Interrogation des résumés de flux de données

Nesrine Gabsi*,**, Fabrice Clérot **
Georges Hébrail*

*Institut TELECOM ; TELECOM ParisTech ; CNRS LTCI
46, rue Barrault 75013 Paris
prénom.nom@telecom-paristech.fr,
** France Telecom RD
2, avenue P.Marzin 22307 Lannion
prénom.nom@orange-ftgroup.com

Résumé. Les systèmes de gestion de flux de données (SGFD) ont été conçus afin de traiter une masse importante de données produites en ligne de façon continue. Etant donné que les ressources matérielles ne permettent pas de conserver toute cette volumétrie, seule la partie récente du flux est mémorisée dans la mémoire du SGFD. Ainsi, les requêtes évaluées par ces systèmes ne peuvent porter que sur les données les plus récentes du flux. Par conséquent, les SGFD actuels ne peuvent pas traiter des requêtes qui portent sur des périodes très longues. Nous proposons dans cet article, une approche permettant d'évaluer des requêtes qui portent sur une période plus longue que la mémoire du SGFD. Ces fenêtres font appels à des données récentes et des données historisées. Nous présentons le niveau logique de cette approche ainsi que son implantation sous le SGFD Esper. Une technique d'échantillonnage associée à une technique de fenêtre point de repère est appliquée pour conserver une représentation compacte des données du flux.

1 Introduction

Contrairement aux systèmes transactionnels classiques où les données sont conservées avant d'être traitées, les Systèmes de Gestion de Flux de Données (SGFD)(Babcock et al. (2002a)) effectuent un traitement à la volée en une seule passe (sans conservation a priori des données) et permettent de poser des requêtes continues. Compte tenu du volume et du débit des données, les SGFD ne conservent que les données récentes ou celles du passé récent dans des structures appelées *fenêtres*. Ces données ne peuvent être analysées a posteriori. Il est ainsi nécessaire, pour tout besoin d'analyse sur les flux, de préciser a priori sa tâche avant l'arrivée des données. Cependant, si un nouveau besoin portant sur des données écoulées est exprimé, ces tâches ne sont plus réalisables. Une solution permettant le traitement a posteriori consiste à conserver un résumé du flux. Il s'agit de résumer le contenu du flux de façon à construire un modèle résumé qui, bien que beaucoup plus petit en taille, permet de répondre à ces tâches mais d'une manière approchée.

Les résumés des flux sont typiquement stockés dans des bases de données permettant aux utilisateurs de les interroger via un langage d'interrogation classique. La conservation de résumé permet de poser des requêtes sur le passé. Cependant, dans plusieurs applications, il est nécessaire de poser des requêtes qui portent à la fois sur des données du passé (*données historiques* mémorisées dans le résumé) et les données du présent (données conservées en mémoire du SGFD). Un exemple de requête dans le domaine financier consiste à envoyer une alerte si l'indice boursier observé sur les 5 dernières heures s'écarte de plus de 20% de celui observé le même jour il y a un mois.

L'évaluation de telles requêtes pose un certain nombre de problèmes, le traitement d'une requête qui porte sur une période très longue ne peut être réalisée ni par les SGBD (qui ne sont pas capables de prendre en considération la période récente du flux) ni par les SGFD (qui ne sont pas capables de prendre en compte le passé).

Ce papier présente une nouvelle approche permettant d'évaluer des requêtes qui portent sur une longue période nécessitant ainsi des données du présent et des données du passé.

2 Travaux connexes

Généralement, il existe trois scénarios possibles pour évaluer des requêtes sur un système de gestion de données. Le premier consiste à interroger des données archivées (*données historiques*) qui, sont présentent dans le système avant l'arrivée de requêtes. Cet axe est largement étudié par les SGBD traditionnels. Le second scénario consiste à poser des requêtes continues sur des données éphémères. Des recherche dans ce sujet ont permis de développer les SGFD ainsi que des nouveaux langages d'interrogation (ex. CQL(Arasu et al. (2003))) qui permettent de traiter les données récente des flux. La dernière classe de requêtes fait intervenir des données archivée et des données récentes. L'interrogation de telles requêtes est un sujet très récent et encore ouvert.

Dans Chandrasekaran et Franklin (2004), les auteurs s'intéressent au traitement de requêtes qui font intervenir le passé. L'idée générale consiste à évaluer des requêtes qui portent sur des données du présent concaténées avec des données historiques. Cependant, l'article s'intéresse à des requêtes de courtes durées (ex. 5 minutes, une plage horaire précise dans la journée, une semaine) et à minimiser le coût des entrées/sorties dans l'interrogation de la base. En revanche, dans cette communication, notre intérêt se porte sur des périodes temporelles glissantes[1] et de longues durées (ex. des mois, des années).

Il existe plusieurs approches de résumé : approches dites simples (ex. Reservoir Sampling(Vitter (1985))) et approches dites complexes (ex. StreamSamp(Csernel et al. (2006))). L'interrogation des résumés simples est triviale mais la qualité des résultats est médiocre. Dans le cas du reservoir sampling les poids associés au échantillons augmentent linéairement, ce qui rend insatisfaisante la qualité des réponses aux requêtes. Tandis que l'interrogation des résumés complexes est difficile mais fournit une meilleur performance. Cet article présente l'interrogation d'un résumé complexe et plus précisément le cas de résumés conçus par l'approche StreamSamp.

StreamSamp. StreamSamp est basé sur une technique de ré-échantillonnage aléatoire des données. Dès leur arrivée, les données du flux sont échantillonnées à un taux α et placées

[1] L'aspect glissant des fenêtres concerne à la fois la partie récente et la partie historique du flux.

dans des échantillons de taille fixe T. Lorsque T est atteint, l'algorithme mémorise l'échantillon ainsi que sa date de début et fin de constitution. Lorsque L échantillons sont constituée, l'algorithme fusionne les deux plus anciens échantillons pour former,par ré-échantillonnage aléatoire, un nouvel échantillon de taille T couvrant ainsi une période temporelle deux fois plus grande. L'idée consiste ainsi à conserver des échantillons de taille constante qui s'étalent sur des périodes temporelles de durée variable, plus courtes pour le présent et plus longues pour le passé lointain. Pour modéliser la représentativité des échantillons, l'algorithme se base sur un système de pondération croissante (voir Csernel et al. (2006) pour plus de détails sur le fonctionnement de cet algorithme).

3 Spécification du problème

Dans cette communication, nous nous sommes intéressés à l'évaluation des requêtes hybrides.

Définition 1.*Une requête est dite hybride si elle porte à la fois sur des données anciennes et des données récentes. Les données anciennes sont disponibles sous forme de résumé en base de données et, les données récentes le sont sous forme de flux.*

Deux grandes familles de requêtes hybrides peuvent se distinguer : (1) requêtes référençant des données récentes et une période passée fixe (ex. vérifier si les indices boursiers actuels s'écartent de plus de 10% de la moyenne de ceux du mois de juin 2006) et, (2) requêtes référençant des périodes récentes et une période passée mobile (ex. une requête qui permet de détecter d'éventuelles fraudes dans les systèmes bancaires en comparant l'activité actuelle de la carte de crédit à l'activité des 30 derniers jours glissants). Le traitement de la première catégorie de requête est simple, il suffit d'appliquer (une seule fois) la requête SQL à la base de résumé et enchaîner par la suite par le traitement de la fenêtre glissante sur les données récentes. Tandis que, le traitement de la seconde catégorie de requête est plus compliqué puisqu'il s'agit d'une requête glissante sur la base de données et, les systèmes de gestion de données existants ne disposent pas des outils nécessaires pour de tels traitements. C'est l'objectif sur lequel se base ce travail. L'idée consiste à étendre un SGFD existant pour l'enrichir de façon à pouvoir évaluer des requêtes glisantes portant sur une base de données associée.

Soit un utilisateur qui définit une requêtes continue sur une période de longueur (N) items. Cette requête est réévaluée tous les Δ items[2]. On note n le nombre d'items que le SGFD peut conserver en mémoire. Ces items sont traités sous la forme d'une fenêtre glissante de $t-n$ jusqu'à t. Deux situations peuvent résulter de cette description :

- Si $n \geq N$: Cela veut dire que la totalité de la période décrite par la requête de l'utilisateur peut être mémorisée dans la mémoire du SGFD. Ainsi, cette requête sera traitée par le mécanisme classique du SGFD.
- Si $n \prec\prec N$: La requête posée par l'utilisateur porte sur des données qui ne peuvent être mémorisées en totalité dans la mémoire du SGFD (cf. figure 1). Le système doit ainsi combiner un traitement sur des données stockées dans le résumé (période $t-N$ à $t-n$) et sur des données présentes dans la mémoire du SGFD (période $t-n$ à t). C'est le cas que nous traitons dans cet article. L'approche définit dans la section suivante consiste à mettre à jour une fenêtre temporelle comprenant les données nécessaires à l'évaluation

[2] Δ est un paramètre fixé par l'utilisateur.

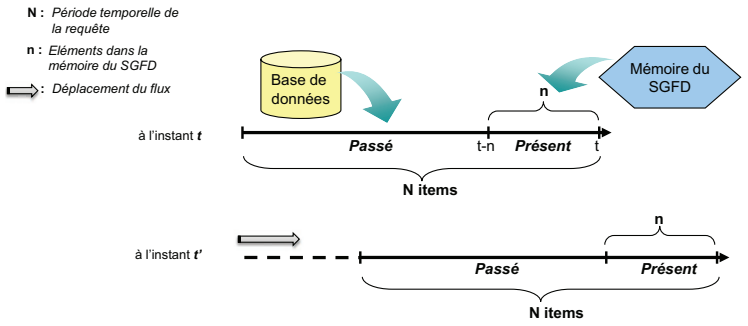

FIG. 1 – *Exemple de fenêtre faisant appel à des données du passé et des données récentes.*

de la requête. Cette fenêtre est mise à jour chaque fois que Δ items du flux sont insérés dans la mémoire du SGFD.

4 Approche proposée

Dans cet article, nous considérons un flux F observé aux instants $1, ..., t$. Au fur et à mesure de l'arrivée des items, le flux est traité par le SGFD et par un algorithme de résumé. Nous considérons ici un résumé par échantillonnage aléatoire qui est conservé dans une base et mis à jour avec l'arrivée du flux. Nous disposons ainsi d'un échantillon pour toute fenêtre du passé. Pour simplifier, nous considérons les requêtes d'agrégat appliquées à l'ensemble de la période interrogée, seul le cas des fenêtres logiques (exprimées en terme de nombre d'items) est traité. Ces requêtes peuvent être alors estimées à partir des échantillons conservés. Pour quantifier l'imprécision liée à l'utilisation d'un résumé, un intervalle de confiance est associé à la réponse de la requête. Il n'y a pas de difficultés à généraliser à des requêtes réalisant une sélection définie par une condition sur des attributs quantitatifs ou qualitatifs.

Pour pouvoir répondre aux requêtes définies précédemment, nous maintenons deux tableaux en mémoire centrale[3] :

- Le premier tableau T_{New} couvre la période $[t-n, t]$ (c'est l'ensemble des éléments qui satisfont la requête de l'utilisateur et qui sont inclus dans la mémoire du SGFD). Ce tableau est maintenu de façon standard par le SGFD.
- Le second tableau T_{Hist} couvrant la période $[t-N, t-n]$ contient les items qui satisfont la requête de l'utilisateur et qui sont extraits à partir du résumé. Le résumé élaboré en utilisant l'algorithme StreamSamp est conservé sur disque dans une table nommée "Résumé".

A chaque rafraîchissement de la fenêtre temporelle, le tableau T_{New} est mis à jour de façon standard par le SGFD. De même, le tableau T_{Hist} doit être lui aussi mis à jour : les items ayant un timestamp inférieur à $t-N$ sont supprimés et remplacés par les items extraits de la base et qui appartiennent à l'intervalle $[t-n-\Delta, t-n]$.

[3]Nous supposons qu'on dispose de l'espace nécessaire pour maintenir en mémoire centrale ces deux tableaux

Temps de latence Vs Débit du flux. Pour que l'approche décrite ci-dessus puisse fonctionner correctement, il faut s'assurer qu'à l'instant t, tous les items qui ont un timestamp inférieur à $(t - n)$ ont déjà été sauvegardés dans le résumé.

Soit τ le délai nécessaire pour traiter, écrire et valider un item dans le résumé. L'item F_t est disponible à partir de l'instant $t + \tau$. Soit d le débit d'arrivée des items du flux. La fenêtre temporelle de n items couvre une période de (n/d). Afin d'assurer le bon fonctionnement de l'approche, il faut que la contrainte $\tau \prec (n/d)$ soit respectée. Cependant, dans la pratique, il est difficile d'agir sur le débit du flux d ou sur le temps de latence τ. La contrainte porte alors sur n : il faut que $n \succ d * \tau$. Dans le cas où les items du flux arrivent avec un débit variable, cette contrainte doit être appliquée sur un débit maximum (d_{max}) du flux. Toutefois, si cette condition ne peut pas être satisfaite, nous pouvons envisager une technique d'échantillonnage sur la fenêtre glissante $[t - n, t]$. Ainsi, la fenêtre de taille n ne contient plus les n items les plus récents du flux mais plutôt un échantillon aléatoire de taille n correspondant à plus de n items dans le flux (voir Babcock et al. (2002b) pour un tel algorithme).

5 Implantation de l'approche proposée

Nous avons testé la solution proposée sous le SGFD Esper. La technique de résumé utilisée est StreamSamp (Csernel et al. (2006)). Un travail antérieur(Gabsi et al. (2009)) a permit d'évaluer les performances de cette approche de résumé.

Esper. Esper est un SGFD open source utilisé au sein d'une communauté très active. Il s'agit d'un ensemble de bibliothèques Java permettant de déclarer un flux et son schéma ainsi que de créer des requêtes continues en EPL[4]. Ces requêtes peuvent s'appliquer sur trois types de fenêtres : glissantes, sautantes ou partitionnées. Dans Esper, les items du flux sont appelés des événements. Une requête EPL reçoit en entrée les événements du flux, et fournit en sortie soit des événements individuels, soit un lot de plusieurs événements. A chaque événement (ou lot d'événements), le serveur Esper[5] invoque une méthode appelée *update*. Les arguments de cette méthode sont deux tableaux qui sont remplis à la volée par le serveur Esper lors de l'exécution d'une requête. Le tableau *oldEvents* contient le(s) dernier(s) événement(s) expiré(s) de la fenêtre alors que le tableau newEvents contient le(s) dernier(s) événement(s) inséré(s) dans la fenêtre.

Implantation de l'approche sous Esper. L'implantation de l'approche sous Esper se traduit par la définition d'un nouvel opérateur de fenêtrage. Cet opérateur reçoit deux paramètres utilisateur : (1) N qui permet de spécifier la taille de la fenêtre sur laquelle va s'appliquer la requête, (2) Δ qui permet de préciser le taux de rafraîchissement du résultat. Dans le cas idéal, le nombre d'items du flux n est fixé par le système en fonction de sa charge, cependant, dans l'implantation proposé, il s'agit d'un paramètre supplémentaire fixé lors de la définition de la fenêtre.

La mise à jour des items de la fenêtre du présent (n) est prise en charge par le SGFD Esper. Cependant, pour les données du passé, un tableau doit être maintenu en mémoire centrale, il contient les items, ainsi que leurs poids, nécessaires à l'évaluation de la requête. Il s'agit de maintenir une fenêtre glissante sur les données de la base. A chaque invocation de la méthode *update*, on supprime du tableau les items ayant expiré de la fenêtre, et à l'aide d'une requête

[4]Event Processing Language : Langage d'interrogation dans Esper

[5]La tâche du serveur Esper consiste à compiler, enregistrer et exécuter les requêtes EPL.

SQL appliquée à la base des résumés, les nouveaux items sont extraits et insérés dans le tableau. Après la mise à jour de ce tableau, l'algorithme de la méthode *update* fait appel aux fonctions de calcul d'agrégats avec intervalle de confiance. L'algorithme ci-dessous illustre le fonctionnement de la méthode *update* dans Esper.

Algorithm 1 update

Require: N : Taille de la fenêtre d'intérêt, Agr : Agrégat de la requête, Tableaux : T_{Hist}, T_{New}
n : Nombre d'items dans la mémoire du SGFD, Δ : Taux de rafraîchissement de la fenêtre
$Update(T_{New})$ {Mise à jour automatique par le SGFD}
$i \leftarrow 0$
while i < T_{Hist}.taille() & $T_{Hist}[i]$.Timestamp < $t - N$ **do**
T_{Hist}.Supprimer(i) {Supprimer l'item d'indice i du tableau T_{Hist}}
$i \leftarrow i + 1$
end while
Req =SELECT * FROM Résumé WHERE Timestamp BETWEEN $t - n - \Delta$ AND $t - n$
Résultat ← Exécuter_requête(Req)
T_{Hist}.Ajouter(Résultat) {Insertion des nouveaux éléments dans le tableau}
Réponse = $Fonction\_Agrégat(Agr)$
return Réponse

Esper ne fournit pas des fonctions d'agrégats avec des intervalles de confiance, mais offre la possibilité d'implanter de nouvelles fonctions. Ainsi, des fonctions faisant intervenir la pondération des items et fournissant un intervalle de confiance de l'agrégat sont à développer. La requête "utilisateur" illustrée ci-dessous est appliquée sur le flux $Flux\_CONSO$ ayant la structure suivante : Timestamp, id_Client, Consommation. Cette requête permet de retourner la consommation moyenne en électricité sur une fenêtre de taille 10^7 items avec un taux de rafraîchissement de 10^2 items.

```
SELECT AVG(CONSOMMATION) FROM
Flux_CONSO.win:Hist:length(10^7,10^2)
```

L'évaluation de cette requête permet de retourner deux valeurs : la valeur de l'estimateur de la moyenne ainsi que son intervalle de confiance. Pour calculer l'intervalle de confiance de l'agrégat, on fait appel à la théorie des sondages. L'exemple ci-dessous illustre le calcul des deux bornes de l'intervalle de confiance de l'agrégat moyenne à partir d'un échantillon de taille m :

$$IC = \left[\widehat{y} - 1.96\frac{S}{\sqrt{m}}; \widehat{y} + 1.96\frac{S}{\sqrt{m}}\right] \quad \widehat{y} = \frac{\sum_{i=1}^{m} v_i w_i}{\sum_{i=1}^{m} w_i}; S^2 = \frac{1}{m-1}\sum_{i=1}^{m}(v_i - \widehat{v})^2$$

m : taille des tableaux T_{Hist} et T_{New} concaténés, w_i : poids d'un item, v_i : item

La solution proposée n'est pas très coûteuse étant donné qu'à chaque mise à jour effectuée par le SGFD sur la fenêtre glissante, Δ éléments sont supprimées du tableau T_{Hist} et sont remplacés par les Δ items qui suivent et qui sont extraits de la base.

6 Conclusion et perspectives

La génération actuelle des Systèmes de gestion de Flux de données a été conçue et optimisée pour interroger le présent d'un flux et se révèle inadaptée pour l'interrogation du passé. Cependant, dans plusieurs applications telle que les applications d'aide à la décision, il est nécessaire d'interroger non seulement le présent d'un flux mais également son passé. Dans cet article nous avons proposé un nouvelle approche permettant de répondre au besoin de telles applications. Une implantation de cette approche est proposée en utilisant Esper comme système de gestion de flux de données et StreamSamp comme algorithme de résumé. Afin d'évaluer les requêtes et de quantifier l'imprécision liée à l'utilisation des résumés, des fonctions d'agrégats doivent être implantées sous Esper. Une perspective de ce travail est d'étudier le cas des fenêtres physiques. Par ailleurs, des travaux sur l'étude des performances de la technique proposée sont en cours.

Références

Arasu, A., S. Babu, et J. Widom (2003). The cql continuous query language : Semantic foundations and query execution. Technical report, VLDB Journal.

Babcock, B., S. Babu, M. Datar, R. Motwani, et J. Widom (2002a). Models and issues in data stream systems. In *ACM PODS*, New York, NY, USA, pp. 1–16. ACM.

Babcock, B., M. Datar, et R. Motwani (2002b). Sampling from a moving window over streaming data. In *SODA*, Philadelphia, pp. 633–634.

Chandrasekaran, S. et M. Franklin (2004). Remembrance of streams past : Overload-sensitive management of archived streams. In *VLDB*, pp. 348–359. VLDB.

Csernel, B., F. Clérot, et G. Hébrail (2006). Streamsamp : Datastream clustering over tilted windows through sampling. In *ECML PKDD*.

Gabsi, N., F. Clérot, et G. Hébrail (2009). Résumé hybride de flux de données par échantillonnage et classification automatique. In *EGC*, pp. 229–240.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Trans. Math. Softw. 11*(1), 37–57.

Summary

Data Stream Management Systems (DSMS) are designed to process large amounts of data. Given that hardware resources cannot store all data, only the most recent part of the stream is stored in the DSMS's buffer. Therefore, DSMS cannot handle queries that relate to past periods. We propose in this paper, a new approach to evaluate queries that include both data on old and recent periods. We present the definition of this approach and its implementation using the Esper DSMS.