Ingénierie de la fouille et de la visualisation de données massives (RCP216)

Fouille de flux de données

Michel Crucianu
(prenom.nom@cnam.fr)
http ://cedric.cnam.fr/vertigo/Cours/RCP216/

Département Informatique
Conservatoire National des Arts & Métiers, Paris, France

26 octobre 2021

Plan du cours

Flux de données : spécificités

- Flux : succession de données de même type, arrivant à intervalles constants ou variables
- Cas d'intérêt :
  - Appliquer un modèle régulièrement sur les dernières données du flux afin de caractériser l'évolution du flux
  - Mettre à jour un modèle régulièrement sur les ($\sim$ dernières) données du flux afin de caractériser le même flux (ou un autre)

$\Rightarrow$ Spécificités :

- Données massives : en raison du *débit* très élevé du flux de données, il n'est pas envisageable de traiter *l'ensemble* des données régulièrement
- Le traitement des « tranches » (*slices*) de données doit être fait en temps réel (ou, au moins, il ne faut pas prendre de retard sur le flux)
- La modélisation doit être *incrémentale* si elle est faite à partir de données en flux
- La mémoire vive étant en quantité limitée et l'accès au stockage de masse trop lent, il est souvent nécessaire de *résumer* les données du flux avant de les traiter de façon plus approfondie

Fouille de flux de données : exemples d'applications

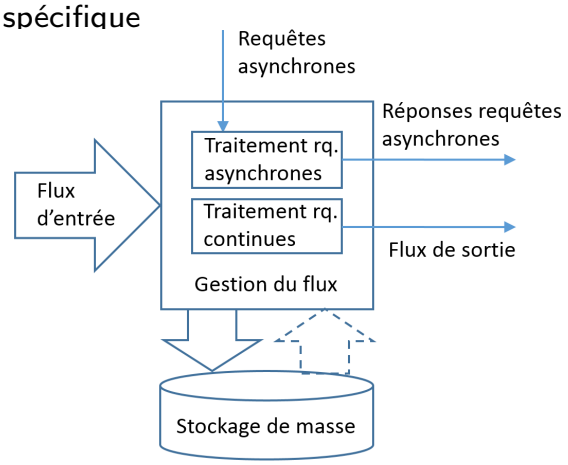
- Réseaux de capteurs : optimiser le fonctionnement des machines, optimiser la planification des maintenances, prévoir les pannes, etc.
  - Exemple de volumétrie : env. 1 To de données générées par heure pour un moteur d'avion de ligne
- Logs de moteurs de recherche : prévoir la demande de produits ou d'informations, prévoir des épidémies à partir de requêtes, etc.
  - Exemple de volumétrie : près de 50.000 recherches par seconde pour Google
- Données entrantes sur sites de partage vidéo : détection d'événements, mesures de popularité, mesures d'impact, etc.
  - Exemple de volumétrie : 300 h de vidéo sont mises en ligne chaque minute sur YouTube

Fouille de flux de données : exemples d'applications (2)

- Données de vidéosurveillance : détection d'événements localisés, détection de corrélations entre faits distants, etc.
  - Exemple de volumétrie : près de 500.000 flux de vidéosurveillance à Londres
- Imagerie satellite : détection d'événements, suivi de l'évolution de l'impact d'événements climatiques, etc.
  - Exemple de volumétrie : les deux satellites Pleiades 2 envoient env. 2 To de données par jour

Architecture d'un système de traitement de flux de données

- Gestion : résumer flux d'entrée, application de modèles sur le flux, construction / mise à jour de modèles à partir du flux
- Stockage du flux d'entrée : entier ou résumé, but d'archivage
- Requêtes :
  - Continues ($\rightarrow$ flux de sortie), par ex. nombre de requêtes par jour concernant une thématique spécifique
  - Asynchrones ($\rightarrow$ réponses), par ex. nébulosité atmosphérique moyenne sur 3 derniers jours sur une aire spécifique



Échantillonnage dans un flux

- Objectif : extraire du flux un échantillon suffisamment représentatif pour permettre de répondre à plusieurs questions et suffisamment petit pour pouvoir être gardé sur un horizon assez long en stockage rapide
- En général, chaque donnée d'un flux donne des valeurs à plusieurs attributs :
  - Réseau de capteurs : identité du capteur (ou position GPS), étiquette temporelle, valeur du paramètre mesuré
  - Logs de moteurs de recherche : adresse IP de l'utilisateur, étiquette temporelle, mots-clés de la recherche, type de navigateur, etc.
  - Imagerie satellite : position du satellite, paramètres de prise de vue, étiquette temporelle, image
- Problème : un échantillonnage ne tenant pas compte des valeurs des attributs ne permet pas de répondre à certaines questions, par exemple
  - Réseau de capteurs : quel pourcentage de capteurs indiquent une variation journalière supérieure à θ du paramètre mesuré ?
  - Logs de moteurs de recherche : quel pourcentage des requêtes de l'utilisateur typique sont répétées durant un intervalle d'un mois ?
  - Imagerie satellite : pour quel pourcentage de positions les paramètres de prise de vue sont les mêmes ?

Échantillonnage dans un flux (2)

- Analyse de l'exemple concernant les logs de moteurs de recherche (quel pourcentage des requêtes de l'utilisateur typique sont répétées durant un intervalle d'un mois, voir aussi [1]) :
  - Supposons que nous souhaitons sélectionner 10% des requêtes et que cette sélection est faite sans tenir compte des valeurs des attributs (adresse IP, étiquette temporelle, mots-clés de la recherche, etc.)
  - Supposons que l'utilisateur typique fait en moyenne s requêtes non répétées par mois, d requêtes répétées une fois et un nombre négligeable de requêtes répétées plusieurs fois
  - Sur les s requêtes non répétées dans le flux, $s/10$ seront présentes (une fois) dans l'échantillon
  - Sur les d requêtes répétées (une fois) dans le flux, seulement $d/100$ $(= \frac{1}{10} \cdot \frac{1}{10})$ seront répétées (une fois) dans l'échantillon et $18d/100$ $(= \frac{1}{10} \cdot \frac{9}{10} + \frac{9}{10} \cdot \frac{1}{10})$ seront présentes dans l'échantillon une seule fois
  - A partir de l'échantillon nous obtenons une estimation de $\frac{d}{10s+19d}$, très loin de la valeur obtenue à partir de toutes les requêtes qui est de $\frac{d}{s+d}$

Échantillonnage dans un flux (3)

- Solution : choisir un utilisateur (ne faisant pas encore partie de l'échantillon) avec une probabilité de 0,1 et garder toutes ses requêtes $\rightarrow$ possible d'analyser complètement le comportement d'env. 10% des utilisateurs
- Si la conservation en mémoire de la liste des utilisateurs de l'échantillon est trop coûteuse, il est possible d'employer une méthode de hachage : par ex., avec 10 valeurs de *hash* différentes, on conserve la requête si la valeur de *hash* est celle fixée *a priori* pour l'échantillon

Échantillonnage dans un flux (4)

- La solution du hachage peut être facilement étendue pour permettre
  - Un réglage plus fin des pourcentages : par ex., avec 100 valeurs de *hash* différentes, en conservant la requête si la valeur de *hash* fait partie d'un ensemble fixé *a priori* à x valeurs, un échantillon de $x\%$ est obtenu
  - Une variation dans le temps du pourcentage (par exemple, lorsque le débit augmente) : un hachage choisi pour le réglage fin permet facilement de réduire progressivement les pourcentages
  - La sélection sur la base des valeurs de plusieurs attributs : si un hachage est mis en place pour chaque attribut dont la valeur compte, il est possible de faire des échantillonnages par rapport à des attributs individuels ou des groupes d'attributs

Filtrage dans un flux

- Objectif : sélectionner une partie du flux afin de lui appliquer un traitement particulier
- Problème : le filtrage par comparaison des valeurs d'un attribut à un ensemble très grand de valeurs peut être très coûteux à la fois en temps et en capacité mémoire mobilisée
- Exemple :
  - Un site de partage vidéo souhaite interdire, pour diverses raisons (présence d'incitations à la haine, contenus dégradants, non respect du droit d'auteur, etc.), la mise en ligne de vidéos répertoriées
  - Les vidéos sont identifiées par un ensemble de descripteurs (« signatures ») extraits de certaines trames vidéo particulières
  - Pour chaque vidéo, l'ensemble des descripteurs (nous l'appellerons « signature » de la vidéo) est volumineux
  - Le nombre de vidéos « interdites » est élevé

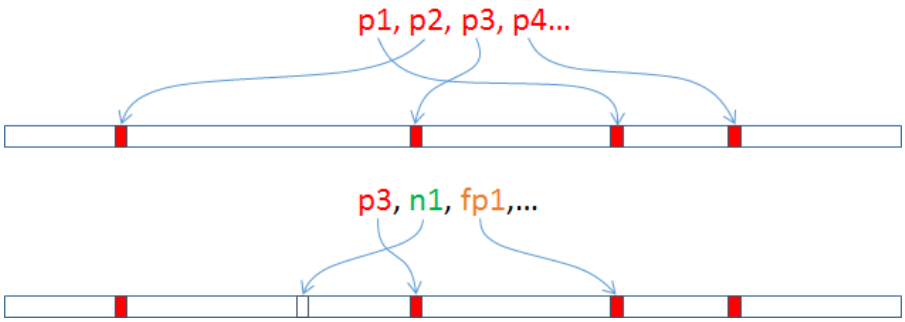
Filtrage dans un flux avec un filtre de Bloom

- Comment effectuer le filtrage
  1. Rapidement et avec relativement peu de ressources mémoire
  2. En garantissant que les vidéos « interdites » ne se retrouvent pas en ligne
  3. En acceptant éventuellement que certaines autres vidéos (« bénignes ») soient également supprimées

→ Principe du filtre de Bloom :

- Considérer une fonction de hachage qui prend en entrée une signature de vidéo et lui associe un *hash* dans un ensemble très grand (= ensemble de bits de la zone mémoire utilisée pour le filtrage) ; mettre à 0 tous les bits de cette zone mémoire
- Appliquer la fonction de hachage à la signature de chaque vidéo interdite et mettre à 1 le bit identifié par le *hash* obtenu ; supposons que la fraction de bits à 1 est $0 < r \ll 1$
- Lorsqu'une vidéo arrive (par *upload*), on calcule sa signature, on applique la fonction de hachage à la signature, si la valeur du bit identifié par le *hash* est 1 alors la vidéo n'est pas mise en ligne
- Si la vidéo fait partie de l'ensemble des vidéos interdites, le bit identifié par le *hash* est 1 par construction (→ pas de faux négatifs)
- Si la vidéo ne fait pas partie de l'ensemble, la probabilité pour que le bit identifié par le *hash* soit 1 (et donc que la vidéo soit à tort considérée interdite) est proche de r (→ il y a donc des faux positifs)

Filtrage dans un flux avec un filtre de Bloom (2)

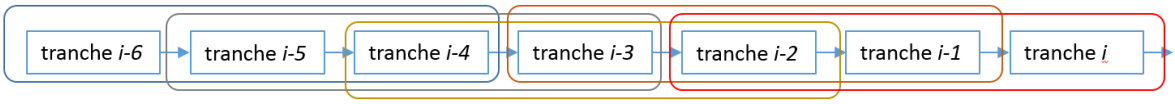


- Pour réduire la probabilité de faux positifs on utilise plusieurs fonctions de hachage indépendantes ; une vidéo est considérée interdite si et seulement si le bit identifié par le *hash* est 1 pour chaque fonction de hachage
- Avec M vidéos interdites, k fonctions de hachage, chacune avec N valeurs de *hash* différentes, alors le taux de faux positifs $\approx (1 - e^{-kM/N})^k$, pour une occupation mémoire de kN bits

Utilisation de fenêtres temporelles

- Motivation :
  - Même en conservant un échantillon du flux, le volume de données accumulées sur une longue période peut être tel que la capacité à traiter l'ensemble de l'échantillon soit limitée
  - Les distributions des valeurs des différents attributs qui décrivent les données du flux sont souvent non stationnaires, remonter loin dans l'historique peut être peu pertinent

$\Rightarrow$ Traitement des données situées dans une fenêtre temporelle glissante, souvent de longueur fixe



Utilisation de fenêtres temporelles (2)

- Fenêtres avec oubli (*decaying windows*) :
  - Fenêtres standard : une tranche est soit dans la fenêtre (donc prise en compte), soit en dehors de la fenêtre (donc ignorée)
  - Si un « lissage » des estimations est souhaité, alors il est possible d'utiliser des pondérations qui diminuent avec l'écart temporel
  - Exemple : flux de nombres m_i d'achats d'un article (i est la tranche du flux), indice de popularité s_i calculé avec décroissance exponentielle (pour comparer la popularité de différents articles) :

$$s_i = m_i + s_{i-1} \cdot d \;\Rightarrow\; s_i = \sum_{j=1}^{i} d^{i-j} \cdot m_j \tag{1}$$

  avec $d < 1$ et $s_0 = 0$
  - Autre avantage (à part le lissage) : il suffit de conserver l'estimation courante et non l'ensemble des éléments d'une fenêtre !

Streaming K-means

- Objectif : actualiser les centres des groupes à partir des données d'un flux et donner des étiquettes de groupe aux données du même flux (ou d'un autre)

Data : Ensembles $\mathcal{E}_t$ de n_t données de $\mathbb{R}^p$, pour $t = 1, 2, \ldots$
Result : après chaque tranche, k groupes (*clusters*) disjoints $\mathcal{E}_{1,t}, \ldots, \mathcal{E}_{k,t}$ (sous-ensembles de $\mathcal{E}_t$) et ensemble $\mathcal{C}_t$ de leurs centres

1 Initialisation aléatoire des centres $\mathbf{m}_{j,1}$, $1 \leq j \leq k$, à partir des données de la première tranche ;
2 **for** *chaque tranche* $\mathcal{E}_t$ *du flux* **do**
3 | Affectation de chaque donnée de la tranche au groupe du centre le plus proche ;
4 | Mise à jour des centres et remplacement des anciens centres par les nouveaux ;
5 **end**

Streaming K-means

- Mise à jour des centres avec les données de chaque tranche, en tenant compte des données plus anciennes à travers les anciens centres :

$$\mathbf{m}_{j,t+1} = \frac{\alpha N_{j,t}\mathbf{m}_{j,t} + n_{j,t}\mathbf{x}_{j,t}}{\alpha N_{j,t} + n_{j,t}}$$

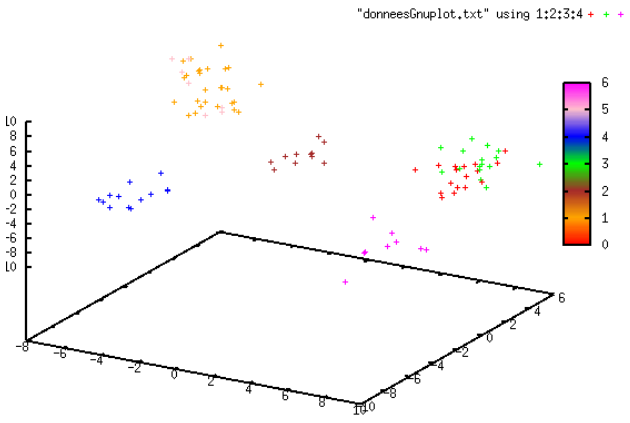
$$N_{j,t+1} = N_{j,t} + n_{j,t}$$

où

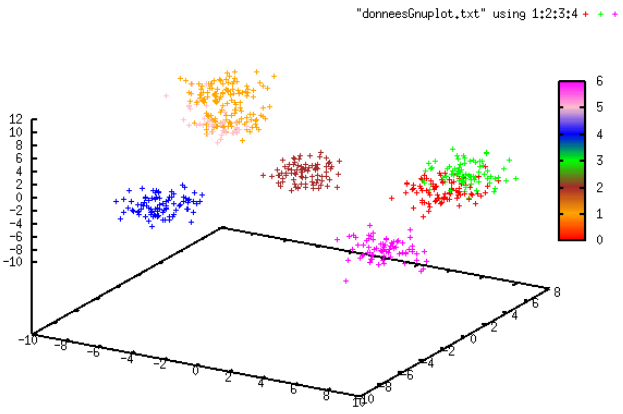
- $\mathbf{m}_{j,t+1}$ est le centre j mis à jour dans l'itération courante
- $\mathbf{m}_{j,t}$ est le centre j avant sa mise à jour
- $n_{j,t}$ est le nombre de données de la tranche courante affectées au centre j
- $N_{j,t}$ est le nombre de données affectées au centre j depuis le début
- $\mathbf{x}_{j,t}$ est le centre correspondant calculé uniquement avec les données de la tranche courante
- α est le facteur d'oubli (aucun oubli si $\alpha = 1$, oubli total si $\alpha = 0$)

Spark Streaming : illustration

- Classification automatique avec *streaming K-means* d'un flux de données de $\mathbb{R}^3$
- Sept lois d'assez faible variance ont été utilisées pour générer les données

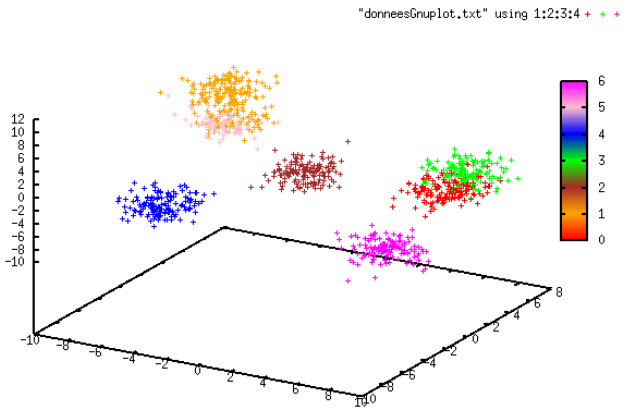


Après l'arrivée des premières données

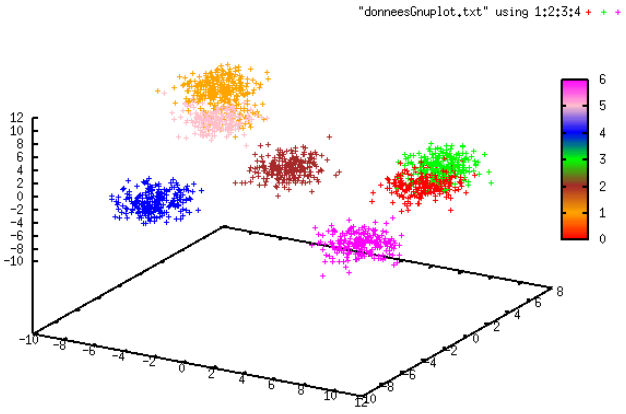


Après l'arrivée de 30% des données

Spark Streaming : illustration (2)



Après l'arrivée de 60% des données

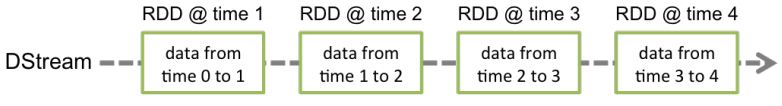


Après l'arrivée de toutes les données

Spark Streaming

- Sources : système de fichiers, *sockets* TCP, Kafka, Flume, Twitter, ZeroMQ, Kinesis

1. API DataFrame/Dataset de flux : opérations de projection, jointure, agrégation, filtrage, transformation
2. API basée sur les RDD : flux discrétisé (`DStream`) = séquence de RDD
   - Créé à partir d'un `StreamingContext`, en précisant la durée d'une tranche (*slice*)
   - Les données du flux sont découpées en tranches en respectant cette durée, chaque tranche est un RDD qui peut ensuite être transformé comme tout RDD

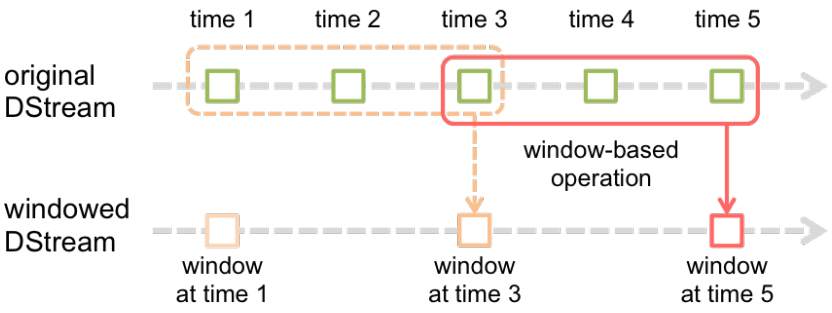


Spark Streaming (2)

- Opérations qui dépendent de l'API : flux transformé → flux, fusion de flux, jointure de flux, jointure flux avec RDD fixe, filtrage d'un flux par un autre, mise à jour d'un « état » à partir d'un flux, construction et/ou application de modèles (*Streaming Linear Regression*, *Streaming KMeans*)
- Utilisation facile à partir d'un programme (inclure les dépendances et construire un .jar avec Maven ou SBT) ou à partir de `spark-shell`
- Toute source (à l'exception de la source *filesystem*) doit avoir un `Receiver` associé pour récupérer les données du flux, les découper en tranches et transmettre les tranches aux nœuds de calcul ; lors d'une utilisation locale avec un `Receiver` il est nécessaire d'avoir au moins 2 *threads* (par ex. `.setMaster("local[2]")` pour `SparkConf`)

Spark Streaming : utilisation de fenêtres glissantes

- Une fenêtre est définie par deux nombres entiers de tranches :
  - Combien de tranches sont dans la fenêtre à un moment donné (longueur de la fenêtre)
  - De combien de tranches sont les déplacements de la fenêtre (pas du glissement)
- Les RDD correspondant aux tranches qui sont dans la fenêtre sont regroupés et traités ; les opérations appliquées sur les RDD individuels (tranches) sont étendues aux fenêtres (par ex. `reduceByKeyAndWindow`)



Références I


A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.