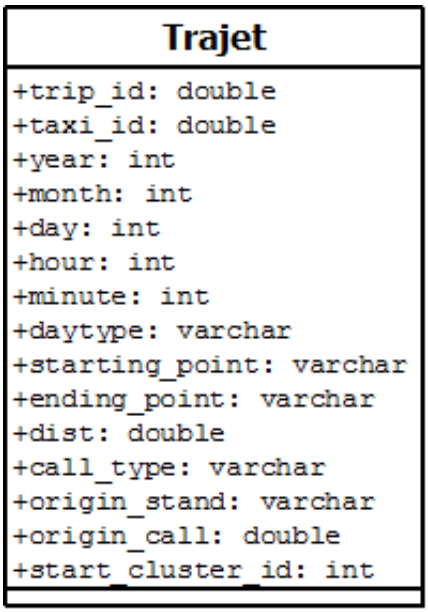

Figure 2. Table des Faits

Nous avons choisi de matérialiser deux tables de faits possédant les mêmes attributs. Ces tables diffèrent au niveau de leurs clés de partition et de clustering. Les clés de partition permettent de découper une table ou un index par rapport à des critères logiques. La table se comporte alors comme un ensemble de tables de dimensions plus petites. Cassandra assure que toutes les lignes d'une partition sont stockées ensemble. Le partitionnement de tables est généralement utilisé pour améliorer la gestion, la performance ou la disponibilité des données. Chaque partition se retrouve sur des serveurs ou des disques différents. Cela permet également d'obtenir une capacité de base de données supérieure à la taille maximum des disques durs ou d'effectuer des requêtes en parallèle sur plusieurs partitions. Les performances de mise à jour peuvent aussi être améliorées car chaque pièce de la table a des index plus petits que ne le serait l'index de l'ensemble complet des données. Les clés de clustering trient et stockent les lignes de données de la table en fonction de leurs valeurs de clé. Ce sont les colonnes incluses dans la définition de l'index. Il n'y a qu'un *index cluster* par table car les lignes de données ne peuvent être triées que dans un seul ordre.

Notre première table de faits permet de faire des requêtes temporelles : elle possède donc une primary key dont la clé de partitionnement contient l'année, le mois et le jour. La clé de clustering est composée de l'heure, des minutes et de l'Id du trajet (trip_Id, car 2 trajets peuvent avoir lieu en même temps). Ce choix d'implémentation nous permet de faire une recherche rapide sur la date du début du trajet.

Notre seconde table de faits (elle possède les mêmes attributs que celle décrite ci-dessus) possède une clé de partitionnement qui contient l'Id du pavé ainsi que l'année. L'heure, les minutes ainsi que l'Id du trajet définissent la clé de clustering. Cette clé primaire permet d'accélérer les requêtes de recherche par zone géographique.

Toutes les tables de faits ne seront pas présentées ici, mais le code de création des tables est fourni en annexe.

La table des clusters

La table des pavés est obtenue grâce à l'implémentation de la méthode des k_means qui calcule les centres des différentes zones géographiques de notre ensemble de points. Les résultats sont stockés dans une table, et des clés étrangères sont utilisés dans la tables de faits pour faire référence à ces pavés. Cette table est définie de la manière suivante :

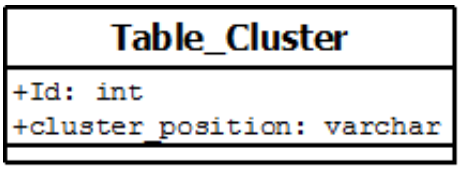

Figure 3. Table des Cluster

Cette table est utilisée pour l'insertion des données dans la table de faits permettant de requêter sur les zones géographiques, dans lesquels sont stockés les identifiants des centroïdes et non pas leurs coordonnées, qui sont

bien plus logiques et faciles à manipuler. Ces informations nous permettront par la suite de pouvoir raisonner par rapport à des zones géographiques.

4 Résultats et Analyses

L'objectif de cette partie est de pouvoir déterminer la typologie des trajets. Pour ce faire, nous avons tenté de les regrouper en classes. Le processus ne revient pas à affecter chaque trajet a une classe mais d'analyser les classes afin de réaliser un résumé des données.

Chaque centre a été obtenu grâce à la méthode des k_means. C'est une méthode itérative, qui prend en paramètre K qui est le nombre de centres a créer (5 dans notre cas), et qui commence par sélectionner alétoirement K centres. Ensuite, chaque point est affecté au centre le plus proche et les centres sont recalculés, et ce jusqu'à convergence. C'est une méthode efficace sur des jeux de données importants car elle converge très vite. Cet algorithme permet donc de résumer chaque classe à un centroïde.

Départs

Nous avons obtenu les centroïdes correspondant aux points de départ suivants :

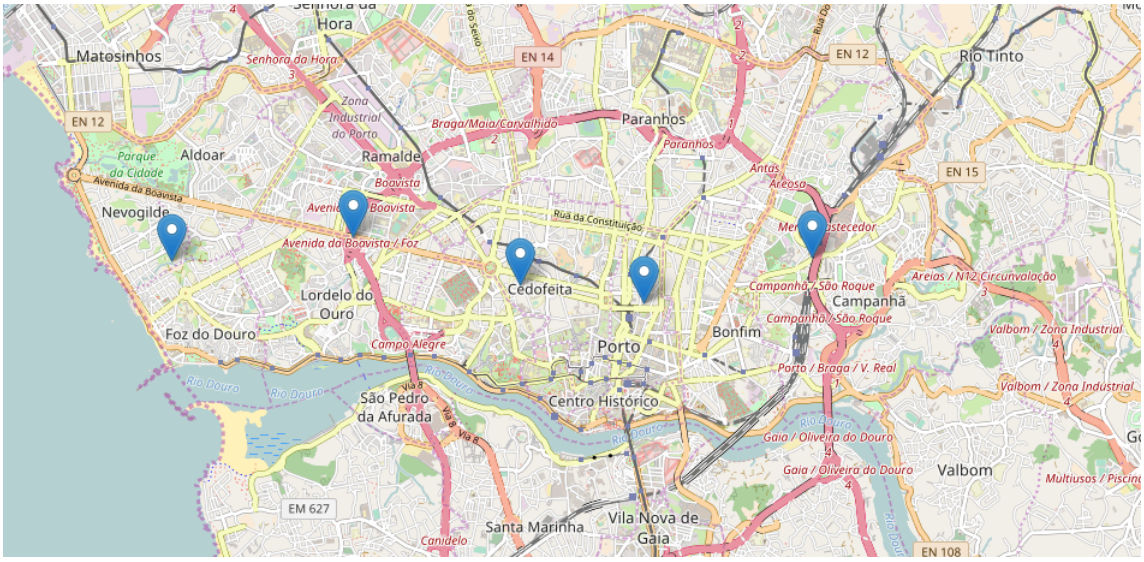

Figure 4. Représentation des centres des classes sur un carte

Nous pouvons désormais fournir à l'entreprise une information concernant les lieux de départs fréquents. Ainsi, ils pourront être capables de placer davantage de taxis dans ces zones et maximiser le nombre de courses en demandant aux taxis de revenir proche des centroïdes de chaque classe. Les centres sont placés dans le centre historique de Porto (très visités par les touristes), près de la gare, près d'un parc fréquenté à l'extérieur de la ville, etc... et constituent donc un ensemble logique de points stratégiques pour les taxis.

Scénarii

En nous positionnant à la place de l'utilisateur, soit le responsable de l'entreprise de taxis, nous avons imaginé différents scénarii. Notre travail est censé aidé l'entreprise de taxis à prendre des décisions stratégiques, concernant les périodes les plus productives, les meilleurs emplacements par exemple. L'objectif final est alors de réduire les coûts, d'augmenter la productivité des chauffeurs, d'augmenter le chiffre d'affaire de l'entreprise, etc...

Requêtes sur les périodes les plus chargées : Afin d'optimiser le nombre de courses, il serait judicieux d'adapter le nombre de taxis en service dans la ville en fonction du nombre de services demandés. Nous avons défini trois cas différents :

— définir les horaires les plus chargés
— déterminer les jours de semaines les plus chargés
— déterminer les mois avec le maximum de trajets

Requêtes sur les lieux de départs des trajets Il peut être intéressant d'étudier si les trajets ont davantage tendance à débuter à la centrale suite à un appel de clients, dans les stands, ou dans la rue. Ainsi, l'entreprise pourra adapter sa répartition de taxis en donnant la consigne de revenir ou non à la centrale. Pour ce faire, nous avons commencé par créer une nouvelle table de faits dont la clé de partitionnement contient l'année et ORIGIN_STAND afin de pouvoir effectuer un GROUP BY sur ce paramètre puis nous avons requêté cette nouvelle table et compté le nombre de 'A', 'B' ou 'C' pour le paramètre ORIGIN_STAND qui renseigne cette information.

Requêtes sur le type de jour : Cette fois, l'entreprise décide de connaître la demande de trajets lors des jours particuliers. En effet, elle souhaite savoir si, même lors de jours fériés, il est utile de mettre en service un nombre important de taxis. Elle voudrait également savoir si les gens ont tendance à demander utiliser le service de taxis la veille des jours de vacances ou des jours fériés, pour se rendre à l'aéroport ou dans les gares par exemple.

De la même manière,nous avons du implémenter une nouvelle base de faits, possédant à nouveau les mêmes renseignements mais utilisant le DAYTYPE comme clé de partitionnement ainsi que l'année. Nous avons donc interrogé la table de faits en comptant le nombre de trajets effectués lors de jours de type B (jours fériés ou de vacances) et le nombre de trajets effectués lors de jours de type C (jours qui ont lieu la veille des jours de type B).

Requêtes sur les taxis effectuant le plus de trajets : Enfin, l'entreprise désire récompenser les taxis les plus efficaces, autrement dit, ceux réalisant le plus de trajets et réalisant les distances les plus longues. L'idée est donc requêté la base de faits en comptant le nombre de trajets par taxis, et de trier en parallèle les taxis en les ordonnant par leur moyenne de distances parcourues en ordre décroissant. Cette table de faits possède alors taxi_id comme clé de partitionnement.

Deux classements sont possibles :

— On commence par trier par nombre de trajets pour chaque taxis, puis par distances parcourues en moyenne pour chaque trajet.

— On commence par trier par distances parcourues en moyenne par trajet puis on compte le nombre de trajets pour chaque taxi.

Malheureusement, nous n'avons pas eu le temps de lancer le script des requêtes et d'analyser les résultats. Ils nous auraient permis d'imaginer les décisions stratégiques à prendre pour l'entreprise.

5 Conclusion

Ce projet nous a permis de nous positionner dans un cas réel de traitement de données. En effet, nous avions à notre disposition une quantité importante de données issuées de cas pratiques. Les données avaient un sens et nécessitaient donc un traitement réfléchi afin de pouvoir en tirer des connaissances performantes.

Nous avons utilisé des connaissances issues d'autres UVs comme SY09 pour implémenter l'algorithme des K_Means par exemple, ce qui montre à quel point l'informatique décisionnel est un domaine complet. Nous avons pu découvrir l'outil Cassandra et nous avons eu l'occasion de programmer en Python ainsi qu'en CQL. Ce TP nous a montré une première approche de l'application des méthodes dans le monde des fouilles de données.