UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ARCHITECTURE D'UN SERVICE D'ANALYSE POUR CDN BASÉ SUR LE CLOUD : CAS DE PRÉDICTION DE POPULARITÉ

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

MAROI ALOUI

OCTOBRE 2020

REMERCIEMENTS

Je tiens à remercier très sincèrement ma directrice de recherche, professeure Halima Elbiaze, pour sa disponibilité, sa grande patience et surtout ses précieux conseils tout au long de mes travaux de recherche.

Je voudrais aussi exprimer mes gratitudes à mes collègues au TRIME pour leur soutien et leur support et pour le bel esprit d'équipe que nous avons toujours au sein de notre laboratoire. Je désire aussi remercier spécialement mon mari pour le soutien qu'il ne cesse de m'apporter et pour son encouragement à terminer mes études.

J'exprime enfin toute ma gratitude à ma famille : mes très chers parents qui m'offrent leur amour infini, qui sont toujours avec moi, et qui ne cessent de me pousser à être la meilleure ; ainsi que mes frères, mes sœurs et mes amis.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

RÉSUMÉ

Ce mémoire propose une nouvelle approche servant à prédire le niveau de popularité des vidéos générées par les utilisateurs en temps réel. Cette approche se base sur une architecture d'analyse des données en tant que service couramment appelé *Analytics as a Service* (AaaS). La nouvelle architecture proposée utilise les services web *RESTful* pour rassembler les traces produites par les utilisateurs sur les réseaux de distribution de contenu (CDN : *Content Delivery Network*), et les stocker dans des formats génériques dans une base de données NoSQL avant d'en calculer la popularité. La collecte des traces des visualisations des vidéos (*logs*) se fait à partir des CDNs basés sur l'infonuagique (*Cloud*). La solution choisit les caractéristiques nécessaires depuis ces (*logs*) pour former un modèle regroupant les vidéos en fonction de leur niveau de popularité pour ensuite envoyer une liste des vidéos les plus populaires nécessaires au CDN. La popularité d'une vidéo est calculée à l'aide d'un modèle de classification préétabli. Ce modèle est construit hors ligne à l'aide d'un algorithme de partitionnement de données (*Clustering*) qui effectue l'entraînement des données récentes disponibles dans la base des données.

L'architecture *RESTful* proposée est adaptée au fonctionnement du réseau CDN grâce à sa facilité de mise en place et de déploiement sur le web. Cette architecture utilise des techniques de paradigme des données massives (*Big Data*) pour traiter des millions de données. Ces données peuvent avoir différentes structures. L'algorithme de *Clustering* n'effectue aucune réduction de dimensions préalable, contrairement à d'autres solutions qui présélectionnent les dimensions comme la régression linéaire. Cet avantage rend la solution générique et augmente sa fiabilité en évitant les risques d'une mauvaise réduction de dimension. Nous avons évalué notre solution à l'aide d'une base de données YouTube ayant plus de 10 millions de données sur des vidéos (temps de visualisation, nombre de vues, nombre de partages de la vidéo, etc.). Notre technique d'analyse de données permet de prédire la popularité des vidéos avec une précision de 99,8 % et un temps d'exécution de moins de 2 secondes.

Mots clés : CDN, Infonuagique, Web service RESTful, Algorithme de clustering.

INTRODUCTION

0.1 Motivation

Notre utilisation quotidienne du web ne cesse de s'accroître, entrainant ainsi une augmentation exponentielle du trafic sur Internet. Au Québec, par exemple, une personne passe en moyenne plus de vingt-deux heures hebdomadaires à consommer du contenu web. C'est donc une quantité gigantesque de données qui sont transférées sur le réseau mondial chaque jour. De plus, les nouvelles utilisations d'Internet, telles que les réseaux sociaux et les jeux en ligne, exigent une connexion ultrarapide, ainsi qu'une importante bande passante. Tout cela demande une infrastructure puissante capable de supporter cette tendance.

Les réseaux de distribution de contenu font partie de cette infrastructure. Le CDN rend le contenu d'un site d'hébergement plus accessible aux utilisateurs, et ce, quel que soit son emplacement dans le monde. Pour ce faire, le CDN duplique l'information sur plusieurs serveurs installés sur toute la planète. L'utilisateur est alors dirigé vers la copie la plus proche de lui, en fonction de son emplacement géographique. Le CDN met alors l'information en cache et distribue le contenu web aux utilisateurs à partir des serveurs d'origine, selon leur emplacement. Le but est d'accélérer l'accès aux données nécessaires avec une meilleure qualité de service et une meilleure qualité d'expérience. Dupliquer systématiquement toutes les informations pourrait être très coûteux en termes de taille de stockage et de bande passante. Il serait donc plus intelligent de faire une duplication sélective de l'information et d'optimiser ainsi l'utilisation de l'infrastructure d'un réseau CDN. Cela constitue la motivation principale de notre travail.

Notre objectif est de proposer une solution intelligente de sélection du contenu qu'un réseau CDN doit considérer en priorité lors de la duplication de contenu sur les différents serveurs (Aloui *et al.*, 2018). Les statistiques mondiales (Planetoscope) donnent des valeurs très élevées pour le visionnage des vidéos partout sur la plateforme de vidéos YouTube [1] . Ainsi, chaque jour, près de 4 milliards de vidéos sont vues sur YouTube par les utilisateurs. En conséquence, nous nous concentrons dans ce mémoire sur le contenu vidéo, en particulier les vidéos partagées sur les réseaux sociaux comme YouTube.

---

1. YouTube. Consulté le 15 octobre 2018 sur https ://www.youtube.com/

La taille limitée des serveurs et le grand nombre des requêtes générées par les utilisateurs des vidéos poussent à stocker ces vidéos dans des serveurs intermédiaires. Ces serveurs, qui sont approvisionnés par les CDNs, doivent contenir les vidéos les plus populaires.

Pour résoudre ce problème, nous proposons une solution d'analyse de données, permettant de traiter les données collectées sur les vidéos en ligne et de fournir des informations permettant la prise de décisions, au niveau du réseau CDN, par rapport à la pertinence de la vidéo.

Nous présenterons donc, dans ce mémoire, une architecture générique d'un service d'analyse pour CDN basée sur *Cloud* et servant à prédire la popularité des vidéos en utilisant de nouvelles technologies comme le *Framework RESTful*. Nous visons ainsi à avoir une application analytique traitant l'historique d'utilisation des vidéos à travers les *logs* générés par le CDN pour ensuite retourner un nombre restreint de vidéos populaires nécessaires à dupliquer au niveau d'un serveur particulier. Toutes ces étapes serviront à prédire quelles vidéos sont amenées à être populaires.

Pour atteindre ces objectifs, nous allons déployer un service d'entraînement qui génère un modèle de classification à partir de l'ensemble des données existantes par l'utilisation des techniques du *Big Data* pour pouvoir gérer des millions de données. Ces techniques seront détaillées dans la prochaine section. Une fois le modèle défini, un autre service de prédiction sera développé, servant à identifier les vidéos populaires et à les rendre disponibles au réseau CDN.

0.2 Contexte

La quantité de données collectées par un CDN sur l'utilisation des vidéos est gigantesque. Par exemple, chaque fois qu'un utilisateur ouvre une vidéo ou en met la lecture en pause, le CDN inscrit l'opération qui est accompagnée d'une dizaine d'informations sur la vidéo, l'utilisateur, l'appareil utilisé ainsi que sur le réseau. Malgré les efforts de standardisation, chaque réseau CDN possède son propre format de données. Cette grande quantité de données ayant des formats et des sources différentes nous oblige à chercher des solutions génériques, fiables et efficaces pour identifier les vidéos pertinentes pour le cache de notre CDN. Il existe différentes technologies et techniques utilisées dans le domaine du *Big Data* pour résoudre des situations de variété, de grandeur de volume ou d'augmentation trop rapide des données. Nous allons nous concentrer sur les méthodes d'analyse et de stockage des données. Ainsi, nous allons trouver un meilleur déploiement sur *Cloud*, que ce soit pour une amélioration de la performance ou pour l'évolutivité de notre architecture.

Ce mémoire présente les contributions suivantes :

- Nous y proposons une architecture des serveurs CDNs basée sur le *Cloud* qui consiste à échanger les demandes de services entre les CDNs et le *Cloud*. Ces échanges passent par les interfaces de programmation applicative, appelées *API (Application Programming Interface)*. Ces communications sont réalisées en se basant sur le *Framework RESTful*, ce qui permet de traiter la quantité massive de données engendrées par les traces d'accès aux vidéos, d'analyser ensuite les informations pertinentes pour utilisateurs et finalement de prédire les vidéos les plus populaires.

- Nous y représentons un modèle d'optimisation qui utilise un algorithme de partitionnement pour l'analyse préliminaire des données. Ces dernières sont stockées dans une base de données et sont classifiées par l'algorithme de classification K-moyennes, résultant de groupes de popularité avec un centroïde représentant la vidéo la plus populaire dans chaque groupe.

Nous montrons que le modèle proposé est facile à déployer dans un environnement d'infonuagique et supporte un déploiement multiple sur des sites d'hébergement dédiés, ce qui permet d'avoir d'une façon optimisée des hébergements des vidéos sur une plateforme distribuée. En identifiant les vidéos populaires qui demandent de la haute disponibilité à travers les sites, nous arrivons à optimiser à la fois la bande passante au moment de la copie des données ainsi que la capacité de stockage physique.

0.3 Structure du mémoire

Ce mémoire est présenté comme suit :

Le chapitre 1 présente les concepts théoriques. Nous y parlons des aspects techniques du *Big Data*, ainsi que des solutions d'optimisation de données par le service d'analyse. Nous y identifions aussi les API utilisées dans l'échange entre le réseau CDN et la plateforme AaaS (*Analytique as a Service*).

Le chapitre 2 détaille notre architecture et nous y développons ses différents composants logiques. Nous y décrivons aussi le service web de type *REST* utilisé pour la communication entre les domaines de notre architecture. Le chapitre contient finalement les différentes étapes du modèle analytique servant à prédire les vidéos populaires à sélectionner selon les utilisateurs.

Le chapitre 3 décrit l'application de service web *RESTful* ainsi que les méthodologies utilisées. Les deux services y sont détaillés, soit le service d'entraînement et le service de prédiction. Nous y décrirons par la suite l'algorithme K-moyennes utilisé pour implémenter notre application analytique.

Le chapitre 4 décrit l'évaluation de notre architecture générique et les résultats obtenus y sont expliqués. Nous y montrons la performance de notre application en termes de précision des prédictions.

Le chapitre 5 fait office de conclusion. Nous y avons fait une récapitulation des objectifs atteints et nous y citons les avantages et inconvénients de notre architecture. Nous montrons à la fin quelques nouvelles idées qui permettront, selon nous, de dépasser quelques limites de notre travail ou de le développer dans le futur.

CHAPITRE I

CADRE THÉORIQUE

Ce premier chapitre présente les aspects théoriques liés à notre projet. Nous y détaillons le cadre théorique en commençant par les techniques et les technologies du *Big Data*, en nous basant sur leurs différents niveaux d'utilisation et leurs rôles. Le concept d'optimisation des réseaux ainsi que l'analyse des données sont aussi étudiés afin de mettre en valeur et d'exploiter l'aspect de la popularité. Nous nous concentrons par la suite sur les différents algorithmes de prédiction qui sont présentés dans la littérature. Finalement, nous étudions les API *REST* (*Representational State Transfer*) utilisées au niveau des interfaces entre CDN et *Cloud* pour améliorer le service d'analyse sur les réseaux.

1.1 *Big Data* : technologies et techniques

Le terme de *Big Data* a été énoncé pour la première fois dans un article scientifique de la bibliothèque numérique de l'association pour les machines de calcul (ACM : Association for Computing Machinery) en 1997. L'enjeu principal derrière le problème du *Big Data* était le manque de capacité par rapport au grand volume de jeux de données (plusieurs millions). Le terme *Big Data* désigne les données générées à chaque fois qu'un utilisateur a recours aux nouvelles technologies pour accéder à l'information, que ce soit à des fins personnelles ou professionnelles. Ces données sont en très grandes quantités et sont aussi très variées, ce qui les rend particulièrement difficiles à analyser. Les techniques du *Big Data* proposent donc des outils adaptés pour manipuler ce genre de données.

Le *Big Data* est devenu très populaire ces dernières années à cause de la croissance de la consommation d'Internet et de la diversification des utilisations du Web dans plusieurs secteurs. Dans le mode du *Big Data*, la grande quantité de données est décrite par cinq facteurs qui sont connus sous le nom des cinq V comme montré dans (Demchenko *et al.*, 2014) et (Chawda et Thakur,

2016) : variété, vélocité, volume, véracité et valeur.

- Variété : différents types de données sont utilisés comme les données structurées (exemple : nom, adresse, etc.) et les données non structurées (exemple : les photos, les vidéos, etc.). La technologie du *Big Data* permet de stocker et d'utiliser ces deux types de données simultanément.

- Vélocité : chaque minute ou bien même chaque seconde, la quantité de données utilisées, comme les photos, les vidéos, etc., s'amplifie rapidement dans tous les domaines technologiques du monde entier. Dans la modalité du *Big Data*, la vélocité indique non seulement la rapidité à laquelle une énorme accumulation de millions de données est générée, collectée et analysée, mais aussi la rapidité de transmission et d'accès aux données.

- Volume : à cause des vidéos, des téléphones, des médias sociaux, etc., chaque seconde, une étonnante quantité de données est générée. Le grand volume de données (des millions d'enregistrements) requiert maintenant d'utiliser des systèmes distribués dans des emplacements différents pour les stocker au lieu d'utiliser des bases de données classiques.

- Véracité : même avec des renseignements manquants ou incomplets, ou bien des fautes dans la saisie de certaines informations, la fiabilité des données et la qualité de précision seront présentes avec le *Big Data*.

- Valeur : avec des informations utiles obtenues par la collecte et l'analyse des données, on peut avoir une valeur ajoutée au résultat décisionnel concerné. Ce qui compte, grâce à la valeur des données du *Big Data*, est d'avoir un impact positif.

Les recherches dans le cadre du *Big Data*, spécialement dans le processus de traitement du *Big Data*, ont convergé vers plusieurs défis comme le stockage des données, la confidentialité des données, l'évolutivité des données, l'analyse des données, etc.

Dans le domaine de la technologie de l'information, on associe généralement le *Big Data* à deux principales utilisations : le stockage des données et l'analyse des données (Ward et Barker, 2013). La première utilisation a pour but de fournir des solutions de sauvegarde et de recherche de données volumineuses et non uniformes, par exemple des bases de données NoSQL. La deuxième utilisation vise plutôt à fournir des plateformes d'analyse de données. Nous nous concentrons dans le cadre de notre travail sur la deuxième utilisation, aussi appelée *Big Data Analytics*. Nous

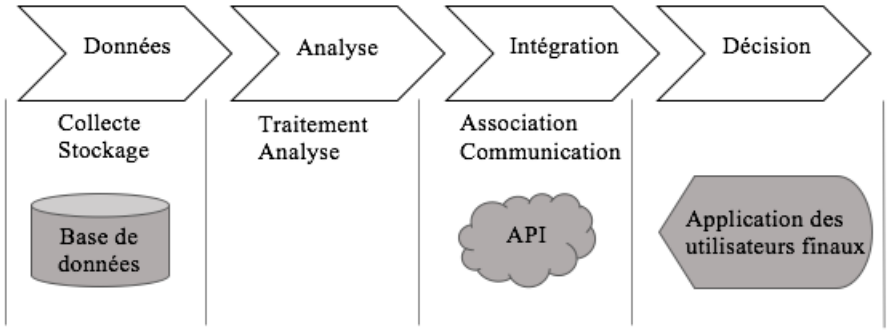

Figure 1.1: Caractéristiques des différentes couches d'analyse.

essayerons donc d'effectuer différents types d'analyses sur de grandes masses de données dans le but de faire ressortir des résultats précis dans un temps d'exécution court.

(Zhang *et al.*, 2017) identifient les problèmes liés à la qualité des données dans le cadre de traitement du *Big Data* et recommandent les principales étapes à déployer comme la collecte, le prétraitement, le stockage et l'analyse de données. Dans notre travail, nous nous concentrons sur le problème d'analyse de données.

Le premier problème de l'analyse de données est la précision des prévisions de données volumineuses à cause de l'instabilité de la précision dans certains cas et domaines, comme la prédiction de maladie, la bourse, etc., qui utilisent le *Big Data*. La solution proposée par (Zhang *et al.*, 2017) pour ce problème est l'utilisation des classificateurs bayésiens naïfs, de l'extraction des règles d'association et des arbres de décision. Le deuxième problème est la faible évolutivité à cause du grand volume du *log* des utilisateurs qui donne une diminution de la performance pour les algorithmes de recommandation. En réalité une même vidéo peut avoir des millions voir des milliards des vues. La solution présentée par ces derniers auteurs pour le dernier problème est l'utilisation des algorithmes de partitionnement de données comme K-moyennes.

Pour arriver à analyser un grand volume de données, il faut passer par plusieurs couches d'implémentation qui constituent l'infrastructure d'analyse, comme illustré par (Praveena et Bharathi, 2017) et aussi (Ahmad *et al.*, 2018). Chaque couche présente une phase très importante et est associée à différents utilisateurs finaux. La Figure 1.1 montre les quatre couches dont on parle.

- La couche de données : est la couche de stockage physique qui contient les différents types de données sauvegardées dans des entrepôts ou bases de données. Les bases de données

NoSQL, telles que Mongo DB, sont un bon exemple de technique de stockage de données hétérogènes, non structurées et volumineuses.

- La couche d'analyse : retient l'application analytique qui va consommer les données stockées dans la couche de base pour produire des résultats en temps réel. Cette couche permet aussi au besoin des opérations de modification des données.

- La couche d'intégration : relie l'application analytique avec l'utilisateur final. Le rôle de cette couche est d'exposer les résultats obtenus dans la couche analytique à une ou plusieurs entités externes, lesquelles l'utiliseront dans leur contexte spécifique. Les API sont très utilisées comme des interfaces d'intégration serveur-serveur et client-serveur.

- La couche de décision : désigne la couche de communication des utilisateurs avec le système. Elle contient les applications des utilisateurs finaux comme les applications mobiles et les applications web interactives.

1.2 Concept d'optimisation du réseau : analyse des vidéos

Pour bien utiliser la bande passante d'un réseau, il est important d'optimiser le contenu web diffusé aux utilisateurs finaux en l'adaptant suivant leurs localisations. Cela diminue en conséquence la charge du trafic web sur les serveurs, ce qui rend les applications web plus rapides et plus performantes.

La popularité des sites web de partage des vidéos est en croissance. Bien que plusieurs techniques sont développées pour l'analyse du contenu vidéo, elles ne sont pas toujours efficaces. Le processus de l'analyse de vidéos a plusieurs fonctions dont la surveillance, la détection de mouvements ou simplement l'obtention de la popularité d'une vidéo comme dans notre cas. L'idée d'indexer les vidéos pour faciliter la recherche et la récupération (Vashisht et Gupta, 2015) n'est pas toujours efficace. Les contenus vidéos sont classés comme des données non structurées, ce qui a pour conséquence que l'analyse de vidéos et l'extraction des informations qui concernent ces vidéos se révèlent difficiles. Il existe deux approches principales d'architecture d'analyse de vidéo décrites dans (Vashisht et Gupta, 2015). La première est l'architecture basée sur le serveur, l'idée dans cette approche est d'avoir un serveur centralisé pour stocker les vidéos qui y sont déjà compressées, et pour faire le processus d'analyse nécessaire. La deuxième est l'architecture basée sur le concept de bordure (*Edge*), où le processus de traitement des vidéos se fait localement sans avoir à envoyer les données vers un serveur distant. Le problème avec ces deux techniques

d'analyse est au niveau de la précision et du coût. La première technique est moins précise, alors que la deuxième technique est coûteuse en traitement des données.

Dans notre contexte précis, optimiser le réseau revient à limiter les vidéos diffusées à celles qui sont pertinentes pour l'utilisateur. Ce sont les vidéos les plus populaires qui doivent être gardées en cache sur le *Cloud* pour une diffusion rapide.

1.3 Analyse des données pour prédire la popularité

À cause du volume important des données provenant de plusieurs sources d'information avec différents niveaux de pertinence, il est indispensable de filtrer les données afin de réduire le volume et d'améliorer la performance des applications et la qualité de service chez les utilisateurs finaux. Ce filtrage des données se fait selon une stratégie spécifique d'analyse de données comme l'analyse par ordonnance, l'analyse prédictive, l'analyse diagnostique ou l'analyse descriptive (Malviya *et al.*, 2016). Pour réaliser une telle analyse, les données passent par quatre principales phases. La phase de préparation permet la collecte et la sélection des données utilisées. La phase de prétraitement comporte le filtrage et la transformation des données. La phase d'analyse effectue l'interprétation des données. Elle utilise les techniques statistiques comme la corrélation, la régression, la prévision, la classification et le regroupement des données. La phase de post-traitement comprend l'interprétation des données (Malviya *et al.*, 2016).

Une fois que l'analyse des données par extraction des informations nécessaires est effectuée, la prédiction a lieu pour avoir une prévision de ce qui va se passer dans le futur. La prédiction utilise des techniques statistiques comme la régression pour prédire des résultats futurs (Hu *et al.*, 2014). Si nous prenons la popularité comme l'information à analyser, plusieurs applications ont besoin de prédire la tendance de popularité pour mieux planifier et optimiser les ressources. La prédiction de la popularité est reconnue comme un but essentiel dans plusieurs utilisations par exemple dans (Tatar, 2011).

La prédiction de la popularité des vidéos a évolué ces dernières années selon les approches utilisées par les chercheurs. Dans (Wu *et al.*, 2016) les auteurs ont développé un modèle d'évolution de la popularité de la vidéo en tenant compte de quatre facteurs externes qui sont la recommandation directe, la recommandation bouche-à-oreille, la popularité intrinsèque de la vidéo et le pourcentage de réaction des utilisateurs. Le modèle est appelé EvoModel. Il fait appel à deux processus : un processus qui décrit la diffusion d'informations sur la vidéo et un autre processus qui représente la réaction constatée quand la vidéo est consommée par l'utilisateur.

Le modèle proposé pour l'analyse de la popularité d'une vidéo et son évolution au fil du temps (avec stimulation de la dynamique de popularité de la vidéo en ligne par un nombre limité de facteurs) ne permet pas de prédire la popularité des vidéos.

La prédiction de la popularité des vidéos en ligne reste encore un problème ouvert, même avec plusieurs tentatives dans la littérature pour le résoudre. Dans la plupart des études, les résultats obtenus sont mesurés seulement par la variable du nombre de vues de la vidéo comme cela est décrit par (Trzciński et Rokita, 2017) et (Su *et al.*, 2016). Ce travail (Trzciński et Rokita, 2017) propose une méthode avec régression vectorielle. L'approche utilisée dans ce travail donne des résultats précis et stables, mais on sait bien que plusieurs autres variables ou facteurs peuvent influencer la popularité au cours du temps, par exemple le nombre de « j'aime » pour une vidéo ou bien le temps de visionnement de l'utilisateur. Il en va de même pour (Su *et al.*, 2016) qui utilise un modèle multilinéaire basé sur le nombre de vues, en utilisant la durée de vie des vidéos comme coefficient dans leur modèle de prédiction de popularité. Ce travail ne prend pas en compte l'influence des autres facteurs sur la popularité des vidéos.

Dans le cadre de notre mémoire, nous allons étudier la prédiction de popularité des vidéos. Tel que discuté dans (Li *et al.*, 2016), la proposition d'un modèle de régression linéaire permet de prédire la popularité future des vidéos. La solution proposée dans (Li *et al.*, 2016) utilise une base de données du site Yourk (c'est un des principaux fournisseur d'hébergement de vidéos en Chine). Cette prédiction se fait par la corrélation de différents paramètres propres aux vidéos comme le nombre de vues, les premières vues et les vues pour une longue durée.

1.4 Interface *REST* entre CDN et *Cloud*

Pour obtenir les meilleurs services d'attribution des vidéos, les fournisseurs d'hébergement des vidéos en ligne les plus connus adoptent des réseaux de distribution de contenu appelés CDNs pour améliorer leurs services de stockage et de récupération de contenu. Les CDNs participent à l'attribution de chaque vidéo au serveur cache concerné, afin de donner une bonne recommandation pour les utilisateurs finaux. La communication entre CDN et *Cloud* se fait à travers un protocole d'échange de documents électroniques. Plusieurs types de protocoles sont utilisés selon les besoins d'échange et la nature du réseau. Les API sont très utilisées comme des interfaces de communication sur le web. Leur utilisation diffère selon les logiciels. Les interfaces API *REST* ou bien *RESTful* API sont très utilisées pour échanger les données entre le client et le serveur sur le web grâce à leur facilité de déploiement. (Pijetlović *et al.*, 2014) ont proposé

une solution d'API *RESTful* pour un fournisseur de contenu de télévision numérique basé sur le *Cloud*, leur solution est flexible et facile grâce à la conception intelligente de style *REST* et de l'infrastructure HTTP qui est déjà bien développée.

Dans notre cas, le client est le CDN et le serveur est l'AaaS sur le *Cloud*. *REST* va assurer la communication entre les deux composants et faciliter le déploiement de la solution en utilisant un algorithme de classification, le K-moyennes.

1.4.1 Architecture *REST*

REST est un style d'architecture qui utilise des contraintes pour réaliser l'existence des services web. Il est apparu pour la première fois dans la thèse de doctorat de Roy Fielding en juin 2000 (Fielding et Taylor, 2000). Les contraintes proposées pour l'architecture, comme cela a été montré dans la thèse en question (Fielding et Taylor, 2000), sont les suivantes :

- Architecture client-serveur : le client et le serveur ont chacun leurs fonctionnalités indépendantes les unes des autres. La même API pourra être utilisée par plusieurs clients, mais la plupart des traitements sont faits du côté du serveur.
- Communication sans état : le serveur reçoit les requêtes de la part des clients avec toutes les informations nécessaires et l'état de la session des clients n'est pas stocké par le serveur. Il est entièrement conservé sur l'ordinateur du client.
- Réponse qui peut s'inscrire sur le cache : le client peut utiliser des copies locales afin d'améliorer la rapidité des réponses délivrées aux utilisateurs sans avoir besoin de communiquer avec le serveur.
- Interface uniforme : il s'agit d'une contrainte fondamentale et sa base d'architecture *REST* lui a permis d'être différente par rapport aux autres architectures. Le transfert des informations se fait en format standard, ce qui est avantageux quand les données sont volumineuses.
- Système en couche : *REST* contient des couches hiérarchiques. Chaque couche ne peut voir au-delà des autres et utilise des intermédiaires partagés pour plus d'évolutivité du système.
- Style de codage à la demande : la dernière contrainte de *REST* est optionnelle, elle permet d'étendre les fonctionnalités du client par l'exécution d'un code sous forme d'un script.

Le but de *REST* est d'avoir des interfaces faciles à mettre en place qui favorisent la fiabilité, la visibilité et l'évolutivité des interfaces utilisateurs.

1.4.2 Interface CDN-Cloud

Le fournisseur de contenu vidéo utilise un mécanisme d'opérations entre le centre de données et les emplacements des caches distribués pour s'adapter aux intérêts des utilisateurs. La grande quantité et la variabilité de popularité des contenus des vidéos générées par les utilisateurs permettent d'utiliser des services de distribution. Les services de distribution se déroulent sur le *Cloud* avec l'utilisation de réseaux de diffusion de contenu. Afin de garantir une bonne qualité d'expérience d'Internet, il faut avoir mis en place une infrastructure réseau fiable capable de livrer du contenu aux utilisateurs finaux sans interruption, quelle que soit leur localisation sur la planète.

L'infrastructure *Cloud* organise un ensemble de serveurs qui sont fournis à la demande pour les déployer partout et à tout moment. Les données de streaming sont déployées par des CDNs. Les CDNs sont liés à des serveurs qui ont une localisation géographique répartie sur le monde entier. Le but du CDN est de donner aux consommateurs du web l'accès au serveur le plus proche d'eux pour avoir une meilleure navigation sur le web et une meilleure qualité de service. Plusieurs contenus web exigent aujourd'hui une bande passante très importante, comme les vidéos qui sont devenues dominantes ces dernières années.

Les CDNs fournissent des services de stockage et de récupération de contenu en ligne. Ils distribuent le contenu avec cohérence entre les serveurs d'origine et les serveurs *Edge*. Dans (Panchal *et al.*, 2013), les auteurs ont présenté un système de récupération et de stockage de contenu et ont proposé un mécanisme d'externalisation de contenu basé sur l'envoi de données vers le CDN. Ce mécanisme effectue un stockage partiel du contenu média du serveur *Cloud* sur plusieurs serveurs *Edge*. L'idée est d'opérer le déploiement du contenu sur les serveurs *Edge* à partir du serveur d'origine pour réduire les coûts de réplication et de mise à jour. Cette solution améliorera les performances de traitement des demandes des clients. Elle réduit également les données redondantes stockées dans les serveurs *Edge* et permet l'utilisation de plusieurs serveurs *Edge* pour chaque demande des clients, en réduisant ainsi la charge existante sur un seul serveur.

Le trafic généré par les infrastructures *Cloud* et CDN a lieu essentiellement entre les centres des données et les utilisateurs finaux qui sont géographiquement dispersés. Les techniques de virtualisation et de délégation de contenu permettent de séparer le service web du serveur de calcul.

Cette séparation complexifie la surveillance du *Cloud* et du CDN. Par contre, les fournisseurs du *Cloud* proposent souvent des APIs pour donner plus d'informations sur l'état interne du *Cloud*. Pour surveiller les données du trafic générées par les *Cloud* et les CDNs, une architecture de plateforme de mesure a été proposée dans (Bermudez *et al.*, 2014). L'évaluation de cette plateforme a permis d'observer un coût augmenté dans le réseau, causé par le transport des données du centre d'hébergement de données vers les utilisateurs finaux éloignés. La question à poser dans ce cas est : quelle est la meilleure façon de traiter les contenus afin de prédire seulement les paramètres nécessaires pour les utilisateurs ?

1.4.3 Algorithme de partitionnement

Les algorithmes de catégorisation consistent à classer de manière automatisée des objets observés selon différents critères d'appartenance. Ces algorithmes ont connu un intérêt particulier suite à l'apparition d'Internet et la nécessité d'organiser de façon rapide des documents numériques. La catégorisation automatique des courriels et l'identification des spams sont de bons exemples d'application de ces algorithmes.

Il existe différents types d'algorithmes de catégorisation. Nous citons les trois familles les plus connues. Cependant, il en existe plusieurs autres.

- Méthodes probabilistes : chaque classe est définie par une loi de probabilité différente (par exemple EM-algorithme).

- Méthodes hiérarchiques : une suite hiérarchique des objets est construite sous forme d'un arbre de clusters pour obtenir une structure organisationnelle (par exemple, classification ascendante hiérarchique).

- Partitionnement : la notion de partition est définie par l'attribution d'un ensemble d'objets en k classes. Chaque objet doit appartenir à une seule classe (par exemple, algorithme K-moyennes).

Des algorithmes d'exploration de données massives sont utilisés afin d'éviter que les données soient faussées par généralisation indue. Puisque la quantité de données est grande, il faut choisir l'algorithme le plus rapide dans le calcul. L'algorithme de K-moyennes est connu comme un algorithme de partitionnement assez rapide, mais aussi facile pour le regroupement (Bu *et al.*, 2014). K-moyennes est un algorithme classique qui est souvent utilisé pour les analyses

de données grâce à sa performance. Comme dans (Niu *et al.*, 2016), la méthode K-moyennes est utilisée pour l'analyse de *Big Data* avec une amélioration de l'efficacité de l'algorithme.

L'idée de la méthode du partitionnement est de former, à partir d'un grand ensemble de données, des petits groupes qui ont les mêmes caractéristiques. L'algorithme K-moyennes est établi par plusieurs utilisations comme décrit dans (Adrian et Heryawan, 2015) et il est pratiqué dans la méthode d'allocation de machines virtuelles. Cette méthode d'allocation de K-moyennes peut augmenter le temps d'exécution et le niveau d'utilisation du processeur en allouant un *cluster* qui contient une machine virtuelle de haute performance sur le premier centre de données et un *cluster* qui contient une machine virtuelle bas de gamme sur le dernier centre de données.

Le logiciel en tant que service connu sous le nom (SaaS) est une partie très importante du *Cloud Computing* puisqu'il contient la maintenance, les mises à jour, l'accessibilité par Internet, etc. Dans (Jagli *et al.*, 2016), les auteurs ont utilisé l'algorithme de partitionnement K-moyennes pour évaluer les attributs de qualité SaaS afin d'offrir une meilleure qualité de service aux clients et aux fournisseurs de service.

1.5 Conclusion

Dans ce chapitre, nous avons présenté l'utilisation des différentes techniques que nous allons utiliser dans notre application. Après une étude de la littérature, nous n'avons pas trouvé des approches qui traitent la prédiction de la popularité des vidéos avec une communication performante entre le CDN et le *Cloud*. Cette étude nous a motivés à travailler sur une architecture *REST* de l'application qui fait la mesure de la popularité par un algorithme de partitionnement. Dans le chapitre suivant, nous détaillerons notre architecture basée analytique en tant que service.

CHAPITRE II

ARCHITECTURE BASÉE SUR AAAS

L'architecture du logiciel de l'application constitue un élément essentiel de notre proposition de solution. Elle permet à la plateforme de prédiction des vidéos populaires d'interagir efficacement avec différents CDNs et de supporter plusieurs formats de *logs* des données. Dans ce chapitre, nous allons expliquer notre architecture d'analyse basée sur la plateforme AaaS. Nous allons examiner les interactions entre CDN et *Cloud* et aussi proposer notre modèle générique d'analyse de données de *logs*, ainsi que la création de l'algorithme de prédiction des vidéos populaires.

2.1 Usage de service d'analyse et d'interface de programmation applicative

Plusieurs entreprises utilisent la plateforme AaaS pour résoudre de grands problèmes du *Big Data*, que cela soit dans le domaine de l'industrie, de la santé ou d'autres domaines encore. AaaS offre un service d'analyse de données à travers des logiciels et des procédures en utilisant la technologie *Cloud*. AaaS désigne une technique de l'environnement *Cloud* qui compose avec d'autres modèles de service comme les logiciels en tant que service (*SaaS : Software as a Service*), les plateformes en tant que service (*PaaS : Platform as a Service*) et l'infrastructure en tant que service (*IaaS : Infrastructure as a Service*). L'objectif commun est de rendre accessible des fonctionnalités à distance à travers des API.

2.1.1 Analytique en tant que service

AaaS est une technique de répartition qui offre des moyens de calcul et d'analyse de données en tant que service autonome. Il est hébergé sur une plateforme complètement isolée et agit indépendamment des applications qui demandent ce service. Les avantages les plus importants de cette technique AaaS sont les suivants.

- Facilité d'utilisation et de déploiement : pour les applications qui veulent ajouter des fonctionnalités ayant besoin d'analyse de données, l'AaaS permet d'accélérer l'accès à une telle technologie et de diminuer le délai de commercialisation (*TTM : time to market*). Grâce aux API *RESTful*, les communications entre l'application client et l'AaaS se font à travers le web standard (protocole HTTP, *HyperText Transfer Protocol*) et ne demandent aucune infrastructure de réseau supplémentaire. En plus, l'interface de programmation applicative *REST* découple complètement la logique applicative locale du client et celle de l'AaaS. Cet avantage facilite l'intégration et simplifie les tests.

- Service à la carte : AaaS supporte un modèle de facturation flexible ou l'application du client (consommateur) paie l'AaaS (fournisseur) seulement lors de son utilisation. Ce modèle est avantageux, parce qu'il offre un coût d'utilisation modeste pour les petits consommateurs (*entry market*) tout en restant raisonnable pour les plus grands consommateurs.

- Scalabilité : l'analyse de données est de loin la tâche qui consomme le plus de mémoire et de CPU aux serveurs applicatifs et cela dégrade la qualité de service à cause des délais engendrés. En sous-traitant cette tâche pénible à un fournisseur, une application web peut offrir la même qualité de service, quel que soit le nombre des utilisateurs connectés en même temps.

Plusieurs entreprises ont besoin de diminuer le coût du matériel et des services informatiques utilisés tout en manipulant une quantité importante de données. Avec AaaS, le développement de nombreux logiciels est remplacé par un accès à des plateformes d'analyse à distance. Cependant, les organisations optent pour des solutions nouvelles au lieu des traditionnelles, avec de nombreuses tâches manuelles pour analyser leurs métadonnées. En prenant l'exemple des entreprises avec des données financières ou bien des données de ventes, utiliser AaaS permet non seulement de faire des économies, mais aussi de mieux connaître les comportements des utilisateurs finaux. Dans ce contexte, une entreprise de commercialisation peut associer des informations concernant les comportements de ses clients avec des informations générales pour obtenir des informations plus précises concernant les ventes de ses produits. Pour les entreprises de santé, l'utilisation d'AaaS est avantageuse du fait de la vaste quantité des informations qui sont stockées dans leurs bases de données, comme les informations des clients, les informations du personnel et bien d'autres. Le fournisseur de *Cloud*, comme *Microsoft* dans ce cas, stocke les données afin de les faire correspondre pour prédire plus de performances et diminuer les risques pour l'entreprise. Pour conclure, les organisations utilisent de plus en plus AaaS, car il

donne de meilleures informations, il est plus rapide et il est moins coûteux que l'utilisation des équipements informatiques et des équipes scientifiques fournissant des données.

2.1.2 Interface de programmation applicative *REST*

Une interface de programmation applicative se révèle être comme un système d'échange et de communication web entre des applications sans tenir compte du système d'exploitation et du langage de programmation de chaque application. *REST* est un modèle d'architecture d'application qui se base sur l'envoi des demandes et la réception des réponses via le protocole de transfert hypertexte (HTTP) comme illustré par la Figure 2.1. Cette architecture représente les deux principaux composants d'un système qui utilise l'approche *REST*. Les deux composants sont le client *REST* et le serveur *REST*, qui font les échanges entre eux via différentes méthodes du protocole HTTP.

Généralement, un utilisateur ou bien un client API a besoin d'accéder à des objets, connus sous le nom de ressources API. Cette ressource peut être sous plusieurs formes comme une ligne dans une base de données, une image ou bien un document, etc. *REST* est connue comme une architecture pour les systèmes distribués. Le critère essentiel d'une API *REST* est basé sur des ressources.

Certaines caractéristiques décrivent l'architecture *REST*.

Premièrement, une API *REST* est illustrée sous une forme distribuée pour rendre disponible une ressource sur un serveur distant. La communication entre le client et le serveur se déroule avec des requêtes HTTP par un localisateur uniforme de ressource (URL : *Uniform Resource Locator*) qui correspond à une ressource.

Deuxièmement, l'identification de la ressource est faite par l'accès à des ressources distantes en utilisant des URLs. Ces derniers peuvent désigner une seule ressource ou bien un ensemble de ressources. L'URL spécifie donc l'accès à une ressource donnée.

Le troisième principe de *REST* est l'utilisation d'une représentation de la ressource. Contrairement à ce qui est le cas dans d'autres APIs comme Soap par exemple, le serveur dans *REST* renvoie une représentation de la ressource au lieu de renvoyer une ressource. Dans ce cas, les échanges sont faits avec un format non imposé, ce qui permet la possibilité d'avoir plusieurs représentations pour une seule ressource dans différents formats comme *HTML, XML, JSON*, etc.

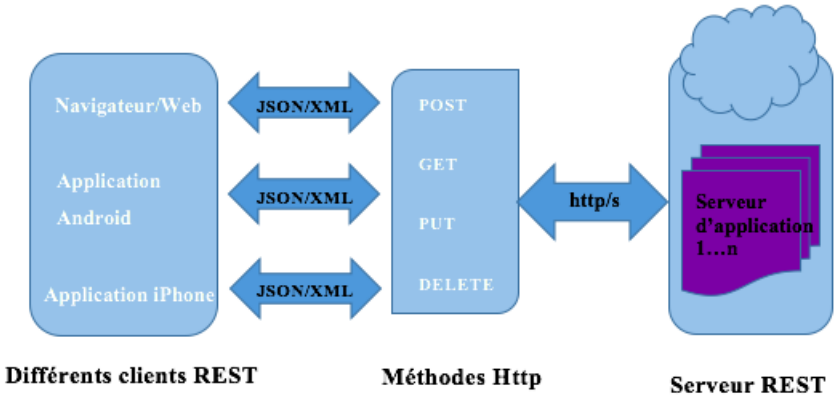

Figure 2.1: Architecture d'une API REST.

Enfin, avec *REST*, les opérations ne sont pas incluses dans l'URL de la ressource, mais le principe ici est d'utiliser les quatre verbes HTTP qui sont connus sous le nom de CRUD pour créer, afficher, mettre à jour et supprimer une ressource. Les spécifications du protocole HTTP sont utilisées dans *REST* qui n'est ni un standard ni un protocole. Les principes d'une architecture *REST* sont les suivants :

- Chaque élément a un ID.
- Les méthodes standards sont utilisées pour effectuer des opérations sur les données : créer (POST), afficher (GET), mettre à jour (PUT) et supprimer (DELETE).
- La liaison entre les éléments est faite.
- Les ressources ont des représentations multiples.
- La communication se fait sans état.

2.2 Architecture proposée

Nous avons choisi une architecture applicative capable de supporter adéquatement les interactions avec des CDN et qui pourrait être hébergée sur le *Cloud* pour une meilleure capacité évolutive. Cette architecture est basée sur une plateforme AaaS. L'architecture contient deux principales composantes du logiciel qui sont les fournisseurs CDNs et la plateforme en tant que service hébergé en *Cloud*, les deux étant liées par l'API *REST*. Ces deux composantes seront

présentées dans les deux prochaines parties. Les avantages d'un tel choix d'architecture sont les suivants.

- - Une meilleure interopérabilité entre la plateforme de prédiction et les différents CDNs grâce aux API *REST* ;
- - Une bonne scalabilité pour répondre à une augmentation de flux des vidéos à traiter grâce à une architecture répartie.

Pour absorber cette quantité de requêtes comme cela a été proposé dans notre architecture, on aura besoin de quatre serveurs virtuels applicatifs : un serveur pour le service de routage (*Routing Service*), un serveur pour le processeur *log* (*LOG Processor*), un autre serveur pour la base de données MongoDB et le dernier serveur pour le modèle analytique (*Analytic Model*).

2.2.1 Domaine de fournisseurs CDN

Les fournisseurs CDN hébergent et traitent les vidéos générées par l'utilisateur. Ils extrairont les fichiers *log* afin de les envoyer vers PaaS via l'API *REST*. Les fournisseurs CDN spécifient chaque modèle analytique selon les fonctionnalités envoyées dans les *logs*.

Chaque client CDN a des fonctionnalités spécifiques selon le type de contenu offert - voir Figure 2.2. Les fonctionnalités d'un CDN sont :

1 Application Analytics : cette fonctionnalité fournit l'ensemble de données de chaque CDN et permet donc au AaaS de bâtir une configuration spécifique à chaque CDN pour la sauvegarde et le traitement des fichiers de *logs*.

2 LogIngestor : cette fonctionnalité permet de renvoyer les *logs* de CDN au service *REST*. Elle joue un rôle de routage dynamique des flux de données provenant d'un CDN.

3 Vidéos populaires : cette fonctionnalité présente le résultat du traitement analytique des données. C'est aussi la réponse de l'AaaS à la demande principale du CDN par rapport aux vidéos populaires.

Selon ces fonctionnalités, AaaS va générer une structure spécifique pour chaque CDN en question.

Le protocole de communication entre les CDNs et le système de *Cloud* est le protocole HTTP. Comme montré dans la Figure 2.2, la communication se fait par les verbes HTTP comme *POST*

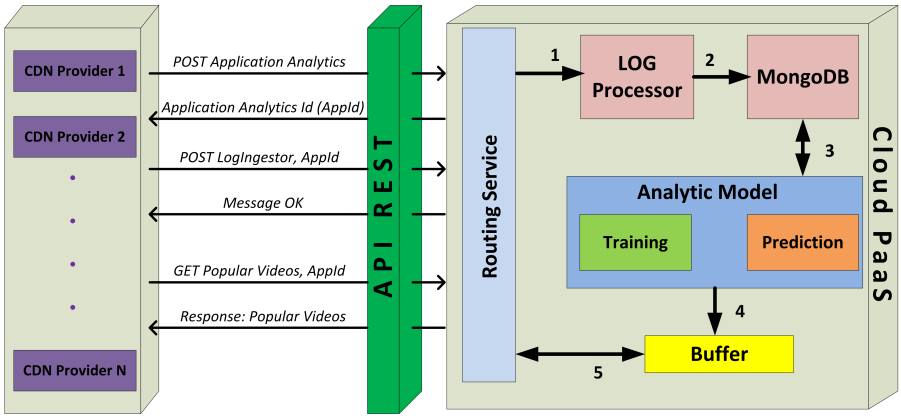

Figure 2.2: Architecture proposée.

et *GET* dans notre cas. Le serveur CDN envoie une demande au système pour déclencher la procédure d'acquisition d'un modèle analytique approprié, en utilisant la méthode de création d'une fonctionnalité. En retour, le système prend en charge la demande du CDN et lui répond en donnant un identifiant pour identifier cette action et faciliter les communications ultérieures. Entre temps, le système va commencer le traitement pour cette demande. De plus, le CDN envoie une demande de création de la fonctionnalité *LogIngestor* avec la méthode *POST*, afin que le système utilise le flux de données provenant du CDN. La dernière demande envoyée par le CDN au système est faite pour lui fournir la liste des vidéos populaires avec l'utilisation de la méthode *GET* par le serveur du CDN.

2.2.2 Domaine Cloud PaaS

La PaaS fournit les ressources de calcul et d'exécution nécessaires aux applications AaaS. Donc PaaS assure l'implémentation et le bon fonctionnement de l'AaaS. La PaaS contient des modules prédéfinis essentiels au modèle analytique comme le service de routage, le *log*, la base de données, les services d'entraînement et de prédiction, et le *Buffer*.

- Service de routage : c'est le module frontal, il reçoit et redirige les demandes provenant des fournisseurs CDN vers le processeur *log*.
- *Log* : contient le squelette créé par le routage et utilisé dans la base de données.
- Base de données : stocke les *logs* qui viennent des différents fournisseurs CDN dédiés. Ces données seront utilisées pour l'entraînement et la prédiction dans le modèle analytique.

- Service d'entraînement et prédiction : construit le modèle analytique de classification et exécute le calcul de la prédiction de la popularité des vidéos en fonction du modèle établi. Ce service sera présenté en détail dans la section 2.4.
- *Buffer* : c'est une structure des données qui garde en mémoire les résultats envoyés par le modèle analytique. Dans notre cas, ce sont les vidéos les plus populaires. Ces résultats seront renvoyés vers les fournisseurs CDN. Le but de *Buffer* est de fournir un accès rapide et plus facile.

Les données échangées sont en format JSON (*JavaScript Object Notation*). La solution ne fait aucune hypothèse sur la structure des données provenant du CDN. Notre architecture est donc générique. Les API *REST* aident le client, qui est le CDN, à faire les échanges des données avec le serveur AaaS en utilisant le protocole HTTP de base.

Certaines bases de données NoSQL, comme MongoDB ou bien Cassandra, sont similaires au niveau des cas d'utilisation, mais différentes quant à la méthode de gestion des données. Comme décrit dans panoply [1], Cassandra évolue sur plusieurs serveurs avec une configuration et un entretien facile. Par contre, MongoDB est utilisé pour l'analyse en temps réel et pour la gestion de contenu, d'où la motivation derrière ce choix pour le traitement de données dans notre système. MongoDB est reconnu pour la prédiction de popularité des vidéos grâce à son meilleur moteur de mise en cache. Subséquemment, le stockage des données non structurées dans des documents MongoDB est mis à jour facilement.

2.3 Traitement de *log*

Chaque CDN possède des traces de l'interaction des utilisateurs avec le contenu qui a été livré. Les traces contiennent des informations de l'utilisateur comme son adresse IP, son pays ou même ses préférences. Elles contiennent aussi des informations sur le contenu comme le type, la taille du fichier et les métas descriptions du contenu. Mais les informations les plus importantes que les traces gardent sont généralement les données d'interaction de l'utilisateur avec le contenu, comme l'heure de la visualisation, le type de terminal utilisé et les différentes actions engagées par l'utilisateur lors de la consultation de ce contenu. Les différentes traces collectées par l'ensemble CDNs sont acheminées vers le PaaS qui doit les sauvegarder en premier lieu, pour ensuite les traiter. L'utilisation des API *RESTful* standardise l'interaction entre CDNs et PaaS et facilite

1. panoply. Consultée le 30 janvier 2020 sur https ://blog.panoply.io/cassandra-vs-mongodb

la collecte des données de *log* à partir des CDNs.

Le CDN envoie les interactions qui se font entre l'utilisateur et la vidéo vers PaaS. Les données sont envoyées dans un format *log* vers une ressource *RESTful*. Les ensembles de *logs* sont regroupés dans un objet générique en format JSON. JSON est un format léger de fichier qui utilise les paires fonctionnalités-valeurs pour représenter les données.

Le premier défi est de pouvoir sauvegarder d'une façon efficace des fichiers de *logs* ayant des formats variés. Le stockage des données volumineuses et structurées de manière hétérogène est très difficile à faire avec les bases de données relationnelles. Ces bases SQL sont plutôt adaptées à des données normalisées pour lesquelles nous connaissons à l'avance la structure de données. La rapidité dépend aussi des index créés manuellement au préalable sur les tables des données relationnelles.

La technique du *Big Data* permet de fournir de nouvelles technologies pour gérer des collections de données dynamiques et non structurées comme les bases de données NoSQL. Ces bases peuvent stocker des millions de données hétérogènes dans un seul schéma générique et les récupérer sous forme d'objets spécifiques à chaque format. Nous avons utilisé dans notre solution la base de données NoSQL, Mongo DB [2] qui est une plateforme open source. L'implémentation de nos modèles se fait en récupérant les fonctionnalités nécessaires des structures des données.

2.4 Modèle analytique

L'application analytique constitue la partie fondamentale de la proposition de notre architecture. Cette application est capable de gérer plusieurs modèles analytiques et supporter différentes données venant de plusieurs CDNs comme cela est illustré dans la Figure 2.3. Notre objectif est d'avoir une architecture qui prend en charge un modèle générique et qui élimine le besoin de modifier le modèle en fonction du CDN. Il faut bien garder en tête que nous parlons ici d'une architecture client-serveur où le client est le CDN et le serveur est l'AaaS *Cloud*. Chaque client CDN doit passer par le même flux de travail et utilise les mêmes API pour envoyer les *logs* et demander les résultats au AaaS.

Le service de routage joue le rôle de procuration sur l'AaaS, chaque requête API passe par ce service en premier lieu. Le service de routage acheminera les demandes de différents CDNs vers leurs applications analytiques dédiées, à travers un identificateur unique de chaque CDN. Il

2. MongoDB. Consulté le 1er avril 2018, sur https ://www.mongodb.com/

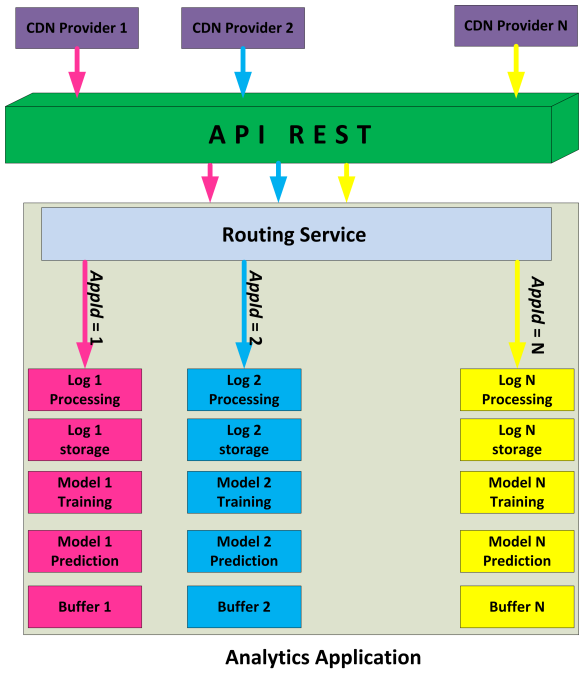

Figure 2.3: Application analytique.

existe une ressource *REST* qui assure la génération d'un nouvel identificateur quand un nouveau CDN est ajouté. Une fois que le CDN a reçu son identificateur, il peut commencer à envoyer les *logs* pour le traitement et le stockage sur le PaaS. L'AaaS fait ensuite le traitement des données, puis la construction du modèle d'entraînement et de prédiction pour se préparer en étape finale à calculer et à envoyer les résultats vers le *Buffer* pour une consommation éventuelle par le CDN.

Le modèle d'entraînement rassemble toutes les informations disponibles sur les vidéos à partir de la base de données MongoDB pour créer un modèle analytique et prédire la popularité des vidéos. D'après les informations collectées dans le modèle d'entraînement, les vidéos populaires de chaque application analytique pour chaque CDN seront traitées dans le modèle analytique. Au fur et à mesure, le modèle d'entraînement sera mis à jour d'après les résultats obtenus de la part du modèle analytique.

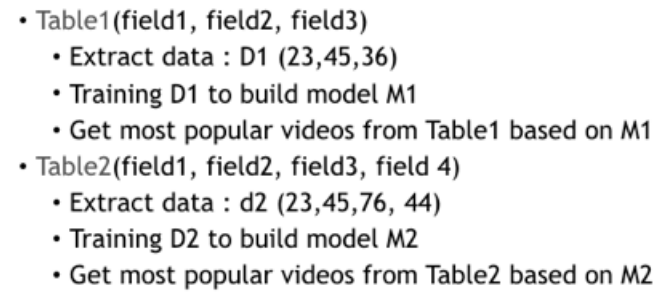

Figure 2.4: Datasets

L'exemple dans la Figure 2.4 illustre les étapes de cycle de vie des Datasets. Le déroulement commence par l'extraction de données (par exemple D1) depuis les collections de bases de données pour ensuite passer au modèle d'entraînement et au modèle analytique.

Ces collections contiennent toutes les fonctionnalités concernant les vidéos. Ensuite, ces données sont traitées par l'entraînement pour construire un modèle (M1). Enfin, l'analyse des données utilise le modèle (M1) pour prédire les vidéos populaires.

2.5 Conclusion

Dans ce chapitre, nous avons présenté une architecture d'analytique en tant que service pour prédire la popularité d'une vidéo. L'approche proposée procède à une communication entre le CDN et le *Cloud* par une API *REST* qui facilite l'implémentation et le déploiement de notre architecture. Elle offre un modèle de fonctionnement générique, peu importe la structure des données supportée par le CDN. En plus, le PaaS permet de prendre en charge plusieurs CDNs. Le même CDN pourra disposer de plusieurs applications analytiques, en fonction des besoins d'affaires. L'explication de l'application de prédiction est présentée dans le chapitre suivant.

CHAPITRE III

APPLICATION DE PRÉDICTION EN UTILISANT LE WEB SERVICE *RESTFUL*

Dans ce chapitre, nous proposons une application analytique en tant que service pour les CDNs basée sur le *Cloud* afin de générer des vidéos populaires aux utilisateurs finaux. L'application de prédiction est implémentée dans une architecture multi-tiers en utilisant des web services *REST* pour échanger les informations entre la frontale (*front-end* en anglais) et la dorsale (*back-end* en anglais) d'une part et entre les différents composants dorsaux d'autre part. Nous allons expliquer uniquement les interactions dorsales qui se font entre les serveurs. Toutes les interactions avec le terminal (navigateur web, application mobile, etc.) ne sont pas indiquées dans cette section.

3.1 Méthodologie

Pour simplifier la communication entre le CDN et le *Cloud* et bien manipuler les informations en échange, nous avons mis en œuvre l'architecture avec le service web *RESTful*. La solution est codée avec le langage de programmation Java pour construire la dorsale ainsi que pour la couche frontale. Pour le développement, notre solution utilise l'algorithme K-moyennes pour prédire la popularité des vidéos. Cet algorithme donne des résultats assez concluants, même dans des cas où toutes les hypothèses sur les données ne sont pas satisfaites (distribution et uniformité). Nous avons utilisé le *Framework Restlet*. Ce *Framework* accélère le développement en offrant un cadre de travail préétabli. Il permet aussi de réutiliser plusieurs modules et codes existants. Un autre avantage du *Framework Restlet* est celui des bonnes pratiques de développement qu'il propose, ce qui rend notre code optimisé et facile à maintenir.

L'idée d'avoir à la fois des solutions et des procédures sûres rend notre modèle d'architecture performant pour tout échange entre client et serveur, et donne des résultats de prédiction instantanée à cause d'un temps d'exécution acceptable. Chaque composant logiciel ou service de notre architecture est codé comme une application Java distincte. Chaque application avait un

port TCP différent pour communiquer les données à travers des API *RESTful*. Nous avons utilisé une base de données MongoDB pour sauvegarder les vidéos générées par les utilisateurs (UGVs : *User Generated Video*). MongoDB permet un accès rapide et simplifié aux données dans le même format JSON utilisé dans les transferts *REST*.

3.2 Attributs et mesure de similarité des vidéos

Les attributs des vidéos ont été fournis par YouTube en recueillant les statistiques des partages et des visualisations d'une centaine de milliers de UGVs. L'exemple suivant illustre les attributs disponibles dans notre jeu de données.

Chaque vidéo a un identifiant (*_id*) constitué par une chaîne de caractères unique. La vidéo possède aussi un objet (*accessControl*) qui liste les autorisations qui lui sont disponibles comme la permission de partager, de commenter ou de lire automatiquement. Elle possède aussi une catégorie, une description, un nom d'auteur, une durée, une date de publication, un titre, un type ou une extension, ainsi qu'une liste des vidéos connexes. Le reste des attributs correspond aux statistiques comme le nombre de partages, la durée de visualisation, le nombre d'abonnés, ainsi que le nombre des vues. Ces attributs sont utilisés pour calculer la popularité dans notre modèle de prédiction.


```
Video {
"_id": "---M7ivXfKQ",
"accessControl": {"comment": {"permission": "allowed"},"list": {"
    permission": "allowed"},"autoPlay": {"permission": "allowed"}},
"category": "Games",
"commentsNumber": "8",
"description": "Hello, so i built a brand new toyota supra",
"author": "stescouser09",
"publishedDate": "2013-07-15T03:45:10.000Z",
"title": "FORZA 4 SO scouser 09 and JSI Dwight tandem practice",
"shares": "14503",
"duration": "145",
"watchtime": "48865",
"subscribers": "145",
"relatedVideos": ["qRKDa7JSp5k", "LWyfxcOPK00"],
"type" : "video/3gpp",
```

"views" : "111465"
}

Pour calculer la similarité entre deux vidéos, nous avons choisi une distance euclidienne en supposant que nos attributs sont indépendants dans un espace multidimensionnel dans lequel chaque attribut peut être tracé. Nous avons seulement considéré les attributs numériques pour le calcul de la popularité d'une vidéo. Les attributs descriptifs comme le titre et la description de la vidéo ont été filtrés. Comme cela a été montré dans l'équation 3.1 la similarité entre deux vidéos $v1$ et $v2$ est calculée alors avec une distance euclidienne entre les attributs retenus dans notre modèle de calcul.

$$MS = (NV(v2) - NV(v1))^2 + (NP(v2) - NP(v1))^2 + (NC(v2) - NC(v1))^2 + (NA(v2) - NA(v1))^2 \quad (3.1)$$

tel que MS est la mesure de similarité, NV est le nombre de vues, NP est le nombre de partages, NC est le nombre de commentaires et NA est le nombre d'abonnés.

3.3 Modèle analytique de l'application

La colonne vertébrale de notre application comporte deux services principaux. Le premier service est le service d'entraînement de données. Il sert à construire le modèle analytique de données à partir des UGVs disponibles dans notre base de données. Le deuxième service est le service de prédiction. Il permet de calculer la popularité d'une nouvelle vidéo en se basant sur le modèle analytique construit.

3.3.1 Service d'entraînement

Le service d'entraînement collecte l'ensemble des UGVs disponibles à partir de la base de données MongoDB. Ensuite, le service applique l'algorithme d'entraînement pour bâtir le modèle analytique qui va servir plus tard dans la prédiction de la popularité d'une nouvelle vidéo. L'entraînement se fait à l'aide de l'algorithme K-moyennes. L'algorithme K-moyennes, une fois appliqué, génère un nombre de groupes de vidéos classées par popularité. Le processus d'entraînement permet de trouver un modèle d'apprentissage machine à partir des données disponibles qui serviront comme données de formation.

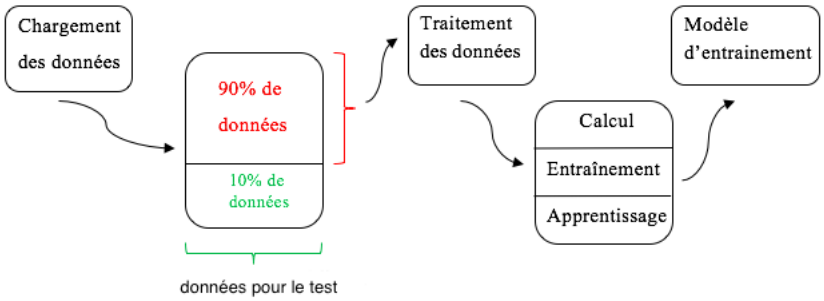

Figure 3.1: Les étapes d'entraînement.

Les données de formation dans notre analyse sont les UGVs de la base de données YouTube CONGAS. CONGAS est une base de données qui regroupe des métriques de popularité des vidéos YouTube. Ces métriques ont été obtenues à l'aide de l'outil YOUStatAnalyzer décrit dans (Zeni *et al.*, 2013). YOUStatAnalyzer a été ainsi développé pour permettre de télécharger les statistiques des vidéos YouTube pour créer une base de données à analyser.

Chaque UGV contient plusieurs attributs qui expliquent son historique de visualisation (nombre de vues, nombre de partages, etc.). Comme montré dans la Figure 3.1, le processus du service d'entraînement commence par le chargement de données avec l'utilisation, approximativement, de 90 % de données générées. L'apprentissage se fait sur ces dernières, afin de trouver un modèle d'entraînement qui soutient le service de prédiction. Les données de formation doivent contenir la réponse exacte, qui porte le nom de cible et que nous cherchons à prédire quand elle n'est pas disponible. Dans notre cas, nous cherchons à répondre à la question suivante : « est-ce que la vidéo en question est populaire ? ». L'algorithme d'apprentissage identifie les liaisons entre les données de formation, qui mettent en correspondance les attributs des données d'entrée avec la cible à prédire. L'algorithme génère alors un modèle d'apprentissage qui essaie de reproduire ces liaisons le mieux possible.

Considérons les vidéos comme un ensemble de n points de données $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ qui doivent être classés en k groupes. L'objectif est d'affecter chaque vidéo à un *cluster* représentant son niveau de popularité. K-moyennes a pour objectif de trouver les positions $\mathcal{M}=\{\mu_1, \mu_2, \ldots, \mu_k\}$ des *clusters* qui minimisent la distance entre les points de données et le centre de gravité du

cluster. L'algorithme K-moyennes résout l'équation suivante :

$$\arg \min_{\mathcal{V}} \sum_{i=1}^{k} \sum_{\mathbf{x}_i \in v(x_i)} d(\mathbf{x}_i, \mu_i) = \arg \min_{\mathcal{V}} \sum_{i=1}^{k} \sum_{\mathbf{x}_i \in v(x_i)} \|\mathbf{x}_i - \mu_i\|_2^2 \tag{3.2}$$

Avec $v(x_i)$ comme l'ensemble des vidéos appartenant au *cluster* i. Le partitionnement K-moyennes utilise le carré de la distance euclidienne $d(\mathbf{x}_i, \mu_i) = \|\mathbf{x}_i - \mu_i\|_2^2$.

Le service d'entraînement crée ainsi un modèle de classification qui dessine les frontières entre les différents groupes de popularité des vidéos. Ce modèle sera utilisé par le service de prédiction pour identifier la popularité d'une nouvelle vidéo.

3.3.2 Service de prédiction

Le service de prédiction permet de prédire la popularité d'une nouvelle vidéo qui vient d'entrer dans le système en se basant sur le modèle de classification déjà établi par le service d'entraînement. L'algorithme K-moyennes est une technique de partitionnement des données très utilisée dans l'apprentissage non supervisé. Elle est efficace pour partitionner un nuage de données en nombre préfini de groupes ou classes (WikiStat, 2020). K-moyennes assigne chaque vidéo au groupe de popularité le plus proche (soit le plus similaire). Un groupe de popularité est présenté par une vidéo fictive qui comporte les moyennes des attributs de toutes les vidéos appartenant à ce groupe. Cette vidéo est le centroïde du groupe de popularité. Cette opération est répétée plusieurs fois (itérations), jusqu'à ce que le centroïde de chaque K-moyennes ne change plus ou change peu. Nous disons donc que l'algorithme K-moyennes a convergé et que les centroïdes représentent désormais les groupes des popularités.

Le service de prédiction utilise la même technique pour prédire la popularité d'une nouvelle vidéo. Il calcule la distance entre la nouvelle vidéo et le centroïde de chaque groupe de popularité. Le groupe qui a le centroïde le plus proche de la nouvelle vidéo sera celui retourné par le service de prédiction. Pour garder le modèle de prédiction à jour, le service d'entraînement lance l'algorithme de partition K-moyennes toutes les heures pour prendre en compte de nouvelles vidéos. Les deux services exposent une ou plusieurs interfaces *REST* pour intercommuniquer. Seul le service d'entraînement accède à la base de données pour mettre à jour les UGVs avec les nouvelles vidéos et le modèle en conséquence.

3.3.3 Choix de l'approche K-moyennes

L'approche K-moyennes est utilisée pour définir des groupes de données. La méthode d'apprentissage non supervisé est bien connue pour sa simplicité et sa puissance, surtout pour les données en temps réel. L'idée est de regrouper un type de données dans un ensemble de données complexes. La méthode examine les données, assimile les observations similaires afin d'identifier les différents groupes.

Avec les grands nombres de données, qui sont les traces des évènements liées aux vidéos, il faut choisir un système à la fois efficace et simple à implémenter. Compte tenu de la grande quantité et la divergence de données utilisées dans notre architecture, nous optons pour un algorithme qui fournit, au cours du temps, un résultat spécifique, mais pas nécessairement optimum. Chaque itération donne des résultats différents à cause du changement de données au cours du temps, ce qui nous permet de déterminer les vidéos populaires à chaque ajout des traces de *logs* liés à ces données.

Différents cas d'utilisation de l'approche K-moyennes pour le regroupement sont cités dans (Trevino, 2016) comme la segmentation comportementale, la catégorisation de l'inventaire, le tri des mesures des capteurs et aussi la détection des bots ou des anomalies. Le processus de division d'un ensemble de données en groupes selon des catégorisations de données est précisément ce dont nous avons besoin dans le cas d'un grand nombre de traces de *logs* d'utilisateurs pour prédire la popularité des vidéos. Le choix de l'algorithme de partitionnement dépend généralement des données à analyser. Dans notre cas, nous avons considéré les hypothèses suivantes pour le choix de K-moyennes comme algorithme de classification.

- Chaque groupe de vidéos a une taille considérable étant donné que les groupes représentent cinq degrés de popularité.
- Les attributs possèdent une distribution sphérique à l'intérieur du même groupe. Cela veut dire que les attributs ont la même variance et qu'ils sont indépendants les uns des autres.
- Les groupes ont une densité similaire.

3.4 Algorithme K-moyennes

K-moyennes est un algorithme d'apprentissage non supervisé. Il est utilisé lorsque nous avons des données non étiquetées, par exemple des données sans catégories ou sans groupes définis.

3.4.1 Avantages et défauts de K-moyennes

Le choix est vaste entre les différents algorithmes de regroupement. Comme pour toute autre approche, K-moyennes possède des bénéfices et des défauts. L'algorithme K-moyennes donne la solution pour le cas des données homogènes comme la segmentation d'images afin de rassembler des pixels proches dans le même groupe. L'utilisation de cet algorithme est simple en cas de grands ensembles de données, la convergence est garantie, et il s'adapte avec les nouveaux changements.

Pour conclure, l'avantage le plus important apporté par l'approche K-moyennes est la généralisation pour les groupes avec des tailles et des formes différentes, surtout ceux qui ont la forme non sphérique comme la forme elliptique. Comme montré dans l'exemple de (developers google, 2020) illustré par la Figure 3.2, après avoir généralisé l'algorithme, les limites de chaque groupe sont identifiées dans chaque cas.

Schéma à gauche : en appliquant l'algorithme K-moyennes sans généralisation, on obtient des groupes qui se chevauchent. Les frontières ne sont pas intuitives à déterminer. Cette technique est efficace seulement dans des cas simples où les groupes ont des formes similaires.

Schéma au centre : en appliquant l'algorithme K-moyennes avec une largeur variable des groupes, les résultats sont bien meilleurs. La séparation entre les groupes est améliorée en adaptant la largeur aux données formant un groupe spécifique.

Schéma à droite : en appliquant l'algorithme K-moyennes généralisé avec une largeur variable des groupes par dimension, on réussit à donner des formes elliptiques au lieu de formes sphériques ordinaires aux différents groupes. Les frontières sont encore mieux dessinées et la séparation des différents groupes est plus intuitive.

Avantages du choix de l'algorithme K-moyennes :

- Il donne des résultats concluants même si quelques hypothèses ne sont pas satisfaites.
- Ordinaire et facile à mettre en place.
- Donne des résultats de classification facile à interpréter.
- Rapide (en termes d'utilisation de l'unité de calcul) et efficient (en termes d'utilisation de la mémoire vive) avec une complexité de $O(K * n * d)$ pour des données numériques, catégorielles ou mixtes ayant une représentation vectorielle.

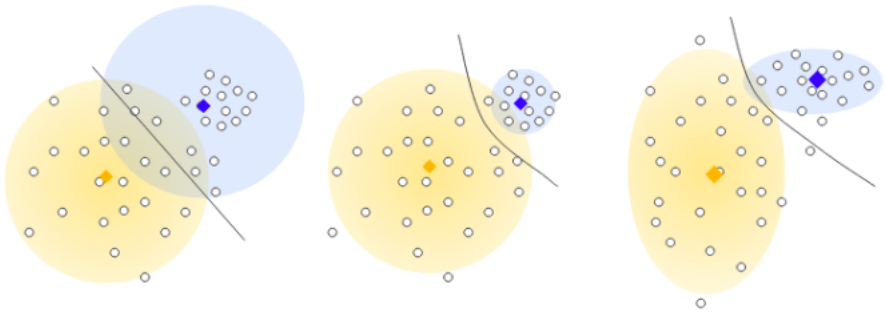

Figure 3.2: K-moyennes généralisé.

Limites du choix de l'algorithme K-moyennes :

- Mauvaise perception sur l'uniformité des données : les groupes créés par K-moyennes ont généralement une taille similaire, ce qui n'est pas nécessairement le cas dans plusieurs situations.
- La distribution sphérique est difficile à obtenir dans la réalité étant donné que les attributs ont généralement des interdépendances.
- L'algorithme donne des résultats non concluants quand il s'agit de groupes ayant différentes densités, même si la distribution est sphérique.
- Difficulté à trouver le nombre K optimal des groupes : il faut faire des essais avec différentes valeurs et garder le K avec le meilleur résultat de classification. Cette technique reste coûteuse si le K est très élevé.
- Le choix initial des centroïdes influence la performance de l'algorithme. Il n'existe pas une façon commune d'initialiser les centroïdes.

3.4.2 Description de l'algorithme

Le but de l'algorithme K-moyennes est alors d'identifier les groupes dans un ensemble de données (c.-à-d., classification des données en groupes). Le nombre de groupes est représenté par la variable K. L'algorithme affecte de manière itérative chaque point de données à l'un des K groupes en fonction de ses attributs. Les données sont ainsi classifiées en groupes en fonction de la similarité des leurs attributs.

L'algorithme K-moyennes se base sur le principe de réaffectation des individus, soit dans notre cas des vidéos réaffectées à des *clusters*, dynamiquement. Comme montré dans l'algorithme 1 (Aloui *et al.*, 2018), les centres de *clusters* ou centroïdes sont eux-mêmes recalculés à chaque itération, dynamiquement jusqu'à leur convergence. La convergence arrive quand les classes ne changent plus de centroïdes, après une affectation complète de chaque vidéo au *cluster* le plus proche. Elle peut aussi arriver si nous atteignons un nombre maximal des itérations *MaxIter*. Cela force l'algorithme à respecter un temps acceptable de traitement. Le nombre de classes K doit être déterminé a priori.

Nous avons effectué des tests avec différents nombres de classes pour arriver au nombre optimal en termes de qualité de classification et de temps d'exécution. Les résultats sont présentés dans le chapitre 4. Pour que l'algorithme puisse utiliser les vidéos, il est important de créer une représentation vectorielle de chaque vidéo x dans R^n mené d'une distance euclidienne. Les critères fournis par la base de données des UGVs (temps de visualisation, nombre de partages de la vidéo, etc.) constituent notre représentation R^n. L'initialisation des *clusters* se fait en tirant aléatoirement K individus ou en affectant simplement une valeur aléatoire à chaque centroïde.

L'algorithme répète donc deux opérations jusqu'à la convergence :

1. Chaque vidéo est affectée au *cluster* dont la distance à son centroïde est la plus proche.
2. Chaque centroïde est calculé en fonction des vidéos appartenant à son *cluster* ainsi constitué.

Algorithme :

- Initialisez les centroïdes : $\mu_i = random$, avec $j = 1, ..., n$
- Répéter :
  - Attribuez chaque vidéo au *cluster* le plus proche :
    $v(x_i) = \{v(x_j) | d(x_j, \mu_i) \leq d(x_j, \mu_l), l \neq i, j = 1, ..., n\}$
  - Définissez le centre de gravité de chaque *cluster* sur la moyenne de toutes les vidéos appartenant à ce *cluster* :
    $\mu_i = \frac{1}{|v(x_i)|} \sum_{j \in v(x_i)} x_j, \forall i$
- Jusqu'à la convergence : $|\mu_i^* - \mu_i| \leq \varepsilon, \forall i$

– La fonction $UpdateCluster\mu_i$ calcule les cordonnées de centroïde en fonction des points appartenant à ce *cluster* :
$\mu_i = avg\ distance(x_i, \mu_i),\ v(x_j) = i,\ j \in 1, ..., n$

L'algorithme 1 montre le pseudo-code de notre algorithme K-moyennes utilisé dans notre application de prédiction de données.

Les résultats de l'algorithme K-moyennes sont :

- Le groupe affecté à chaque point de donnée (étiquette) sachant que chaque point de données est affecté à un seul groupe.
- Les k centroïdes de chaque groupe : le centre de gravité d'un groupe représente l'ensemble des données formant ce groupe. L'examen des poids des points centroïde peut être utilisé pour interpréter qualitativement le groupe en question.

3.5 Ressource *RESTful* utilisée

L'application AaaS utilise des services web *RESTful* pour communiquer avec l'extérieur. Ces services permettent d'exposer des fonctionnalités que d'autres applications peuvent utiliser par la suite. L'interface de programmation d'application *REST* que nous appelons aussi service web *RESTful* utilise des requêtes HTTP pour échanger l'information.

Les fonctionnalités d'un web service de type *REST* sont organisées selon un ensemble de ressources. Une ressource est donc un groupement des fonctionnalités par entité logique ou modèle des données. Chaque ressource de l'architecture *RESTful* est identifiée par un URI (*Uniform Resource Identifier*) unique. Les ressources *REST* utilisent quatre opérations fondamentales qui sont les suivantes :

- *PUT* : Pour la création d'une nouvelle ressource.
- *POST* : Pour la modification d'une ressource.
- *GET* : Pour la récupération d'une ressource.
- *DELETE* : Pour supprimer une ressource existante.

Toutes les opérations qui se font au cours de l'échange entre client et serveur utilisent des ressources *REST*. Dans le cas de notre approche AaaS, les ressources sont présentées dans le tableau

Algorithme 1 : K-means algorithm

Entrée : $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ (set of data points to be clustered), k (number of clusters), *MaxIter* (limit of iterations)

Résultat : $\mathcal{M}$=$\{\mu_1, \mu_2, \ldots, \mu_k\}$ (set of centroids) $\mathcal{V} = \{v(x)|x = 1, 2, \ldots, n\}$ (set of cluster labels of $\mathcal{X}$)

`/* Initialize the centroids */`

pour chaque $i \in 1, ..., k$ **faire**
| $\mu_i \leftarrow random$

fin

pour chaque $\mu_i \in \mathcal{M}$ **faire**
| $v_i \leftarrow argminDisatence(x_i, \mu_j)$, $j \in 1, ..., k$

fin

$changed \leftarrow false$;

$iter \leftarrow 0$;

répéter

  pour chaque $\mu_i \in \mathcal{M}$ **faire**
    UpdateCluster(μ_i);

  fin

  pour chaque $x_i \in \mathcal{X}$ **faire**
    $minDist \leftarrow argminDisatence(x_i, \mu_j)$, $j \in 1, ..., k$

  fin

  si $minDist \neq v(x_i)$ **alors**
    $v(x_i) \leftarrow minDist$;
    $changed \leftarrow true$;

  fin

  $iter$++ ;

jusqu'à *changed= true and iter* $\leq$ *MaxIter*;

3.1. La communication entre le client et le serveur commence par la création d'une nouvelle application analytique. Cela se fait par l'envoi d'une méthode *POST* à l'URI *ApplicationAnalytics*. Une fois la ressource *ApplicationAnalytics* bien créée, l'AaaS retourne en *callback* un identificateur unique que le CDN utilisera plus tard pour envoyer ses *logs* à l'AaaS. Le CDN utilise la ressource *LogIngestor* pour en envoyer les *logs* à l'AaaS. À l'aide de la méthode *POST*, le CDN transmet les *logs* dans un format JSON en plusieurs lots. L'identificateur *ApplicationAnalytics* est ajouté à chaque requête pour s'assurer d'envoyer les *logs* à la bonne application analytique.

Le serveur AaaS effectue l'entraînement de chaque application analytique pour construire son modèle de prédiction. Le traitement est déclenché par une routine interne du serveur avec une période de temps prédéfinie. Pour obtenir les résultats de la prédiction, dans notre cas les vidéos populaires, le CDN envoie la demande à la ressource *AnalyticsModel* à l'aide de la méthode *GET*. L'AaaS retourne la liste des vidéos populaires au CDN. Cette liste sera utilisée par la suite par le CDN pour mettre à jour son cache, avec la liste des vidéos populaires qui ont le plus de chance d'être consultées par les utilisateurs.

Tableau 3.1: Resource description

Resources	**Method**	**hyperlink**	**Results**
Application Analytics	POST : Create a new Application Analytics ressource	https ://ApplicationAnalytics Exemple.com	201 Created <ApplicationAnalytics> <id>11 </id> </ApplicationAnalytics>
LogIngestor	POST :Add for each Application Analytics their logs	https ://LogIngestor Exemple.com/ {AppId}	200 Ok
Analytics Model	GET : provide the result of popular video	https ://Analytics-Model Exemple.com/ {AppId}	<video> <id>1 </id> <name>video1 </name> <order>2 </order> </video>

3.6 Conclusion

Dans ce chapitre, nous avons expliqué notre architecture de prédiction basée sur des services web *RESTful*. Nous avons aussi expliqué le cycle de vie d'une interaction entre CDN et AaaS qui commence par la création de l'application analytique et l'envoie des *logs* de CDN. Ensuite, se fait l'entraînement des données et finalement la construction d'un modèle analytique et l'envoi des résultats sous forme d'une liste des vidéos populaires. Les résultats de calcul obtenus par notre modèle seront détaillés dans le chapitre suivant.

CHAPITRE IV

ÉVALUATION ET RÉSULTATS

Nous présentons dans ce chapitre l'évaluation de notre architecture en étudiant la performance de la méthode utilisée dans l'implémentation. Il est à constater que, dans la littérature courante, il n'y a aucun modèle de prédiction de popularité utilisé dans une architecture *REST* à comparer avec notre application. Les résultats obtenus font suite à l'implémentation des services utilisés qui sont le service d'entraînement, qui construit le modèle analytique basé sur les UGVs rassemblés, et le service de prédiction, qui calcule la popularité d'une nouvelle vidéo basée sur le modèle analytique de manière *ad hoc*. Afin de bien évaluer notre modèle proposé et d'avoir une architecture fiable, nous nous sommes concentrés sur deux principales idées. La première idée est de confirmer que l'architecture de notre application analytique est générique, qu'elle peut traiter n'importe quel type d'UGV quels que soient leur taille et leur format. La deuxième idée est de prouver que notre modèle de régression, et plus précisément nos services d'entraînement et de prédiction, donnent des résultats précis.

4.1 Évaluation de notre approche

Dans cette section, nous évaluons l'efficacité de notre architecture de prédiction de vidéos populaires en utilisant les attributs calculés dans l'algorithme de partitionnement.

4.1.1 Architecture générique

Pour démontrer l'efficacité de notre application de prédiction, nous avons choisi un style *REST* pour notre architecture pour donner la flexibilité d'utilisation, surtout que nous avons travaillé sur des données de type vidéo. Notre idée est d'avoir une architecture générique, peu importe les limites de la communication entre le CDN et le *Cloud*. Cette opportunité est atteinte par l'utilisation de deux avantages, le premier est de mettre en pratique les différentes contraintes

d'une approche *REST*, et le deuxième est de mettre en œuvre le *Cloud* avec l'utilisation de notre modèle de prédiction.

Une architecture générique offre l'opportunité de maîtriser n'importe quel modèle de données utilisées. La généricité d'un modèle d'application rend leur utilisation facile, fiable et rapide. Nous avons choisi une solution qui s'adapte à plusieurs formats de données. Plusieurs procédures essentielles sont utilisées pour traiter chaque format *log*. Pour chaque format de données spécifiques, nous avons effectué dans notre application les étapes suivantes :

- Créer un client *REST* dédié à simuler l'interaction avec le CDN qui est dans notre cas le moteur de prédiction.
- Générer des données fictives en fonction du format de CDN pour former un banc d'essai.
- Effectuer l'entraînement des données générées.
- Simuler une requête pour les vidéos populaires et valider les résultats retournés par notre service de prédiction.

4.1.2 Description du jeu de données

La base de données Congas [1] que nous avons étudiée contient des statistiques sur des vidéos visionnées sur la plateforme YouTube. YouTube conserve un certain nombre de statistiques sur l'accès aux vidéos visionnées, sauf si le contenu indique le contraire. En effet, l'éditeur de la vidéo peut décider si les statistiques pourront être accessibles aux téléspectateurs ou non. Ces statistiques sont enregistrées selon le format suivant :

Watch Time : cette statistique indique le temps passé en minutes par les téléspectateurs à visualiser la vidéo. Cette information a été introduite à la base de données seulement en octobre 2012.

Subscribers : cette statistique indique le nombre d'abonnements au service concerné.

Views : cette statistique indique le nombre de vues marquées par la vidéo, c.-à-d le nombre des fois des utilisateurs de YouTube ont commencé à regarder la vidéo.

Shares : cette statistique indique le nombre de fois que la vidéo a été partagée par les utilisateurs YouTube.

---

1. Gongas. Consulté le 5 janvier 2017, sur https ://research.google.com/youtube8m/

Comments : cette statistique indique le nombre des commentaires écrits par les utilisateurs YouTube.

Toutes ces statistiques ont été fournies à la fois sur une base quotidienne et cumulative (depuis la publication du contenu). Un exemple de données de ces statistiques est présenté ci-dessous :

```
id:0Hzpbs9JKdo comments:22 nbrviews:76816 nbrShare:180 nbrWatchtime
    :145697.483333 subscribers:12
id:0I-jif-gIIc comments:1 nbrviews:16317 nbrShare:0 nbrWatchtime
    :9242.13333333 subscribers:2
id:0I0xisKZFKw comments:8 nbrviews:41989 nbrShare:19 nbrWatchtime
    :63401.4666667 subscribers:8
id:0I5spMisa5g comments:0 nbrviews:166 nbrShare:0 nbrWatchtime:109.45
     subscribers:0
id:0IJoKuTlvuM comments:25844 nbrviews:31565353 nbrShare:15577
    nbrWatchtime:1.47283238E7 subscribers:4185
```

Le but d'exporter ces statistiques depuis la base de données est de pouvoir les appliquer dans le calcul de notre algorithme. Notre approche considère toutes les statistiques sans éliminer aucun attribut de vidéo qui pourra influencer le calcul et par conséquent les résultats de la prédiction. Cette technique permet de pondérer le résultat de la prédiction de la popularité d'une vidéo en fonction de toutes les statistiques disponibles.

Nous avons calculé le minimum, le maximum, la moyenne, l'écart type et la variance du nombre de vues d'une vidéo sur l'ensemble de données de notre base de données. Le tableau 4.1 offre une illustration des calculs effectués. Le nombre de vues varie entre 0 pour les vidéos sans visualisation et 1.7 milliard pour les vidéos les plus populaires. La moyenne est de 5 millions de nombres de vues avec un écart type de 10 millions de vues. Cette grande variabilité dans le nombre de vues amène automatiquement une variabilité de popularité des vidéos étant donné la relation directe entre nombre de vues et popularité. La grande variabilité de la popularité des vidéos a alors nécessité un nombre de *clusters* supérieur ou égal à deux pour bien représenter la topologie des vidéos en termes de popularité. Ce nombre a été identifié en variant le paramètre k de l'algorithme K-moyennes.

Tableau 4.1: Statistique du jeu de données.

	Minimum	**Maximum**	**Moyenne**	**Écart type**	**Variance**
Nombre de vues	0.0	1.74E9	5.26E6	1.09E7	1.19E14

4.1.3 Mesure de la popularité

Le score de popularité est calculé à travers les différents attributs caractérisant la vidéo. Les attributs nécessaires qui décrivent une vidéo sont le temps de visionnement pendant lequel un utilisateur regarde la vidéo, le nombre des vues pour une vidéo, le nombre de partages d'une vidéo, le nombre d'abonnés et aussi le nombre de commentaires faits sur une vidéo.

La popularité d'un groupe est calculée en fonction du centroïde. Notre algorithme de K-moyennes a l'avantage de calculer les centroïdes pour chaque nouvelle vidéo pour chercher à quel groupe elle appartient. Le groupe le plus populaire est celui qui a un centroïde avec la plus grande norme. La norme du centroïde est la racine carrée de la somme des attributs suivants au carré : le temps de visionnement (Tv), le nombre de vues (Nv), le nombre de partages (Np), le nombre d'abonnés (Na) et le nombre de commentaires (Nc). La popularité d'une vidéo est aussi calculée selon la formule décrite dans 4.1.

$$\textit{Score de popularité} = || \textit{ centroide } || = \sqrt{Tv^2 + Nv^2 + Np^2 + Na^2 + Nc^2} \tag{4.1}$$

Notre approche, qui utilise tous les attributs pour calculer les centroïdes au lieu d'en utiliser un seul, permet à chaque attribut de jouer un rôle pour préciser la valeur de centroïde. De plus, prendre en considération plusieurs attributs dans le calcul rend notre algorithme plus fiable face aux attributs manquants pour une ou plusieurs vidéos.

En effet, avec la mesure de centroïde, chaque vidéo va se diriger vers son groupe d'appartenance. Pour identifier la popularité d'une nouvelle vidéo, nous calculons sa distance avec les centroïdes des groupes de popularité. Ensuite, nous affectons la vidéo au groupe le plus proche, c'est-à-dire avec la plus petite distance. Le calcul de la distance se fait en utilisant la distance euclidienne.

4.2 Résultats et discussion

Dans cette section, nous discutons les résultats d'application de l'algorithme de notre approche sur notre modèle analytique proposé. Mais avant de discuter les résultats obtenus, il convient de donner une idée sur les faux positifs et les faux négatifs.

Nous avons calculé le nombre de faux positifs quand une vidéo non populaire est identifiée comme populaire par notre algorithme de prédiction, ainsi que le nombre de faux négatifs quand une vidéo populaire est identifiée comme non populaire par notre calcul. Nous avons obtenu un taux de faux positifs de 0,21 % et un taux de faux négatif de 0,065 %. Seulement 65 vidéos ont été identifiées comme non populaires alors qu'elles avaient un nombre de vues très important. Le nombre de faux positifs est trois fois plus important. Nous pouvons conclure que notre modèle évalue légèrement à la hausse la popularité des vidéos analysées.

4.2.1 Métrique de succès des résultats

La métrique de succès est exprimée par le pourcentage de correspondance (*matching percentage* en anglais). Ce dernier donne le pourcentage de correspondance entre le résultat de l'entraînement et le résultat de la prédiction. Cet indicateur explique aussi la précision de la prédiction de notre algorithme. Pour chaque vidéo faisant partie du jeu de test de validation, nous avons comparé la popularité fournie par le nombre de vues (connue) et la popularité retournée par notre algorithme de prédiction (calculée). Si les deux informations concordent (populaire dans les deux cas ou non populaires dans les deux cas), on incrémente le nombre des résultats dits « bons ». Si, par contre, les deux observations ne correspondent pas, nous incrémentons le nombre des résultats dits « mauvais ». Le pourcentage de correspondance est ainsi calculé en divisant le nombre des résultats « bons » par le nombre total des observations.

$$Pourcentage\ de\ correspondance = \frac{nombre\ des\ (bons)\ résultats}{nombre\ total\ des\ observations} \tag{4.2}$$

Une prédiction est dite « bonne » si la popularité de la vidéo connue en fonction du nombre de vues correspond à la popularité retournée par notre algorithme de prédiction. Si la prédiction est bonne, nous incrémentons le nombre d'observations correctes. Cette information correspond à la colonne match comme dans le tableau 4.2. Si la prédiction est fausse, nous incrémentons plutôt le nombre d'observations incorrectes, la colonne non match dans le tableau 4.2. Cette opération est faite selon différents nombres des groupes en appliquant l'algorithme K-moyennes

Tableau 4.2: Précision de la prédiction selon le nombre de groupes.

Number of clusters	**Match**	**Non match**	**Execution time (ms)**
2	99964	36	141
3	99931	69	159
4	99864	136	300
5	99785	215	305
6	99724	276	578
7	99595	405	710
8	99546	454	815
9	99586	414	907
10	99142	858	1248
20	98374	1626	2578

avec un K différent à chaque fois.

La deuxième métrique que nous avons calculée pour évaluer notre algorithme est le temps d'exécution. Pour différents nombres de groupes, nous avons calculé le temps nécessaire à l'algorithme K-moyennes pour terminer l'entraînement du même ensemble de données, ce qui est à voir dans le tableau 4.2. Le temps commence avec l'initialisation de l'algorithme et se termine quand K-moyennes converge. Nous excluons le temps nécessaire pour charger les données à partir de la base de données, le temps de digestion des données, ainsi que le temps de formatage et la sauvegarde des résultats obtenus.

4.2.2 Performance du partitionnement

Nous avons effectué nos tests de performance sur une machine virtuelle. La machine utilise le système d'exploitation Ubuntu version Desktop. Nous avons utilisé comme banc d'essai une base de données réelle provenant de YouTube et contenant 100 000 vidéos. Pour évaluer la validité de notre solution, nous avons effectué l'entraînement sur 90 000 données et nous avons utilisé les 10 000 données restantes pour valider la fiabilité de notre solution. À chaque fois que notre moteur de prédiction effectue un calcul de popularité, nous comparons cela aux résultats réels et nous incrémentons le nombre des classifications correctes et le nombre des classifications incorrectes en conséquence. Le pourcentage des classifications correctes est visualisé dans le graphe de la Figure 4.1. Nous avons aussi calculé la durée d'exécution dont notre application a eu besoin

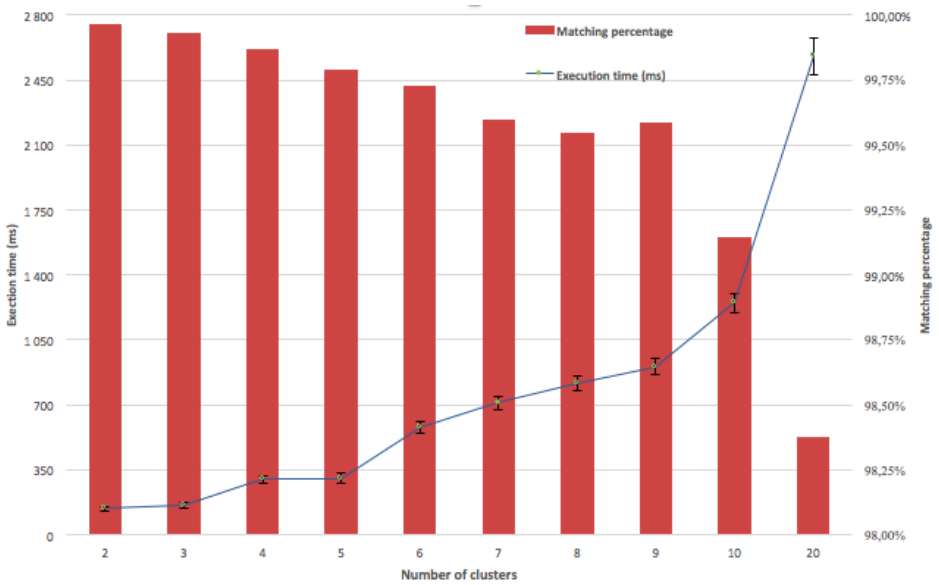

Figure 4.1: Temps d'exécution et pourcentage de correspondance en fonction du nombre de clusters.

pour prédire les 10 000 données. Enfin, pour observer comment notre solution réagit en fonction du nombre des *clusters*, nous avons effectué le même test sur des données groupées en 2, 3, 4, 5, 6, 7, 8, 9, 10 et 20 *clusters*.

Nous avons donc trouvé les résultats suivants :

- Notre solution a besoin de 1,4 seconde pour faire la prédiction de 10 000 vidéos avec un modèle de 10 *clusters* et 0,2 seconde pour un modèle de 2 *clusters*. Ce temps d'exécution est très acceptable dans un contexte de services *REST*. Avoir utilisé des NoSQL *database* a grandement contribué à la rapidité de notre application.
- Le temps d'exécution augmente linéairement en fonction du nombre de *clusters*, ce qui est acceptable contrairement à d'autres algorithmes qui réagissent en temps exponentiel. Le temps d'exécution reste en dessous de 3 secondes pour 20 *clusters*, ce chiffre est très acceptable en comparaison à d'autres algorithmes linéaires.
- Notre solution a donné 99 % de résultats valides, ce qui est très bon pour un algorithme de prédiction.
- La fiabilité de l'algorithme est meilleure quand nous avons moins des *clusters*, c'est presque

100 % avec 2 *clusters*. Par contre, si nous observons la tendance, nous pouvons nous attendre à de bons résultats, même avec un grand nombre de *clusters*. Cependant, la différence est faible entre les résultats.

– Pour mesurer la conformité de nos résultats, nous avons ajouté un calcul avec k=20. Nous avons obtenu qu'avec un modèle de 20 *clusters*, notre solution a besoin de 2,5 secondes. Le temps d'exécution est plutôt linéaire, même s'il ne l'est pas parfaitement. Donc l'allure linéaire de notre algorithme est validée.

À partir de la Figure 4.1, nous pouvons aussi identifier que le nombre optimal des *clusters* est deux. Par contre, diviser les vidéos en cinq groupes de popularités permet un bon compromis entre la rapidité d'exécution de notre algorithme (587 millisecondes) et la qualité de classification des données (99,8 % classifications réussies).

Effectivement, avec k=2, nous obtiendrons une qualité de solution suffisante. Par contre, nous avons ajouté une nouvelle contrainte à notre problème pour chercher une ségrégation entre les vidéos populaires. Cette ségrégation permettra de grouper les vidéos populaires en 3 groupes supplémentaires en fonction de leur degré de popularité. Cette contrainte peut se présenter dans plusieurs contextes où nous avons une limite sur la taille des *Buffers* des vidéos populaires à afficher ou sauvegarder.

Nous obtiendrons ainsi cinq *clusters* au lieu de deux. Nous avons démontré que la qualité des résultats reste très similaire dans le cas de deux *clusters* en termes de pourcentage de correspondance et aussi en termes de temps d'exécution de l'algorithme. Les cinq *clusters* ainsi formés sont les vidéos très populaires, assez populaires, moyennement populaires, peu populaires et impopulaires. Comparée à une prédiction binaire où le résultat indique si une vidéo est populaire ou non, la prédiction du niveau de popularité sur une échelle de 1 à 5 apporte une meilleure compréhension du résultat. D'ailleurs, le système d'évaluation 5 étoiles est souvent utilisé pour exprimer la popularité dans plusieurs domaines comme le web, la restauration, l'hôtellerie et autres services. Le résultat de la prédiction sera alors plus facilement compréhensible par le consommateur de la vidéo si jamais elle est affichée. Pour un CDN, avoir cinq niveaux de popularité apporte plus d'agilité sur le cache des vidéos. Nous avons considéré des scénarios où le serveur CDN enregistre les vidéos populaires progressivement en fonction de la bande passante et l'espace disque disponible. Le CDN peut enregistrer en priorité les vidéos avec une popularité plus élevée (*cluster* 5) qui est beaucoup plus petite que l'ensemble des vidéos populaires

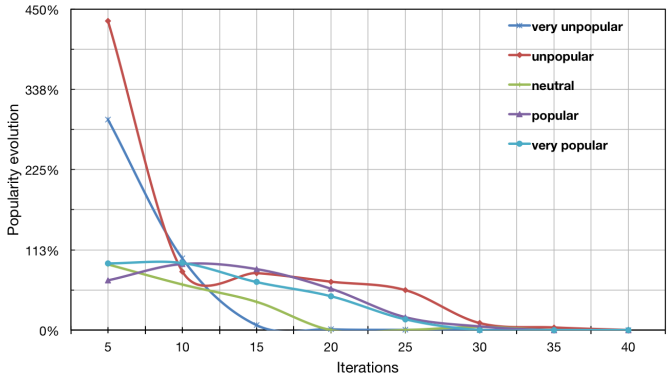

Figure 4.2: Les centroïdes.

en cas de classification binaire, c'est-à-dire des vidéos populaires et des vidéos non populaires. Avec ces cinq groupes, l'algorithme de formation converge très rapidement comme décrit dans la Figure 4.2, il n'a fallu que 30 itérations de K-moyennes pour regrouper les 100 000 vidéos en cinq groupes de popularité distincts.

4.2.3 Comparaison de notre modèle avec le modèle de régression hybride

Pour évaluer la précision de notre modèle d'entraînement, nous avons effectué une comparaison avec un modèle de régression hybride utilisé dans la prédiction de la popularité des vidéos (Ben Abdelkrim *et al.*, 2016). Comparé au modèle de régression hybride qui limite le nombre de variables utilisées dans la classification à un nombre réduit, notre modèle inclut l'ensemble des variables fournies par la base de données.

Nous avons étudié l'évolution des centroïdes d'un *cluster* donné, soit dans notre cas, le groupe des vidéos le plus populaire, pour les variables *Watch Time*, *Views* et *Shares* dans deux configurations. La première configuration *All fields* utilise l'ensemble des variables existantes dans la base de données YouTube pour faire la classification. La deuxième configuration *Reduced* utilise seulement deux variables, le *Watch time* et le *Shares*.

La Figure 4.3 montre l'évolution de centroïdes du *cluster* pour les variables *Watch time*, *Views* et *Shares* pendant dix périodes. Chaque période correspond à 10 000 vidéos.

Si nous comparons par exemple les deux courbes pour *Watch time*, nous remarquons qu'elles évoluent presque de la même façon. Nous pouvons donc constater que le partitionnement des vidéos sera similaire à la régression hybride, par contre notre approche est plus simple parce

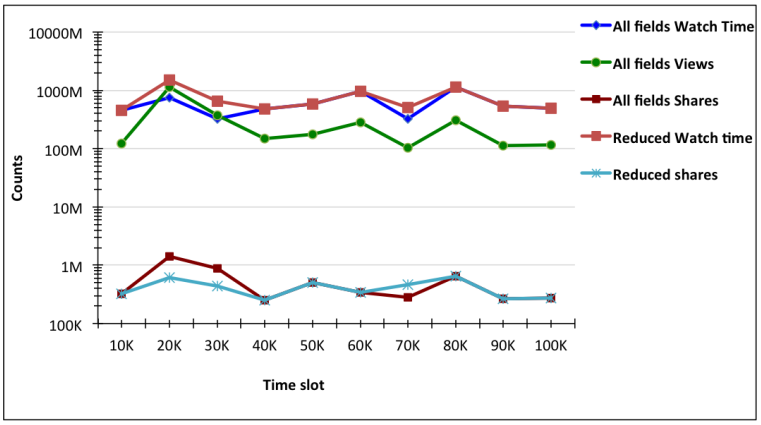

Figure 4.3: Évolution des centroïdes d'un *cluster* donné en fonction de *time slot* pour différentes configurations des variables.

qu'elle inclut toutes les variables et évite un processus additionnel de sélection préalable. Un tel processus pourra introduire des erreurs dans le cas où les variables principales de classification se font écarter par négligence.

Si nous nous penchons maintenant sur la répartition des vidéos en *clusters* de popularité à la convergence de notre algorithme d'entraînement, nous remarquons que seulement 255 vidéos appartiennent aux *clusters* 2, 3, 4 et 5 alors que plus de 99 % des vidéos appartiennent au *cluster* 1. En analysant les données, nous avons remarqué qu'effectivement seulement un nombre limité des vidéos est vraiment populaire, alors que la plus grande majorité a un nombre de vues égal à zéro. En réalité, YouTube garde les statistiques pour une période limitée dans le temps. La figure 4.4 décrit le changement au niveau de la répartition des vidéos par groupes en fonction du nombre d'itérations de l'algorithme de partitionnement.

L'évolution d'un groupe i à l'itération j est calculée de la manière suivante :

$$N_{i,j} = \frac{|n_{i,j} - n_{i,(j-1)}|}{n_{i,(j-1)}} \tag{4.3}$$

tel que $n_{i,j}$ est le nombre de vidéos dans le groupe i à l'itération j. $n_{i,(j-1)}$ est le nombre de vidéos dans le groupe i à l'itération $j-1$.

Cette mesure sert comme critère de convergence pour notre algorithme. L'algorithme s'arrête quand il enregistre très peu de mouvement de classification des données entre les *clusters*.

Iterations	2		5		10		15	
	Count	% Evolution	Count	% Evolution	Count	% Evolution	Count	% Evolution
Cluster 1	11651	295%	46033	100%	92228	7%	98470	1%
Cluster 2	6801	433%	36261	-82%	6445	-80%	1298	-68%
Cluster 3	1317	-92%	105	-64%	38	-39%	23	0%
Cluster 4	51373	-69%	15692	-93%	1173	-85%	171	-58%
Cluster 5	28858	-93%	1909	-94%	116	-67%	38	-47%

Iterations	20		25		30		35	
	Count	% Evolution	Count	% Evolution	Count	% Evolution	Count	% Evolution
Cluster 1	99467	0%	99717	0%	99739	0%	99745	0%
Cluster 2	418	-56%	184	-10%	166	-4%	160	0%
Cluster 3	23	0%	23	-4%	22	0%	22	0%
Cluster 4	72	-18%	59	-5%	56	0%	56	0%
Cluster 5	20	-15%	17	0%	17	0%	17	0%

Figure 4.4: Nombre de vidéos par *cluster* en fonction de nombre d'itérations de l'algorithme de partitionnement.

Par exemple, si un groupe double le nombre des vidéos dans une itération, le pourcentage d'évolution affiche 200 %. Le partitionnement converge quand il y a peu de changements ou quand il n'y en a plus dans les répartitions de vidéos entre les groupes. On peut interpréter à partir de ce tableau que le partitionnement de 100 000 vidéos converge après 30 itérations. Cette mesure explique la rapidité du traitement.

4.3 Conclusion

Dans ce chapitre, nous avons évalué les performances de notre application d'analytique en tant que service proposé pour la prédiction. Les résultats obtenus montrent que notre solution donne de très bonnes performances en termes d'exactitude de prédiction des vidéos populaires. En plus, le temps d'exécution de l'algorithme d'entraînement et de prédiction est très acceptable pour un déploiement presque en temps réel. Nous avons aussi expliqué comment nous avons réussi à déployer une architecture générique qui ne dépend pas des attributs spécifiques.

CONCLUSION

Nous avons proposé une application analytique pour la prédiction des vidéos populaires. Cette application a été développée en tant que service web que nous avons appelé AaaS. Nous avons aussi présenté une architecture *RESTful* pour communiquer notre AaaS aux CDNs ayant besoin d'utiliser les résultats de l'analyse pour mieux gérer le cache des vidéos à diffuser. Cela permet au CDN de mieux optimiser la bande passante et l'espace de stockage.

Pour implémenter notre solution de prédiction, nous avons utilisé différentes techniques et technologies qui sont plutôt connues dans le domaine du *Big Data* et dans le développement des applications web et mobiles modernes. Nous avons par exemple utilisé des bases de données NoSQL pour sauvegarder les *logs* provenant de différents CDNs. Nous avons aussi développé des API *REST* pour faciliter les communications entre le CDN et l'AaaS. Pour le codage et le déploiement, l'application analytique a été codée en Java et déployée sur un serveur applicatif web. Nous avons utilisé l'algorithme K-moyennes pour la classification des vidéos provenant de chaque CDN en dix groupes de popularité et pour construire un modèle de prédiction spécifique à chaque CDN et à chaque application analytique.

L'utilisation de ces techniques nous a permis de développer une architecture générique. Notre application analytique peut supporter différents CDNs quel que soit le format des traces *logs* générées. Notre AaaS sauvegarde les données de *log* pour chaque CDN. Ensuite, il effectue l'entraînement pour construire un modèle de prédiction spécifique qui servira plus tard à déterminer quelles sont les vidéos populaires qui devront être priorisées dans la gestion du cache du côté du CDN.

Pour tester notre solution, nous avons utilisé la base de données Congas qui contient l'historique d'utilisation de cent mille données collectées par le CDN YouTube. Nous avons utilisé une partie des données pour faire l'entraînement et l'autre partie pour valider notre modèle de prédiction. Nous avons obtenu un pourcentage de 99 % des prédictions correctes des vidéos populaires.

Pour valider l'efficacité de notre modèle générique, nous avons comparé les résultats de notre modèle avec celles du modèle de régression linéaire. La comparaison a montré que notre modèle analytique donne les mêmes performances de prédiction, mais en utilisant l'ensemble des attri-

buts collectés par le CDN. La régression linéaire effectue une réduction préalable des dimensions qui peut introduire des erreurs quand la structure des données change ou dans le cas où de nouveaux attributs sont ajoutés. Notre solution de prédiction est générique, il est donc facile de l'utiliser pour optimiser d'autres types de contenu comme les images à haute résolution. Notre architecture pourra aussi être déployée dans d'autres domaines d'application comme la gestion des incidents ou en sécurité dans l'identification des comportements anormaux. Par contre, il serait important de prévoir une implémentation sur une infrastructure distribuée pour assurer une meilleure capacité de supporter la charge et cela pourrait être le sujet d'un travail futur.

RÉFÉRENCES

Adrian, B. et Heryawan, L. (2015). Analysis of k-means algorithm for vm allocation in cloud computing. Dans *2015 International Conference on Data and Software Engineering (ICoDSE)*, 48–53.

Ahmad, S., Singh, P. et Sagar, A. K. (2018). A survey on big data analytics. Dans *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, 256–260.

Aloui, M., Elbiaze, H., Glitho, R. et Yangui, S. (2018). Analytics as a service architecture for cloud-based cdn : Case of video popularity prediction. Dans *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 1–4.

Ben Abdelkrim, E., Salahuddin, M. A., Elbiaze, H. et Glitho, R. (2016). A hybrid regression model for video popularity-based cache replacement in content delivery networks. Dans *2016 IEEE Global Communications Conference (GLOBECOM)*, 1–7.

Bermudez, I., Traverso, S., Munafò, M. et Mellia, M. (2014). A distributed architecture for the monitoring of clouds and cdns : Applications to amazon aws. *IEEE Transactions on Network and Service Management*, *11*(4), 516–529.

Bu, F., Chen, Z., Zhang, Q. et Wang, X. (2014). Incomplete big data clustering algorithm using feature selection and partial distance. Dans *2014 5th International Conference on Digital Home*, 263–266.

Chawda, R. et Thakur, G. (2016). Big data and advanced analytics tools. Dans *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, 1–8.

Demchenko, Y., de Laat, C. et Membrey, P. (2014). Defining architecture components of the big data ecosystem. Dans *2014 International Conference on Collaboration Technologies and Systems (CTS)*, 104–112.

developers google. (2020). *Clustering in Machine Learning*.

Fielding, R. T. et Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Irvine.

Hu, H., Wen, Y., Chua, T. et Li, X. (2014). Toward scalable systems for big data analytics : A technology tutorial. *IEEE Access*, *2*, 652–687.

Jagli, D., Purohit, S. et Nalla, S. C. (2016). Implementation of k-means clustering for evaluating saas on the cloud computing environment. Dans *2016 International Conference on ICT in Business Industry Government (ICTBIG)*, 1–5.

Li, C., Liu, J. et Ouyang, S. (2016). Characterizing and predicting the popularity of online videos. *IEEE Access*, *4*, 1630–1641.

Malviya, A., Udhani, A. et Soni, S. (2016). R-tool : Data analytic framework for big data. Dans *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, 1–5.

Niu, K., Gao, Z., Jiao, H. et Deng, N. (2016). K-means+ : A developed clustering algorithm for big data. Dans *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, 141–144.

Panchal, P., Ramaswamy, N. M., Su, X. et Dong, Y. (2013). Content delivery networks in the cloud with distributed edge servers. Dans *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, 526–532.

Pijetlović, S., Jovanov, N., Vukobrat, V. et Basicevic, I. (2014). One solution of a restful api for a cloud based dtv content provider. Dans *2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin)*, 384–387.

Praveena, M. D. A. et Bharathi, B. (2017). A survey paper on big data analytics. Dans *2017 International Conference on Information Communication and Embedded Systems (ICICES)*, 1–9.

Su, B., Wang, Y. et Liu, Y. (2016). A new popularity prediction model based on lifetime forecast of online videos. Dans *2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, 376–380.

Tatar, Antoniadis, D. L. L. F. (2011). Predicting the popularity of online articles based on usercomments. *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*.

Trevino, A. (2016). Oracle ai and data science blog.

Trzciński, T. et Rokita, P. (2017). Predicting popularity of online videos using support vector regression. *IEEE Transactions on Multimedia*, *19*(11), 2561–2570.

Vashisht, P. et Gupta, V. (2015). Big data analytics techniques : A survey. Dans *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, 264–269.

Ward, J. et Barker, A. (2013). Undefined by data : A survey of big data definitions.

WikiStat. (2020). *Classification non supervisée*.

Wu, J., Zhou, Y., Chiu, D. M. et Zhu, Z. (2016). Modeling dynamics of online video popularity. *IEEE Transactions on Multimedia*, *18*(9), 1882–1895.

Zeni, M., Miorandi, D. et De Pellegrini, F. (2013). Youstatanalyzer : A tool for analysing the dynamics of youtube content popularity. Dans *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '13, p. 286–289., Brussels, BEL. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Zhang, P., Xiong, F., Gao, J. et Wang, J. (2017). Data quality in big data processing : Issues, solutions and open problems. Dans *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation*, 1–7.