Extension et interrogation de résumés de flux de données

Nesrine Gabsi

TELECOM ParisTech

École Doctorale d'Informatique, Télécommunications et Électronique de Paris

Thèse

Extension et interrogation des résumés de flux de données

présentée et soutenue publiquement le 31 Mai 2011

pour l'obtention du grade de

Docteur de l'École Nationale Supérieure des Télécommunications

spécialité : Informatique et Réseaux

par

Nesrine GABSI

Composition du jury

Rapporteurs :	Pascal Poncelet	Université Montpellier 2
	Jean-Marc Petit	INSA-Université de Lyon
Examinateurs :	Gilbert Saporta	CNAM
	Antoine Cornuéjols	AgroParisTech
Directeurs de thèse :	Georges Hébrail	Télécom ParisTech
	Fabrice Clérot	Orange R&D

Pour toute grande oeuvre il faut de la passion et pour la Révolution,
il faut de la passion et de l'audace à hautes doses. [Che Guevara]
À mon cher pays.

Remerciements

Je remercie en premier lieu tous les membres du jury de m'avoir fait l'honneur de leur participation à ma soutenance de thèse. Je remercie Monsieur Pascal PONCELET et Monsieur Jean-Marc PETIT d'avoir accepté de rapporter mon travail de thèse. Je vous remercie pour l'intérêt que vous avez porté à ce travail et pour vos remarques pertinentes. Je remercie également Monsieur Gilbert SAPORTA et Monsieur Antoine COURNUEJOLS de m'avoir fait l'honneur d'être présents à ce jury de thèse ainsi que d'avoir examiné ce travail à travers de regards critiques et avisés.

L'accomplissement de ce travail n'aurait pu aboutir sans l'aide et le soutien de certaines personnes que je tiens vivement à remercier :

Ma dette de reconnaissance va en premier lieu à mon directeur de thèse Monsieur Georges HEBRAIL pour m'avoir permis de réaliser cette thèse au sein de Télécom ParisTech. Un grand merci pour la confiance que vous m'avez accordé, pour votre soutien, votre disponibilité ainsi que l'intérêt et le temps dédiés pour le suivi de cette thèse. Je remercie également les membres de l'équipe Data Stream pour leurs remarques constructives qui m'ont permis d'accroître la pertinence de mes travaux.

J'exprime également ma profonde gratitude et mes remerciements les plus sincères à Monsieur Fabrice CLEROT pour m'avoir permis de réaliser cette thèse au sein d'Orange Labs. Je voudrais vous exprimer toute ma reconnaissance pour l'enseignement que vous m'avez apporté, pour vos permanents encouragements ainsi que votre disponibilité. Votre apport scientifique, votre expertise et votre regard critique m'ont été de précieux éléments à l'aboutissement de mes travaux de recherche décrits dans ce manuscrit. Je remercie tous les membres de l'équipe Profiling et data-mining avec qui j'ai eu le plaisir de partager des moments inoubliables.

Je ne pourrais sans doute oublier mes chers collègues de laboratoire que j'ai eu le plaisir de côtoyer durant ces quelques années. Un grand merci pour les échanges aussi bien culturels qu'académiques, sans oublier les moments de détente. Je voudrais par ailleurs remercier mes amis à TUNIS, PARIS et LANNION pour vos encouragements. Je vous remercie pour tous les moments de bonheur et de délire que j'ai partagé avec vous. Je voudrais particulièrement remercier WAEL pour sa disponibilité, son énergie, son aide, sa patience et son écoute.

Je remercie du fond du coeur ma grande famille principalement les familles GABSI, SASSI, JOUINI et KHEMIRI. Vos encouragements et vos prières m'ont toujours aidé à surmonter les moments les plus difficiles. Merci pour l'amour que vous m'avez procuré et pour la confiance. Un grand merci aux SASSI de Paris qui ont toujours répondu présent dans mes moments de stress. Merci pour votre compréhension et votre support sans condition.

Je remercie tout particulièrement ma grand-mère NANA, mes parents SALEM et SAMIRA, mon frère ENIS et CYRINE. Votre soutien et votre amour m'ont aidé à dépasser

les moments difficiles de ma vie. Je voudrais vous exprimer mon éternelle reconnaissance et mon indéfectible attachement. J'espère que ce travail réalisé grâce à vous sera à la hauteur de la confiance et de l'apport que vous m'avez donné.

Enfin, pour tout et bien plus encore, je remercie Dieu.

À la mémoire de ma grand-mère KHADIJA
et mes grands pères, ALI et KHALIFA.

Résumé

Au cours de ces dernières années, un nouvel environnement s'est développé dans lequel les données doivent être collectées et traitées instantanément dès leur arrivée. La gestion de cette volumétrie nécessite la mise en place d'un nouveau modèle et de nouvelles techniques de traitements de l'information. Il s'agit du traitement des flux de données. Ces derniers ont la particularité d'être continus, évolutifs, volumineux et ne peuvent être stockés, dans leur intégralité, en tant que données persistantes. Plusieurs travaux de recherche se sont intéressés à cette problématique ce qui a engendré l'apparition des systèmes de gestion de flux de données (SGFD). Ces systèmes permettent d'exprimer des requêtes continues qui s'évaluent au fur et à mesure sur un flux ou sur des fenêtres (sous ensembles finis du flux). Toutefois, dans certaines applications, de nouveaux besoins peuvent apparaître après le passage des données. Dans ce cas, le système ne peut répondre aux requêtes posées car toutes les données n'appelant aucun traitement sont définitivement perdues. Il est ainsi nécessaire de conserver un résumé du flux de données. De nombreux algorithmes de résumé ont été développés. Le choix d'une méthode de résumé particulière dépend de la nature des données à traiter et de la problématique à résoudre.

Dans ce manuscrit, nous nous intéressons en premier lieu à l'élaboration d'un résumé généraliste permettant de créer un compromis entre la vitesse de construction du résumé et la qualité du résumé conservé. Nous présentons une nouvelle approche de résumé qui se veut performance face à des requêtes portant sur des données du passé lointain.Nous nous focalisons par la suite sur l'exploitation et l'accès aux évènements du flux conservés dans ces résumés. Notre objectif consiste à intégrer les structures de résumés généralistes dans l'architecture des SGFD existantes de façon à étendre le champ de requêtes possibles. A cet effet, l'évaluation des requêtes qui font appel aux données du passé lointain d'un flux (i.e. données expirées de la mémoire du SGFD) serait possible au même titre que les requêtes posées sur le passé proche d'un flux de données. Nous présentons deux approches permettant d'atteindre cet objectif. Ces approches se différencient par le rôle que détient le module de résumé lors de l'évaluation d'une requêtes.

Mots-clés: Résumé généraliste de flux de données, régénération du résumé, approche hybride, approche statique, approche dynamique

Abstract

In the last few years, a new environment, in which data have to be collected and processed instantly when arriving, has emerged. To handle the large volume of data associated with this environment, new data processing model and techniques have to be set up ; they are referred as data stream management. Data streams are usually continuous, voluminous, and cannot be registered integrally as persistent data. Many research works have handled this issue. Therefore, new systems called DSMS (Data Stream Management Systems) appeared. The DSMS evaluates continuous queries on a stream or a window (finite subset of streams). These queries have to be specified before the stream's arrival. Nevertheless, in case of some applications, some data could be required after their expiration from the DSMS in-memory. In this case, the system cannot treat the queries as such data are definitely lost. To handle this issue, it is essential to keep a summary of data stream. Many summaries algorithms have been developed. The selection of a summarizing method depends on the kind of data and the associated issue.

In this thesis, we are first interested with the elaboration of a generic summary structure while coming to a compromise between the summary elaboration time and the quality of the summary. We introduce a new summary approach which is more efficient for querying very old data. Then, we focus on the querying methods for these summaries. Our objective is to integrate the structure of generic summaries in the architecture of the existing DSMS. By this way, we extend the range of the possible queries. Thus, the processing of the queries on old stream data (expired data) becomes possible as well as queries on new stream data. To this end, we introduced two approaches. The difference between them is the role played by summary module when the query is evaluated.

Keywords: Data streams' generic summaries, Hybrid approach, Static approach, Dynamic approach

Table des matières

Liste des illustrations

Liste des tableaux

Introduction générale

Ces dernières années, de nouvelles applications ont émergé pour lesquelles les données sont générées de façon continue et rapide sous forme de flux. Parmi ces applications nous citons : l'analyse financière (détection de fraude, échanges internationaux, etc.), la supervision des systèmes et des réseaux (surveillance du trafic, détection des attaques, etc.), les réseaux de capteurs (supervision du trafic routier, les compteurs relevant la consommation électrique, etc.), etc. Ces données sont produites sous forme de séquences d'évènements obtenues généralement en temps réel, de manière continue et ordonnée. Elles sont par nature imprévisibles et potentiellement infinies. Les systèmes se trouvent ainsi confrontés à des flux très importants de données pour lesquels un accès rapide et un stockage permanent de la totalité des évènements serait très coûteux, voir impossible.

De part leurs caractéristiques, l'apparition des flux de données a soulevé un certain nombre de problèmes et de contraintes liées à la technologie actuelle ainsi qu'aux méthodes de traitement, d'extraction et d'analyse des données existantes. Bien que tous les flux possèdent les mêmes spécificités de volume et de débit, ils ne sont généralement pas tous de même grandeur. Prenons l'exemple d'un flux de données généré à partir de l'ensemble des conversations et messages qui circulent entre les différents agents boursiers. Selon [6], le trafic de ces messages peut atteindre 100k messages/sec. Ou encore, dans le domaine de l'énergie des particules par exemple, les données produites par les expérimentations peuvent atteindre un débit de 800Go/s [142]. Tous ces exemples nécessitent un accès en temps réel aux données à des fins de supervision, de surveillance et de suivi. L'émergence de ce type d'applications est essentiellement due à deux causes [154] : (i) l'augmentation de l'utilisation et le déploiement des dispositifs de télédétection, et (ii) des raisons de sécurité tel que le terrorisme. C'est l'exemple du réseau mondial ECHELON qui utilise plus de 150 satellites ainsi que des récepteurs radio à large spectre permettant d'analyser les différentes communications en indiquant un ensemble de mots clés potentiellement intéressants [53].

Dans un nombre non négligeable de situations faisant intervenir des flux de données, l'utilisation d'outils classiques de gestion de données (tels que les Systèmes de Gestion de Base de Données (SGBD)) ne convient pas. On entre dans le domaine de la gestion de flux dès lors que (i) les données n'ont pas vocation à être stockées, (ii) elles nécessitent un traitement immédiat et (iii) les requêtes sont exécutées continuellement. C'est ainsi que sont apparus les Systèmes de Gestion de Flux de Données (SGFD). Ces systèmes ont pour

objectif de répondre au même type de besoin qu'un système de gestion de bases de données classique mais en posant des requêtes continues sur des flux de données qui sont par nature éphémères, là où les SGBD posaient des requêtes ponctuelles sur des tables relativement statiques. Ces SGFD permettent le requêtage ainsi que les tâches de fouille sur les flux ou sur des fenêtres qui sont un sous ensemble fini du flux, afin d'extraire les informations les plus pertinentes à partir d'une gigantesque masse de données.

Il est dans la nature des données volatiles de n'être plus disponibles pour les analyses après leur expiration, ce qui suppose de poser a priori sur un flux l'ensemble des requêtes dont on pourra avoir besoin par la suite. Or, de nouveaux besoins peuvent parfois apparaitre après le passage des données. Dans ce cas, le SGFD ne pourra pas répondre aux requêtes posées car toutes les données n'appelant aucun traitement sont définitivement perdues. Une des solutions proposées dans la litterature est la conservation d'une représentation compacte du flux de données appelée résumé. De nombreuses structures de résumé ont été développées tels que les sketches, l'échantillonnage, etc. Le choix d'une méthode de résumé particulière dépend de la nature des données à traiter et de la problématique à résoudre. La majorité des algorithmes de résumé existants sont destinés à des tâches spécifiques tel que le calcul du nombre d'événements. Une telle spécificité est un défaut si le but du résumé est de permettre une certaine flexibilité dans les traitements. Il est ainsi nécessaire de mettre en place un résumé opérationnel pour tout type d'application et sur lequel une large panoplie de requêtes peut être appliquée. De plus, le résumé doit pouvoir s'adapter aux variations des flux de données et permettre une bonne estimation aux différentes requêtes. C'est dans ce cadre que s'inscrit la première contribution de cette thèse. Il s'agit de l'élaboration d'un résumé généraliste de flux de données.

C'est dans un contexte d'utilisation et d'interrogation des résumés généralistes de flux de données que s'introduit le sujet de cette thèse. Les objectifs du travail sont doubles. Le premier axe consiste à étudier des extensions des résumés généralistes. Le second consiste à développer des outils permettant l'interrogation et l'intégration de ces résumés dans les SGFD.

1 Résumé généraliste de flux de données

Quand le volume de données augmente, il devient très coûteux (voire impossible) de les stocker avant leur traitement : il est donc nécessaire de développer des traitements à la volée ou de mettre au point le stockage d'un seul résumé intelligent de celles-ci. Nous considérons un résumé intelligent tout résumé généraliste permettant de répondre a posteriori et approximativement à n'importe quelle requête portant sur le flux, de façon optimale compte tenu des contraintes de ressources.

Notre première contribution consiste à analyser ou étudier les requêtes sur certains résumés généralistes (StreamSamp et CluStream) et à proposer des extensions de ces résumés.

2 Interrogation des résumé de flux de données

La gestion des résumés de flux de données pose deux problématiques fondamentales. La première concerne la construction et la conservation des résumés. Tandis que la seconde problématique implique leur interrogation. En effet, les systèmes actuels permettent d'évaluer des requêtes qui portent sur les données de la mémoire du SGFD. Cependant, ils sont incapables de traiter les données expirées. Afin d'évaluer des requêtes à posteriori, nous proposons dans ce manuscrit des mécanismes permettant l'intégration des résumés généralistes à l'architecture actuelle des SGFD.

3 Plan du manuscrit

Ce document est composé de trois parties principales :

1. État de l'art du domaine de la gestion et de l'interrogation des flux de données. Les chapitres 1, 2 et 3 sont dédiés à l'introduction des notions de flux de données ainsi qu'aux méthodes existantes pour l'interrogation des flux et, la conservation de traces de l'histoire des flux. Récemment, ce domaine a suscité l'intérêt de plusieurs communautés (bases de données, analyse de données, statistique, etc.) et ceci aussi bien du côté industriel qu'académique. Un intérêt qui se reflète dans le nombre important d'applications et de systèmes émergents confrontés à manipuler des flux de données.
2. Étude approfondie des algorithmes de résumés généralistes. Après avoir défini les algorithmes StreamSamp et CluStream, nous présentons une nouvelle approche de résumé généraliste qui permet de pallier les inconvénients des deux techniques évoquées. Le chapitre 4, permet d'aborder les limites de chacun des deux algorithmes. Nous présentons par la suite, dans le chapitre 5, l'approche hybride qui permet de combiner les avantages de StreamSamp et ceux de CluStream. L'idée consiste à proposer une méthode assurant une conception rapide des résumés et une bonne garantie des résultats pour les requêtes posées.
3. Interrogation des résumés de flux de données. Après avoir étudié la construction des résumés généralistes, nous proposons dans le chapitre 6 deux approches permettant l'interrogation et l'exploitation de ces résumés. Chacune des deux approches intègre un module de résumé dans l'architecture même du SGFD. Ce module se charge de conserver l'historique du flux, ce qui rendra possible l'évaluation des requêtes sur le passé.

Finalement, le chapitre 7 conclut ce travail et donne quelques perspectives de recherche.

Chapitre 1

Les flux de données

Sommaire

1.1 Introduction

Avec le développement des nouvelles technologies, les applications sont de plus en plus contraintes à faire face à une masse importante de données disponibles en temps réel, de manière continue et ordonnée. Cela a entraîné l'apparition de très grandes bases de données, puis plus tard des entrepôts de données (*Datawarehouse*). Cependant, la masse de données est devenue si importante qu'il devient impossible de la conserver en totalité. Un nouveau modèle a ainsi fait son apparition, il s'agit des *flux de données* (ou flot de données, *Data Stream*) où les données sont en circulation avec des débits en général importants. Les flux de données constituent un champ de recherche récent et se situent au carrefour de plusieurs domaines (fouille de données, bases de données, réseaux de capteurs, statistiques, etc.).

De nouvelles caractéristiques (e.g. données séquentielles et éphémères) et de nouvelles contraintes (e.g. un seul accès possible aux données, débit variable) liées à la nature même de ces données nécessitent de reconsidérer les approches traditionnelles de stockage, de traitement, d'extraction et d'analyse de l'information ainsi que les systèmes destinés à gérer ces données. Sous toutes ces contraintes, le modèle classique de communication dans lequel les données sont d'abord collectées et stockées puis analysées doit être révisé pour permettre une collecte et un traitement "à la volée" de ces données dès leur arrivée. Une nouvelle famille de systèmes a fait son apparition : les Systèmes de Gestion de Flux de Données (SGFD) (*Data Stream Management System*). Les données traitées par ces systèmes sont sous forme de flux transitoires et temporairement stockés dans la mémoire centrale rendant ainsi le processus de traitement de l'information instantané.

Outre ces caractéristiques, les flux de données proviennent de plusieurs sources géographiquement distribuées pour alimenter un ou plusieurs serveurs. Ce mode de traitement fait apparaître de nouveaux défis : l'hétérogénéité des sources, le passage à grande échelle, la fiabilité, la sécurité, etc. Une partie de ce chapitre sera consacrée au caractère distribué des flux de données cependant, dans le cadre de cette thèse nous nous intéressons à un flux de données unique.

La présentation des différentes caractéristiques et systèmes de flux de données constitue le corps de ce chapitre, suivie d'une illustration des principaux domaines d'utilisation des flux.

1.2 Concepts de base des flux de données

1.2.1 Définition

Une définition d'un flux de données est présentée par Golab et Özsu dans [90] : "*A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety. Likewise, queries over streams*

run continuously over a period of time and incrementally return new results as new data arrive. These are known as long-running, continuous, standing, and persistent queries." [1]

Une deuxième définition est aussi proposée dans [60] : *"Un flux de données est une séquence de données structurées que l'on peut considérer comme infinie d'éléments générés de façon continue à un rythme rapide et parfois variable. "Rapide" signifie que la vitesse d'arrivée des nouveaux éléments est grande devant les capacités de traitement et de stockage disponibles".*

Plusieurs définitions peuvent être proposées, néanmoins, toutes s'accordent sur les principales caractéristiques de ces évènements. Nous distinguons dans ce qui suit la notion d'évènement qui représente un élément du flux et la notion de données qui est la description contenue dans les évènements.

- **Continues** : les évènements arrivent d'une manière continue et séquentielle.
- **Rapides** : les évènements sont produits à un rythme rapide par rapport aux capacités de traitement du système qui les reçoit (e.g. $\simeq$ 100K messages par seconde dans le marché des transactions par Internet [39]) ;
- **Ordonnés** : l'ordre des évènements est généralement défini par rapport au temps soit implicitement par temps d'arrivée ou explicitement par une estampille temporelle de production à la source (*timestamp*). En fonction de la représentation du temps dans le flux, cette estampille est soit physique (e.g. une date), soit logique (e.g. un nombre incrémental associé aux évènements).
- **Volume illimité** : le nombre d'évènements dans un flux de données est potentiellement infini (e.g. communications téléphoniques de l'AT&T : 1Go / heure [39]). La mémorisation de la totalité des données est non-envisageable. Cependant, des techniques d'archivage peuvent être déployées en parallèle du traitement classique du flux (cf. chapitre 2).
- **A caractère push** : les évènements se présentent d'eux-mêmes. Les sources sont programmées afin d'envoyer régulièrement leurs mesures. Le système recevant le flux n'a aucun contrôle sur ces sources.
- **Sources incontrôlables** : aucun contrôle n'est possible sur le débit des sources de données. Le débit peut varier de quelques kilo octets par seconde (e.g. évènements provenant des capteurs) à des giga octets par seconde (e.g. évènements circulant sur un canal OC-768 pour la transmission des données par fibres optiques). Les évènements n'arrivent pas à un rythme constant. Le débit avec lequel ils arrivent contribue dans le choix du système de gestion de flux de donnés utilisé.

1. Il s'agit d'une séquence d'évènements continue et ordonnée arrivant en temps réel. Il est impossible de contrôler l'ordre dans lequel les évènements arrivent, ni de stocker localement un flux dans son intégralité. De même, les requêtes sur ces flux s'exécutent de façon continue sur une période donnée et retournent un nouveau résultat chaque fois qu'une nouvelle donnée arrive au système. Ces requêtes sont de nature persistante, continue et nécessitent des exécutions de longue durée.

– **Incertitude** : les évènements qui arrivent au système sont fiables. Cependant, l'imprécision porte sur la sémantique du flux (e.g. l'ordonnancement et la complétude des évènements). Il peut en effet manquer quelques évènements, avoir des duplications, etc. Ces perturbations peuvent être liées à une déficience de la source, à des problèmes de transmission ou à des problèmes de réseau, etc.

1.2.2 Structure et types de données

1.2.2.1 Flux de données structurées

Dans les flux de données structurées, les évènements arrivent sous forme d'enregistrements respectant un schéma relationnel spécifique comprenant le nom des champs et leurs types. Les évènements d'un flux diffèrent des n-uplets d'une base de données par le fait qu'ils sont ordonnés. Un exemple de flux de données généré par l'ensemble des communications relatives à un opérateur téléphonique est illustré dans la Figure 1.1. Chaque communication est ici représentée par un évènement du flux.

Timestamp	Code client	Zone Géographique	Type d'appel	Durée(sec)
...	...	...	...	...
31/01/2008-12 :43	A21138	SAVOIE MOUTIERS	National	320
31/01/2008-16 :44	A21809	SAVOIE MOUTIERS	International	120
10/02/2008-16 :44	B81297	COTE D'OR DIJON	National	100
23/02/2008-11 :23	B52089	PARIS ILE-DE-FRANCE	National	86
...	...	...	...	...

FIGURE 1.1 – Flux de données structurées de communications téléphoniques.

1.2.2.2 Flux de données semi-structurées

Les évènements d'un flux semi-structuré arrivent sous forme de balises XML [147]. XML est devenu le standard pour les données échangées sur le Web. Plusieurs applications, allant des applications e-commerce et B2B (Business-to-Business) à des applications scientifiques de surveillance de capteurs ou des applications qui gèrent des données transmises par satellites [127], exigent le traitement en ligne d'une grande quantité de données arrivant dans un format spécifique : le format XML. La quantité de ces données est de plus en plus importante face à des ressources mémoires limitées. Ces systèmes sont donc amenés à subir les contraintes caractéristiques d'un flux de données ainsi que les contraintes liées au format XML :

– La validation des documents XML par rapport à la DTD [2] associée : la difficulté consiste à valider le document en une seule passe et avec une quantité mémoire restreinte. Des travaux se sont intéressés à ce sujet, ils proposent d'utiliser des techniques basées sur les Automates à États Finis (FSA : finite-state automaton) pour résoudre ce problème [160] [91].
– L'évaluation des requêtes XPath sur un flux de données XML : plusieurs travaux se sont intéressés à l'évaluation des requêtes XPath sur des flux XML. Un évènement XML est constitué d'un ensemble de balises. La réponse aux requêtes devient difficile lorsque le système n'a pas encore reçu la totalité des balises qui constituent l'évènement en question. Plusieurs travaux se sont intéressés à l'utilisation des automates déterministes à états finis. L'idée revient à transformer toutes les expressions XPath en un seul automate et de l'évaluer par rapport au flux XML. D'autres travaux ont porté sur la réponse aux expressions XPath complexes contenant des agrégations et de multiples prédicats [91].

1.2.2.3 Flux de données non-structurées

Dans un flux de données non-structurées, les événements ont des structures différentes telles que les pages HTML. Un des SGFD permettant la gestion de ce type de données est le système WebCQ [132]. Ce système permet le suivi et la surveillance des pages Web afin de notifier tous les changements qui se produisent. C'est l'exemple de la base des films IMDB (*Internet Movies Database*) (`http://www.imdb.com/`). L'utilisateur enregistre sa requête (une sentinelle) à l'aide d'un formulaire HTML et spécifie le type d'information qu'il souhaite surveiller sur cette base. Lorsque des changements intéressants sont détectés, une notification lui est envoyée. Ces changement sont détectés chaque jour. Les informations sont extraites à partir des flux des pages statiques [3] et des flux des pages dynamiques [4] du site.

Dans le cadre de cette thèse, nous nous intéressons aux flux de données structurées.

1.2.2.4 Types de données dans un flux

Du fait de la grande variété des applications, les données transportées par les évènements du flux sont sous différents formats. Ces données nécessitent une phase de prétraitement dans laquelle les données passent de leur format brut à un format adapté aux algorithmes existants. Cette étape est aussi l'occasion de nettoyer les données en traitant le

2. La DTD de l'anglais *Document Type Definition* est une grammaire associée à un document XML. Elle a pour rôle de définir les balises possibles et leurs attributs ainsi que l'imbrication des balises.
3. Une page statique est une page web qui a été écrite par le développeur en utilisant le langage HTML.
4. Une page dynamique est une page qui a été généré en utilisant un langage serveur tel que PHP, ASP, etc.

cas des éléments vides, incomplets, ou celui des valeurs aberrantes. Deux types de données standard sont alors représentés :

- Les données dites *quantitatives (ou numériques)* : c'est l'ensemble des données qui peuvent être mesurées, elles proviennent de calculs mathématiques ou de mesures. Elles sont soient continues, soient discrètes. Les données continues sont celles dont les valeurs forment un sous-ensemble infini de l'ensemble **R** des réels (e.g. le salaire, le poids, le coût, etc.). Les données discrètes sont celles dont les valeurs forment un sous-ensemble fini ou infini de l'ensemble **Z** des entiers relatifs (e.g. nombre d'abonnés, genre, etc.).
- Les données dites *qualitatives (ou catégorielles)* : une variable qualitative prend un nombre fini de valeurs, appelées modalités de la variable. Si un ordre entre les différentes modalités existe (e.g. petit, moyen, grand), la variable est dite ordinale, sinon elle est nominale. Les adresses IP, la couleur, la texture, l'état, etc. sont des exemples de données qualitatives. Les données binaires sont considérées comme étant un cas particulier des données qualitatives à deux modalités (0/1, ON/OFF, etc.).

1.3 Modèles de données

Le modèle associé à un flux permet de représenter la relation qui existe entre les données de ce flux et le signal sous-jacent qu'il décrit [142]. D'une façon générale, les données d'un flux arrivent au système, séquentiellement, évènement par évènement. Une observation est définie par une paire (t_i,e_i) où t_i est une estampille temporelle et e_i est un évènement du flux. Il existe dans la littérature plusieurs modèles pour définir un signal sous-jacent à partir de différentes observations [141] [88] [107] [111] [23]. Le choix du modèle dépend de l'application finale et des besoins de l'utilisateur.

1.3.1 Modèle des séries temporelles

Une série temporelle (appelée aussi série chronologique) est une suite d'observations (t_i, e_i). Dans ce modèle, chaque évènement du flux représente une nouvelle observation à analyser qui donne directement la valeur du signal sous-jacent. C'est l'exemple des durées d'appels, dans la Figure 1.1, observés pour chaque estampille temporelle. Dans ce modèle, les nouvelles observations sont ajoutées à la fin de la série.

1.3.2 Modèle de la caisse enregistreuse

Une seconde modélisation consiste à ne plus considérer les évènements du flux comme étant un évènement d'un signal sous-jacent mais comme une variation du signal. Dans ce modèle chaque observation donne un incrément positif à la valeur précédente du signal pour obtenir la nouvelle valeur. Un évènement est défini comme suit : $e_i = (j, v_i)$ avec $v_i \geq 0$ représente le nouvel état du signal et j définit le numéro de l'attribut pour un flux

multidimensionnel [5].
Prenons l'exemple du flux illustré dans la Figure 1.1. Si le signal décrit la durée totale des communications depuis le 31/01/2008 12 :43, on parle ainsi d'un modèle de caisse enregistreuse.
Il existe une variante de ce modèle. Il s'agit du modèle du tourniquet dans lequel, la variation v_i peut aussi bien être positive ou négative.

1.3.3 Modèle probabiliste

Le modèle de flux de données probabiliste a été introduit par Jayram et al dans [113]. Il s'agit d'une généralisation du modèle classique des flux dans laquelle on manipule des évènements dits *"probabilistes"*. Généralement, les évènements probabilistes proviennent des deux types de sources suivants :

- Des capteurs : suite à des défaillances techniques certains capteurs peuvent produire des données erronées. Une incertitude liée aux mesures générées par ces capteurs est à considérer ;
- Des post-traitements : où les données d'origine subissent des traitements avant d'être envoyées sous forme de flux.

Dans [113], Jayram et al définissent un flux probabiliste comme suit : "*A probabilistic stream is a data stream* $A = (a_1, a_2, ..., a_m)$ *in which each data item* a_i *encodes a random variable that takes a value in* $[n] \cup \perp$. *We consider that* $[n]$ *is a set of integers and* $\perp$ *is a special symbol. In particular each* a_i *consists of a set of at most* l *tuples of the form* (j, p_j^i) *for some* $j \in [n]$ *and* $p_j^i \in [0, 1]$. *These tuples define the random variable* X_i *where* $X_i = j$ *with probability* p_j^i *and* $X_i = \perp$ *otherwise. We define* $p_\perp^i = Pr[Xi = \perp]$ *and* $p_j^i = 0$ *if not otherwise determined.*" [6]

Afin de mieux comprendre le principe qui découle de ce modèle, on considère un flux A contenant des paires d'évènements (e_i, p_i), avec e_i l'événement d'indice i et p_i la probabilité avec laquelle cet évènement appartient "réellement" au flux $A = (< x, 1/2 >, < y, 1/3 > , < z, 1/4 >)$. Les probabilités associées à ces événements sont calculées sans prendre en considération l'ordonnancement des événements. Nous observons ainsi huit combinaisons possibles du flux appelées "mondes possibles" (cf. Figure 1.2). Nous déduisons ainsi la probabilité d'apparition de chacun de ces mondes (cf. Figure 1.3).
$P(Flux1) = P(\overline{x}) \times P(\overline{y}) \times P(\overline{z}) = (1 - 1/2) \times (1 - 1/3) \times (1 - 1/4) = 1/4$
$P(Flux2) = P(x) \times P(\overline{y}) \times P(\overline{z}) = 1/2 \times (1 - 1/3) \times (1 - 1/4) = 1/4$

5. Dans un flux multidimensionnel, un évènement est caractérisé par plus qu'une dimension (appelée aussi attribut) à part l'estampille temporelle.

6. Un flux de données probabiliste est un flux A = (a_1, a_2, ..., a_m) dans lequel chaque évènement a_i décrit une variable aléatoire qui prend ses valeurs dans l'espace $[n] \cup \perp$ avec $[n]$ l'espace des entier et $\perp$ un symbole spécial. Particulièrement, chaque évènement a_i est composé d'au plus l n-uplets de la forme (j, p_j^i) avec tout $j \in [n]$ et $p_j^i \in [0, 1]$. Ces n-uplets permettent de définir la variable aléatoire X_i où $X_i = j$ pour une probabilité égale à p_j^i et $X_i = \perp$ sinon. Nous définissions $p_\perp^i = Pr[Xi = \perp]$ et $p_j^i = 0$ si la probabilité ne peut pas être déterminée.

Monde possible	Événements du flux	Commentaire
Flux 1	$\varnothing$	Le flux ne contient aucun évènement
Flux 2	x	Le flux ne contient que l'évènement x
Flux 3	y	Le flux ne contient que l'évènement y
Flux 4	z	Le flux ne contient que l'évènement z
Flux 5	x, y	Le flux ne contient que les évènements x et y
Flux 6	x, z	Le flux ne contient que les évènements x et z
Flux 7	y, z	Le flux ne contient que les évènements y et z
Flux 8	x, y, z	Le flux contient les évènements x, y et z

FIGURE 1.2 – Les mondes possibles du flux A.

$P(Flux3) = P(\overline{x}) \times P(y) \times P(\overline{z}) = (1 - 1/2) \times 1/3 \times (1 - 1/4) = 5/24$
$P(Flux4) = P(\overline{x}) \times P(\overline{y}) \times P(z) = (1 - 1/2) \times (1 - 1/3) \times 1/4 = 1/12$
$P(Flux5) = P(x) \times P(y) \times P(\overline{z}) = 1/2 \times 1/3 \times (1 - 1/4) = 1/8$
$P(Flux6) = P(x) \times P(\overline{y}) \times P(z) = 1/2 \times (1 - 1/3) \times 1/4 = 1/12$
$P(Flux7) = P(\overline{x}) \times P(y) \times P(z) = (1 - 1/2) \times 1/3 \times 1/4 = 1/24$
$P(Flux8) = P(x) \times P(y) \times P(z) = 1/2 \times 1/3 \times 1/4 = 1/24$

G	Flux1	Flux2	Flux3	Flux4	Flux5	Flux6	Flux7	Flux8
$Pr[G]$	1/4	1/4	5/24	1/12	1/8	1/12	1/24	1/24

FIGURE 1.3 – Probabilité d'apparition des flux.

Dans l'exemple illustré ci-dessus, le calcul des probabilités et de l'ensemble des mondes possibles n'est pas très coûteux étant donné que le nombre d'évènements dans le flux A est petit. Cependant, si on se réfère à la définition d'un flux et en particulier à son caractère infini, le calcul de l'ensemble des mondes possibles n'est plus envisageable. Les algorithmes classiques appliqués aux flux de données ne sont pas adaptés à ce modèle de flux probabilistes. Il s'agit d'un domaine de recherche assez récent dans lequel plusieurs travaux se sont intéressés au calcul d'agrégats sur ce modèle de données [113] [120] [112].

1.3.4 Modélisation temporelle des flux de données

L'aspect temporel est une caractéristique importante dans les flux de données. Une estampille temporelle est attribuée à chaque évènement, elle permet de situer la position de l'évènement dans le flux. Le marquage temporel n'est pas automatiquement attribué par la source qui produit les évènements. Deux modélisations possibles permettant l'attribution d'une étiquette temporelle à un évènement peuvent être distinguées :

– **Attribution explicite** : dans cette modélisation, le marquage temporel fait partie du schéma du flux. L'estampille temporelle est basée sur le temps d'observation (date à laquelle l'évènement a été produit). Cette méthode conserve l'estampillage de la source, cependant, elle ne garantit pas l'arrivée et le traitement des évènements dans l'ordre. La Figure 1.1 présente un exemple de flux avec un estampillage explicite.
– **Attribution implicite** : il s'agit d'attribuer une estampille temporelle à chaque évènement dès son entrée dans le système (e.g. un entier qui permet de numéroter les évènements successivement). Cette modélisation permet de garantir l'ordre des évènements. Cependant, l'estampille temporelle générée par la source n'est plus connue, ce qui peut présenter parfois des inconvénients par rapport à certaines requêtes (e.g. le nombre d'évènements produits entre le 01/10/03 et le 03/10/03). La Figure 1.4 représente des évènements du flux illustrant différentes mesures enregistrées par des compteurs électriques avec un estampillage implicite.

Timestamp	Puiss. Active(kW)	Puiss. Réactive(kVAR)	U(V)	I(A)
...	...	...	...	...
820154237	4,365	0,562	234,72	19
820154238	4,572	0,512	234,21	19
820154239	3,682	0,506	257,01	16,7
820154240	3,540	0,474	253,64	15
...	...	...	...	...

FIGURE 1.4 – Flux généré par un compteur électrique.

1.4 Modèle de communication

Le système classique de traitement de l'information se base sur un modèle dit "data-pull", dans lequel la collecte des données est réalisée par des commandes liées aux Système de Gestion des Bases de Données. Dans ce modèle, il faut aller chercher et collecter l'information à partir de différentes sources (base de données, systèmes pair-à-pair, etc.). L'augmentation du volume de données a amené à la création de très grandes bases de données et par la suite à des entrepôts de données.

D'un point de vue général, la principale fonction d'un entrepôt de données concerne l'intégration et l'historisation des données. L'alimentation de ces entrepôts se fait de façon périodique par l'utilisation d'un ETL [7]. La Figure 1.5, illustre l'architecture actuelle des systèmes d'information d'aide à la décision. Une première étape consiste à collecter les

7. Extract, Transform and Load : outil décrivant les données, leur provenance et réalisant l'extraction et les transformations.

informations à partir de plusieurs bases de production. Ces informations sont par la suite utilisées afin d'alimenter l'entrepôt de données en respectant les étapes suivantes :

- **Découverte des données** : il s'agit d'identifier, à partir des sources, les données à conserver dans l'entrepôt. Une sélection judicieuse de ces données est nécessaire. Un mauvais choix peut considérablement compliquer les phases suivantes de l'alimentation ;
- **Extraction des données** : il s'agit de collecter les données utiles dans les systèmes de production. Une phase d'identification des données modifiées est importante afin d'importer le minimum d'information dans l'entrepôt ;
- **Transformation des données** : cette étape permet, à l'aide de filtres, de rendre les données cohérentes avec la structure de l'entrepôt.

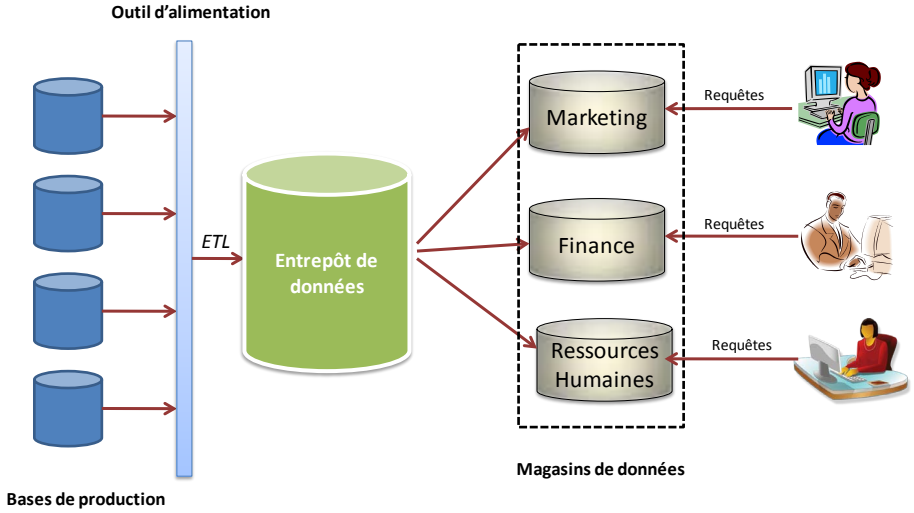

FIGURE 1.5 – Architecture actuelle des systèmes d'aide à la décision

Les informations stockées dans l'entrepôt font l'objet d'interrogation et/ou de fouille. L'analyse et le traitement des données se font traditionnellement dans les entrepôts où elles sont agrégées. Pour alléger les opérations de fouille, l'utilisation de bases d'études (*Datamarts* ou magasins de données) est souhaitée. Une base d'étude représente une vue partielle de l'entrepôt mais orientée métier ce qui permet de réduire le nombre d'opérations sur les bases de production.

Avec l'émergence des flux, les données sont produites à une vitesse et dans des quantités telles que la technologie actuelle ne permet pas de les traiter de façon satisfaisante poussant ainsi ce modèle classique de communication à être re-visité. L'architecture doit s'adapter à la distribution et l'hétérogénéité des données, à leur volumétrie croissante et à leur taux d'arrivée. Les méthodes d'extraction, de transformation et d'analyse doivent être étendues à la logique des flux où les données arrivent d'elles mêmes avec une vitesse très importante selon un modèle dit "data-push". L'utilisation de méthodes et de techniques

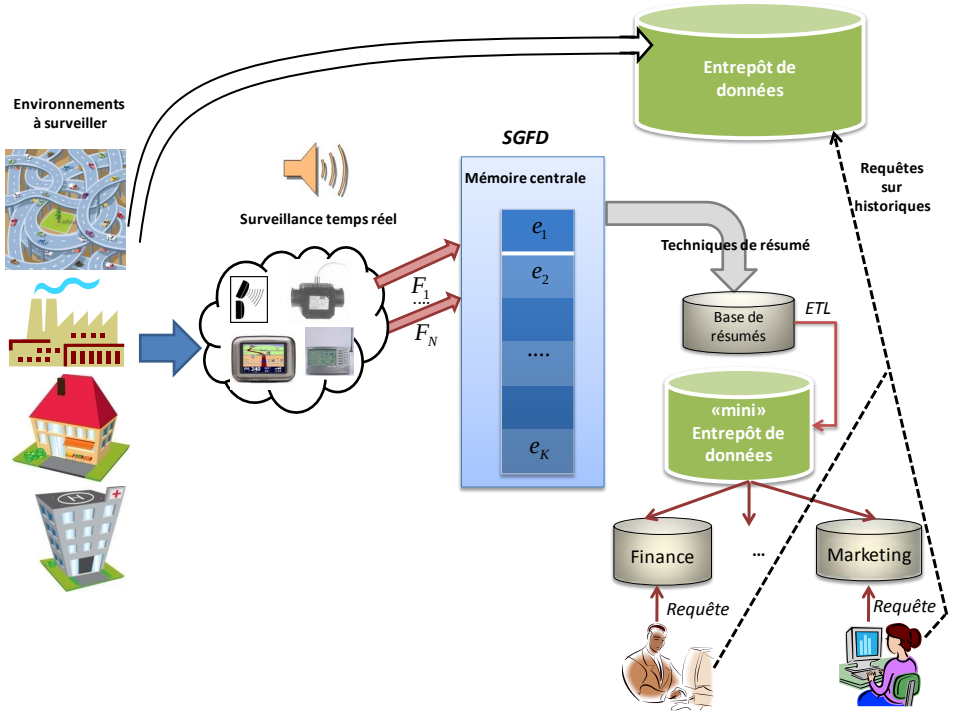

FIGURE 1.6 – Architecture des systèmes d'aide à la décision en présence de flux

de traitements en temps réel devient nécessaire. Pour les traitement "rapides" [8] supportant l'imprécision, il est possible de passer par des entrepôts résumés appelés "*mini entrepôt de données*". Cependant, à terme, cette structure devient à son tour énorme tandis que l'entrepôt de données de départ devient impossible à maintenir. La logique de cette nouvelle architecture, décrite dans la Figure 1.6, consiste à traiter les données à la volée sous forme de flux et à appliquer en temps réel les opérations d'analyse et de contrôle. Afin de satisfaire ces contraintes, de nouveaux outils appelés Systèmes de Gestion de Flux de Données (SGFD) sont apparus. Dans ces systèmes, les données issues de différentes sources sont transitoires et temporairement stockées dans la mémoire centrale du SGFD. Afin d'appliquer des analyses a posteriori sur les données des flux, des techniques de résumé permettant de conserver une trace de l'histoire du flux peuvent être prévues. Les bases de résumé ainsi construites contiennent les informations nécessaires pour effectuer des requêtes d'agrégats ou des analyses statistiques sur les données, mais sous une forme plus compacte. La qualité des résultats est ajustée en fonction des ressources disponibles. Ces résumés sont périodiquement remontés jusqu'aux entrepôts (au moyen d'un ETL) et par la suite transformés en base d'étude sur lesquelles se font des opérations d'analyse.

8. On entend par rapide, tout traitement qui doit être retourné dans les brefs délais.

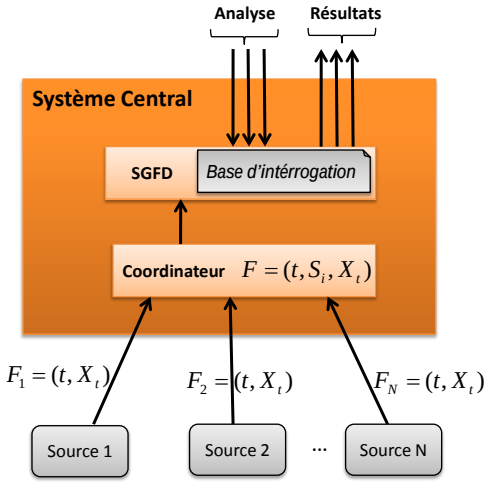

FIGURE 1.7 – Architecture centralisée

1.5 Flux de données distribués

Avec le développement des technologies de l'information et de la communication, de plus en plus d'informations sont générées et échangées par les applications localisées sur différents noeuds physiquement distribués. De nouvelles architectures sont à prévoir afin de gérer efficacement les contraintes liées à cette diversité. Certains systèmes ont adopté une architecture centralisée tandis que d'autres sont basés sur une architecture distribuée.

Dans le modèle centralisé [23](cf. Figure 1.7), les données sont envoyées par les différentes sources (i.e. noeuds) vers un coordinateur avant de passer au système central (i.e. SGFD). A l'aide d'index [9], le coordinateur peut accéder rapidement aux données. L'analyse de ces données est réalisée localement dans le SGFD. L'avantage de cette architecture consiste à minimiser les coûts par rapport à une architecture distribuée et permet de faciliter les jointures entre les flux. Cependant, celle-ci présente plusieurs inconvénients :

- Des temps de réponses importants étant donné que l'ensemble des tâches d'analyse est envoyé vers un système unique ;
- Une saturation du réseau : le fait d'utiliser un système centralisé restreint les capacités disponibles en bandes passantes provoquant ainsi des goulots d'étranglements et par conséquent l'indisponibilité du système ;
- L'architecture centralisée est moins adaptée aux flux ayant des débits importants et variables. Ces flux bloquent l'arrivée des données vers le noeud central et empêchent le système d'intégration de collecter l'information et de l'acheminer vers le système central (cette contrainte dépend du nombre de flux différents).

9. Un index est un élément de redondance que l'on va spécifier pour permettre au Système de Gestion de Base de Données d'optimiser certaines requêtes

Pour pallier les limitations d'une architecture centralisée, plusieurs travaux [54] [52] [57] ont proposé un modèle qui prend en considération l'aspect distribué des sources de données, des ressources de calcul et des liens de communication.
Dans une architecture distribuée (cf. Figure 1.8), une partie des calculs est effectuée sur les différents noeuds et les résultats sont envoyés vers le coordinateur (i.e. distribution du traitement et centralisation du résultat). Les avantages de ce système sont multiples :

- Réduction de temps de réponse : cette architecture fournit un grand degré de parallélisme ce qui minimise les temps de latences ;
- Réduction de la charge : le traitement et l'analyse des données sont partagés ce qui conduit à une charge plus faible destinée aux SGFD. Par ailleurs, si des erreurs de traitement surgissent, celles-ci peuvent être localisées et contrôlées directement sur le noeud en question. Par conséquent, le système reste toujours opérationnel ;
- La réduction de la charge de communication entraîne une réduction de la consommation d'énergie étant donné que chaque noeud sera réservé à une source (i.e. parallélisme au niveau de la bande passante).

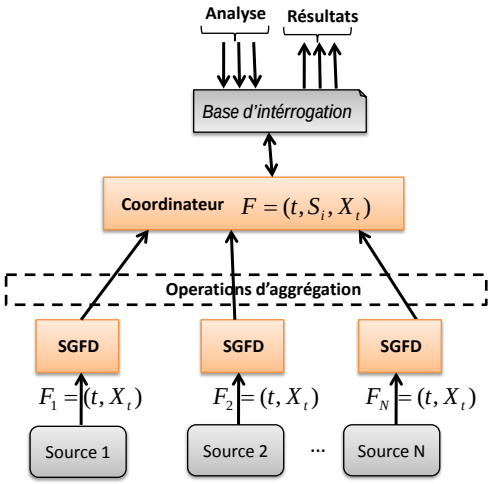

FIGURE 1.8 – Architecture distribuée

1.6 Tâches de fouille sur flux de données

Les algorithmes de fouille sur les flux de données ont été élaborés afin d'extraire des connaissances à partir des données des flux. La conception de ces algorithmes doit tenir compte de la variabilité du débit des flux ainsi que du caractère infini de ce dernier. Nous présentons dans cette section une liste non exhaustive des principaux travaux élaborés pour différentes tâches de fouille de flux de données.

1.6.1 Détection d'anomalie dans un flux de données

Dans plusieurs tâches d'analyse, on est face à un nombre considérable de variables qui seront sauvegardées ou échantillonnées. Une des premières étapes pour obtenir des analyses cohérentes serait la détection des anomalies. Les anomalies sont considérées comme une erreur ou un bruit. Ce sont des données aberrantes qui peuvent engendrer des estimations biaisées et des résultats incorrects. Il est ainsi important de les identifier avant de commencer l'analyse [173][129].

Une définition exacte d'une anomalie dépend souvent de plusieurs hypothèses sur la structure des données et de la méthode de détection appliquée. Cependant, certaines définitions sont considérées suffisamment générales pour faire face à divers types de données et de méthodes[152][117][29]. Dans [152], Hawkins définit une anomalie comme une observation qui s'écarte tellement des autres observations au point d'éveiller les soupçons.

Les techniques actuelles de détection d'anomalies peuvent être divisées en quatre catégories : (i) méthodes statistiques (ii) méthodes basées sur le calcul des distances (iii) méthodes basées sur la densité et (iv) méthodes basées sur la classification. La plupart de ces méthodes souffrent de l'hypothèse sur la distribution ainsi que de la multidimensionnalité des données. C'est l'exemple des histogrammes qui sont efficaces lors de l'analyse d'un seul attribut mais qui perdent leur efficacité pour les données ayant de grandes dimensions, car ils n'ont pas la capacité d'analyser simultanément plusieurs caractéristiques [62]. De plus, la plupart de ces techniques supposent que la distribution des données est stationnaire or cette caractéristique ne peut pas être appliquée aux données du flux (concept drift [10]).

La nature infinie du modèle de flux de données signifie qu'il est impossible de stocker toutes les données ou de réaliser plusieurs passes sur elles. Pour cela, les algorithmes de détection de changements doivent satisfaire les contraintes suivantes : les besoins en mémoire doivent être constants ou augmenter logarithmiquement, et l'algorithme doit se limiter à un seul passage sur les données. Plusieurs techniques ont été proposées dans la littérature. Nous les avons classés selon la dimensionnalité et le type de données utilisé dans chaque technique.

– Flux de données mono-dimensionnels avec des données quantitatives : dans les travaux énoncés ci-dessous, un flux de données est considéré comme un ensemble de valeurs réelles. Dans [121], Kifer et al. utilisent une technique basée sur deux fenêtres glissantes afin d'obtenir deux échantillons dont il est statistiquement possible de déterminer s'ils appartiennent à des distributions différentes. Dans [146], Papadimitriou

10. Une des tâches de fouille dans les flux de données est de prédire la classe ou la valeur des nouvelles instances dans le temps étant donné une certaine connaissance de l'appartenance des instances précédentes. Les techniques d'apprentissage classiques peuvent être utilisées pour l'apprentissage de cette tâche de prédiction à partir d'exemples étiquetés. Cependant, dans plusieurs applications, la distribution des instances ou des règles peut changer au fil du temps (la classe à prédire ou à la valeur cible à prédire). Ce problème est appelé *concept drift*

et al utilisent une modélisation basée sur les ondelettes. Dans [143], Muthukrishnan et al proposent une méthode basée sur les tests d'hypothèses.

– Flux de données multidimensionnels avec des données quantitatives : dans le cadre des flux de données multidimmensionnels, Aggarwal et al présentent dans [12] une technique basée sur la visualisation des flux de données en utilisant l'estimation de la densité. Cette visualisation utilise 2 dimensions et il n'est pas possible d'estimer les densités avec une dimensionalité qui croît au cours du temps. Dans [13], Aggarwal et al présentent une technique de régression polynomiale pour détecter les changements dans le flux.

  Une autre technique a été présenté dans [64] par Dasu et al. Cette approche utilise la distance de Kullback Leibler [125] afin de mesurer la différence entre deux distributions. Les expérimentations présentées dans le papier ont été réalisées avec un flux multidimensionnel et des données quantitatives. Cependant, cette méthode semble mieux s'adapter aux flux ayant un nombre réduit de dimensions (<10).

  Widmer et Kubat, présentent dans [172] une technique qui utilise des fenêtres à tailles variable pour l'apprentissage à la volée dans le domaine du concept drift. La taille de la fenêtre est ajustée en utilisant des heuristiques.

– Flux de données multidimensionnels avec des données qualitatives et quantitatives : Chen et al, proposent dans [49] une technique permettant de visualiser le changement sur des clichés du flux. L'inconvénient de cette technique est que les clichés sont capturés et analysés manuellement. Récemment, Calders et al ont proposé dans [35] une alternative à la mesure "support minimum" dans les flux de données appelé "fréquence maximale". Cette mesure utilise des fenêtres flexibles qui maintiennent les fréquences maximales des différentes valeurs dans le flux.

1.6.2 Recherche d'évènements fréquents dans un flux

Les évènements fréquents (en anglais *Frequent Items* ou *Heavy hitters* ou *Hot items*) sont les éléments qui apparaissent le plus souvent dans un flux de données. Un utilisateur définit un seuil sur la fréquence de ces évènements. Les évènements considérés comme fréquents sont ceux qui apparaissent avec une fréquence supérieure à celle définie par l'utilisateur.

Il existe plusieurs algorithmes utilisés pour la recherche des évènements fréquents tels que Apriori, Eclat, FPgrowth, etc [32]. Ces méthodes ne peuvent pas être appliquées dans le cadre des flux de données étant donné qu'elles nécessitent plusieurs passes pour calculer les évènements les plus fréquents.

Trouver exactement les évènements fréquents en un seul passage requiert $O(min(|A|, N))$ (A : l'alphabet du flux. Souvent $|A|$ n'est pas connu à l'avance. N : taille du flux) en espace mémoire[56]. Les évènements fréquents peuvent être définis sur la totalité du flux ou sur des fenêtres de taille fixe ou variable. Avoir des solutions exactes au pro-

blème des évènements fréquents engendre des coûts élevés. C'est pourquoi, dans plusieurs travaux, les auteurs ont opté pour des solutions approchées.

Plusieurs algorithmes[56][67][74][115][119][134][139] ont été proposés pour résoudre les problèmes d'approximation des évènements fréquents. Metwally et al ont divisé ces algorithmes dans[139] en deux classes : algorithmes basés sur les compteurs et algorithmes basés sur les sketchs.

Les algorithmes basés sur les compteurs supervisent la fréquence d'un sous-ensemble de A. Ils gardent un compteur pour chaque évènement surveillé. Le compteur d'un évènement surveillé (E_i) est mis à jour chaque fois que E_i est observé dans le flux. Ces algorithmes se distinguent principalement par leur manière de gérer les évènements non-surveillés [139].

Contrairement aux algorithmes basés sur les compteurs, les algorithmes basés sur les sketchs ne surveillent pas un sous-ensemble de l'alphabet mais fournissent, avec une moindre garantie, une estimation de la fréquence pour tous les évènements en utilisant des tableaux de compteurs. Chaque compteur surveille les fréquences d'un sous-ensemble de l'alphabet. De plus, chaque évènement est haché dans un espace de compteurs à l'aide d'une famille de fonctions de hachage. Le compteur de l'évènement haché est mis à jour pour chaque apparition de cet évènement [139].

D'autres travaux concernant l'extraction de motifs séquentiels dans les flux de données ont récemment été proposés. Dans [150][149], les auteurs proposent une approche pour l'extraction de nouvelles connaissances permettant des analyses de tendances et des détections de connaissances atypiques.

1.6.3 Classification non supervisée d'un flux de données

La classification non supervisée (en anglais *Clustering*) est largement étudiée dans le domaine de la fouille de données. Cependant, les approches classiques présentent quelques difficultés pour s'étendre à un environnement de flux de données. La classification non supervisée cherche à former des groupes d'individus partageant certaines caractéristiques. L'objectif est soit de détecter des groupes naturels, faciles à analyser, soit de réduire la volumétrie. Le principe général repose sur une minimisation des distances intra-classe et sur une maximisation des distances inter-classes.
Il existe une multitude de méthodes de classification non supervisées basées sur la classification hiérarchique ou le partitionnement [110]. Ces techniques souffrent de la nécessité d'avoir plusieurs passes sur les données. Cependant, dans le cadre des flux de données, plusieurs de ces techniques ont été réadaptées pour fonctionner en une seule passe. Nous illustrons dans ce qui suit les principales techniques qui ont prouvé leur utilité à la fois théoriquement et pratiquement pour la conception et l'analyse de méthodes de regroupement dans le contexte dynamique des flux de données.

Dans[98][97], les auteurs s'intéressent au problème $k-median$. L'idée consiste à trouver les k évènements du flux qui minimisent la somme des distances [11]. Dans leur démarche, les auteurs ont abordé le problème en deux temps. La première étape consiste à fournir un algorithme qui utilise un espace restreint et, la deuxième étape, permet le regroupement des éléments en une seule passe. L'algorithme nécessite $O(nk)$ en temps, et $O(n^{\epsilon})$ en espace mémoire (avec k nombre de classes, n nombre d'éléments dans le flux et $\epsilon < 1$). Dans [97], l'algorithme $k-center$ est proposé en extension de l'algorithme $k-median$. L'idée consiste à rechercher les k éléments du flux de données qui minimisent la distance maximale entre un centre de classe et l'élément affecté le plus éloigné. L'algorithme réalise une seule passe sur les données, avec un temps de traitement en $O(nk\ log\ n)$ et n^{ϵ} en quantité mémoire (avec $0 < \epsilon < 1$). Contrairement à la mesure $k-median$ qui est moins sensible aux bruits, la mesure $k-center$ est sensible aux données aberrantes et atypiques.

D'autres techniques de classification non supervisée pour les flux de données ont vu le jour : Smaller Space[98], STREAM[140], CluStream[15], EMADS[116], ODAC[153], etc.
Smaller Space et STREAM traitent les éléments du flux par paquets. Dans chaque paquet, une classification de type $k-means$ est réalisée et les barycentres résultants sont conservés. Lorsqu'une quantité suffisante d'éléments est observée, ces centroides sont à leur tour classifiés formant ainsi des centroides d'un plus haut niveau. Smaller Space présente deux inconvénients : (i) il est insensible aux changements dans le flux et (ii) il est difficile d'évaluer des requêtes sur la totalité du flux. Les auteurs ont présenté dans [140] une extension à cet algorithme en présentant l'approche STREAM. STREAM présente de meilleurs résultats que Smaller Space, ce dernier n'est pas adapté aux analyses faisant intervenir les données récentes.
Dans[15], Aggarwal et al présentent l'algorithme CluStream, permettant de créer des classes sur un horizon temporel défini par l'utilisateur. L'idée principale de cette approche consiste à maintenir de façon incrémentale k micro-classes dans une mémoire de taille bornée et de conserver par la suite, dans des clichés, l'évolution de ces micro-classes avec une précision qui diminue avec l'ancienneté. Comme CluStream, l'algorithme EMADS se base sur la constitution en ligne des classes qui seront par la suite utilisées dans une étape hors ligne permettant de réaliser la vraie classification des données. Contrairement aux autres algorithmes qui utilisent $k-means$, EMADS utilise l'algorithme EM[92] pour réaliser la classification non supervisée des données.

1.6.4 Classification supervisée d'un flux de données

La classification supervisée (en anglais *Classification*) a pour but d'identifier les classes auxquelles appartiennent des objets à partir de traits descriptifs (attributs, caractéristiques, features). Les classes sont connues et l'on dispose d'exemples de chaque classe. Étant donné n_C différentes classes, un algorithme de classification supervisée construit un

11. Il s'agit de minimiser la distance entre les évènements et les centres des classes.

modèle qui prédit pour chaque élément I, la classe C à laquelle il appartient. L'objectif est de construire un classifieur, i.e. un procédé qui permet l'attribution d'une étiquette (ou modèle) à un élément quelconque au vu de ses caractéristiques.

La plupart des méthodes classiques de classification telles que les plus proches voisins et les arbres de décision sont appliquées dans le cadre des données stationnaires. Généralement, les flux de données ne sont pas stationnaires, ces algorithmes ont été réadaptés pour gérer la non-stationnarité des données. Nous illustrons ci-dessous des exemples de classifieurs qui ont été adaptés de façon à pouvoir être utilisés sur des flux de données :

- Plus proche voisin (en anglais *Nearest Neighbor Classifier*) : dans cette technique, la classe majoritaire parmi k proches voisins de l'élément cible est utilisée pour effectuer la classification. Étant donné que les voisins les plus proches ne peuvent être facilement définis sur la totalité du flux, un échantillon du flux est utilisé pour effectuer la classification [37]. Cet échantillon peut être mis à jour dynamiquement en un seul passage en utilisant une technique appelée échantillonnage réservoir [168]. Afin de faire face au caractère évolutif du flux, l'échantillonnage réservoir biaisé [14] peut être appliqué dans ce sens. Dans ce type d'échantillonnage, une fonction du temps est utilisée pour maintenir un échantillon favorisant les éléments récents.
- Les arbres de décision (en anglais *Decision Tree Classifier*) : dans cette technique, les arbres de décision doivent être construits en un seul passage. Une méthode connue sous le nom de VFDT (Very Fast Decision Tree) a été proposée dans [71]. Elle permet la classification en ligne des flux de données stationnaires en utilisant les fenêtres glissantes. L'idée consiste à utiliser l'inégalité de Hoeffding [12] afin de déterminer le nombre minimum d'éléments requis à un noeud pour choisir la meilleure variable de division. l'inégalité d'Hoeffding est utilisée afin de s'assurer que l'arbre généré produit le même arbre que celui généré en utilisant la totalité des données du flux. Une extension à cet algorithme a été présentée dans [108] afin de traiter les flux de données non-stationnaires.
- Cluster-based Classifier : Aggarwal et al ont adopté l'idée des micro-classes introduite dans CluStream pour présenter le principe de la classification à la demande [16]. Cette méthode divise le processus de classification en deux composants : un premier composant permet de résumer le flux de données sous forme de statistiques. Le second composant permet d'utiliser ces statistiques pour effectuer la classification. Chaque micro-classe est associée à un label. Il s'agit de l'étiquette de l'ensemble des éléments associés à la micro-classe. Cette technique est appellée approche de classification à la demande car l'ensemble des éléments de test peuvent arriver sous la forme d'un flux de données et peuvent être ainsi classifiés efficacement à la demande. Au même

12. L'inégalité de Hoeffding permet de borner la valeur réelle d'une variable aléatoire par rapport à son espérance et un terme d'erreur.

moment, le résumé sous forme de statistiques peut être mis à jour efficacement avec l'arrivée des événements.

- Ensemble Classifier : cette technique de classification a été proposée par Wang et al [170] afin de répondre aux problèmes liés au concept drift dans les flux de données (évolution des classes ainsi que de la distribution des données). L'idée consiste à construire un modèle en utilisant un ensemble de classifieurs tels que les arbres de décisions (C4.5, Ripper, etc.) afin d'améliorer l'exactitude des prédictions. Cette méthode a été présentée comme solution à différents défis dans le domaine de la classification des flux de données :
  - Concept drift : plusieurs classifieurs présentent des résultats sensibles aux concept drift dans les flux de données. L'échantillonnage est généralement utilisé pour étudier la distribution des données. Cependant, si le flux est stationnaire, il n'est pas intéressant d'ignorer plusieurs parties du flux. Le rôle du classifieur est de décider sur quelle partie du flux, la classification doit être réalisée.
  - Robustesse : la méthode de classification utilisant plusieurs classifieurs a été mise en place afin de rendre les modèles plus robustes. L'idée principale consiste à éviter le sur-apprentissage des classifieurs utilisés.

1.7 Systèmes de gestion de flux de données

Les flux de données touchent aujourd'hui de nombreux domaines dans des secteurs très variés (e.g. médical, télécommunications, financier, surveillance, etc.). En fonction du domaine et des exigences, plusieurs traitements peuvent s'appliquer sur ces données. Traditionnellement, pour répondre aux besoins des utilisateurs, on utilise un Système de Gestion de Bases de Données (SGBD). Ces systèmes ont un contrôle absolu sur le stockage physique des données et sont capables de déterminer quand et comment elles sont accessibles lors d'un traitement. Cependant, l'apparition des flux génère quelques difficultés (au niveau du traitement et de l'analyse), essentiellement liées à la taille et au taux d'arrivée de ces données. L'information doit être extraite instantanément sur des données continues rendant ainsi l'utilisation d'un SGBD inadaptée. Afin de pallier cet inconvénient, de nouveaux systèmes ont récemment fait leur apparition : les Systèmes de Gestion de Flux de Données (SGFD).

1.7.1 Applications de bases de données *Vs.* applications de flux de données

Nous consacrons cette partie à une comparaison entre les applications de bases de données (*Data Based Applications*) et les applications de flux de données (*Stream Based Applications*). L'idée revient à étudier la capacité des systèmes traditionnels à répondre aux exigences fonctionnelles imposées par les applications de flux de données. Une

comparaison des deux systèmes est illustrée dans le tableau 1.1.

– **Traitement de requêtes continues :** les applications basées sur les SGBD traditionnels permettent de lancer des requêtes qui s'évaluent une seule fois. Dès que la requête est transmise, elle est évaluée et le résultat est instantanément renvoyé à l'utilisateur. Tandis que les applications de flux de données exigent une réévaluation continue des requêtes depuis leur inscription dans un registre jusqu'à leur suppression ou l'arrêt du flux. Un exemple de requête continue consiste à calculer la durée moyenne des communications téléphoniques émises par un client. La Figure 1.9 illustre le mécanisme général pour le traitement de requêtes par les SGFD et les SGBD.

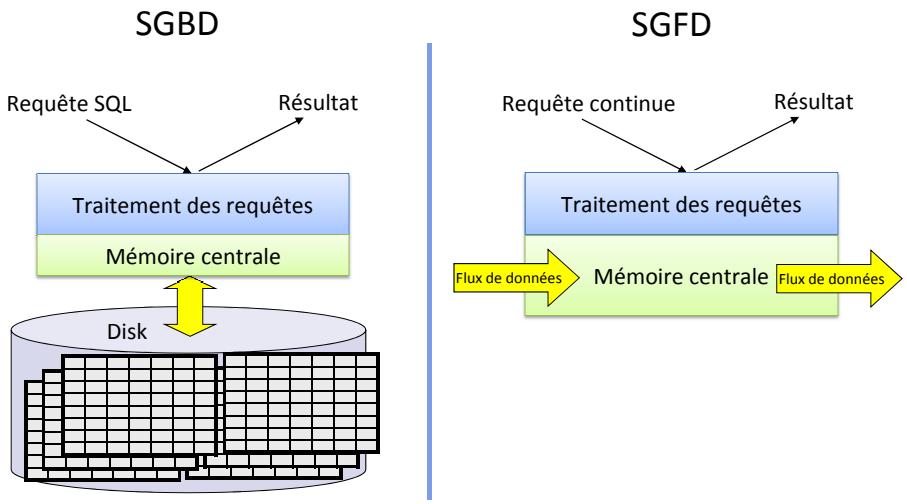

FIGURE 1.9 – Mécanisme général de traitement de requêtes

– **Temporalité des données :** généralement, la temporalité des données n'est pas importante dans les applications de base de données (i.e. en excluant le cas des BD temporelles ainsi que les systèmes de versions). La mise à jour d'un attribut entraîne la suppression de son ancienne valeur et l'insertion d'une nouvelle instance. Cependant, l'aspect séquentiel des flux rend la temporalité des données fondamentale. Chaque mise à jour est une valeur à part. Les traitements peuvent soit porter sur la dernière valeur de l'entité, soit sur l'historique de toutes les valeurs. C'est l'exemple d'un opérateur téléphonique qui veut calculer la durée moyenne des communications d'un client durant les 10 dernières minutes.

– **Infinité des données :** dans les applications de base de données, les requêtes sont évaluées dans un environnement fini. En effet, chaque requête représente une transaction unique, i.e. dès son lancement l'état de la BD reste inchangé. Cependant,

Applications de base de données	Applications de flux de données
Évaluation unique des requêtes	Évaluation continue des requêtes
Relations persistantes	Flux transitoire
Accès aléatoire aux données	Accès séquentiel aux données
Comportement passif	Comportement actif

TABLE 1.1 – Applications de base de données Vs Applications de flux de données

dans les applications de flux de données, l'environnement est complètement évolutif. Le système n'a aucun contrôle sur l'arrivée des données et les requêtes doivent alors être réévaluées tant que les données continuent d'arriver.

1.7.2 Caractéristiques attendues d'un SGFD

Un système de gestion de flux de données doit disposer d'un certain nombre de caractéristiques lui permettant de répondre aux besoins des applications citées précédemment. Dans [162], Stonebraker et al définissent les caractéristiques attendues d'un SGFD en proposant un ensemble de règles qui touchent aux fonctionnalités des systèmes et aux performances attendues.

1.7.2.1 Exigences sur les fonctionnalités des systèmes

Traitement de requêtes continues. Les requêtes sur flux de données sont continues et retournent des réponses sous forme de flux. Pour formuler ces requêtes, l'approche la plus adoptée est d'utiliser le modèle relationnel et de construire des requêtes avec un langage similaire à SQL. En raison du caractère infini des données, un système de gestion doit être capable de délimiter une opération afin de générer une réponse. Des traitements basés sur le temps ou l'ordre des données doivent alors être envisagées. Des techniques de fenêtrages permettent de poser des limites temporelles sur les flux. A cet effet, la sémantique du standard SQL doit être étendue en ajoutant la notion de fenêtres et de nouveaux opérateurs spécifiques aux flux de données.

Traitement des données statiques et continues. Dans le cadre de la supervision ou de détection de fraude, il est commun de comparer en temps réel des données du passé avec des données du présent. Les données du passé sont généralement mémorisées dans des bases de données. Le système de gestion de flux doit ainsi être capable de combiner des données statiques en bases des données aux données dynamiques du flux. Par ailleurs le système doit autoriser le stockage, l'accès et la modification de données statiques.

Disponibilité des données. Les systèmes doivent garantir la disponibilité des données à tout instant. De plus, ils doivent faire face aux pannes systèmes. Ainsi une opération de sauvegarde du contenu de la mémoire du SGFD doit être prévue. Celle-ci est assurée par la mise en place d'un système secondaire (un clone) qui prend la place du système principal face à une panne.

1.7.2.2 Exigences sur la performance des systèmes

Résistance aux imperfections du flux. Les systèmes de flux doivent traiter les données sans aucune obligation de stockage. Par ailleurs, ils doivent prendre des précautions concernant les retards, les pertes et les données inattendues. Une longue attente d'une donnée peut engendrer le blocage des opérations. Prenons l'exemple du traitement suivant où on cherche à calculer le prix moyen des 20 dernières ventes. Pour pouvoir répondre à cette requête, le système est obligé d'attendre l'arrivée des résultats de ces 20 ventes. Cette requête est alors bloquée tant que les données nécessaires ne sont pas arrivées. Certes, dans un environnement temps réel, il est impossible d'attendre indéfiniment l'arrivée d'un tuple. La solution est de poser un délai maximum de retard (time out) pour d'éventuelles opérations bloquantes [13]. Ainsi, tous les calculs pouvant être bloquants sont traités dans les délais même si les données utilisées sont partielles.

Traitement de la volumétrie. dans le cas d'une grande volumétrie des données, le système doit être capable d'utiliser des techniques tel que le délestage (*load shedding*) [164][25] permettant de réduire la réduction de charge du système.

La plupart des SGFD respectent les règles liées aux fonctionnalités des systèmes. Cependant les règles associées aux performances des systèmes sont plus difficiles à satisfaire.

1.7.3 Gestion du temps et fenêtrage

Le caractère dynamique du flux fait apparaître la notion d'opérations dites bloquantes (e.g jointure, groupement, ordonnancement, etc.). Afin d'appliquer ces traitements, il est nécessaire de faire un compromis entre le temps de traitement et le niveau de détail de l'analyse. Cet équilibre s'exprime sous la forme d'une fenêtre [23] [80] [90] [141] [82].

Une fenêtre est une portion temporelle et finie du flux qui correspond à la vue accordée sur les données. Le principe du fenêtrage repose sur un découpage du domaine d'agencement des éléments, et sur la spécification d'une taille de fenêtre et d'un décalage entre deux fenêtres successives.

Le découpage du domaine d'agencement peut être :

13. Une opération est dite bloquante si elle fait intervenir un opérateur qui bloque le traitement de la requête jusqu'à l'arrivée de tous les évènements nécessaire au traitement (e.g. JOIN, la moyenne sur la totalité du flux, etc.)

- **Physique** : les fenêtres physiques sont définies au moyen d'intervalles de dates. Par exemple : la fenêtre du 15/01/2008 au 15/06/2008
- **Logique** : les fenêtres logiques sont définies en terme de nombre d'éléments à considérer dans la fenêtre. Par exemple : la fenêtre du 1^{er} au 1000^{eme} élément.

La modélisation du temps a une grande importance dans la conception des algorithmes. En effet, la taille d'une fenêtre logique est connue a priori, alors que celle d'une fenêtre physique n'est connue qu'à posteriori.

Selon les instants de début et de fin, on distingue trois types de fenêtres :

- **Fenêtre Fixe :** il s'agit d'une portion du flux dont les bornes sont précises, e.g. une fenêtre entre le 1^{er} juin et le 1^{er} juillet 2008.
- **Fenêtre Glissante :** elle se caractérise par des bornes qui évoluent avec le temps, e.g. les 10 derniers événements du flux. Ces fenêtres ont une durée fixe dans le cas des fenêtres physiques et un nombre fixe d'évènements dans le cas de fenêtres logiques. On compte trois variantes de fenêtres glissantes. Elles se distinguent par leur taille et leur pas de décalage :

  1. Si le pas de décalage est inférieur à la taille de la fenêtre, on parle de fenêtre purement ***glissante*** (ou *sliding window*) ;
  2. Si le pas de décalage est égal à la taille de la fenêtre, on parle de fenêtre ***sautante*** (ou *jumping window*) ;
  3. Si le pas de décalage est supérieur à la taille de la fenêtre, on parle de fenêtre ***bondissante*** (ou *hopping window*).

     La Figure1.10 illustre la différence entre ces trois types de fenêtres.

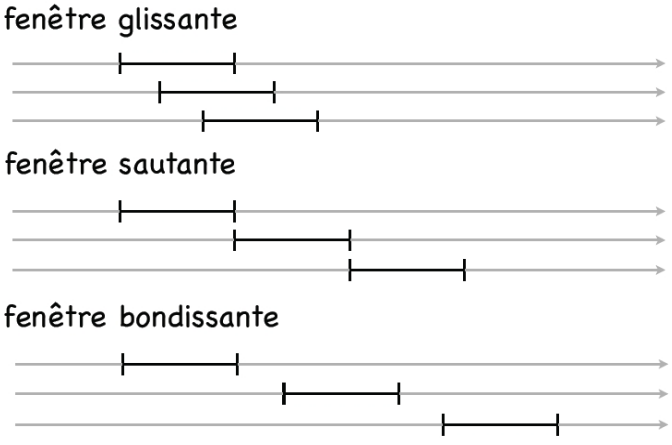

FIGURE 1.10 – Exemples des fenêtres glissantes
[77]

– **Point de repère :** les fenêtres point de repère (ou *landmark window*) commencent à une date précise et se terminent à la date courante. Par exemple : une fenêtre entre le 1^{er} janvier et le temps courant.
Une caractéristique importante de ce type de fenêtre est la taille de la structure qui augmente avec le déroulement du flux. La Figure 1.11 illustre le principe de cette structure.

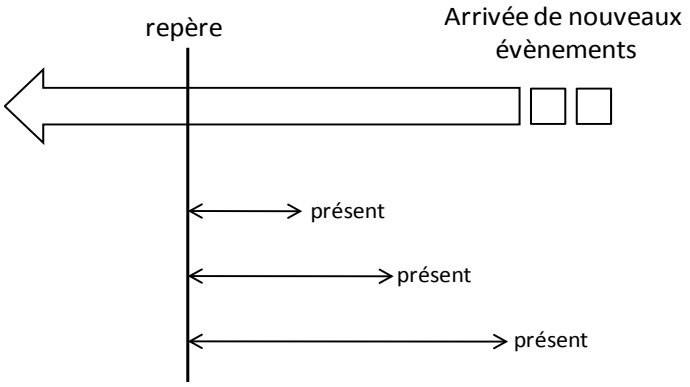

FIGURE 1.11 – Fenêtre point de repère

1.7.4 SGFD *Vs.* Systèmes de gestion d'événements complexes (SGEC)

Depuis quelques années on voit apparaître plusieurs types de SGFD qui peuvent être classés en deux catégories : (i) SGFD permettant le traitement de flux de données, (ii) SGEC permettant la gestion des évènements complexes. Ces derniers plus connu sous le nom de CEP (pour *Complex Event Processing*).

1.7.4.1 Systèmes traditionnels de gestion de flux de données

Les SGFD traditionnels assurent les fonctionnalités requises pour le traitement des flux de données. Ces systèmes peuvent être classés en deux catégories : (1) les SGFD spécifiques à une problématique particulière et (2) les SGFD généralistes conçus pour traiter des flux à structure quelconque.

Aurora est un système de gestion de flux de données généraliste, orienté workflow pour les applications de supervisions (Monitoring) [9] [26]. Il dispose d'une interface graphique pour la spécification du plan de requêtes. La version commerciale de ce système a été proposée en 2003 sous le nom de *StreamBase*. La Figure 1.12 illustre l'architecture du système Aurora. Dans ce qui suit, nous allons décrire les principales fonctionnalités de chaque composant de cette structure.

– *Les catalogues ("catalogs") :* les catalogues permettent de stocker les différentes méta-données sur la topologie d'un réseau, les fonction QoS (Qualité de Service), les

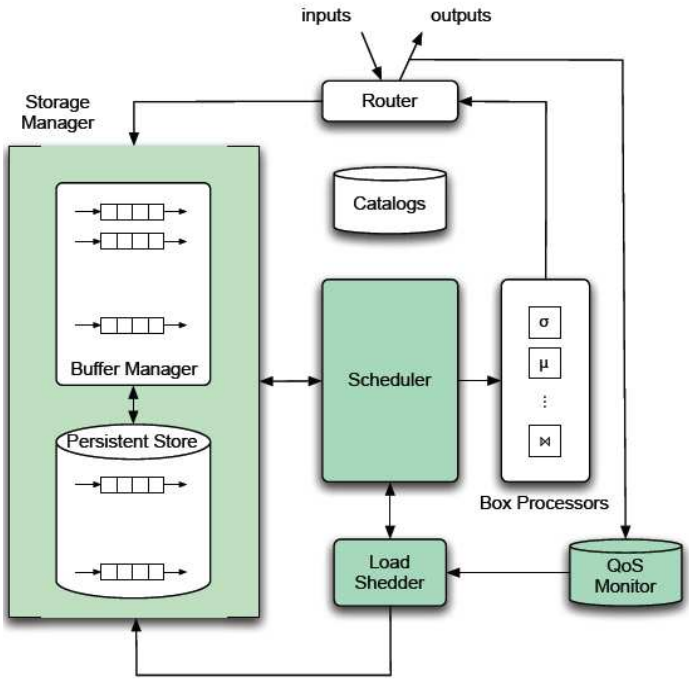

FIGURE 1.12 – Architecture du système Aurora

statistiques d'exécution telles que le coût moyen de traitement d'un opérateur de la requête, etc. Ces informations sont importantes pour les opérations effectuées par les autres composantes de l'architecture. Ces catalogues sont ainsi accessibles par toutes les autres composantes.

– *Le routeur ("router") :* le routeur est chargé de transmettre les flux de données aux différents composants d'exécution. Il reçoit en entrée les évènements issus des différentes sources de données ainsi que les évènements à lancer qui arrivent de la boîte de transformation (*"processor box"*).
Si le traitement d'une requête sur un évènement du flux est achevé, le routeur envoie cet élément pour alimenter des applications externes. Sinon, il le transmet au gestionnaire de stockage (*"Storage manager"*) où il est mis sur une file d'attente pour un traitement ultérieur.

– *Le gestionnaire de stockage ("Storage manager") :* cette composante est responsable de l'efficacité de stockage et d'extraction des données dans les files d'attente. Il gère la mémoire tampon qui mémorise les items du flux pour un usage immédiat par les boîtes des traitements ainsi q'un stockage persistant qui lui permet de garder un historique des traitements potentiels des requêtes ad-hoc. Les boîtes des traitements contiennent l'ensemble des opérateurs (exemple : opérateur de jointure).

Contrairement aux requêtes prédéfinies qui sont définies avant l'arrivée du flux, les requêtes ad-hoc sont ajoutées au fur et à mesure que le flux arrive.

- *Le répartiteur ("The scheduler") :* c'est la composante principale de cette architecture. Elle sélectionne une boîte pour l'exécuter, s'assure du traitement à effectuer et passe la main au processeur. Celui-ci exécute l'opération appropriée et renvoie les évènements de sortie vers le routeur.

- *Le moniteur de QoS ("QoS Monitor") :* cette composante supervise les performances du système et active le mécanisme de load shedding (*"load Schedder"*) si la QoS diminue.

- *Load Schedder :* cette composante est responsable du traitement des surcharges dues à un débit trop élevé d'évènements en entrée. Il consulte le catalogue afin d'avoir des informations statistiques sur le système. Il fait aussi certaines modifications dans le plan des requêtes, telles que la suppression des évènements pour minimiser la dégradation, afin qu'il soit possible de les exécuter par le CPU le plus rapidement possible.

La version commerciale d'Aurora (StreamBase)[8] est un système générique proposant une extension du langage SQL (*StreamSQL*) incluant la définition de la structure de flux, la définition de fenêtres et la manipulation de données statiques. Pour le traitement des données distribuées, une version de Aurora a été développée, il s'agit de Borealis[51].

TelegraphCQ : TelegraphCQ [124] [42] est un SGFD généraliste développé dans le cadre d'un projet mené par l'Université de Berkeley. C'est une extension du système de gestion de base de données PostgreSQL permettant de poser des requêtes continues sur des flux de données. TelegraphCQ a été conçu afin de faire face aux flux de haut débit. TelegraphCQ est l'extension des systemes Eddy[22] et Psoup[44]. Eddy est un mécanisme d'exécution de requête sur flux qui permet d'ordonner continuellement les opérateurs dans les plans de requête. Psoup est un moteur de requête basée sur TelegraphCQ permettant de mémoriser momentanément les résultats des requêtes afin de permettre les interrogations hors ligne. Ce moteur a pour objectif d'évaluer les requêtes définies a priori comme celle définies a posteriori de l'arrivée des évènements du flux.

La Figure 1.13 illustre l'architecture du système TelegraphCQ qui contient trois composantes principales :

- *Front-End* : cette composante permet de traiter les requêtes statiques ainsi que la définition des données et des requêtes ;
- *Back-End* : cette composante permet l'exécution partagée des requêtes continues ;

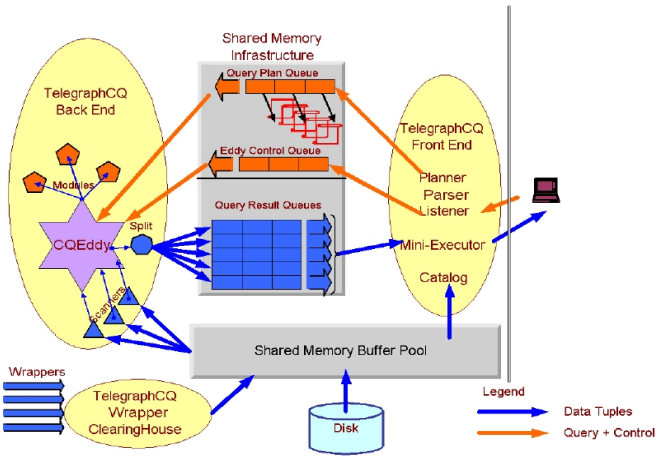

FIGURE 1.13 – Architecture de TelegraphCQ

– *Wrapper* : cette composante garantit que les données en entrée ne forment pas un obstacle à l'avancement de l'exécution de la requête.

Ces trois composantes communiquent à travers une structure de mémoire partagée (*Shared Memory*). Pour traiter une requête, un écouteur (*Listener*) est mis en place afin d'accepter les commandes des clients. Si la commande est une définition ou une requête sur une table, elle sera envoyée à la composante Front-End. Cependant, s'il s'agit d'une requête continue, elle sera envoyée à la composante Back-End à travers la file d'attente des requêtes.
L'exécuteur traite continuellement les éléments de la file d'attente et les places dynamiquement dans l'ensemble des requêtes évaluées. Les résultats des requêtes sont placés dans des files d'attente spécifiques à chaque client.

Nous avons défini précédemment deux SGFD généralistes pouvant être utilisés dans divers domaines. Cependant, la liste est encore longue nous pouvons citer le système Hancock[58], STREAM[18], Nile[105], etc.

1.7.4.2 Systèmes de gestion d'événements complexes

Le développement des applications de suivi et de gestion d'événements complexes a été motivé par l'apparition de systèmes qui doivent réagir à des "situations", comme la gestion des chaînes d'approvisionnement basées sur les capteurs RFID ou la supervision des réseaux de capteurs. En effet, ces applications doivent procéder à des traitements continus des flux de données afin de détecter des événements dits « intéressants ». Ces événements forment une abstraction de plusieurs autres événements (événements membres) qui surviennent en respectant un certain motif (pattern) déclenchant ainsi une séquence d'actions prédéfinies

une fois que la situation est détectée. Par exemple la crise financière mondiale [5] qui a débuté en juillet 2007 est un événement complexe créé à partir de plusieurs autres événements tels que : le dégonflement des bulles de prix (dont la bulle immobilière américaine des années 2000), les pertes importantes des établissements financiers, etc.

Les SGFD traditionnels ne sont pas conçus pour supporter facilement ces événements complexes. Prenons l'exemple de la requête suivante : « Préviens moi si le client A téléphone au client B puis au client C ». Cette requête peut être exprimée par les SGFD classiques. Cependant, ces derniers n'offrent pas le même niveau d'optimisation que les systèmes de gestion d'événements complexes (SGEC) [14]. En effet, ces systèmes présentent des particularités sur les traitements réalisés en proposant une sémantique et une algèbre particulière (e.g. SNOOP, CEL, SASE+) ainsi qu'une classe de requêtes plus riche (e.g. opérations de « pattern matching ») pour spécifier des patrons complexes d'enchaînements d'évènements.

Plusieurs systèmes (e.g. Esper, SASE, CORAL8, etc.) prétendent être à la fois des systèmes de gestion de flux de données et des systèmes de traitement d'événements complexes. En effet, il est important de préciser que le traitement des événements complexes est complémentaire au traitement classique des flux de données.

Esper : [1] est un SGEC open source et gratuit développé en Java. Il est utilisé comme moteur de gestion de flux de données dans le produit Oracle BEA [7]. Esper est à la fois un logiciel de traitement de flux d'événements (ESP pour Event Stream Processing) ainsi qu'un moteur de corrélation [15] (CEP) qui fonctionne en temps réel et déclenche des actions lorsque des conditions sur des événements surviennent dans le flux. Le langage d'interrogation associé à Esper est EPL (*Event Processing langage*). Il permet l'expression de requêtes complexes qui incluent des fenêtres temporelles et des opérations de jointure sur les flux d'événements. Les bibliothèques java fournies dans le package permettent d'effectuer plusieurs opérations sur les flux :

- Déclarer un flux et son schéma ;
- Créer et exécuter des requêtes en EPL (Event Processing Language, le langage d'interrogation d'Esper) ;
- Instancier un serveur Esper, dont la tâche est de compiler, enregistrer et exécuter les requêtes EPL ;
- Contrôler l'instance serveur (démarrage, arrêt, etc.) ;
- Gérer les processus légers et les accès concurrents.

L'enregistrement des requêtes EPL dans le moteur de corrélation est réalisé en utilisant des objets Java (POJO, JavaBean) pour représenter les événements. Un composant "*Listener*" est appelé par le moteur de corrélation lorsque la condition sur l'évènement est satisfaite.

Esper offre la possibilité de poser des requêtes avec des fenêtres glissantes ou sautantes.

14. SGEC = traitement de flux de données + traitement d'évènements complexes
15. le moteur de corrélation permet la gestion et la corrélation de plusieurs flux de données en entrée.

– Exemple de requête avec fenêtre glissante :
`SELECT * FROM StreamInEvent.WIN:LENGTH{3}`
L'expression "$WIN : LENGTH(3)$" de la requête ci-dessus définit une fenêtre glissante. Cette fenêtre contient les trois derniers événements du flux "StreamInEvent".
– Exemple de requête avec fenêtre sautante :
`SELECT * FROM StreamInEvent.WIN:LENGTH_BATCH{3}`
L'expression "$WIN : LENGTH\_BATCH(3)$"' de la requête ci-dessus définit une fenêtre sautante. Cette fenêtre contient le dernier lot de trois événements du flux "StreamInEvent".

Coral8 : est un SGEC commercial issu du système STREAM [18] développé à l'Université de Stanford. La Figure1.14 illustre les différentes composantes de ce système.

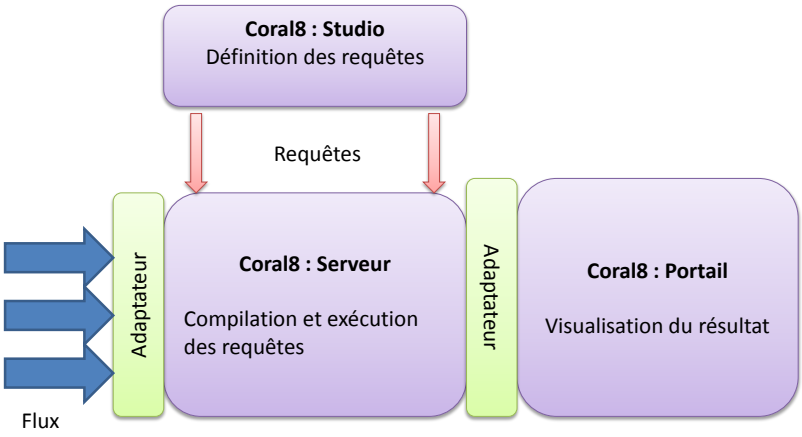

FIGURE 1.14 – Architecture haut niveau du système Coral8

– Coral8 Studio : est une application graphique permettant à l'utilisateur de créer des requêtes, des fenêtres, de lier un adaptateur à un flux, etc. Les adaptateurs permettent de transformer des fichiers CSV, XML en flux, de se connecter à une base de données, etc.
– Coral8 Portal : est un client Web permettant la visualisation en temps réel des flux manipulés par Coral8.
– Coral8 Server : est un ensemble de bibliothèques permettant le développement d'applications client en C/C++, Java, etc.

Coral8 utilise CCL (*Continuous Computation Language*) comme langage de définition de requête. Ce langage est basé sur le standard SQL. Quand un utilisateur pose sa requête en CCL, un compilateur procède à la transformation de cette requête en un ensemble de primitives de plus bas niveau. CCL permet la définition de fenêtres glissantes et sautantes. Cependant sa particularité réside dans sa capacité à rechercher des séquences d'événements dans plusieurs flux. La requête suivante, par exemple dans le cadre d'une application de

supervision basée sur des puces RFID, vérifie si un tag a été vu par les lecteurs A et B, puis C, mais pas D, dans une fenêtre de 10s :

```
Insert Into StreamAlerts
Select StreamA.Id
From StreamA a, StreamB b, StreamC c, StreamD d
Matching [10 seconds : a && b, c, !d]
On a.Id = b.Id = c.Id = d.Id
```

Le système Coral8 est capable de distribuer les traitements afin d'améliorer les performances, résister aux chutes du serveur, ou encore de fédérer les traitements. Récemment, Aleri [16] et Coral8 ont fusionné afin de faire face aux géants des SGBD qui se sont intéressés aux systèmes de gestion d'événements complexes tel que Oracle et IBM.

Dans cette section nous avons présenté une liste non-exhaustive de SGFD et SGEC. Le tableau 1.2 résume les principales caractéristiques de ces systèmes (y compris des systèmes non décrt ci-dessus).

16. Système de gestion d'évènements complexes[3]

	Nom de la classe	Description				
Aurora	Réseaux de capteurs	Fixes, Glissantes Point de repère	SQuAI	Généraliste	StreamBase	Dans StreamBase
Cayuga[68]	Supervision des réseaux sociaux Analyse des actions boursières	Glissantes	CEL	Généraliste	–	O
Cougar[181]	Réseaux de capteurs	Fixes, Glissantes Point de repère	Orienté Objet	Spécifique	–	N
Esper	Tout domaine	Tout type	EPL	Généraliste	Esper Entreprise Edition	O
GicaScope[59]	Supervision de réseaux	Glissantes	GSQL	Spécifique	–	N
Hancock	Analyse de flux transactionnel	Glissantes Point de repère	Langage C	Spécifique	–	N
NiagaraCQ[48]	Traitement de flux XML distribué	Glissantes	XML-QL	Spécifique	–	N
SASE[101]	Réseau de capteurs	Glissantes	SASE+	Généraliste	–	O
Statstream [184]	statistique	Fixes, Glissantes Point de repère	Non renseigné Objet	Spécifique	–	N
STREAM[18]	Tout domaine	Glissantes	CQL	Généraliste	Coral8	Dans Coral8
TelegraphCQ	Réseaux de capteurs	Tout Type	StreaQuel	Généraliste	Truviso	N

TABLE 1.2 – Résumé des systèmes de gestions de flux de données existants.

1.8 Domaines d'application

Le domaine des flux de données a, depuis quelques années, suscité l'intérêt de différentes communautés de chercheurs (base de données, analyse de données, statistique, etc.) d'un point de vue académique et industriel. Un intérêt qui se reflète dans le nombre croissant de techniques, applications et systèmes industriels qui permettent de contrôler et mesurer cette masse importante d'information. Les utilisations des flux de données sont multiples et diverses en fonction du domaine et du type d'application [11]. Dans cette section nous nous intéressons aux domaines émergents d'application de flux de données.

1.8.1 Télécommunications

Pour stimuler la croissance des revenus et prévenir l'érosion des marges, les fournisseurs de services de télécommunications doivent constamment répondre à une demande accrue pour de nouveaux services. Cet environnement dynamique met à rude épreuve les réseaux, les systèmes de soutien des opérations et doit offrir un service de prévision, de surveillance, et de facturation.

Les entreprises de télécommunications génèrent une quantité énorme de données qui contient des détails sur : les appels qui traversent les réseaux de télécommunication, l'état des composantes matérielles du réseau, les données du client, etc. Dans [171], les auteurs décrivent les différentes tâches pouvant être réalisées en utilisant ces données :

- Détection de fraude : Les fraudes conduisent à de grandes pertes annuelles pour les entreprises de télécommunication. Elles peuvent être divisées en deux catégories : les fraudes de souscription et les fraudes de superposition. Les fraudes de souscription se produisent quand un client ouvre un compte avec l'intention de ne jamais payer les frais du compte. Les fraudes de superposition implique des activités illégales par une personne autre que le titulaire du compte. La méthode la plus commune pour identifier les fraudes est de construire un profil du comportement des appels du client et de comparer l'activité récente avec ce comportement.
- Profilage des clients : Les entreprises de télécommunications maintiennent une quantité importante de données sur leurs clients et stockent des enregistrements d'appels téléphoniques qui décrivent précisément les habitudes d'appel de chaque client. Cette information peut être utilisée pour le profilage des clients. Ces profils peuvent ensuite être utilisées à des fins de marketing et/ou de prévision.
- Détection des défaillances du réseau : Les réseaux de télécommunications ont des configurations matérielles et logicielles extrêmement complexes. La plupart des éléments du réseau génèrent plusieurs messages d'état et d'alarme. Afin de gérer efficacement le réseau, les alarmes doivent être analysées automatiquement afin d'identifier les défauts du réseau au moment opportun ou avant qu'ils ne surviennent. Une réponse pro-active est essentielle pour maintenir la fiabilité du réseau.

Le système *Apama* présenté dans [4] offre un ensemble de services dans le domaine des télécommunications. Dans ce système, les opérateurs téléphoniques utilisent une plateforme de gestion d'événements complexes afin de gérer une large gamme de réseaux de télécommunications. L'objectif de ce système consiste à surveiller les infrastructure de prestation de services et détecter des événements dits "d'intérêt" sur leur réseau. Par ailleurs, le système permet le suivie et l'analyse en temps réél des évènements du réseau.

1.8.2 Analyses financières

Les marchés financiers constituent une source importante de flux de données. Reuters transmet par exemple plus de 275,000 prix par jour [178].

Acheteurs		Trade	Vendeurs	
Volume	Prix	12.03	Prix	Volume
14.533	12.03	1	12.04	42.450
28.850	12.02	2	12.06	20.540
23.000	12.02	3	12.07	8.261
2.121	11.99	4	12.09	35.000
41.000	11.98	5	12.10	120.515

TABLE 1.3 – Exemple de données dans un marché financier

L'analyse en ligne des cours des actions implique la découverte de corrélations, l'identification de tendances, la détection de fraudes, etc. En effet, plusieurs opérations de fouille peuvent être appliquées sur ces flux financiers :

- Prédiction des séries financières : les investisseurs dans les marchés financiers veulent voir leurs revenus augmenter en achetant ou en vendant leurs investissements (actions boursières par exemple). Cependant, étant donnée la forte variabilité des flux financiers, prédire les futures tendances et les faire évoluer devient un vrai défi. Plusieurs travaux ont été introduits dans [123] afin de présenter des méthodes pour la prédiction des marchés financiers. Ils sont surtout basés sur les réseaux de neurones, les arbres de décision, etc.

- Recherche de modèles : Nesbitt et Barrass ont présenté dans [144], un outil qui permet d'aider les traders à détecter les tendances et prédire l'orientation du marché boursier. Les données boursières contiennent de nombreux attributs que les traders doivent analyser.

– Requêtes : pour l'évaluation des requêtes en temps réel sur des flux de données financiers, le moteur de recherche Tradebot [2] a été développé. C'est un SGFD spécifique permettant d'évaluer des requêtes continues sur des flux financiers.

1.8.3 Réseaux de capteurs

Dans [100], on définit un réseau de capteurs comme étant un réseau ad-hoc auto configurable composé de capteurs intelligents, chacun ayant une certaine capacité de calcul, de stockage et de communication sans fil. Les capteurs sont utilisés dans différents domaines tels que la médecine, le domaine militaire, la surveillance environnementale, etc.

Dans ce qui suit, nous allons décrire quelques travaux récents dans le domaine de la surveillance ainsi que celui de l'énergie. En particulier, nous allons nous intéresser aux techniques de suivi dans lesquelles les données sont traitées en temps réel.
Généralement les capteurs sont équipés de technologie leur permettant de détecter des objets intéressants (ou des événements). Pour cela, les capteurs doivent collaborer ensemble. Si l'un des capteurs détecte un objet, il collabore avec les autres capteurs pour prédire sa trajectoire.

Dans [130], les auteurs présentent une méthode de collaboration dynamique entre les capteurs dans le contexte de la localisation des objets en temps réél. L'objectif est de mettre en place un environnement collaboratif pour la localisation et la poursuite d'un ou d'un groupe d'objets. A chaque instant, un groupe relit les capteurs qui fournissent des informations sur l'objet traqué en excluant ceux qui sont moins instructifs ou qui fournissent des données redondantes. Lorsqu'un objet est détecté par un des capteurs, il envoie un message à tous les autres capteurs. Dès l'arrivée de ce message, les capteurs choisissent un "leader". Seul le "leader" connaît la répartition géographique de la collaboration. Il a pour rôle la gestion et la maintenance du groupe (suppression et ajout des membres).

Dans [103], Halkidi et al se sont intéressés au suivi des objets mobiles au sein d'un réseau de capteurs. Leur objectif consiste à mettre en place des techniques permettant la coopération de nombreux capteurs du réseau, et l'échange en temps réel des données de ces capteurs. Le système proposé utilise des probabilités afin de prédire le futur emplacement d'un objet. Cette prédiction est réalisée à partir des observations passées de l'objet qui ont été envoyées par les différents capteurs du réseau. Les auteurs proposent une approche à deux niveaux : le premier est effectué localement au niveau de chaque capteur. Chaque capteur doit détecter la présence de l'objet et estimer sa trajectoire en combinant les données locales de l'objet aux anciennes observations. Le second niveau consiste à combiner les estimations de trajectoire réalisées par les capteurs individuels et prédire avec une probabilité élevée la trajectoire de l'objet dans l'ensemble du système. Le

système est réparti en cellules, chaque cellule contient un "leader" qui est responsable de recueillir les données de suivi de tous les capteurs dans sa cellule et de communiquer avec les dirigeants (*leaders*) des autres cellules. En recueillant les données de suivi de sa cellule, le "leader" estime la trajectoire de l'objet et décide de manière dynamique quelle cellule va poursuivre le processus de suivi, il informe ainsi le "leader" de la cellule en question qui prend le relais.

Une autre utilisation des capteurs figure dans le domaine de l'énergie électrique. En effet, plusieurs fournisseurs d'énergie mettent en place des capteurs intelligents [10]. Ces capteurs vont envoyer les détails de la consommation électrique journalière de plusieurs foyers. Les données récoltées de ces capteurs se présentent sous la forme d'un flux de données. Ce flux permet par exemple d'avertir les ménages d'une sur-consommation ou d'une défaillance au niveau d'un appareil électroménager.

1.8.4 Télésurveillance médicale

Dans [161], Stoa et al illustrent une méthode permettant l'analyse en temps réel de flux de données issues de capteurs qui sont installés au domicile des patients afin de détecter des maladies critiques nécessitant une intervention instantanée. Dans l'article en question, les auteurs se sont intéressés au cas des patients atteint d'ischémie myocardique (*déséquilibre entre les besoins du coeur en oxygène et l'apport issu de la circulation sanguine coronarienne*). Pour cela, les auteurs ont implémenté un nouveau type de fenêtres dans Esper permettant l'exploitation des agrégats qui ciblent uniquement les capteurs transmettant le rythme cardiaque du patient.

Dans [157], les auteurs présentent le système "@Home" qui permet la télésurveillance médicale de patients à leur domicile.
Ce système permet de mesurer et collecter plusieurs informations relatives à l'état du patient tels que la pression artérielle, le niveau d'oxygène, l'enregistrement de son rythme cardiaque, etc. Ces informations sont envoyées d'une manière continue à l'hôpital en utilisant le système de communication des téléphones mobiles(GSM) ou le (PSTN).
Le système @Home est autonome et n'a pas besoin d'interagir avec le patient. Il est constitué de trois parties principales :

- Home sub system : Il est responsable de l'acquisition, la collecte, l'analyse et la transmission des signes vitaux du patient.
- Hospital sub system : Il est responsable de l'analyse, l'évaluation, l'archivage et la représentation des données.
- Telematics sub system : C'est le système de communication, qui transmet l'enregistrement des données provenant des capteurs au serveur de l'hopital.

Dans [34], les auteurs présentent l'intérêt qu'apporte le contrôle des flux de données dans le domaine médical. En effet, la télésurveillance médicale permet aux institutions de santé (les hôpitaux, les polycliniques, etc.) de surveiller l'état des patients à distance, au moment de leurs activités quotidiennes. Les informations proviennent de différents capteurs de petite taille que les patients transportent quotidiennement. Ces données arrivent sous forme d'un flux à un système qui permet de détecter par exemple une faiblesse dans le rythme cardiaque ou une hypoglycémie. Les acteurs de santé (urgentistes, médecins, infirmiers,etc.) peuvent ainsi agir rapidement pour venir en aide à la personne concernée. Les auteurs présentent le système *osiris-se*. Ce système fournit une plate-forme flexible pour les différents types d'applications de surveillance et garantit un haut niveau de fiabilité. En cas d'évolution dans l'état de santé du patient, son médecin est automatiquement informé, et il est capable de récupérer toutes les données médicales. Les événements critiques, tels que les problèmes cardiaques, peuvent être détectés par l'analyse des données transmises sous forme de flux.

1.9 Synthèse

Avec le grand *boom* des nouvelles technologies, plusieurs applications se sont développées, exploitant une quantité importante de données créées sous forme de flux. Ces données se caractérisent par leur volumétrie et leur débit rendant ainsi difficile, voire impossible, leur stockage. Le traitement de ces données doit se faire à la volée et les opérations classiques d'analyse sont impossibles à réaliser sans l'apport de nouveaux modèles et algorithmes puisque la majorité des approches classiques requièrent plusieurs passages sur les données. D'autre part, de nouveaux systèmes (les Systèmes de Gestion de Flux de Données) ont été développés afin de gérer et interroger un ou plusieurs flux de données via une série de requêtes. Contrairement au Systèmes de Gestion de Bases de Données où les données sont persistantes et les requêtes sont transitoires (ou éphémères), dans les SGFD, les requêtes sont persistantes et les données transitoires. Les données arrivent sous forme de flux infinis sur lequel les requêtes sont évaluées de manière continue. En raison du caractère spécifique des flux de données, les SGFD se voient confrontés à différents problèmes : traitement des opérateurs dits bloquants (e.g. Jointure, groupement, etc.), gestion de ressources dans le cas de requêtes multiples, gestion des éléments en retard, gestion des variations de débit, etc.

Dans ce chapitre, nous avons introduit le concept de flux de données ainsi que l'ensemble des caractéristiques associées. Dans le modèles des flux de données, les évènements se présentent de façon spontanée et à fréquence variable nécessitant un traitement en temps réel et à la volée. Ceci implique que pour tout besoin d'analyse sur les flux, il est nécessaire de préciser a priori sa tâche avant l'arrivée des données. Cependant, pour plusieurs applications, il est important de garder une trace de l'histoire d'un flux, soit pour effectuer un traitement non prévu a priori, soit dans un but de réaliser des tâches de fouilles hors-ligne. Une solution permettant de procéder à des traitements a posteriori consiste à conserver

une représentation compacte du flux de données appelée *résumé*. Cette approche revient à 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. Dans le chapitre suivant, nous allons nous intéresser aux structures de résumés pouvant être construites sur des flux de données.

Chapitre 2

Résumé de flux de données

Sommaire

2.1 Introduction

Les systèmes de gestion de flux de données (SGFD) permettent de répondre aux requêtes continues via l'utilisation de fenêtres temporelles définies sur la mémoire du système. L'expiration des données rend impossible l'évaluation de toute requête non prévue a priori. Ceci suppose de définir à l'avance sur un flux l'ensemble des requêtes dont on pourra avoir besoin par la suite. Pour l'évaluation des requêtes a posteriori, il est nécessaire de conserver une trace de l'histoire du flux car l'utilisation exclusive d'un SGFD ne permet pas de satisfaire un tel besoin. Il en est de même pour les systèmes de gestion de base de données (SGBD) qui se trouvent inadaptés pour répondre à ce type de besoin. En effet ces systèmes sont capables de conserver des historiques, cependant ils font face à deux contraintes qui les rendent inadaptés dans un contexte de flux de données :

- Taille des données : un flux de données est théoriquement infini. Il est par conséquent impossible de le conserver en totalité dans une structure de données ;
- Débit d'écriture : les opérations d'insertion dans les SGBD sont suivis de plusieurs opérations qui prennent un temps d'écriture considérable (opération de verrouillage, de journalisation, d'indexation, etc.). Ceci ne peut s'appliquer à des données arrivant avec des débits importants.

Pour pallier les insuffisances de ces systèmes, une solution permettant le traitement de données historiques des flux consiste à conserver des représentations compactes appelées

résumés. L'objectif est de conserver l'histoire du flux sans dépasser l'espace de stockage ni le temps de latence [17] accordés pour l'application. De nombreux travaux ont été récemment proposés pour développer des techniques de résumé qui s'adaptent au caractère non stationnaire des flux de données. Leur objectif consiste à trouver un compromis entre les contraintes imposées par les systèmes (traitement en une seule passe, ressources limitées, débits variables, etc.) et la précision des résultats. Ce chapitre dresse l'état de l'art dans le domaine de résumé de flux de données individuels.

2.2 Définition d'un résumé de flux de données

Par définition, un résumé de flux permet de conserver une trace des évènements apparus dans un flux de données. Cette notion est apparue avec le besoin de traiter l'ensemble des analyses qui ne sont pas prévues à l'avance pour un flux particulier. Par ailleurs, la conservation d'un résumé du flux permet de réaliser des avancées dans des domaines tels que la gestion des données en retard, les techniques de réduction de charge (en anglais *load shedding*)[164][25] ou encore l'approximation de certaines requêtes contenant des opérateurs bloquants.

Dans ce mémoire, nous proposons la définition suivante d'un résumé de flux de données : Un résumé de flux de données consiste à conserver avec un minimum d'évènements, les représentations statistiques de toutes les périodes temporelles du flux.

Nous distinguons dans la littérature trois classes de résumé de flux de données :

- **Les synopsis**. Les synopsis (aussi appelés résumés spécialisés) sont des structures de résumé de flux de données ayant pour objectif de traiter une requête particulière sur les évènements du flux[81]. Ces requêtes sont définies avant l'arrivée du flux. Il existe dans la littérature différents travaux sur les synopsis tel que les sketchs de Flajolet-Martin [78], les filtres de Bloom [31], etc.
- **Les historiques**. Ces structures permettent de résumer les flux de données à travers un ensemble d'agrégats. Ces agrégats sont spécifiques à un besoin particulier et sont définis à l'avance, avant l'arrivé du flux. Dans la plupart des architectures de SGFD, les évènements historiques sont appelés évènements archivés [41] [43].
- **Les résumés généralistes**. Ces structures permettent l'accès à un résumé de l'histoire du flux. La particularité d'un résumé généraliste réside dans sa capacité à réaliser des tâches de fouilles ou de requêtes sur le passé d'un flux. Quelque soit la période temporelle interrogée, les résumés généralistes ont pour objectif de répondre de façon approchée à n'importe quelle requête portant sur les évènements du flux. On peut citer comme algorithme de résumé généraliste, l'algorithme StreamSamp [61] qui permet de concevoir un résumé sous forme d'échantillons et, l'algorithme CluStream [15] qui fournit un résumé permettant d'assurer l'accès aux densités des données.

17. Le temps de latence associé à un évènement en sortie est le temps total du traitement de l'évènement par le SGFD.

2.3 Conception d'un résumé de flux de données

Un résumé de flux de données est conçu en se basant sur une organisation de la dimension spatiale et/ou une organisation de la dimension temporelle. L'organisation spatiale définit la structure utilisée pour disposer en mémoire les données conservées. Tandis que l'organisation temporelle définit la gestion de l'historique conservé en fonction du temps. Dans ce qui suit, nous illustrons les différentes techniques permettant une organisation spatiale et/ou temporelle des données du flux.

2.3.1 Conception simple d'un résumé de flux de données

Plusieurs techniques permettent de conserver en mémoire une version allégée des données du flux. Ce sont en réalité des approches qui existaient déjà dans le cadre du traitement de données massives et qui par la suite ont été adaptées au contexte des flux. Nous illustrons dans ce qui suit une liste non exhaustives des principales techniques utilisées dans la littérature pour la construction d'un résumé de flux de données.

2.3.1.1 Échantillonnage

Une approche particulièrement adaptée aux flux de données est l'approche par échantillonnage aléatoire. L'objectif de cette technique est de fournir des informations sur une large population à partir d'un échantillon représentatif extrait de celle-ci. Généralement, ces techniques nécessitent une seule passe pour échantillonner les données. Cet atout facilite leur adaptation dans le cas des flux de données. Cependant, les algorithmes d'échantillonnage exigent en général l'intégralité des données afin de sélectionner un échantillon représentatif. Cette contrainte rend impossible leur utilisation dans le cadre des flux. Pour pallier cet inconvénient, des algorithmes d'échantillonnage séquentiels ont été développés. Nous nous intéressons dans cette section aux principales méthodes qui ont été adaptées au contexte des flux de données. Nous désignons dans ce qui suit par k la taille de l'échantillon, n la taille du flux ou de la fenêtre glissante et par c les constantes utilisées dans les formules de complexités.

Échantillonnage réservoir. L'échantillonnage réservoir est introduit dans [168]. Cette technique est largement utilisée dans le cadre des flux de données. Elle permet d'obtenir de façon incrémentale un échantillon uniforme aléatoire de l'ensemble d'un flux, sans exiger une connaissance à priori de la taille du flux. La taille maximale du réservoir est fixée à l'avance et la mise à jour du réservoir se fait en temps constant. Le principe de l'approche consiste à maintenir un réservoir de taille k à partir des données du flux. L'étape d'initialisation permet d'insérer dans le réservoir les k premiers évènements du flux. Lorsqu'un nouvel évènement i ($i > k$) arrive, il remplace de manière aléatoire et avec une probabilité d'inclusion $\frac{k}{i}$ un des évènements du réservoir. Cette approche a l'avantage

d'être simple à implémenter et à mettre en oeuvre. En revanche, plus on avance dans le flux, plus la probabilité d'inclusion est réduite (puisque $lim_{i \mapsto \infty} \frac{k}{i} = 0$).

Dans [14], Aggarwal présente une méthode d'échantillonnage par réservoir biaisé. Une fonction de biais est associée au processus pour constituer les échantillons. Celle-ci se focalise sur les comportements récents ou les plus anciens afin de constituer un échantillon du flux. Ces comportements sont définis par les contraintes de l'application.
Fonction de biais : soit $p(r,t)$ la probabilité que l'évènement i du flux introduit à l'instant r soit toujours présent dans le réservoir à l'instant t. La fonction de biais $f(r,t)$ est proportionnelle à cette fonction de probabilité. Dans son approche, Aggarwal utilise une fonction de biais dite *exponentielle* définie comme suit :

$$f(r,t) = \exp^{-\lambda(t-r)} \ \textit{avec } \lambda \textit{ le taux de biais}, \ \lambda \ \in \ [0,1].$$

L'avantage de cette approche réside dans l'utilisation d'une fonction de biais qui permet l'inclusion de méthodes de remplacement simples. Cependant, la taille de l'échantillon n'est plus fixé dés le départ comme c'est le cas dans l'échantillonnage réservoir de Vitter. Une borne supérieure déterministe sur la taille du réservoir est contrôlée par les fonctions exponentielles utilisées. Cette technique est biaisée étant donné qu'elle assure une importante probabilité d'inclusion associée aux évènements récents, ce qui favorise leur insertion dans le réservoir.

Échantillonnage aléatoire pondéré. Dans cette approche, la probabilité d'inclusion d'un évènement dans l'échantillon est proportionnelle à un paramètre lié à l'évènement tel que sa taille. Il existe plusieurs techniques permettant d'obtenir un échantillon aléatoire pondéré. Ces techniques diffèrent par l'efficacité d'échantillonnage et de mise à jour de l'échantillon. Nous illustrons dans ce qui suit l'algorithme "Échantillonnage par Acceptation/Rejet". D'autres algorithmes sont décrit dans la thèse de Frank Olken [].

Supposons que nous souhaitons tirer un échantillon aléatoire de taille 1 à partir d'une population de taille N. Soit r_j un élément de la population avec une probabilité d'inclusion proportionnelle à son poids w_j. Soit w_{max} le poids maximum qui peut être associé à un évènement.

Pour créer cet échantillon, il suffit de générer aléatoirement un entier j, entre 1 et N. L'évènement r_j a une probabilité d'inclusion p_j :

$$p_j = \frac{w_j}{w_{max}}$$

Insérer l'évènement dans l'échantillon revient à générer aléatoirement un réel u_j entre 0 et 1. L'évènement r_j est accepté si $u_j < p_j$. Si l'évènement r_j est rejeté, on répète le processus jusqu'à ce qu'un évènement soit accepté.

Cette technique ne peut pas être appliquée dans le cadre des flux de données étant donné qu'aucune politique de remplacement n'a été prise en compte. Une amélioration

de cette technique a été présentée dans [73] par Efraimidis et al. L'idée est d'associer un échantillonnage réservoir dans le cas où la taille de la population est inconnue (exemple des flux de données).

Échantillonnage sur fenêtres glissantes. Dans le cadre d'un échantillonnage sur fenêtre glissante, la principale difficulté à laquelle les algorithmes doivent faire face est le maintien d'un échantillon représentatif de la fenêtre. En effet, les évènements qui expirent de la fenêtre doivent être remplacés dans l'échantillon. Plusieurs techniques ont été développées, la plupart se basent sur un échantillonnage de type réservoir. Dans le cas des fenêtres logiques, Babcock et al ont introduit dans [24] plusieurs algorithmes permettant de maintenir un échantillon de taille k sur une fenêtre glissante de taille n.

Échantillonnage périodique. L'idée consiste à maintenir un échantillonnage réservoir sur les k premiers évènements du flux. Lorsqu'un évènement de cet échantillon expire, il est remplacé par le nouvel évènement qui arrive. Cette approche a l'avantage d'être efficace en terme de mémoire (k évènements dans l'espace mémoire à chaque étape). Elle permet de maintenir un échantillon aléatoire uniforme sur une fenêtre glissante de taille n. Cependant, cette méthode souffre d'un défaut de périodicité. Si l'évènement i est inclus dans l'échantillon, l'évènement $i + cn$ sera aussi inclus pour tout entier $c > 0$, ce qui n'est pas acceptable pour un certain nombre d'applications. Pour pallier ce défaut, la technique de l'échantillonnage avec réserve peut être utilisée.

Échantillonnage avec réserve. Dans cet algorithme, on dispose d'un échantillon dit "de réserve" (ou "backing sample"). Chaque évènement du flux est ajouté à cet échantillon avec une probabilité $(2ck\ log\ n)/n$. On génère ensuite un échantillon aléatoire à partir de l'échantillon de réserve. Lorsqu'un évènement expire, on le supprime de la réserve. Des arguments sur les bornes de Chernoff garantissent que la taille de la réserve reste en $O(k\ log\ n)$ pour assurer que la taille de l'échantillon reste entre k et $4ck\ log\ n$.

Échantillonnage chaîné. Une autre technique pouvant s'appliquer au cas des fenêtres logiques est l'échantillonnage en chaîne. Cette approche permet de maintenir une chaîne de remplaçants pour chaque évènement i dans l'échantillon. Au total, on maintient k chaînes de remplaçant pour un échantillon de taille k. Chaque évènement i est ajouté à l'échantillon avec une probabilité $\frac{1}{min(i,n)}$ (avec i : indice de l'évènement courant et n : taille de la fenêtre). Comme l'évènement d'indice i est ajouté à l'échantillon, il faut choisir l'indice de son remplaçant. Cet indice est sélectionné aléatoirement dans l'intervalle $[i + 1, i + n]$ selon une loi uniforme. Lorsque l'évènement de cet indice arrive, il est ajouté à l'échantillon et on sélectionne son remplaçant, de la même façon que précédemment. On construit ainsi, une chaîne de remplaçants potentiels. Quand un évènement est éliminé de l'échantillon, la chaîne de ses remplaçants est aussi supprimée. Cette méthode a l'avantage

de garantir une taille mémoire en $O(k)$.

Échantillonnage par priorité. Les approches présentées précédemment exigent une connaissance à priori du nombre d'évènements dans la fenêtre. Par conséquent, ces méthodes ne peuvent être appliquées aux fenêtres physiques. Un échantillonnage par priorité [24], très similaire à l'échantillonnage en chaîne permet de contourner ce problème. L'idée est d'associer aléatoirement à chaque évènement i de l'échantillon une priorité p_i ($p_i \in [0, 1]$) ainsi qu'une chaîne de remplaçants. Les k évènements encore valides (n'ayant pas encore expiré) et ayant la priorité la plus élevée font partie de l'échantillon de taille k. On choisit par la suite la liste des candidats. Pour cela, on stocke pour chaque priorité, les évènements ayant à la fois une priorité plus élevée et une étiquette de temps plus récente que tous les évènements stockés. Cette liste est ordonnée par étiquette temporelle puis par priorité. Le tableau 2.1 illustre les complexités des méthodes présentées par Babcock [24].

Échantillonnage	Complexité spatiale	Fenêtre
Périodique	$O(k)$	Logique
Avec réserve	$O(k\ log\ n)$	Logique
Chaîné	$O(k)$	Logique
Par priorité	$O(k\ log\ n)$	Physique

TABLE 2.1 – Complexités des méthodes (k taille de l'échantillon, n taille de la fenêtre glissante).

Échantillonnage par couplage aléatoire. Dans [84], Gemulla et al présentent l'algorithme RP (*Random Pairing*) qui permet de maintenir un échantillon uniforme sur fenêtres glissantes. Leur approche combine l'algorithme d'échantillonnage réservoir de Vitter et l'algorithme d'échantillonnage sur fenêtres glissantes de Babcock. L'algorithme opère sur n'importe quelle structure S_c qui tolère les insertions et les suppressions mais dont la taille reste fixe. Chaque suppression dans S_c engendre une suppression dans l'échantillon associé E. L'algorithme maintient deux mesures : c_1 et c_2. La première mesure permet d'enregistrer le nombre de suppressions effectuées sur des évènements de S_c qui existent dans l'échantillon E. La deuxième mesure enregistre le nombre de suppressions dans S_c des évènements qui n'existent pas dans l'échantillon E. La suppression de ces évènements n'aura aucun impact sur E. La stratégie d'insertion des évènements dans E dépend du nombre de suppressions réalisées sur S_c. Si aucune suppression n'a été observée, l'insertion de l'évènement dans E suit le principe d'insertion dans un échantillon réservoir. Sinon, l'évènement est inséré dans E avec une probabilité d'inclusion égale à $\frac{c_1}{c_1+c_2}$.

Cet algorithme permet de garder un échantillon uniforme sur une structure de taille fixe. Pour les structures évolutives (où la taille est variable), les auteurs proposent un algorithme permettant le redimensionnement périodique de la taille de l'échantillon. Cette approche

transforme l'échantillon uniforme E en un échantillon de Bernouilli pour augmenter la taille de E.

Échantillonnage stratifié. Dans [83], Gemulla et al présentent une technique permettant de maintenir un échantillon à partir d'une fenêtre glissante physique. Il s'agit de partitionner une fenêtre W en l strates temporelles ($l > 1$). Un échantillon uniforme E_i est maintenu sur chacune des strates (avec $1 < i < l$). Chaque échantillon possède une taille fixe n. En plus de ces échantillons, l'algorithme maintient, pour chaque strate, sa taille N_i ainsi que l'estampille temporelle de son évènement le plus récent. Ces deux mesures sont importantes pour le maintien des échantillons.
Le principal défi de cet échantillonnage est le partitionnement des frontières des strates temporelles. Ce partitionnement a un impact important sur la qualité de l'échantillon généré. Dans le cas le plus simple, la fenêtre W est divisée en strates selon un partitionnement temporel équivalent (*Equi-width stratification*). Une seconde méthode revient à diviser la fenêtre en strates de tailles égales (en terme de nombre d'évènements). Cette approche est appelée stratification équi-distante (*Equi-depth stratification*).
Une parfaite stratification équi-distante est impossible à réaliser. Dans [83] les auteurs présentent un algorithme basé sur la fusion des strates pour se rapprocher le plus possible d'une stratification équi-distante. L'idée revient à fusionner les deux strates voisines ce qui réduit l'information mémorisée dans l'échantillon représentatif de ces deux strates (qui lui reste de taille fixe). Ceci crée un espace libre à la fin de l'échantillon. Cet espace peut être utilisé pour d'autres évènements.

Échantillonnage de jointure. Dans le cadre d'applications réelles (e.g. réseaux de capteurs), il est fréquent de traiter de multiples flux de données qui partagent des informations de nature relationnelle permettant de les relier entre eux. Des jointures sont alors réalisées sur ces flux afin de produire un flux unique. C'est par exemple le cas dans le domaine de la météorologie où on est amené à joindre les différents flux produits par les stations météo sur l'attribut température. Dans le contexte des flux de données, l'opération de jointure s'avère complexe. En effet, les exigences liées aux opérateurs dits "bloquants" ajoutent de nouvelles contraintes : pour chaque nouvel élément dans le flux F_1, l'intégralité du flux F_2 doit être présent. Or, les données de ces flux sont éphémères, ce qui rend impossible ce procédé.

Dans [47], Chaudhuri et al présentent une méthode d'échantillonnage biaisée pour une opération de jointure entre deux relations R_1 et R_2. Cette méthode procède en deux phases. La première consiste à utiliser un échantillonnage réservoir pondéré afin de produire un échantillon biaisé E_1 à partir de la relation R_1. E_1 est biaisé sur la fréquence de la clé de jointure de chaque élément $e_2(i)$ dans R_2. La deuxième phase consiste à joindre les éléments $e_1(i)$ de l'échantillon E_1 avec un élément $e_2(i)$ tiré aléatoirement tel que $e_2(i)$.clé $= e_1(i)$.clé. L'avantage de cet algorithme est que chaque élément de l'échantillon soit tiré en une seule passe. Cependant, pour pouvoir l'utiliser, il faut avoir l'ensemble des fréquences

de la clé de jointure dans R_2. Ce qui limite son utilisation à des opérations de jointure entre flux et relations.

Dans [63], Das et al présentent une approche de jointure entre deux flux de données. L'algorithme utilise trois réservoirs : r_1 et r_2 pour les flux F_1 et F_2 et r_3 pour le résultat de la jointure. Quand un évènement i arrive sur F_1, on effectue une jointure avec les évènements inclus dans r_2. Si le résultat de la jointure n'est pas vide, il est inséré dans r_3. Des estimations sont ensuite réalisées sur le réservoir r_3 afin de biaiser les tirages et améliorer la qualité des échantillons extraits de F_1 et F_2.

Dans le contexte des bases de données, Haas et al présentent dans [102] l'algorithme Ripple Join. L'idée consiste à produire, à n'importe quel instant, une estimation d'une requête d'agrégat d'un utilisateur en fournissant un intervalle de confiance. L'utilisateur peut ainsi arrêter la requête quand il estime que le résultat est suffisamment correct. Cette approche a été développée pour des données statiques. Son utilisation dans le cadre des flux de données nécessite des extensions étant donné que les ressources mémoire et temporelles nécessaires ne peuvent pas être connues par avance.

Dans [79], Féraud et al étendent le modèle de l'algorithme Ripple Join pour la jointure de deux flux. Ils utilisent un double échantillonnage à réservoir pondéré pour réaliser la jointure de F_1 et F_2. Les auteurs abordent cette jointure comme un problème d'estimation. Pour cela, il comparent plusieurs estimateurs afin obtenir un résultat de taille importante avec une faible variance et respectant les contraintes sur les ressources. Pour cela, les auteurs comparent quatre estimateurs basés sur l'échantillonnage réservoir (simple, déterministe, actif et pondéré). Les expérimentations ont montré que l'estimateur basé sur la méthode d'échantillonnage réservoir simple n'est pas robuste (forte variance et résultat de jointure petit). Cependant, avec l'utilisation d'un estimateur d'échantillonnage réservoir pondéré, une faible variance avec un résultat de jointure large soient observés. L'estimateur basé sur un échantillonnage réservoir déterministe présente une variance faible comparé à l'échantillonnage pondéré avec un large résultat de jointure pour une utilisation limitée des ressources. L'estimateur basé sur un échantillonnage réservoir actif ne pourra pas être utilisé étant donné qu'il nécessite $O(n^3)$ en temps.

2.3.1.2 Histogrammes

Une autre façon permettant de résumer un flux de données consiste à utiliser des histogrammes. Cette structure est souvent utilisée pour résumer des données qualitatives ou quantitatives.

Un histogramme sépare la population en un ensemble de groupes ou classes suivant des attributs particuliers. Pour chaque groupe, on conserve ses fréquences. Il est relativement facile d'utiliser l'histogramme pour des variables quantitatives pour répondre à différents types de requêtes telles que les requêtes portant sur un intervalle. Il suffit de déterminer l'ensemble des segments inclus dans l'intervalle spécifié par l'utilisateur.

Dans le domaine des flux de données, cette approche devrait permettre une meilleure représentation d'un échantillon issu des flux. Plusieurs types d'histogrammes sont proposés dans la littérature, voici une liste non exhaustive de quelques approches. Dans tout ce qui suit, nous considérons n le nombre total d'évènements et B le nombre d'intervalles (groupes) dans le histogramme.

Histogramme Equi-Profondeur et Quantiles. Les histogrammes Equi-Profondeur sont des histogrammes simples mais présentent deux caractéristiques principales :

- Le nombre d'évènements dans le groupe (backet) est le même(cf. Figure 2.1) ;
- Les attributs présentent des valeurs contiguës et continues.

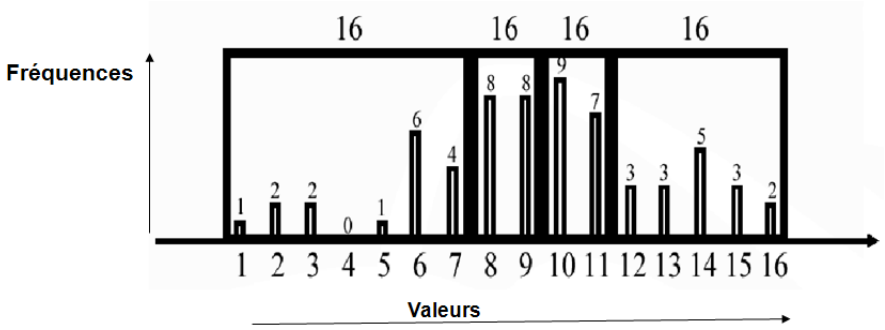

FIGURE 2.1 – Histogramme Equi-profondeur

Greenwald et Khanna ont présenté dans [92] un algorithme déterministe à une seule passe pour calculer les quantiles d'une manière efficace. Leur algorithme nécessite $O(1/\epsilon\ Log\ \epsilon n)$ espace et garantit une erreur égale à ϵn. Ils maintiennent pour cela un échantillon des rangs possibles pour chaque quantile. L'erreur associée à chaque quantile représente la largeur de la classe. Par la suite, ils fusionnent les quantiles avec des erreurs "similaires" tant que l'erreur du quantile ne dépasse pas ϵn. Cet algorithme améliore les précédents travaux [136] [137] [46].

Dans [96], Guha et McGregor présentent un algorithme à une seule passe permettant de déterminer les B quantiles d'un échantillon E de taille n maintenu aléatoirement. L'algorithme utilise $O(Blogn)$ en espace mémoire et avec une erreur de $O(n^{-1/2})$. Cette erreur tend vers zéro quand l'échantillon augmente de taille.

Histogramme V-Optimal. L'objectif des méthodes d'approximation par histogramme est de construire un histogramme qui contient au plus B intervalles et qui minimise l'erreur d'approximation. L'une des mesures d'erreur les plus utilisées est $\sum(x_i - \hat{x_i})^2$ avec (x_i) le résultat réél d'une requête d'agrégat et $(\hat{x_i})$ son approximation. Cette mesure est aussi appelée mesure V-optimal.

Plusieurs travaux ont présenté des optimisations de la mesure de l'erreur. On retrouve ainsi : l'erreur relative [99] [81] , la range query [89] [95] [122], etc. Chacune de ces erreurs est appliquée dans un domaine particulier.

Dans [109], Jogadish et al ont présenté un algorithme pour construire le meilleur histogramme V-Optimal avec une complexité temporelle [18] $O(n^2B)$ et une complexité spatiale [19] $O(B)$. Cet algorithme est basé sur une programmation dynamique généralisée pour une grande variété de mesures d'erreur. Cependant, le temps d'exécution quadratique ne convient pas dans le contexte des flux de données. Les auteurs ont également présenté un algorithme d'approximation qui a une complexité temporelle de $O(n^2B)$ et utilise (B) intervalles tout en garantissant la qualité des intervalles construits. Cependant, l'aspect quadratique existe toujours dans le temps d'exécution.

Le premier algorithme d'approximation par histogramme pour les flux de donnée a été présenté dans [93] par Guha et al. Cet algorithme suppose que le flux de données est déjà trié sur l'attribut d'intérêt. Une approximation à $(1 + \epsilon)$ près de l'histogramme optimal peut être calculée en utilisant $O(\epsilon^{-1}B^2log_n)$ espace mémoire et $O(\epsilon^{-1}B^2log_n)$ unité de temps de traitement par évènement du flux.

Dans [86], Gilbert et al ont levé l'hypothèse que le flux doit être trié. Ils ont utilisé une méthode basée sur les sketchs pour calculer la norme L_2. L'idée est de considérer chaque évènement du flux comme étant une mise à jour d'un vecteur de taille n. L'objectif est d'approximer ce vecteur en utilisant le meilleur histogramme B-intervalles. Le temps de traitement d'un évènement du flux, le temps de reconstruction de l'histogramme ainsi que la taille du sketch sont respectivement limités à $poly(B, logn, 1/\epsilon)$ avec ϵ l'erreur relative tolérée.
Le tableau 2.2(extrait de [94]) résume les différentes complexités des algorithmes de construction d'un histogramme V-optimal.

Papier	Complexité Temporelle	Complexité Spatiale	Flux trié
Jagadish et al [109]	$O(n^2B)$	$O(nB)$	Oui
Guha et al [93]	$O(nB\tau)$	$O(B\tau)$	Oui
Gilbert et al [87]	$n\tau^{O(1)}$	$\tau^{O(1)}$	Non

FIGURE 2.2 – Algorithme d'approximation par histogramme

Histogrammes Fin-Biaisée et requêtes Iceberg. Plusieurs applications maintiennent des agrégats simples (tel que le COUNT) dans un attribut afin de détecter des

18. La complexité temporelle permet d' évaluer quel va être la rapidité d'exécution d'un algorithme en fonction d'un ou de plusieurs paramètres dépendant des données en entrée.

19. La complexité spatiale permet d'évaluer l'occupation mémoire que va nécessiter un algorithme en fonction de certains paramètres

valeurs d'agrégats au-dessus d'un certain seuil. Ces requêtes sont dénommées requêtes iceberg [76]. Elles sont utilisées dans différents domaines tels que l'extraction de données, la fouille de données, la détection de copies, et le regroupement. Par exemple, un moteur de recherche pourrait être intéressé par la collecte des termes de recherche qui figurent dans plus de 1% des requêtes. D'autre part, les évènements fréquents sont largement utilisés pour l'analyse des tendances.

Fang et al ont présenté dans [76] un algorithme efficace pour calculer les résultats des requêtes Iceberg sur des données stockées sur disque. Leur algorithme nécessite plusieurs passes, ce qui est incompatible avec les flux de données.

Manku et Motwani ont présenté dans [135] un algorithme aléatoire et un autre déterministe afin de calculer la fréquence dans les requêtes iceberg sur des flux de données. L'algorithme aléatoire utilise l'échantillonnage adaptatif dont le principe est que tout évènement qui représente une fraction ϵ des évènements fera probablement parti de l'échantillon uniforme de taille $1/\epsilon$. L'algorithme déterministe maintient un échantillon des évènements distincts avec leur fréquence. Cet algorithme nécessite $O(1/\epsilon\ Log\ n)$ espace.

2.3.1.3 Sketchs

Les sketchs sont des structures de données compactes et de petite taille. Ils sont utilisés pour des tâches bien spécifiques tel que l'estimation du nombre d'évènements distincts dans une liste. Les sketchs sont apparus bien avant l'apparition des flux de données. Mais, étant donné leur rapidité de traitement, ils peuvent être utilisés dans le contexte des flux. Ils constituent alors un résumé du flux de données très compact permettant d'estimer les réponses à certaines requêtes. Nous présentons dans cette section quelques types de sketches qui sont utilisés dans le cadre des flux de données.

Flajolet Martin. Le sketch Flajolet Martin (FM) [78] permet d'estimer le nombre de valeurs distinctes dans une série d'évènements. Cette méthode a été développée en 1985 pour résumer une masse importante de données dans les grandes bases de données. En raison de ses principales caractéristiques (incrémental et rapide), il est parfaitement adapté au contexte des flux de données.
Le sketch FM est composé d'un mot binaire de taille L, initialisé à 0 et d'une fonction de hachage $h(x)$ qui répartit uniformément les évènements d'un ensemble M dans l'ensemble des mots binaires de taille L. A chaque nouvelle insertion, l'évènement est tout d'abord traité par la fonction de hachage qui génère l'indice i de l'évènement dans le tableau binaire. Pour estimer le nombre d'évènements de l'ensemble, il suffit de sélectionner l'indice i du zéro de plus faible poids dans L. On a alors $E(i) = log_2(\Phi n)$ avec $\Phi = 0.77351$. Les détails liés à l'obtention de ces bornes sont donnés dans [78].

L'inconvénient de cette approche est qu'elle ne présente de bons résultats que pour un seul agrégat (DISTINCT COUNT). De plus, elle fournit un estimateur avec une forte

variance pour cet agrégat. Cependant, plusieurs approches ont été proposées pour réduire cette variance, parmi elles nous citons [72].

Count Min. Le sketch Count Min (CM) a été introduit dans [55] par Cormode et Muthukrishnan. Ce sketch a été spécialement conçu pour les flux de données. Il se base sur le modèle de la caisse enregistreuse où un flux F représente les variations d'un signal sous jacent s de structure $(s_1, s_2, ..., s_n)$ (avec $s_i > 0$).
On note e la base de la fonction logarithmique ln. Un sketch CM est représenté par un tableau T(count) à deux dimensions de taille $w = [\frac{e}{\epsilon}]$ et de profondeur $d = [ln\frac{1}{\lambda}]$ $(count[1,1], ..., count[d,w])$. La structure de ce sketch est définie par deux paramètres (ϵ, λ) qui sont respectivement la précision et la probabilité d'échec. Il est basé sur une solution de type PAC (Probably Approximately Correct) [20][133].
Chaque entrée dans le tableau est initialisée à zéro. De plus, on choisit d fonctions de hachage tel que :

$$h_1...h_d : 1...n \rightarrow 1...w$$

Étant donné que ce sketch suit le modèle de la caisse enregistreuse, lorsqu'un évènement $(i(t), c(t))$ arrive, cela revient à dire que l'évènement d'indice i arrivant à l'instant t est mis à jour par la valeur $c(t)$. Cette valeur $(c(t))$ est insérée dans une case de chacune des lignes de la matrice T. La case choisie est désignée grâce à la fonction de hachage h_j associée à la ligne j $(1 \leq j \leq d)$:

$$count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t$$

L'objectif du Count Min est de pourvoir répondre à des requêtes de types :

- *Point Query* : $Q(i) = s_i$
- *Range Query* : $Q(l, r) = \sum_{i=l}^{r} s_i$ (avec l et r les bornes inférieure et supérieure de la requête)
- *Inner product Query* : $Q(\overrightarrow{s}, \overrightarrow{p}) = \sum_{i=l}^{n} s_i p_i$.

Les requêtes de type "Range Query" et "Inner product Query" sont des extensions de la requête "Point Query". La réponse à la requête $Q(i)$ est donnée par :

$$\widehat{s}_i = min_j count[j, h_j(i)]$$

On a la garantie que, d'une part, $s_i \leq \widehat{s}_i$ et d'autre part, qu'avec une probabilité $1 - \lambda$: $\widehat{s}_i \leq s_i + \epsilon ||s||_1$.

20. PAC est une plateforme d'analyse mathématique dans laquelle le système reçoit un ensemble d'échantillons et doit sélectionner une fonction de généralisation (appelée l'hypothèse) à partir d'une certaine classe de fonctions possibles. L'objectif est que la fonction sélectionnée génère, avec une forte probabilité, un faible taux d'erreur

Sketch Count Le sketch Count permet de répondre aux requêtes dites "$top-k$". Cela revient à déterminer les k évènements les plus fréquents dans une série de valeurs.
L'algorithme reçoit en entrée, un flux F, un entier k et un réel ϵ représentant la précision souhaitée. L'algorithme fournit une liste de k évènements de F, tel que chaque évènement i respecte la contrainte suivante :

$$n_i > (1-\epsilon)n_k$$

avec n_i la fréquence d'un évènement d'indice i dans la liste des évènements classés par fréquence.

Soit F un flux de données composé d'un ensemble d'évènements $\{s_1, ..., s_n\}$. Un sketch COUNT est représenté sous la forme d'un tableau bi-dimensionnel $(w \times d)$, d'un ensemble de fonctions de hashage $h_1, ..., h_w$ tel que $h : [F] \longrightarrow [1, d]$, et d'un ensemble de fonctions de hashage $p_1, ..., p_w$ tel que $p : [F] \longrightarrow \{-1, +1\}$. Le sketch COUNT maintient en parallèle une liste FI initialement vide contenant les évènements les plus fréquents. A l'arrivée d'un évènement s_i du flux, l'algorithme effectue deux opérations :

- Une case de chaque ligne du tableau est incrémentée ou décrémentée de 1 suivant l'opération suivante : $h_j[s_i] += p_j[s_i]$ $(1 \leq j \leq w)$;
- Si $s_i \in FI$ alors incrémenter son compteur de 1, sinon l'ajouter à FI. On estime la fréquence de s_i en évaluant la valeur de $median_j\{h_j[s_i].p_j[s_i]\}$. Soit S_m, l'évènement de FI ayant la plus petite fréquence et C_m sa fréquence. Si $median_j\{h_j[s_i].p_j[s_i]\} > C_m$, alors éliminer S_m de FI.

Filtre de Bloom Un Filtre de Bloom est un outil compact introduit par Burton H. Bloom dans [31]. Cette structure permet de mémoriser dans un espace fini un ensemble d'évènements. L'intérêt d'une telle structure est de vérifier rapidement si un évènement a déjà été inséré ou non dans le filtre.
Un filtre de Bloom est composé d'un ensemble d'évènements $F = \{s_1, s_2, ..., s_n\}$ à insérer, d'un tableau binaire de taille w et de d fonctions de hachage $h_1, ..., h_d$ tel que $h : [F] \longrightarrow [1, w]$. Toutes les cellules du tableau binaire sont initialisées à zéro.
Pour insérer un nouvel évènement s_i dans le filtre, l'algorithme opère en deux phases :

- On applique les fonctions de hachage sur l'évènement à insérer ;
- Tous les indices retournés par $h_j(s_i)$ (avec $j = 1..d$) sont mis à 1 dans le tableau binaire.

Pour tester l'existence d'un évènement dans le filtre, il suffit de lui appliquer les fonctions de hachage et de vérifier si dans chacune des positions résultantes, la valeur 1 est associée. Si tel est le cas, on considère que l'évènement appartient à l'ensemble, sinon, on est sûr que la valuer n'a été vue.
Avec une telle structure, des problèmes de collision peuvent surgir. Une collision a lieu lorsqu'un évènement est identifié à tort comme faisant partie de l'ensemble des évènements appris. Le résultat d'une interrogation peut ainsi être faussé. La probabilité d'erreur (faux

positif) est calculée comme suit :

$$\left(1 - \left(1 - \frac{1}{w}\right)^{dn}\right)^d \approx \left(1 - e^{\frac{dn}{w}}\right)^d$$

On peut contrôler la probabilité de collision, en respectant la formule suivante pour le choix des paramètres du filtre de Bloom :

$$d = \ln 2 \times w/n$$

Plusieurs extensions ont été développées autour des filtres de Bloom. Almeida et al ont introduit dans [17] le *"scalable Bloom filter"* qui permet de construire des filtres de tailles variables. De plus, dans [75], Fan et al présentent le *"counting Bloom filter"* afin de gérer la suppression dans ces filtres. Cependant, avec toutes ces extensions la construction du filtre n'est plus aussi rapide que la première version.

Le tableau ci-dessous illustre les complexités des différents sketchs (avec M = taille du sketch, d = nombre de fonctions de hachage).

Sketch	Complexité Temporelle	Temps de mise à jour	Temps de réponse
FM	$O(M)$	M	M
Count-Min	$O(1/\epsilon log 1/\delta)$	1	1
Count	$O(klogn/\delta)$	d	d
Filtre de Bloom	$O(M)$	d	d

FIGURE 2.3 – Tableau des complexités

Rôle des sketchs dans les flux de données distribués. Il existe une multitude d'applications qui font appel aux flux distribués tel que la gestion et la supervision du trafic dans un réseau informatique. Dans ces applications, répondre à des requêtes d'agrégats appliquées sur ces réseaux revient à transmettre et combiner tous les flux de données à un point d'agrégation (i.e. pour avoir l'union des flux) et de répondre localement aux requêtes. Une telle solution souffre de plusieurs problèmes tels que : coût de maintenance, contraintes mémoire dues au taux d'arrivée des flux, goulot d'étranglement causé par la transmission de toutes les données vers un seul point. Pour pallier ces inconvénients, une solution revient à utiliser un sketch initial pour chaque flux de données. Ces sketchs sont ensuite acheminés vers un ou plusieurs points d'agrégation. Plusieurs travaux se sont focalisés sur l'utilisation des sketchs dans les flux de données distribués [155][179].

2.3.2 Conception complexe d'un résumé de flux de données

Dans ces approches, des techniques de fenêtrage, donnant la possibilité d'archiver des résumés couvrant l'intégralité du flux avec un espace borné, sont utilisées afin de produire

des résumés dits généralistes.

D'après [60], un résumé généraliste doit remplir quatre conditions : traiter un large champ de requêtes (e.g., sélection, médiane), permettre des analyses supervisées des données (e.g., arbres de décision), autoriser des tâches d'analyse exploratoire (e.g.,classification) et prendre en compte la dimension temporelle. Ce qui revient à définir ce résumé selon deux aspects : l'aspect fonctionnel (réaliser des analyses variées) et l'aspect conceptuel (combiner une organisation spatiale avec une organisation temporelle).

Selon [65], un résumé de flux de données est dit généraliste s'il respecte les quatre conditions suivantes :

1. Respecter des contraintes d'espace-mémoire et de puissance de calcul ;
2. S'exprimer sous forme de variables appartenant à TxD (T : espace des estampilles temporelles, D : espace de description des évènements du flux (organisation spatiale)) ;
3. Permettre une approximation de toute requête de type SELECT et COUNT sur TxD. Ceci revient à approximer la densité jointe sur l'espace TxD ;
4. Permettre le calcul de l'erreur d'approximation en fonction des ressources disponibles.

2.3.2.1 Organisation temporelle complexe

Il s'agit d'une organisation particulière qui gère l'historique en pondérant le niveau de détail par l'âge des données. Ces fenêtres ont une longueur variable qui croît avec le temps, de sorte que les évènements les plus récents sont observés dans des fenêtres plus petites et les évènements les plus anciens sont observés dans des fenêtres plus grandes. L'idée sous-jacente est que les évènements les plus récents soient décrits de façon plus précise.
Dans [114], Han et al définissent deux techniques permettant l'implantation de ces structures. On distingue alors les fenêtres inclinées et le système de clichés.

Système de fenêtres inclinées (*Tilted Time Frame*). Les fenêtres inclinées sont des structures qui permettent de modéliser et compresser la dimension temporelle. Leur modèle s'inspire fortement du mécanisme d'oubli de la mémoire humaine : il permet de stocker avec une précision maximale les données les plus récentes et, avec le temps, ces données sont de plus en plus oubliées.
Formellement, soit $T = [t_1, ..., t_k]$ une fenêtre inclinée de taille k avec t_1 le temps courant. Chaque intervalle $[t_i, t_{i+1}]$ (avec $1 \leq i < k$) inclus dans T représente une fenêtre avec un niveau de précision.
Dans [85][50], deux implémentations du système de fenêtres inclinées sont présentées : le modèle des fenêtres temporelles naturelles et, le modèle logarithmique. Ces deux modèles

se distinguent par leur découpage du temps. Les niveaux de granularité sont précisés par les applications.

Modèle naturel de fenêtres inclinées. Dans ce modèle, la période est structurée en plusieurs granularités basées sur une échelle naturelle du temps. Comme le montre la Figure 2.4, on stocke pendant une heure les informations à la minute. Ensuite pendant une journée les informations sont stockées à l'heure, etc. Ce modèle permet ainsi une réduction importante du nombre de valeurs conservées. Dans ce modèle, pour conserver les données d'une année, on a besoin de 117 valeurs $(60 + 24 + 30 + 12)$ au lieu de 525600 valeurs $(365 * 24 * 60)$.

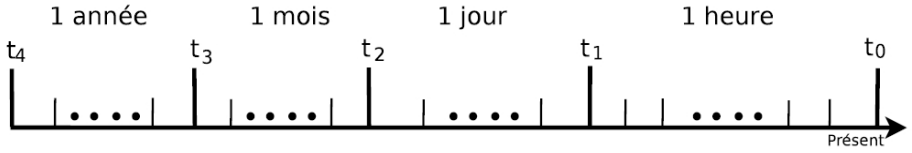

FIGURE 2.4 – Modèle naturel de fenêtres inclinées

Modèle logarithmique des fenêtres inclinées. Dans ce modèle la période est structurée en plusieurs granularités basées sur une échelle logarithmique. Dans le cas d'un log en base 2 (cf. Figure 2.5) pour des données d'une année et avec une précision de 15 minutes, nous avons besoin de : $Log_2(365 \times 24 \times 4) + 1 \simeq 17$ unités de temps au lieu de $365 \times 24 \times 4 = 35136$ unités.

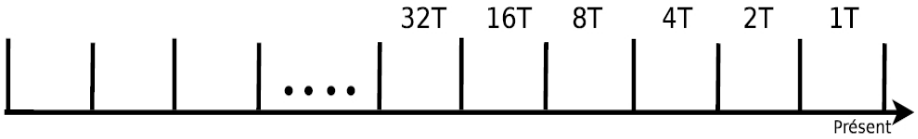

FIGURE 2.5 – Modèle logarithmique de fenêtres inclinées

Système de clichés (*Progress Logarithmic Tilted Time Frame Window*). Pour décrire ce modèle, il faut tout d'abord définir une structure appelée *cliché (snapshot)*. Un cliché est une image qui décrit l'état du système à un instant donné. Par exemple, l'ensemble des tuples d'une table à un instant t est un cliché de cette table.
Dans ce modèle, on mémorise des clichés à des intervalles de temps variables. Ces clichés sont représentés par leurs estampilles temporelles. Il existe deux règles à respecter afin de conserver un cliché à l'instant t :

– Règle 1 : si $(t\ mod\ 2^i) = 0$ et $(t\ mod\ 2^{i+1}) \neq 0$, alors le cliché est inséré dans la liste des clichés d'ordre i. Si $(i \succ ordre\_max)$ alors il est inséré dans la liste des clichés d'ordre $ordre\_max$.
– Règle 2 : le nombre de clichés d'un ordre i ne doit pas dépasser une capacité maximale de stockage intitulé $max\_capacity$. Si au moment de l'insertion du cliché dans la liste, le nombre des clichés de cet ordre est égal à $max\_capacity$, alors on supprime le plus ancien cliché de cette liste et on insère le nouveau.

Ordre	Clichés
0	69 67 65
1	70 66 62
2	68 60 52
3	56 40 24
4	48 16
5	64 32

FIGURE 2.6 – Système de clichés

Prenons l'exemple de la Figure 2.6. Pour un cliché capturé à l'instant $t = 70$, on a $(70\ mod\ 2^1 = 0)$ et $(70\ mod\ 2^2 \neq 0)$. Ainsi le cliché sera inséré dans la liste des clichés d'ordre 1. Supposons que le plus ancien cliché de cet ordre a été capturé à $t = 58$ et supposons que la liste des clichés soit déjà pleine, alors, le cliché capturé à $t = 58$ sera supprimé. Avec cette technique l'ordre maximal des clichés sauvegardés ($ordre\_max$) vérifie l'inégalité suivante : $log_2 t - max\_capacity \leq ordre\_max \leq log_2 t$.
L'avantage de cette méthode est qu'elle permet, si la structure de données utilisée supporte la soustraction [21], de générer un résumé d'une période quelconque de l'histoire du flux.

Remarque : dans le modèle logarithmique des fenêtres inclinées et le modèle du système de clichés, nous avons pris en considération que la base logarithmique est 2. Cependant, les mêmes règles sont appliquées si on choisit une autre base α, avec $\alpha \in N$ et $\alpha \succ 1$.

2.3.2.2 Algorithmes de résumés généralistes

Les avantages des algorithmes de résumés généralistes sont multiples : ils permettent de prendre en compte l'évolution des données ainsi que l'exploration des données sur différentes parties du flux. Récemment plusieurs algorithmes de résumés généralistes sont apparus, certains se basent sur des techniques d'échantillonnage (e.g. StreamSamp [61]), ou sur un système de clustering (e.g. CluStream [15]) pour résumer les données d'un flux, d'autres utilisent une structure d'automate pour l'indexation des motifs fréquents du flux

21. La soustraction de deux clichés pris à l'instant t_1 et t_2 définit l'état du système sur la période $[t_1,t_2]$.

de données [163][167] :
Nous nous intéressons dans cette thèse aux algorithmes de résumés basés sur l'échantillonnage et le clustering des données.

StreamSamp. C'est un algorithme de résumé généraliste basé sur l'échantillonnage de flux de données. Développé par Csernel et al [61], StreamSamp combine à la fois une approche spatiale qui est l'échantillonnage aléatoire et une approche temporelle qui est le système de fenêtres inclinées.

Principe des fenêtres inclinées dans StreamSamp. Pour gérer l'historique des connaissances extraites à partir du flux avec une granularité plus fine pour les données récentes, StreamSamp s'appuie sur le modèle logarithmique des fenêtres inclinées. En adaptant cette représentation, il se base sur un plan d'échantillonnage du flux pour biaiser l'échantillon en fonction du temps. L'échantillon est constitué d'une série de fenêtres couvrant des périodes de temps différentes : plus l'échantillon est ancien, plus il couvre une période de temps importante. Comme le montre la Figure 2.7, les résumés ont une taille constante mais s'étalent sur des périodes temporelles de durées variables, plus courtes pour le présent et plus longues pour le passé lointain.
Lorsqu'on atteint L périodes de durée T, on fusionne les deux plus anciennes périodes afin de former une nouvelle de durée $2T$. L'utilisation des fenêtres inclinées ne permet donc pas de privilégier (en terme de précision) une autre période que le passé récent.

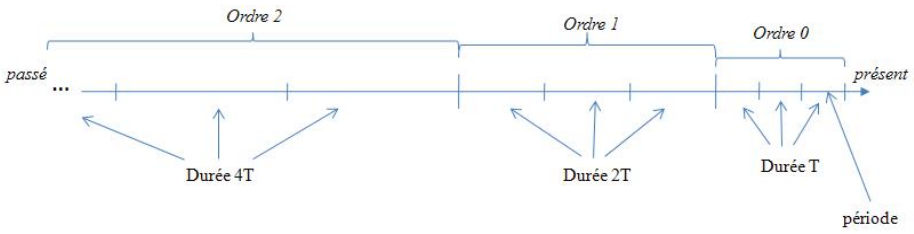

FIGURE 2.7 – Fenêtres inclinées dans StreamSamp

Principe de StreamSamp. Cet algorithme remplit trois critères : il est à la fois généraliste et rapide dans son élaboration. Son principe est basé sur deux étapes :

1. **Traitement en ligne :** cette étape permet de constituer les échantillons à partir du flux de données. Dès leur arrivée, les évènements du flux sont échantillonnées à un taux fixé α et placées dans des échantillons de taille T. Lorsque T est atteint, StreamSamp mémorise sur le disque l'échantillon ainsi que sa date de début et fin de constitution. L'ordre 0 est associé à cet échantillon. On procède par la suite à un second échantillonnage sur le flux. Plus on avance dans le temps, plus le nombre d'échantillons constitués, d'ordre 0, augmente. Comme il est impossible de conserver

à l'infini ces échantillons, StreamSamp procède à la fusion de ses échantillons suivant le principe de fenêtres inclinées (énoncé dans le paragraphe précédant). Le nouvel échantillon construit couvre la même période temporelle que ses deux échantillons parents mis bout à bout. Il est conçu par ré-échantillonnage aléatoire (sélection de $\frac{T}{2}$ évènements de chacun de ses échantillons parents).

2. **Traitement hors ligne :** cette étape permet l'exploitation du résumé conservé pour une période donnée P. On débute par constituer l'échantillon final E_f sur lequel va porter l'analyse. Pour cela, on concatène les échantillons d'ordres différents (inclus dans P) afin de reconstituer le flux de cette période. On associe par la suite un poids à chaque évènement appartenant à E_f de façon à conserver la même représentativité pour tous les évènements de cet échantillon. Pour le calcul du poids, on fait intervenir l'ordre auquel est associé l'élément comme suit : $2^{Ordre} \times \frac{1}{\alpha}$.
L'échantillon final peut ainsi être exploitable pour toute application des techniques classiques de fouille de données. Si un traitement porte sur l'intégralité des données du flux, il suffit alors de fusionner la totalité des échantillons conservés afin de reconstituer un échantillon de tout le flux. [22]

Grâce à son échantillonnage aléatoire, StreamSamp ne dépend pas du taux d'arrivée des flux. Ainsi, plus le taux d'échantillonnage augmente, plus les évènements sont pris dans l'échantillon. Des expérimentations pour valider l'efficacité et la rapidité de cet algorithme sont disponibles dans la thèse de Csernel [60] où il a effectué des comparaisons avec CluStream et l'algorithme STREAM [98] [140] un des premiers algorithmes de classification automatique de flux de données. En revanche, les performances de StreamSamp se dégradent avec l'ancienneté. En effet, les anciens évènements ont un poids exponentiellement croissant pour une taille d'échantillon constamment fixe. Par conséquent, si un échantillon contient des évènements récents (poids beaucoup plus faible) et des évènements anciens, ces derniers vont considérablement augmenter la variance dans le résultat des requêtes.

CluStream. [15] C'est un algorithme dont l'objectif affiché est la classification automatique des données numériques. Cependant, il fournit une structure particulièrement adaptée au résumé de flux de données. Cette méthode se base sur le croisement d'une approche spatiale permettant de conserver des statistiques sur des classes (cette structure ressemble beaucoup aux histogrammes) et d'une approche temporelle qui permet de gérer le résumé au cours du temps : il s'agit du système de clichés (défini dans la section 2.3.2.1).
L'objectif de Clustream consiste à construire un résumé sous forme d'un ensemble de micro-classes évolutives régulièrement mémorisées dans des clichés. Pour cela, Aggarwal s'est inspiré de l'algorithme BIRCH [182] en utilisant la structure des CFVs (Cluster Feature Vector) utilisée pour représenter une classe ou une sous-classe. Cette structure conserve

22. Plus de détails sur la constitution de l'échantillon final ainsi que l'effet de la pondération dans la qualité des échantillons conservés seront davantage détaillés dans le chapitre4

des données statistiques qui résument l'ensemble des évènements d'une micro-classe sans pour autant conserver les évènements eux même. On conserve l'effectif total de l'ensemble et pour chaque variable, la somme de toutes les valeurs prises dans l'ensemble des évènements ainsi que la somme de leurs carrés. Aggarwal étend la notion de CFV en lui intégrant l'étiquette temporelle des évènements comme une variable additionnelle. A partir de ces valeurs, on peut reconstituer pour toute variable les moments d'ordre 1 et 2 au sein de la micro-classe.
Afin de résumer des ensembles d'évènements issus du flux de données, CluStream opère en trois étapes :

1. **Etape d'initialisation :** il s'agit d'initialiser k micro-classes de départ. Pour cela, il utilise l'algorithme de classification k-means sur les évènements du début du flux. Ces micro-classes vont contenir les informations statistiques des CFVs.
2. **Etape en ligne :** il s'agit de la mise à jour des micro-classes. En effet, lorsqu'un évènement arrive, l'algorithme détermine la classe qui lui est proche (calcul de distance [23] entre l'évènement et le barycentre de chaque micro-classe déjà constituée). L'évènement est alors absorbé par cette micro-classe. Le CFV associé à la micro-classe est ainsi mis à jour en conséquence. Cependant, si aucune classe ne satisfait le critère de distance (critère fixé sur le rayon de la micro-classe), une nouvelle micro-classe est créée ne contenant que cet élément. Étant donné que la taille de la mémoire est bornée, on ne peut pas créer un nombre infini de micro-classes. Deux stratégies sont alors possibles : soit l'algorithme fusionne deux micro-classes, soit il supprime la micro-classe la plus ancienne (celle qui n'a pas été mise à jour depuis longtemps). Un critère de vieillesse est alors donné par Aggarwal, il est basé sur l'hypothèse que les étiquettes temporelles des évènements au sein de chaque micro-classe suivent une distribution gaussienne. Par ailleurs, un système de cliché est mis en place pour mémoriser l'état de toutes les micro-classes. Ces clichés sont ensuite conservés selon la structure temporelle présentée à la section 2.3.2.1.
3. **Etape hors-ligne :** dans cette étape, on utilise les propriétés mathématiques (additives et soustractives) des CFVs afin de suivre l'évolution des micro-classes. On effectue des opérations de soustractions sur les clichés afin d'obtenir le contenu du flux entre deux prises de clichés. Pour ce faire, le système se réfère aux historiques des micro-classes qui ont été conservés (historiques de fusion, d'insertion et de suppression). Cette technique permet de fournir, dans la limite du nombre de clichés pris, une version du résumé pour n'importe quelle section du flux située entre la prise de deux clichés consécutifs ou non.

En se basant sur une technique de clustering couplée avec une structure de système de clichés, CluStream permet de conserver des clichés représentatifs pour n'importe quel horizon temporel. Ceci permet de suivre l'évolution du flux de données au cours du temps.

23. La distance utilisé dans CluStream est la distance euclidienne.

Cependant, son inconvénient est le traitement relativement lourd des calculs de distance. Ce qui ne convient pas pour des flux de données très rapides. Une deuxième limite de cet algorithme concerne le nombre élevé de paramètres et dont certains dépendent de la nature même du flux de données et de la vitesse d'arrivée des évènements (e.g. le critère de vieillesse d'une micro-classe).

Une extension de cet algorithme a été présentée par Yang et al afin de traiter des données qualitatives. Il s'agit de l'algorithme HClustream [180]. Cet algorithme conserve les caractéristiques classiques des CFVs mais leur ajoute une matrice qui représente le nombre d'occurrences pour les valeurs des attributs qualitatifs présentes dans chaque micro-classe. Pour l'étape d'initialisation des micro-classes, il se base sur l'algorithme $k - prototype$. Cependant, le fait de conserver les fréquences des données qualitatives sous forme de matrice prend beaucoup d'espace de stockage et alourdit encore davantage l'algorithme.

Une deuxième extension de CluStream a été présentée par Ong et al avec l'algorithme SCLOPE [126]. Cet algorithme s'intéresse aux données binaires (chaque évènement est défini par un ensemble d'attributs dont les valeurs 0/1). Pour traiter ces données, SCLOPE utilise les "CLUSTER HISTOGRAM". Il s'agit d'une structure associée à chaque micro-classe. Cette structure comprend pour chaque attribut, la fréquence $freq(x, \mu_c)$ ainsi que le nombre d'évènements contenues dans la micro-classe. $freq(x, \mu_c)$ représente le nombre d'évènements de la micro-classe qui ont une valeur vrai pour l'attribut x. Deux mesures dérivent de ces informations à savoir :

- La largeur : le nombre d'attributs ayant des fréquences non nulles ;
- La taille $|x : freq(x, \mu_c) \succ 0|$: le nombre d'attributs de valeur "vrai" sur l'ensemble des évènements contenus dans la micro-classe ;

L'algorithme utilise les arbres FPTree[106] comme structure de stockage et d'indexation des micro-classes. Ces arbres peuvent subir des opérations d'élagage pour réduire leur volumétrie en dépit de la précision. Ces arbres sont ensuite stockés selon une structure de systèmes de clichés.

DenStream. C'est un algorithme développé par Cao et al dans [169] pour la classification automatique des données numériques. Cet algorithme hérite du fonctionnement de CluStream. Il fournit un résumé sous forme d'un ensemble de micro-classes évolutives. Contrairement à CluStream, DenStream définit deux types de micro-classes. Des micro-classes pour les évènements atypiques nommées o-micro-classe, et des micro-classes pour le reste des évènements nommées p-micro-classe. Quelque soit son type, chaque micro-classe garde trois statistiques : $\overline{CF^1}$, $\overline{CF^2}$ et w. $\overline{CF^1}$ représente la somme pondérée des évènements absorbés par la micro-classe, $\overline{CF^2}$ représente la somme quadratique pondérée des évènements et w le poids de la micro-classe représentant son nombre d'évènements.

L'algorithme commence par une phase d'initialisation des p premiers évènements du flux. Il applique l'algorithme DBSCAN[183] pour créer les premières p-micro-classes. Quand un évènement s_i arrive, l'algorithme l'affecte dans une micro-classe en respectant les étapes suivantes :

- L'algorithme essaye d'insérer s_i dans la p-micro-classe la plus proche (c_p). Si r_p, le rayon de c_p est inférieur ou égal à ϵ (seuil défini par l'utilisateur), alors on insère s_i dans c_p.
- Sinon, l'algorithme essaye d'insérer s_i dans la o-micro-classe la plus proche (c_o).
  Si r_o, le rayon de c_o est inférieur ou égale à ϵ, on insère s_i dans c_o. L'algorithme vérifie par la suite le poids de c_o. Si ce poids devient supérieur à $\beta\mu$ (défini par l'utilisateur) alors on conclut que cette o-micro-classe n'est plus considérée comme une classe d'évènements atypiques.
  Si le rayon est supérieur à ϵ, une nouvelle o-micro-classe est créée dans laquelle on insère l'élément s_i

Étant donné, qu'on ne peut conserver toutes les micro-classes, l'algorithme procède à une opération de nettoyage. Cette opération assure que le nombre de micro-classes évolue d'une manière logarithmique, ce qui est adapté au flux de données. Cependant, comme CluStream, les opérations de calcul et de mise à jour des micro-classes ne sont pas adaptées dans le cas des flux de données très rapides. De plus, étant donné, les opérations effectuées pour la mise à jour des micro-classes, cet algorithme ne permet le traitement que des flux de données numériques.

2.4 Synthèse

Les SGFD permettent d'exprimer des requêtes continues qui s'évaluent au fur et à mesure sur un flux ou sur des fenêtres. Ces requêtes doivent être spécifiées avant l'arrivée du flux. De nouveaux besoins peuvent parfois apparaître après le passage de l'information. Dans ce cas, le système ne pourra plus répondre aux requêtes posées car toutes les données n'appelant aucun traitement sont définitivement perdues. Il est donc nécessaire dans certains cas de conserver un résumé du flux de données.

Au cours de ce chapitre, nous avons défini la notion de résumé de flux et présenté les différentes approches permettant de résumer un flux de données individuel. Les résumés créés peuvent être classés selon leur fonctionnalité : certains sont destinés à une tâche particulière (sketch, etc.), d'autres permettent de réaliser des analyses variées sur les données. Le choix d'une approche n'est pas arbitraire mais doit respecter les exigences et besoins attendus (e.g. précision des résultats, structure du flux, domaine d'application) et les ressources disponibles. Une étude approfondie de ces méthodes est présentée dans le chapitre 4.

Chapitre 3

Traitement des requêtes continues

Sommaire

3.1 Introduction

Dans le contexte du World Wide Web (WWW), les internautes peuvent publier à chaque instant de nouveaux documents sur la toile. Cette autonomie dans la production et le partage de l'information est devenue importante au point qu'il est difficile de naviguer, traiter ou pister les données d'intérêt. Le problème s'accentue avec des données qui changent continuellement. C'est l'exemple de l'encyclopédie Wikipedia qui est constituée à partir des contributions de plusieurs utilisateurs. Ces derniers peuvent écrire et publier de nouveaux articles ou mettre à jour des articles existants. Dans un tel environnement, la réponse à une requête se fait manuellement. Les utilisateurs doivent fréquemment filtrer les documents afin d'extraire les données d'intérêt. Ces tâches sont pénibles et difficiles à réaliser. Pour

cela, les utilisateurs interrogent les bases de données en utilisant des requêtes "ad-hoc" [24] Ces requêtes permettent la recherche d'information dans la base de données. Cette opération consiste à créer une sous-table contenant les enregistrements répondant à certains critères et appartenant à certains champs. Ces requêtes portent à la fois sur les lignes et les colonnes d'une table, ou de plusieurs tables liées par des relations. Cependant, l'utilisation des requêtes ad-hoc dans un environnement continuellement évolutif constitue une tâche lourde pour les utilisateurs. Les requêtes continues ont ainsi été définies afin d'automatiser ces tâches.

La notion de requête continue a tout d'abord fait son apparition dans un contexte de base de données. En 1992, Terry et al [165] ont introduit cette nouvelle catégorie de requêtes afin de détecter et surveiller des changements sur les données. L'idée consiste à produire un nouveau résultat chaque fois que de nouvelles données qui concernent une requête sont modifiées. Une définition des requêtes continues est présentée dans [39] : "*Continuous queries are processed repeatedly against new data to produce new results. They are long-running and produce results continuously. Typically, continuous queries consist of relational operators such as select, project, join, and other aggregation operators.*" [25]

A première vue, les requêtes continues ont été développées pour des bases de données qui ne tolèrent que les opérations d'insertion "*Append-only database*". Des travaux ont par la suite étendu cette notion pour supporter les mises à jour des données de la base [165]. Ces requêtes sont similaires aux requêtes traditionnelles à la différence qu'elles sont définies une seule fois mais exécutées en continu quelque soit le modèle de données défini (relations ou flux de données). Elles sont écrites dans un langage proche de SQL et permettent aux utilisateurs de recevoir de nouveaux résultats. Dès leur apparition, les requêtes continues ont été traitées en se basant sur des approches incrémentales afin d'éviter les calculs redondants et de ne retourner à l'utilisateur que les nouveaux résultats. Une évolution est apparue avec le système NiagaraCQ [48] permettant d'inclure les actions et les événements temporels dans la définition de ces requêtes.

Dans ce chapitre nous comparons le traitement des requêtes continues dans un environnement de base de données au traitement de ces requêtes dans un contexte de flux de données.

24. Dans certaines solutions logicielles, les requêtes sont accessibles via des rapports, formulaires, ou via des fenêtres de requête. Les requêtes ad-hoc sont celles qui n'ont pas été déjà définies et qui ne sont pas nécessaires pour un traitement régulier sur une base. Ce sont des requêtes que l'utilisateur peut créer en dehors du champ d'application.

25. Les requêtes continues sont traitées plusieurs fois tant que les évènements continuent d'arriver afin de produire de nouveaux résultats. Ce sont des requête de longue durée produisant des résultats en continu. En règle générale, les requêtes continues se composent d'opérateurs relationnels tels que la sélection, la projection, la jointure et d'autres opérateurs d'agrégation.

3.2 Traitement de requêtes continues dans les SGBD

Dans un environnement de bases de données où les données sont continuellement mises à jour, les utilisateurs peuvent souhaiter poser une requête et être averti lorsque de nouvelles données entrent dans le système. C'est l'exemple d'une requête permettant de surveiller les transactions bancaires de l'ensemble des clients. Suite à chaque transaction, le solde des clients est mis à jour et lorsque le solde atteint un seuil négatif, une alerte est envoyée à la banque afin de prévenir le client. Ce type de requête est inscrit une seule fois dans le système et pour chaque opération de mise à jour, la requête doit être réévaluée.

Dans [159], les auteurs introduisent le système Alert qui, comme son nom l'indique, permet de notifier les utilisateurs lors d'un changement dans les données. Ce système se base sur la notion de déclencheurs (trigger)[66] appelés aussi règles ECA (Événement, Condition, Action). Les règles ECA sont définies dans les systèmes de bases de données dites actives [26]. Les utilisateurs posent des requêtes en spécifiant trois parties essentielles :

- Événement : spécifie l'événement qui permet l'évaluation de la condition ;
- Condition : il s'agit d'une condition logique. Si elle est validée, l'action est exécutée ;
- Action : il s'agit d'un ensemble d'invocations, de mises à jour, qui sont réalisées sur les données de la base.

La notion de requête continue présente plusieurs similarités avec les règles ECA, cependant il y a plusieurs différences au niveau de l'architecture, des fonctionnalités et du modèle d'exécution. En effet, dans [131], Liu et al définissent une requête continue comme un triplet $< CQ, T_{cq}, Stop >$ où CQ est une requête standard en SQL, T_{cq} est un déclencheur et $Stop$ définit la condition d'arrêt de l'évaluation de la requête. Chaque fois que la condition du déclencheur est vérifiée, la requête est ré-exécutée et un nouveau résultat est retourné à l'utilisateur. Les conditions de déclenchement et d'arrêt d'une requête continue CQ sont spécifiées à travers des événements. Les auteurs distinguent trois types d'événements :

- Les événements primitifs : un événement primitif peut être de type basique (tels que les opérations d'insertion, de suppression et de mise à jour) ou temporel (le 01 Janvier 2008 à 10h :00, toutes les 8 minutes, etc.) ;
- Les événements conditionnels : il s'agit d'un événement sous la forme, [nomAttribut <opérateur de comparaison> valeur] ou d'un ensemble de conjonctions ou disjonctions d'événements (lorsque la température est entre 60 degré et 80 degré) ;
- Les événements composés : il s'agit d'une composition d'un ensemble d'événements.

Formellement, le résultat d'une requête continue CQ sur l'état S_i de la base de données à l'instant i est défini par $CQ(S_i)$. Le résultat de cette requête est une séquence $\{CQ(S_1), CQ(S_2), ..., CQ(S_n)\}$ obtenue en exécutant CQ sur la séquence d'état de base de

26. Une base de données classique est dite active si elle offre la possibilité aux utilisateurs d'inscrire un ensemble de règles ECA. Généralement, ces règles sont utilisées pour la surveillance, la gestion des processus workflow, etc.

données $\{S_1, S_2, ..., S_n\}$. $CQ(S_i)$ est déclenchée chaque fois que la condition $T_{cq} \wedge \neg Stop$ est vérifiée.

Pour mieux comprendre les différences entre les règles ECA et les requêtes continues, nous nous basons sur un exemple d'une application de gestion de stock dans laquelle, le gérant veut être notifié chaque fois qu'il y a un risque de rupture d'un article du stock. Pour cela, l'ensemble des articles en stock et les articles commandés ne doit pas être inférieur à un seuil défini par le gérant. En utilisant la notion de requête continue définie précédemment, les besoins du gérant peuvent être exprimés comme suit :

```
Q1
CQ : Select articles_stock, articles_commandés, seuil
                From BD
T_{cq} : articles_stock(articleID) + articles_commandés(
    articleID) < seuil(articleID)
Stop : six mois
```

La requête Q1 est exécutée chaque fois que la condition T_{cq} est vérifiée. Cette condition est évaluée, pendant les six prochains mois, chaque fois qu'une mise à jour survient sur la liste des articles en stock, la liste des articles commandés ou le seuil de l'utilisateur. A chaque évaluation de Q1, la liste des articles présentant un risque de rupture est retournée à l'utilisateur.

En utilisant les règles ECA, les besoins du gérant peuvent être exprimées ainsi :

```
Q2
Événement : Mise à jour du stock Ou Mise à jour des articles
    commandées Ou Mise à jour du seuil
Condition : articles_stock(articleID) + articles_commandés(
    articleID) < seuil(articleID)
Action : Afficher articles_stock, articles_commandés, seuil
```

Contrairement à $Q1$, dans la requête $Q2$, une définition explicite de tous les événements est nécessaire pour obtenir le même résultat que $Q1$. L'avantage des requêtes continues par rapport au règles ECA, est que les évènements qui provoquent le déclenchement sont déduits implicitement à partir des déclencheurs.

Dans les requêtes continues définies sur les bases de données, la condition d'arrêt de l'évaluation de la requête est spécifiée explicitement par l'utilisateur. Avec les règles ECA, la requête doit être évaluée tant qu'elle n'a pas été supprimée explicitement du système.

L'environnement des bases de données a également introduit la notion de vues matérialisées. Ces vues sont utilisées essentiellement à des fins d'optimisation et de performance dans le cas où la requête associée est particulièrement complexe ou lourde, ou pour faire des réplications de table. Le défi associé à ces structures consiste à assurer leur mise à jour en fonction de l'évolution des données dans la table, ce qui se traduit par une évaluation conti-

nue des structures. Une vue matérialisée peut être considérée comme une requête continue étant donné qu'elle est associé à une ou plusieurs tables qui évoluent au cours du temps. Une réévalution continue de la vue est ainsi nécessaire afin d'assurer la cohérence entre le contenu des tables et le résultat de la vue. Malgré ces similarités, les vues matérialisées présentent une différence par rapport aux requêtes continues définies dans un environnement de base de données. En effet, le résultat des vues matérialisées est mémorisé dans la bases de données contrairement aux requêtes continues où les résultats sont retournés à l'utilisateur sous la forme de flux.

Nous détaillons dans ce qui suit les principaux systèmes qui ont marqué le traitement des requêtes continues dans les bases de données.

3.2.1 Requêtes continues dans Tapestry

Tapestry [165] est le premier système qui a permi aux utilisateurs de poser des requêtes continues sur une base de données. Ces requêtes ont été développées et incorporées au sein du système afin de filtrer les flux des documents électroniques tels que les courriels (e-mails), les articles, etc. Ce système maintient les informations sur les documents tels que leur auteur, leur date de création, leurs mots clés, etc. La base de données utilisée pour stocker ces documents ne permet que les opérations d'insertion. Aucune opération de suppression ou de mise à jour n'est permise. Les requêtes continues sont utilisées pour identifier "les documents d'intérêt" pour les utilisateurs.

Les requêtes continues dans Tapestry sont exprimées en TQL (pour Tapestry Query Language). A l'instant t, le résultat d'une requête Q contient l'ensemble des n-uplets obtenus en exécutant Q comme une requête SQL classique. Cette requête est évaluée à chaque instant $\tau \leq t$ et le résultat final est obtenu en effectuant l'union de tous les résultats. Comme le montre la Figure 3.1, l'approche utilisée par Tapestry pour implémenter les requêtes continues porte sur deux phases :

– La requête Q posée à l'instant t est convertie en une requête monotone $Q_M(t)$. Une requête est dite monotone si son résultat ne change pas au cours du temps. Prenons l'exemple de la requête ci-dessous :

```
Q3
SELECT * FROM étudiants
WHERE
étudiant.nom = 'Samuel';
```

Cette requête est monotone car lorsqu'un n-uplet est ajouté dans la table, il peut satisfaire ou non la requête. Lorsqu'il répond à la requête, un n-uplet ne disparaît pas de l'ensemble des résultats vu que la base ne permet que les opérations d'insertion.
La requête ci-dessous ($Q4$) permet de sélectionner l'ensemble des étudiants inscrits durant le dernier mois. La valeur "Sysdate" indique la date actuelle. Cette requête est dite non-monotone car un n-uplet peut satisfaire la requête pendant une période

temporelle et ne pas la satisfaire pour une autre période étant donné que la valeur de SysDate change au cours du temps.

```
Q4
SELECT * FROM etudiants
WHERE
etudiant.dateInscription > SysDate - 30;
```

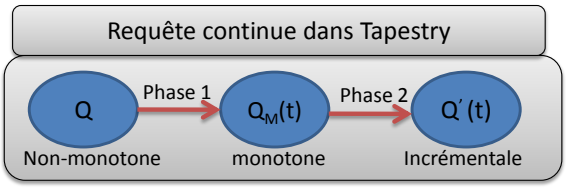

FIGURE 3.1 – Traitement des requêtes continues dans Tapestry

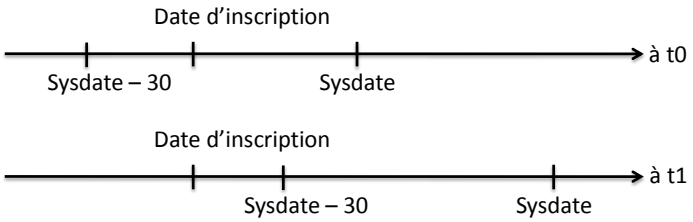

FIGURE 3.2 – Requête non monotone

Dans Tapestry, la conversion d'une requête non-monotone en une requête monotone consiste à transformer Q en $Q_M(t)$ pour qu'elle soit exécutée à chaque instant $\tau \leq t$. Si la requête de l'utilisateur est déjà monotone, alors on utilise directement la requête Q.

$$Q_M(t) = \bigcup_{\tau \leq t} Q(\tau)$$

En reprenant l'exemple de la Figure3.2, le résultat de la requête $Q4_M(t1)$ est :

$$Q4_M(t1) = Q4(t1) \cup Q4(t0)$$

– La requête monotone $Q_M(t)$ est par la suite convertie en une requête incrémentale $Q^{'}(\tau, t)$ qui peut rapidement calculer un résultat approché de $Q_M(t) - Q_M(\tau)$. Dans l'exemple précédant $Q4^{'}(t0, t1)$ est calculée comme suit :

$$Q4^{'}(t0, t1) = Q4_M(t1) - Q4_M(t0).$$

Les requêtes incrémentales ont été introduites dans le système Tapestry pour des raisons de performance. Une requête incrémentale $Q^{'}$ possède deux paramètres : τ l'instant de la

dernière exécution de la requête et t le temps courant. $Q^{'}(\tau, t)$ correspond à l'ensemble des n-uplets retournés par la requête Q_M exécutée sur l'intervalle de temps $[\tau, t]$. Les requêtes incrémentales se caractérisent par leur fonctionnement sur un espace de données réduit. Ceci permet une exécution plus efficace des requêtes continues au lieu d'opérer sur la totalité de la base de données.

La procédure suivante illustre la syntaxe utilisée pour définir des requêtes continues dans Tapestry.

```
τ ← −∞
Tant que (Vrai) Faire
    t ← temps courant
    Exécuter la requête Q'(τ, t)
    Retourner le résultat à l'utilisateur
    τ ← t
Fin Tant que
```

Une extension de cette technique a été présentée dans [28] par Barbara afin de permettre les opérations de suppression et de mise à jour dans la base de données. Considérons la requête suivante :

```
Q5
SELECT *
FROM BD
WHERE (A > 1000)
```

Soit A un attribut d'une relation. Si un n-uplet D_1 satisfait la requête à un instant donné et qu'une mise à jour est effectuée sur cet attribut (la valeur de A devient inférieure à 1000) alors, l'évènement ne satisfait plus la requête. Il peut par conséquent être ignoré lors de l'évaluation de la requête à l'intervalle suivant. Pour garantir la sémantique des requêtes continues, la requête doit être évaluée chaque fois qu'une mise à jour se produit sur un attribut de la requête. Pour tolérer les opérations de suppression dans Tapestry, l'auteur propose une méthode qui prend en considération les n-uplets supprimés lors de l'évaluation de la requête.
Soit D_τ^k l'ensemble des n-uplets qui ont été supprimés de la base avant $k\tau$ et qui ont contribué pour la première fois à l'évaluation de la requête à un quelconque instant durant l'intervalle $[(k-1)\tau, k\tau]$.
Prenons l'exemple de la requête suivante : "sélectionnez tous les messages tel que t > message.id" et un message M tel que $M.id = 450$. Ce message arrive à l'instant $t = 250$ et est supprimé du système à $t = 350$. Supposons que $\tau = 100$ et que l'évaluation de la requête est effectuée aux instants $t = 0, 100, ..., 400, 500, etc.$ Le message M satisfait pour la première fois la requête à $t = 450$. Cependant, ce message est supprimé avant l'évaluation de la requête à l'instant $t = 500$. Ainsi M appartient à l'ensemble des n-uplets D_{100}^5.

Pour l'évaluation des requêtes sur une base de données permettant les suppressions, l'auteur présente le corollaire suivant : soit Q une requête. Si Q est monotone, alors son évaluation est effectuée comme suit :

$$Q(k\tau) - Q((k-1)\tau) = \bigcup_{t_0 \leq s \leq k\tau} Q(s) - \bigcup_{t_0 \leq s \leq (k-1)\tau} Q(s) - D_\tau^k$$

L'évaluation incrémentale utilisée dans Tapestry pour répondre aux requêtes continues est basée sur un principe de réécriture de requêtes. Cependant, cette approche devient inefficace dans le cas de relations très volumineuses. Des calculs répétitifs doivent être réalisés sur un volume important de données historiques dès que de nouvelles données arrivent.

3.2.2 Requêtes continues dans OpenCQ

OpenCQ [131] est un système de bases de données distribuées permettant la définition et l'évaluation de requêtes sur un large réseau. Les requêtes ont pour but de notifier aux utilisateurs les changements dans les bases de données. Dans ce système, les requêtes sont composées de trois parties : la requête, la condition de déclenchement et la condition d'arrêt. Quand la condition du déclencheur est validée, la requête est exécutée continuellement jusqu'à la validation de la condition d'arrêt.

Dans OpenCQ, la première exécution d'une requête continue est activée dès son installation dans le système. Cette activation est réalisée à travers un ensemble d'actions. Nous reprenons dans ce qui suit l'exemple de la requête Q1 :

- **Identification de la requête.** La requête Q est enregistrée dans le système avec un identifiant unique (cq_{id}) ;
- **Réécriture de le requête.** L'utilisateur précise la requête à évaluer ainsi que ses conditions de déclenchement et d'arrêt. Cette requête est notée Q. La requête qui sera évaluée par OpenCQ est notée $Q^{'}$. Il s'agit d'une réécriture de la requête Q qui intégre les conditions de déclenchement et d'arrêt saisies par l'utilisateur ;
- **Exécution de la requête.** Cette étape n'inclut que la première exécution de la requête :
  1. La requête est exécutée au moins une fois, sa première exécution ne tient compte ni de la condition de déclenchement, ni de la condition d'arrêt.
  2. Les événements d'intérêt sont par la suite identifiés à partir de la condition de déclenchement (e.g. dans la requête Q1, les événements d'intérêt sont : articles_en_Stock, articles_commandés et, seuil).
- **Ré-exécution de la requête.** Chaque fois que la condition de déclenchement est vérifiée, l'ensemble des actions suivantes sont réalisées :
  1. Pour chaque événement d'intérêt identifié, le système définit un triplet qui contient : l'événement (e.g. articles_en_Stock), une condition de son déclen-

chement (e.g. article_en_Stock = 10) et un connecteur pour passer au triplet suivant (e.g. WHERE). Le connecteur spécifie le passage d'un triplet à un autre ;

2. Pour chaque événement d'intérêt, si une mise à jour est détectée sur l'un de ces événements, la condition de déclenchement est réévaluée ;
3. Si la condition de déclenchement est vérifiée, le système ré-exécute la requête et retourne à l'utilisateur le nouveau résultat. Ce résultat est la différence entre l'exécution courante de la requête et son ancienne exécution.

L'avantage de OpenCQ réside dans sa capacité à traiter une multitude de requêtes continues dans un environnement évolutif tel que le WEB. Ceci est réalisé grâce à sa gestion distribuée des requêtes.

3.2.3 Requêtes continues dans NiagaraCQ

NiagaraCQ [48] est un système de gestion de plans de requêtes continues sur des bases de données de l'internet. Tout comme OpenCQ, ce système a été développé pour supporter et gérer un grand nombre de requêtes dans un environnement dynamique où les requêtes sont continuellement ajoutées et supprimées du système.

Le système est distribué et permet de construire des requêtes dans un langage proche de XML-QL [69] sur des bases XML distribuées. Chaque requête est constituée de quatre blocs : le corps de la requête, sa date de première exécution, sa date d'expiration et le délai séparant deux exécutions.

Ce système supporte le traitement évolutif des requêtes continues. L'efficacité de la gestion de ces requêtes est assurée par le groupement des requêtes similaires, pour un traitement non redondant. Pour cela, NiagaraCQ utilise une stratégie d'optimisation de groupes avec un re-groupement dynamique. L'idée consiste à rassembler dans un même groupe les requêtes qui partagent la même signature [36] [27]. Chaque groupe est constitué de trois parties :

- La signature du groupe : il s'agit de la signature commune à toutes les requêtes du groupe ;
- Les constantes du groupe : il s'agit des parties constantes ou fixes de toutes les requêtes du groupe. Ces constantes sont mémorisées dans un fichier XML ;
- Le plan du groupe : il s'agit du plan de requêtes commun déduit à partir de toutes les requêtes du groupe.

Lorsqu'une nouvelle requête est enregistrée dans le système, son plan de requête est étudié afin de chercher une correspondance entre sa signature et la signature d'un des groupes existants. Si aucun groupe ne correspond à la signature de la requête, un nouveau groupe est créé avec cette signature.

Le cycle d'une requête continue dans NiagaraCQ est constitué de trois phases : l'installation de la requête, son exécution et sa suppression.

27. La signature d'une requête définit la structure de celle-ci

Installation de la requête continue. Comme le montre la Figure 3.3, pour traiter les requêtes continues, NiagaraCQ fait appel à quatre composantes : le gestionnaire des requêtes continues, le détecteur d'événements, le moteur de requêtes et le gestionnaire de données. Lorsqu'une requête continue entre au système, les informations temporelles et le nom des sources de données sont envoyés au détecteur d'événements (phase 1). Le détecteur d'événements demande par la suite au gestionnaire de données de surveiller les sources concernées (phase 2). Cette étape est nécessaire pour détecter les changements dans les sources de données.

Le détecteur d'événements surveille deux types d'événements : les événements temporels (e.g. chaque 30 secondes) et les événements liés aux modifications dans les sources. Si l'un de ces événement survient, le détecteur d'événements notifie le gestionnaire des requêtes continues de ces modifications (phase 3). Dans le cas d'un événement temporel, le détecteur d'événements signale une ré-exécution de la requête adéquate. Dans le cas d'une mise à jour d'une source, ce composant signale au gestionnaire les requêtes qui doivent être relancées sur la source de données en question.

Le gestionnaire de données dans NiagaraCQ capture les E/S pour les requêtes standard et les requêtes continues. Les entrées sont les sources de données, et les sorties sont les résultats des requêtes. Les sources de données peuvent être de type push ou pull. Pour les systèmes push, le gestionnaire de données est informé d'une mise à jour dans la sources de données (phase 4), il notifie alors le détecteur d'événements. Dans le cas d'un système pull, le détecteur d'événement demande périodiquement au gestionnaire de données de vérifier les modifications dans les sources de données.

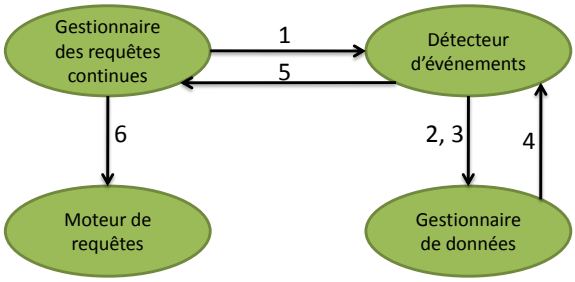

FIGURE 3.3 – Traitement des requêtes continues dans NiagaraCQ

Exécution de la requête continue. Lors de l'évaluation d'une requête continue, son identifiant ainsi que les noms de ses sources de données sont envoyés au gestionnaire des requêtes (phase 5). Ce dernier invoque le moteur de requêtes qui se charge d'exécuter la requête sur les nouvelles données et retourne à l'utilisateur le nouveau résultat (phase 6).

Suppression de la requête continue. Chaque requête continue possède un nom unique. Chaque utilisateur peut utiliser le nom d'une requête pour récupérer son état ou la supprimer du système. Les requêtes qui ont expiré sont automatiquement supprimées.

Système de cache dans NiagaraCQ. Pour que NiagaraCQ puisse facilement passer à l'échelle, un système de cache est utilisé pour obtenir de bonnes performances avec une quantité limitée de mémoire. Pour cela, plusieurs règles sont prises en considération :

- Les plans de requêtes groupées résident en mémoire en supposant que le nombre de groupes de plan de requête est relativement faible.
- Les requêtes non regroupées peuvent être mises en cache en utilisant une politique de gestion de la mémoire (LRU : Least Recently Used) qui favorise les requêtes fréquemment utilisées.
- NiagaraCQ met en cache les fichiers récemment utilisés ainsi qu'un ensemble de fichiers intermédiaires générés par les opérateurs. NiagaraCQ met en place une politique qui favorise la mise en cache de ces petits fichiers qui permet d'économiser considérablement le taux d'E/S vers le disque.

Le principal avantage du système NiagaraCQ est qu'il est évolutif. En effet, ce système ne prend en considération que la partie du fichier XML qui a été modifiée et non la totalité du document. Cette stratégie permet un gain important dans le temps d'évaluation de la requête. Par ailleurs, cette technique évite les calculs répétitifs et redondants qui alourdissent le traitement, en ne retournant que les nouveaux résultats à l'utilisateur.

Un second avantage de ce système est sa capacité à traiter des sources hétérogènes. En effet, NiagaraCQ peut surveiller et détecter les changements des sources de données en utilisant les deux modèles push et pull.

Les systèmes OpenCQ et NiagaraCQ permettent de répondre à de nombreuses requêtes continues sur de grandes bases de données. Cependant, leurs architectures ne permettent pas de passer à l'échelle et de traiter une large gamme de données. En effet, dans le web actuel, avec des données qui changent continuellement, tous les profils des utilisateurs doivent être redéfinis afin de vérifier s'il y a de nouvelles données d'intérêt pour les utilisateurs. NiagaraCQ essaye de regrouper les requêtes similaires pour passer à l'échelle. Cependant, ceci n'apporte qu'une faible amélioration au problème du passage à l'échelle dans la version actuelle du web.

3.2.4 Requêtes continues dans Oracle

Récemment, le SGBD relationnel Oracle dans sa version 11g a introduit les requêtes continues dans sa syntaxe. Une requête continue dans Oracle sort du contexte des requêtes continues définies précédement. En effet, il s'agit d'un objet SQL ayant une définition similaire aux vues matérialisées. Dans cet objet, on spécifie une requête continue Q, les

contraintes sur Q telles que les contraintes d'intégrité fonctionnelles [28], l'endroit où les données seront stockées ainsi qu'un ensemble de caractéristiques sur le calcul de la requête. C'est l'exemple de la requête suivante qui vérifie l'ensemble des comptes courants présentant un solde débiteur. Le résultat de la requête continue peut être une table persistante dans la base de données ou dans une file d'attente. Dans la requête ci-dessous, les résultats de la requête continue sont mémorisés dans la table "*dest_table*".

```
Q6
CREATE CONTINUOUS QUERY verif_compte
PRIMARY KEY (id_compte)
COMPUTE ON COMMIT
DESTINATION dest_table
SELECT id_compte, sum(montant) montant
FROM banking_transactions
GROUP BY id_compte
HAVING sum(montant) < 0;
```

Une des particularités de la requête Q6 est qu'elle contient le prédicat "PRIMARY KEY". La contrainte sur la clé primaire n'est pas définie sur la table de destination mais sur la requête continue. Ce prédicat permet de regrouper les opérations d'insertion et de suppression suivant la valeur de la clé primaire et de retourner la différence (Δ mise à jour).

Le calcul des requêtes continues dans Oracle peut être invoqué selon trois modes :

- Lors de la modification d'une des tables concernées par la requête ;
- A la demande, par appel explicite de la méthode, *recompute_cq*(*nom de la requete*) ;
- Périodiquement, en spécifiant l'estampille temporelle de la première exécution de la requête ainsi que sa périodicité. Dans l'exemple suivant, on spécifie pour la requête *verif_compte* la date de première exécution (01-01-2007) ainsi que sa périodicité (chaque heure).

```
Q7
ALTER CONTINUOUS QUERY verif_compte
COMPUTE START WITH '01-01-2007' PERIOD 1 HOUR;
```

3.3 Traitement de requêtes continues dans les SGFD

Dans le contexte des flux de données, les requêtes continues sont évaluées tant qu'il y a de nouvelles données qui arrivent dans le système. Ces requêtes sont constituées d'un

28. Une contrainte d'intégrité est une clause permettant de contraindre la modification de tables, faite par l'intermédiaire de requêtes d'utilisateurs, afin que les données saisies dans la base soient conformes aux données attendues.

ensemble d'opérateurs relationnels tels que la sélection (Select), la projection (Project) et la jointure (Join) ainsi que d'un ensemble d'opérateurs d'agrégation. Un plan de requête logique, analogue à un arbre de requête utilisé dans un SGBD traditionnel, est généré à partir de la spécification d'une requête continue. Il est ensuite transformé en un plan de requête composé d'algorithmes utilisés pour le calcul de chaque opérateur.

Nous illustrons dans ce qui suit le processus général de traitement des requêtes continues et plus particulièrement le traitement de ces requêtes sur des fenêtres glissantes étant donné qu'il s'agit du type de fenêtres le plus utilisé dans le contexte des flux de données.

3.3.1 File d'attente et stratégie de planification

Contrairement aux SGBD, les opérateurs dans les SGFD sont insérés dans le plan de requête suivant un modèle data push. Chaque opérateur possède une file d'attente permettant aux sources d'insérer les données dans le plan de requête et dans les opérateurs. Une stratégie de planification simple alloue une tranche de temps à chaque opérateur, au cours de laquelle les évènements sont extraits à partir des file(s) d'attente. L'opérateur se charge par la suite de traiter les évènements suivant l'ordre chronologique et envoie les évènements résultants dans la file d'attente de l'opérateur suivant. La tranche de temps accordée à chaque opérateur peut être fixe ou dynamique ou elle est calculée en fonction de la taille de la file d'attente et de la vitesse du traitement. Cette opération peut être optimisée en planifiant le traitement d'un ou plusieurs évènements par plusieurs opérateurs à la fois. En général, plusieurs critères doivent être pris en compte dans le choix de la stratégie de planification tels que : la taille des files d'attente, le taux d'arrivée des données, le temps de latence des évènements résultants des opérateurs, etc.

3.3.2 Requêtes continues sur fenêtres glissantes

L'hypothèse fondamentale sur laquelle repose un système de gestion de flux de données est que de nouvelles données sont générées en permanence, ce qui rend impossible le stockage d'un flux dans son intégralité. Ainsi, des fenêtres glissantes peuvent être maintenues afin de contenir les données récemment arrivées, ce qui signifie que les anciennes données doivent être supprimées au fil du temps. En outre, étant donné que le contenu des fenêtres évolue au cours du temps, le résultat d'une requête continue sur la fenêtre glissante doit refléter les mises à jour.

Nous illustrons dans ce qui suit deux techniques permettant de maintenir l'état des fenêtres glissantes dans le traitement des requêtes continues : l'approche des évènements négatifs [19][104][105] et l'approche directe [104][105].

– Approche des évènements négatifs : dans l'approche des évènements négatifs, chaque fenêtre référencée dans la requête est associée à un opérateur qui génère explicitement un évènement négatif lors de chaque expiration. De plus, cet opérateur assure l'envoi des nouveaux évènements (issus de l'opérateur) dans le plan de requête. Les

évènements négatifs se propagent dans le plan de requête et sont traités par les opérateurs comme des évènements "normaux". Cependant, ils provoquent la suppression des évènements réels. La Figure 3.4 a) illustre le traitement d'un évènement négatif dans une opération d'agrégation précédé par une jointure sur fenêtres glissantes. L'évènement négatif est généré suite à une expiration sur la fenêtre définie sur le flux 1. L'évènement négatif est traité par tous les opérateurs dans le plan de requête. Par conséquent, l'opérateur d'agrégation peut éventuellement recevoir un ensemble d'évènements négatifs correspondant à tous les résultats de jointure. L'inconvénient de cette approche est qu'il faudra traiter deux fois plus d'évènements étant donné que pour chaque évènement expiré, un évènement négatif est généré. De plus, de nouveaux opérateurs doivent être ajoutés au plan de requête pour générer des évènements négatifs à chaque glissement de la fenêtre.

– Approche directe : les requêtes évaluées sur des fenêtres basées sur l'estampille temporelle ont comme propriété le temps d'expiration des évènements qui peut être déterminé par l'estampille t_{exp}. Ainsi, les opérateurs n'ont plus besoin des évènements négatifs pour trouver les évènements expirés. L'approche directe est illustrée dans la Figure 3.4 b). Pour chaque nouvel évènement dans l'une des fenêtres de la jointure, l'expiration est réalisée au moment du traitement du nouvel évènement. L'approche directe porte l'avantage de ne pas inclure les coûts générés par les évènements négatifs. Cependant, cette approche peut être plus lente que l'approche précédente dans le cas où plusieurs fenêtres sont utilisées. Par exemple, si les évènements dans la mémoire tampon sont triés par leur estampille temporelle, les insertions seront simples. Cependant, les suppressions exigent un parcours séquentiel des données. D'autre part, trier toutes les données selon la date d'expiration simplifie l'opération de suppression mais alourdit l'opération d'insertion qui nécessite une analyse séquentielle de la mémoire tampon.

  L'inconvénient de l'approche directe est que les nouvelles données ne peuvent être traitées immédiatement par tous les opérateurs du plan de requête. Ainsi, les résultats intermédiaires peuvent être retardés par rapport aux données entrées. Une solution permettant de garantir un résultat correct consiste à maintenir une horloge au sein de chaque opérateur. Cette horloge maintient l'estampille temporelle de l'évènement le plus récent traité par l'opérateur parent.

3.3.3 Expiration des données

En plus du traitement des évènements dans les files d'attente, les opérateurs doivent supprimer les anciens évènements de la mémoire tampon et mettre à jour le résultat de la requête. L'expiration des données diffère en fonction du type de la fenêtre. Nous détaillons dans ce qui suit la stratégie d'expiration pour les fenêtres physiques et les fenêtres logiques.

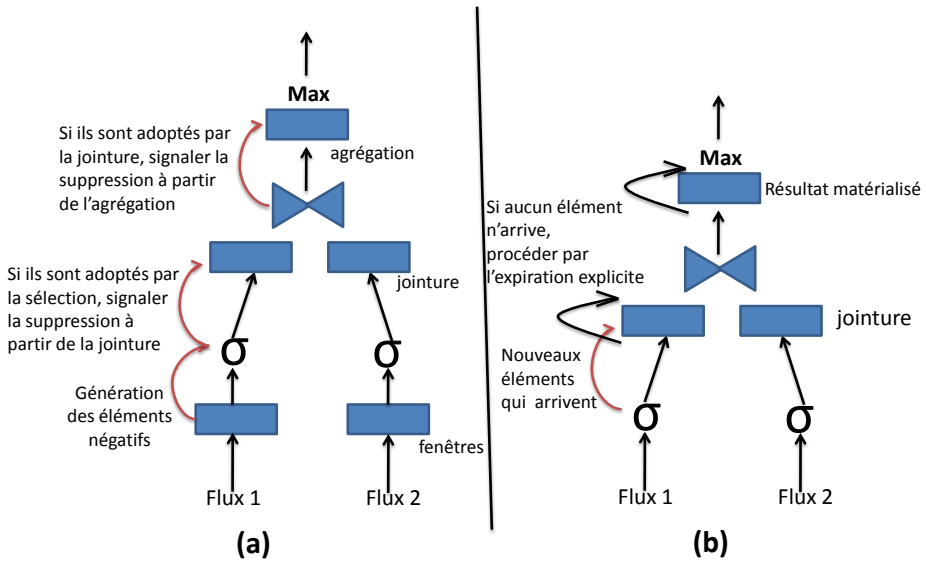

FIGURE 3.4 – (a) Approche des évènements négatifs (b) Approche directe

L'expiration des évènements d'une fenêtre glissante basée sur les estampilles temporelles est simple : un évènement expire si son estampille temporelle n'est plus dans la période de la fenêtre. Si un évènement entre dans le système avec une estampille temporelle t, le système lui associe automatiquement une nouvelle estampille temporelle t_{exp} indiquant le moment de son expiration ($t_{exp} = t$+taille de la fenêtre). En effet, chaque évènement dans la fenêtre peut être associé à un intervalle d'une longueur égale à la taille de la fenêtre. Cependant, si on effectue une opération de jointure entre cet évènement et un évènement d'une autre fenêtre ayant comme estampille d'insertion $t^{'}$ et estampille d'expiration $t^{'}_{exp}$, alors l'évènement résultant (résultat de la jointure) aura comme estampille d'expiration $min(t_{exp}, t^{'}_{exp})$. Ainsi l'évènement résultant expire dès qu'au moins un de ses évènements "source" expire.

Dans une fenêtre logique, le nombre d'évènements dans la fenêtre reste constant au fil du temps. Par conséquent, le processus d'expiration peut être mis en place en remplaçant l'évènement le plus ancien par le nouvel évènement. Dans certains cas de figure, l'expiration doit être provoquée explicitement. C'est l'exemple d'une opération de jointure entre deux flux. Dans tel cas, l'expiration est provoquée en utilisant les évènements négatifs générés par le système mais qui ne seront pas pris en considération dans le traitement.

3.3.4 Traitement des requêtes continues dans Esper

Etant donné que nous utilisons dans nos contributions le SGFD Esper (cf. chapitre6), nous consacrons cette section à la description du modèle d'exécution des requêtes continues dans ce système.

Dans Esper, pour qu'un utilisateur puisse accéder au résultat d'une requête continue, il doit lui associer un objet de type "écouteur" (Listener). Cet objet implante une fonction permettant de récupérer les évènements résultants de l'exécution de la requête. Il s'agit de la méthode "update (objet [] newEvents, objet [] oldEvents)". Cette méthode est invoquée chaque fois que la requête fournit un évènement ou un lot d'évènements. Les arguments de la méthode "update" sont deux tableaux d'évènements. Ces tableaux sont remplis à la volée par le serveur Esper lors de l'exécution d'une requête. Le tableau "newEvents" contient le dernier évènement (ou le dernier lot d'évènements) renvoyé par la requête lors de la dernière invocation de la méthode "update". Le second tableau "oldEvents" n'est utilisé que si la requête interroge une fenêtre. Dans ce cas, le tableau "oldEvents" contient le dernier évènement (ou le dernier lot d'évènements) expiré de la fenêtre alors que "newEvents" contient le dernier évènement inséré dans la fenêtre.

Nous illustrons dans la Figure 3.5 (extraite de [118]) le remplissage des tableaux ("oldEvents" et "newEvents") par le serveur Esper lors de l'interrogation d'une fenêtre glissante de taille 3. A chaque insertion d'un évènement dans la fenêtre glissante, il est automatiquement ajouté au tableau "newEvents". Lorsque la fenêtre est saturée, l'évènement le plus ancien est supprimé et, il est ajouté au tableau "oldEvents".

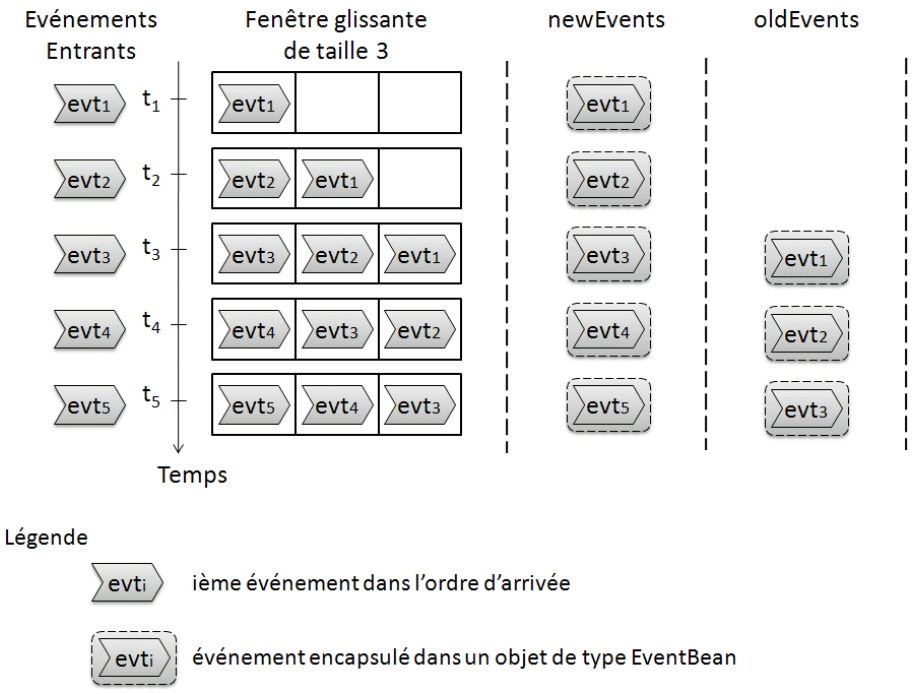

FIGURE 3.5 – Remplissage des tableaux oldEvents et newEvents dans le cas des fenêtres glissantes

3.3.5 Notion de requêtes hybrides continues sur flux de données

Plusieurs tâches d'analyse sur les flux de données nécessitent à la fois des informations arrivant en temps réel dans le flux et des informations plus anciennes stockées sur disque, concernant l'historique du flux. L'évaluation de telles requêtes n'est pas facile étant donné qu'elles nécessitent des quantités de données historisées qui dépassent la taille de la mémoire principale.

3.3.5.1 Définition d'une requête hybride

Dans [43], Chandrasekaran et Franklin définissent une requête hybride comme une requête continue qui porte à la fois sur des données anciennes et sur des données récentes. Les données anciennes sont généralement historisées dans une structure de données persistante, telle qu'une base de données. Les données récentes sont les données du flux mémorisées pendant un court instant dans la mémoire du SGFD. Ainsi, pour évaluer ce type de requête, il faut accéder aux données enregistrées sur disque et aux données qui circulent dans le flux. C'est l'exemple des requêtes qui permettent de détecter d'éventuelles fraudes dans les systèmes bancaires en comparant l'activité actuelle de la carte de crédit aux activités passées.

3.3.5.2 Évaluation des requêtes hybrides

L'utilisation conjointe des flux de données et des bases de données a été largement étudiée dans la littérature. En effet, plusieurs travaux se sont focalisés sur la jointure entre les flux et les relations [148][166][19][38]. Toutefois, dans cette section, nous nous intéressons aux travaux qui utilisent les données récentes du flux et les données historisées. À notre connaissance, peu de travaux se sont intéressés à l'évaluation des requêtes hybrides.

PSoup [45] est le premier système à combiner des données sous forme de flux avec des données persistantes. L'idée consiste à évaluer des requêtes qui nécessitent un accès aux données déjà arrivées (données persistantes) et des données qui arriveront dans le futur. Le système PSoup traite les requêtes et les données séparément sous forme de flux mais en associant une dualité à ces deux notions. L'utilisateur interagit avec le système par un premier enregistrement de la requête : il enregistre les spécifications de celle-ci. Le système attribut alors un identifiant unique et procède à une évaluation continue de cette requête sur les données en entrée et le résultat est conservé dans une structure auxiliaire (appelée structure résultat). Après l'invocation de la requête, le système calcule la taille de la fenêtre pour retourner le résultat adéquat. Pour cela, il applique les clauses BEGIN et END, définies dans la requête, sur la structure résultat. Le système permet de répondre à des requêtes monotones de la forme SELECT-FROM-WHERE. La structure des requêtes rappelle celle des fenêtres définies dans 1.7.3 :

Q11

```
Select *
From Flux F
Where (F.a < v1 ^ F.b > v2)
Begin (NOW - 100)
End (NOW)
```

Les clauses BEGIN et END permettent de spécifier le début et la fin de la fenêtre qui est associée à la requête. Par ailleurs, les valeurs choisies pour ces clauses permettent de spécifier soit des snapshots (où BEGIN et END sont constants), soit des fenêtres points de repère (dans ce cas le BEGIN est constant et le END est variable), soit des fenêtres glissantes (dans tel cas le BEGIN et le END sont tous les deux variables).
Le principal apport du système PSoup est double : le premier consiste à appliquer les nouvelles requêtes sur des données anciennes ce qui revient à évaluer les requêtes sur des données historiques du flux. Et, le second aspect consiste à appliquer les nouvelles données sur des requêtes anciennes, ceci revient à évaluer des requêtes continues sur de nouvelles données en entrées. Cependant, l'approche présentée nécessite le stockage des données et des requêtes dans des structures. Or, une des contraintes d'un environnement flux de données est la limite de l'espace de stockage disponible face à la volumétrie de données générées.

Dans [43], Chandrasekaran et Franklin ont été les premiers à s'intéresser à l'évaluation des requêtes hybrides combinant des données en temps réel sous forme de flux et des données historisées. L'objectif consiste à accéder de façon simultanée aux données du flux et aux données historisées. Les auteurs se sont intéréssés dans cet article au coût nécessaire pour extraire les données historisées. Ce coût peut avoir une conséquence sur le traitement des données en temps réel (i.e. perte de certaines informations si le débit du flux est important). Les auteurs proposent le système OSCAR, une solution qui permet de réduire les coûts d'accès aux données historisées en récupérant, en fonction de la charge du système, une version réduite des données. Ce système utilise TelegraphCQ comme SGFD et PostgreSQL comme SGBD. Le SGFD TelegraphCQ se charge de traiter les flux en entrée. Par la suite, le système se charge de conserver temporairement des échantillons du flux dans le SGBD. Ces échantillons représentent des versions du flux avec différents taux de réduction (e.g. 25%, 50%, etc.). Pour répondre à une requête, le système sélectionne en fonction des ressources disponibles, la version du flux la plus appropriée. Pour permettre l'évaluation des requêtes hybrides, les auteurs ont effectué plusieurs extensions et changements au niveau du moteur de requête de TelegraphCQ. Ces changements sont assez complexes et difficiles à étendre à d'autres SGFD.

Dans [151], Reiss et al se sont intéressés aux index Bitmap pour mémoriser et évaluer les données historisées du flux. Les index Bitmap [40] [145] [156] [174] [175] sont connus pour être efficaces afin de répondre à des classes importantes de requêtes sur un volume important de données où peu de mises à jour sont effectuées. En effet, les mises à jour

incrémentales sur ces index ne sont pas pratiques et sont très coûteuses. Les auteurs ont amélioré l'utilisation de ces index afin de réduire les coûts de mises à jours des données historisées. Pour cela, ils proposent une architecture qui combine le système TelegraphCQ pour la gestion des flux de données et, FastBit [176] pour la gestion des données historisées. L'utilisation conjointe de ces deux systèmes permet de répondre rapidement à des requêtes qui portent sur les données historisées. En effet, étant donné que le système FastBit indexe les données historisées, la recherche devient plus rapide. Cependant, pour assurer une telle rapidité les auteurs supposent que les données historisées ne sont mises à jour que par des opérations d'insertion. Avec une telle hypothèse, on estime qu'on dispose de l'espace mémoire nécessaire pour conserver la totalité du flux si on a des requêtes qui portent sur le début du flux.

Dans [138], Maskey et al présentent une approche fondée sur le principe de régénération de flux pour le traitement des révisions dans un flux de données. Les auteurs se placent dans un contexte où les données du flux peuvent être erronées et qu'il faudra envoyer à nouveau les évènements pour corriger ceux préalablement envoyées. Suite à cette opération, toutes les requêtes déjà évaluées sur les évènements non-corrigées doivent être réévaluées. C'est l'exemple des flux d'information envoyées par les agences de presse qui sont corrigés en cas d'erreur. Dans cet article, les auteurs présentent une extension d'un moteur de traitement de flux pour supporter les révisions. Cette technique suppose qu'il existe une archive du flux sur le disque. Lorsqu'une source envoie une révision, le système met à jour l'archive et réévalue les requêtes qui portent sur ces données mises à jours. Ceci suppose qu'aucun système de résumé n'est mis en place autre que les données archivées, l'intégralité du flux étant maintenu sur disque. La mise à jour de l'archive consiste à remplacer les données les plus anciennes par les données les plus récentes.

Récemment, dans [70], Dindar et al présentent DejaVu. Il s'agit d'un système conçu pour permettre la reconnaissance des formes à partir d'un flux de données et d'une archive. Contrairement aux approches classiques, DejaVu est conçu sur un système de gestion de base de données au lieu d'un SGFD. Pour cela, les auteurs étendent le système MySQL pour permettre le traitement des flux de données. Cette extension inclut un ensemble de nouveaux composants : une machine à états finis pour le modèle d'exécution des requêtes de reconnaissance de forme, de nouveaux moteurs de stockage pour les flux de données à temps réel et la gestion des données archivées, un moteur de traitement de requêtes continues, etc. En outre, [70] est un papier de démonstration, dans lequel les auteurs ne présentent pas d'explications approfondies sur le fonctionnement interne du système, ni sur la synchronisation entre les données archivées et les données récentes.

Dans [27], Balazinska et al présentent le système Moirae. Ce système est conçu pour la supervision des réseaux et la notification en cas de panne. Les auteurs s'intéressent aux données historisées afin d'apprendre des anciennes pannes détectées. Moirae permet de répondre à deux types de requêtes. Les requêtes standard (e.g. sélectionner les activités historiques de l'intrus pour chaque attaque sur le réseau) et, les requêtes contextuelles

(e.g. si le réseau tombe en panne, sélectionner les alertes similaires détectés dans le passé). Ces requêtes font ainsi appel à des données à temps réél et des données historisées. Cependant, toutes les données anciennes sont archivées sous formes d'historiques prédéfinis avant l'arrivée du flux. Seules les requêtes faisant appel à ces historiques peuvent être évaluées.

3.4 Synthèse et positionnement

Les systèmes informatiques d'aide à la décision (SIAD) visent à donner les moyens de déclencher des alertes de gestion, de suivre l'évolution de l'activité et de disposer d'outils d'investigation de sujets ou phénomènes particuliers à partir de l'étude des activités récentes et passées. Ceci est d'abord réalisé en utilisant les requêtes continues dans un environnement de base de données. Cependant, dans le contexte des flux de données, ces requêtes font face à de nouvelles contraintes suite au caractère fortement évolutif des données. Dans ce chapitre, nous avons défini la notion de requête continue et présenté les différents systèmes qui ont permis l'évaluation de ce type de requêtes. Les requêtes continues sont apparues dans deux environnements : l'environnement des bases de données (Tapestry, OpenCQ, NiagaraCQ, etc.) et l'environnement des flux de données (STREAM, Aurora, TelegraphCQ, etc.).

Il apparaît au terme de cet état de l'art que le domaine des flux de données est très actif et continue de susciter l'intérêt d'un grand nombre de communautés de recherche (Base de données, Statistique, Ingénierie logicielle, etc.). Plusieurs défis sont apparus avec ce nouveau modèle imposant l'extension des approches existantes d'extraction, de traitement et d'analyse.

Les systèmes de gestion de flux de données (SGFD) ont été développés afin de répondre à un besoin particulier : l'interrogation à la volée et en temps réel des données du flux par des requêtes continues. Ces systèmes disposent d'une mémoire de taille limitée qui met à disposition un certain nombre de données de détail. L'ensemble des évènements inclus dans cette mémoire sont appelés évènements du passé proche. Ainsi, suite à ce qui a été présenté dans l'état de l'art, les SGFD permettent à ce jour d'accéder aux données de détail qui couvrent le passé proche. Toute forme de requête peut ainsi être appliquée sur ces évènements. Cependant, ces données sont éphémères et les capacités de stockage sont limitées. Par conséquent, tout besoin exprimé sur la base de ces évènements après leur expiration de la mémoire ne peut être considéré. Il s'agit alors d'évaluer des requêtes sur des données du passé lointain. Les évènements du passé lointain sont les évènements du flux qui ont expiré de la mémoire du SGFD et qui par conséquent sont perdus à jamais à moins de planifier un mécanisme de conservation explicite.

Le besoin d'interroger des requêtes continues sur les données du passé lointain a engendré l'apparition de la notion de résumé de flux de données. Comme défini précédemment, un résumé de flux de données est une structure compacte permettant d'être mise à jour au fur et à mesure de l'arrivée des évènements du flux et permettant de répondre à des

requêtes évaluées sur le passé lointain d'un flux. Nous avons distingué au cours de cet état de l'art (cf. section 2.2) trois types de résumés de flux de données à savoir : (i) les synopsis qui ont pour objectif de traiter une requête particulière sur le passé proche et sur le passé lointain, (ii) les historiques qui permettent de résumer les flux de données à travers un ensemble d'agrégats et, (iii) les résumés généralistes qui permettent la réponse à un éventail plus large de requêtes sur le passé lointain.

Nous nous intéressons dans ce manuscrit à la conception et l'interrogation des résumés généralistes de flux de données, plus précisément à deux algorithmes qui fournissent des résumés de structures différentes : les algorithmes StreamSamp [61] et CluStream [15]. Nous présentons dans le chapitre 4 une étude détaillée de ces deux algorithmes (paramétrage, comportement, qualité des résultats, etc.). L'analyse réalisée met en avant les avantages et les inconvénients de ces deux techniques.

Nous proposons dans le chapitre 5, une nouvelle approche pour l'amélioration de ces algorithmes de résumés généralistes. L'approche développée, intitulée "*approche hybride*", permet de résumer des flux de données numériques.

Dans le chapitre 6, nous nous focalisons sur l'interrogation des résumés généralistes. Notre objectif sera alors l'évaluation des requêtes continues sur les résumés de flux de données conservés à travers un SGFD. Nous proposons pour cela une extension de l'architecture actuelle des SGFD afin d'assurer l'accès aux résumé sur le passé lointain. L'architecture proposée repose sur deux aspects importants : (1) la construction des résumés au sein du SGFD et, (2) l'évaluation des requêtes par reconstitution des données du passé lointain.

Chapitre 4

Etude des résumés de flux : cas de StreamSamp et CluStream

Sommaire

4.1 Introduction

La conception des résumés généralistes a fait l'objet du second chapitre dans lequel nous avons présenté une panoplie de techniques permettant de résumer un flux de données individuel. Cependant, une fois ces résumés conçus, il est important de les exploiter efficacement. La gestion des résumés de flux de données pose deux problématiques fondamentales. La première concerne la construction et la conservation des résumés. Tandis que la seconde problématique implique leur utilisation. Nous portons notre étude sur les

algorithmes StreamSamp [61] et CluStream [15]. Le choix de ces algorithmes n'est pas arbitraire. D'une part ces techniques ont montré leur efficacité dans la qualité du résumé conçu, en associant une approche mémoire de compression de données et une approche permettant une bonne organisation temporelle. D'autre part, ils mettent en oeuvre des structures très différentes.

Nous nous intéressons à l'exploitation de ces résumés. Cela consiste à adapter les algorithmes des requêtes aux structures de résumé. En effet, nous présentons pour chaque tâche d'analyse les différentes étapes à appliquer sur le résumé. Nous présentons dans un premier temps une analyse détaillée de la structure des résumés utilisés ainsi que l'importance de chaque paramètre dans la constitution d'une version compacte du flux. Nous établissons par la suite une étude comparative des performances fournies par chacun des algorithmes sur plusieurs tâches de requêtage et de fouille.

4.2 Résumé de StreamSamp

Comme nous l'avons décrit dans le chapitre 2, l'algorithme StreamSamp [61] se base sur une technique d'échantillonnage aléatoire et un système de fenêtres inclinées pour la construction du résumé au cours du temps. Il peut être appliqué à la fois sur des données qualitatives et quantitatives. Une fois le résumé construit, il est conservé sur disque afin de maintenir une trace du flux pour d'éventuels post-traitements. La construction du résumé se fait en ligne, en temps réel, tandis que son exploitation est une opération hors ligne.

Dans cette section nous présentons en premier lieu la structure du résumé fournie par l'algorithme StreamSamp. Nous détaillons par la suite les différentes tâches d'interrogation et de fouille réalisées sur ce résumé.

4.2.1 Structure du résumé

A la différence des techniques d'échantillonnage existantes permettant de résumer le flux tel que l'échantillonnage réservoir [29], le résumé StreamSamp se traduit par un échantillonnage progressif. Contrairement aux fenêtres du passé proche, les fenêtres du passé lointain contiennent des échantillons ayant subi plusieurs opérations de ré-échantillonnage. Au cours de ces opérations des évènements du flux sont supprimés. Cette fonction d'oubli inspirée du principe des fenêtres inclinées permet de libérer de l'espace, mais privilégie (en terme de précision) le passé proche.

La structure du résumé est comme suit : nous disposons d'un ensemble de fenêtres. Chaque fenêtre illustre un échantillon $E_i =< t_{début}, t_{fin}, e_1, e_2, ..., e_n >$. $t_{début}$ et t_{fin} constituent les estampilles temporelles qui délimitent la période représentée par l'échantillon. e_1, ... e_n représentent l'ensemble des évènements du flux conservés dans l'échantillon. Les

29. L'échantillonnage réservoir conserve un seul échantillon couvrant la totalité du flux

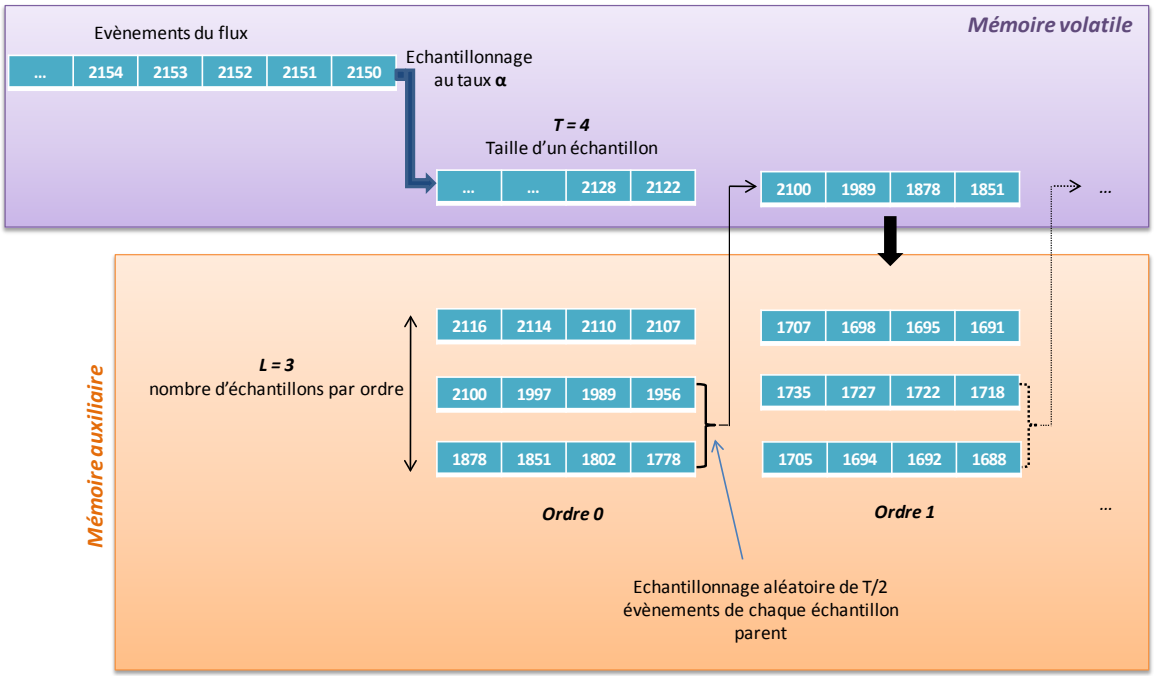

FIGURE 4.1 – Processus de ré-échantillonnage de StreamSamp.

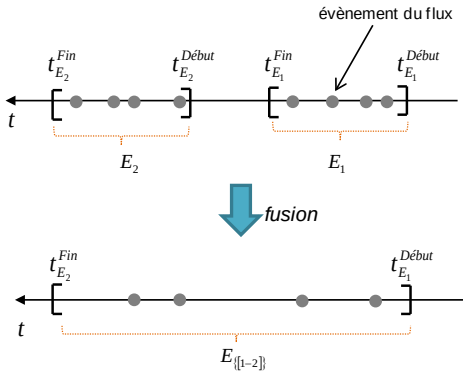

FIGURE 4.2 – Opération de fusion entre deux échantillons

estampilles temporelles des évènements n'étant pas conservées, chaque évènement e_i est décrit par un ensemble de valeurs $< v_1, v_2, ..., v_d >$ avec d la dimension du flux.

Comme illustré dans la Figure 4.1, lorsqu'on atteint le nombre maximum d'échantillons par ordre, le système fusionne les deux plus vieux échantillons de cet ordre. Un nouvel échantillon d'ordre supérieur est alors créé par ré-échantillonnage aléatoire. L'opération de ré-échantillonnage engendre ainsi le vieillissement des échantillons.

Lors de la fusion de deux échantillons (E_1 et E_2 par exemple), on conserve l'estampille temporelle du début de l'échantillon E_1 et l'estampille temporelle de fin de l'échantillon E_2 (cf. Figure 4.2). L'échantillon ainsi constitué contient le même nombre d'évènements

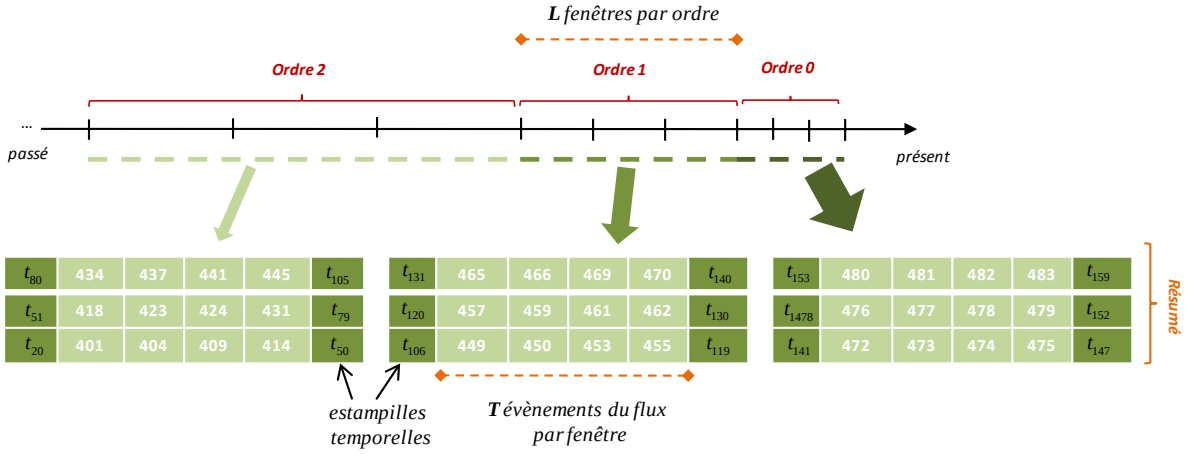

FIGURE 4.3 – **Struture du résumé de StreamSamp.** Dans cet exemple, le résumé contient 3 fenêtres par ordre ($L = 3$). Chaque fenêtre comporte 4 évènements du flux ($T = 4$). Pour chaque fenêtre, les estampilles temporelles de la période sont conservées.

(T) mais, ces derniers couvrent des périodes temporelles de plus en plus longues au fur et à mesure du vieillissement du résumé. La concaténation de tous les échantillons conservés permet de fournir un échantillon de la totalité du flux. La Figure 4.3 illustre la composition du résumé conçu par StreamSamp.

4.2.1.1 Pondération

Pour garantir la représentativité des évènements, un poids est associé à chaque événement de l'échantillon. Nous étudions dans la suite l'effet de cette pondération dans la qualité de la réponse aux requêtes. Cette pondération est obtenue en fonction de l'ordre auquel appartient l'évènement dans le résumé à l'instant t (instant de l'évaluation de la requête). En effet, le processus d'échantillonnage récursif fait progresser les évènements du flux dans des ordres croissants et, en fonction de la position de l'évènement à l'instant t, un poids lui est associé. Ce poids est égal à $2^{o_t} \times \frac{1}{\alpha}$ (avec o_t l'ordre auquel appartient l'évènement à l'instant t et $\alpha \in]0, 1]$ le taux d'échantillonnage initial). Le même poids est attribué à l'intégralité des évènements contenus dans un échantillon d'ordre o.

La pondération des évènements conservés est ainsi liée à l'ordre auquel ils appartiennent. Plus l'ordre est grand, plus le poids est important. Cependant, la vitesse de croissance des poids des évènements est fortement liée aux paramétrages de l'algorithme du résumé. Nous étudions dans la section suivante, l'effet du choix de ces paramètres sur la pondération des échantillons.

4.2.1.2 Effet des paramètres

A l'instant de l'évaluation d'une requête sur un résumé de StreamSamp, nous ne disposons que d'un ensemble d'échantillons. Ainsi, la réponse ne peut être exacte mais fournie par des estimations. Nous nous intéressons en premier lieu à l'effet de l'échantillonnage sur la qualité des estimations. Nous nous basons sur la théorie des sondages [20] pour étudier cet effet. Un des points fondamentaux est la nature de l'aléa introduit dans l'échantillonnage. L'échantillonnage fait appel essentiellement à trois notions : le biais, la variance et l'erreur quadratique moyenne.

On considère une population de taille N finie dans laquelle on tire un échantillon sans remise de taille n finie. L'échantillon est tiré avec un taux de sondage de n/N. Nous limitons les analyses proposées dans ce qui suit aux variables quantitatives ainsi qu'aux requêtes d'agrégat de type Count, Somme, Moyenne, Variance et Médiane.

Soit X une variable quantitative, m la moyenne exacte de la population et $\overline{X}$ la moyenne estimée calculée sur l'échantillon.

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.1}$$

$\overline{X}$ est un estimateur sans biais de la moyenne. Nous avons ainsi $\mathbb{E}[\overline{X}] = m$.

Pour quantifier l'erreur liée à l'opération d'échantillonnage, nous calculons l'erreur quadratique moyenne. Cette erreur est en fonction du biais et de la variance :

$$EQM(\overline{X}) = Var(\overline{X}) + \left(Biais(\overline{X})\right)^2 \tag{4.2}$$

$$Biais(\overline{X}) = \mathbb{E}[\overline{X}] - m \tag{4.3}$$

$$Var(\overline{X}) = \mathbb{E}[\overline{X} - \mathbb{E}(\overline{X})]^2 \tag{4.4}$$

L'intervalle de confiance de la moyenne pour un coefficient de risque a est :

$$[\overline{X} - t.\sqrt{Var(\overline{X})}; \overline{X} + t.\sqrt{Var(\overline{X})}] \tag{4.5}$$

avec t un nombre tel que $\prod(t) = 1 - \frac{a}{2}$ ($\prod(t)$ est donnée par la table de la loi de Student).

En théorie de sondage, l'écart type $\sigma(\overline{X})$ [30] et la variance permettent de mesurer la précision. Plus ils sont grands, moins bon est le sondage. Il faut soit agir sur l'expression de l'estimateur ($\overline{X}$), soit modifier la probabilité du tirage.

Nous étudions par la suite l'effet des paramètres de l'algorithme sur le volume occupé et la qualité des résultats. Nous posons N_e le nombre d'évènements passés avant l'arrivée de l'évènement e. Ainsi l'ordre de e est déterminé par la formule suivante :

30. $\sigma(\overline{X}) = \sqrt{Var(\overline{X})}$

$$Ordre(e) = log_2 \left\lfloor \frac{N_e}{L * T} \right\rfloor \tag{4.6}$$

En tenant compte de l'échantillonnage initial, l'ordre de l'évènement devient alors :

$$Ordre(e) = log_2 \left\lfloor \frac{N_e}{(L * T)/\alpha} \right\rfloor \tag{4.7}$$

Nous rappelons que les paramètres de l'algorithmes StreamSamp sont α, T et L. Le taux d'échantillonnage α contribue au remplissage des échantillons du premier ordre (*ordre 0*). Dans le reste de ce chapitre, nous considérons que la totalité des évènements du flux sont traités ($\alpha = 1$).
Les paramètres L et T ont un effet sur le vieillissement du résumé en agissant sur la vitesse de croissance des ordres. Par conséquent, ces deux paramètres ont un rôle important sur le processus de mise à jour du résumé ainsi que la qualité des résultats des requêtes. Pour ralentir le vieillissent des échantillons au cours du temps et avoir un résumé "*jeune*", il suffit d'augmenter la valeur de $L * T$, compte tenu de l'espace mémoire disponible.

1. **Volume occupé :** une des principales contraintes qui doit être prise en considération est l'espace mémoire disponible pour conserver les résumés de flux. Cet espace est généralement inférieur au volume du flux. Pour cela, il est nécessaire de bien paramétrer les algorithmes de résumé en prenant en considération cette contrainte. Dans ce qui suit, nous étudions l'importance du produit $L * T$ dans le volume occupé à l'instant courant t. Nous désignons par V_{Max} la taille maximale du résumé. Il s'agit de la composition du résumé lorsque tous les ordres sont complètement remplis (cf. Figure4.3 où pour chaque ordre on a $L * T$ évènements du flux).
   - Si, on a un seul ordre, le résumé contient $L * T$ évènements ;
   - Si, on a 2 ordres, le résumé contient $2 * L * T$ évènements ;
   - ...
   - Si, on a n ordres, le résumé contient $n * L * T$ évènements.

   Ainsi, le volume maximum occupé est :

   $$V_{Max}(t) \approx (L * T) * nb_{Ordres}(t) \tag{4.8}$$

   avec $nb_{Ordres}(t)$, le nombre d'ordres permettant de stocker les t évènements du flux.

   $$nb_{Ordres}(t) = Ordre(t) + 1 \tag{4.9}$$

   D'après 4.6, nous avons :

   $$Ordre(t) = log_2[\frac{t}{L * T}] \tag{4.10}$$

   Ainsi,

   $$nb_{Ordres}(t) = log_2[\frac{t}{L * T}] + 1 \tag{4.11}$$

Lorsqu'on atteint le volume maximale V_{Max} et qu'un nouvel échantillon entre au système à l'instant t, des mises à jours sont propagées pour créer un nouvel ordre. Cet ordre va contenir l'échantillon de taille T propagé par le processus de ré-échantillonnage. Une fois cet échantillon placé dans le nouvel ordre, il reste alors $t-T$ évènements à distribuer entre les ordres précédents (au nouvel ordre créé). Ces ordres contiennent $L-1$ échantillons. Ainsi, le volume atteint à t noté V_{Min} est donné par la formule suivante (pour $t \geq T$) :

$$V_{Min}(t) \approx T + (L-1) * T[log_2(1 + \frac{t-T}{(L-1)*T})] \tag{4.12}$$

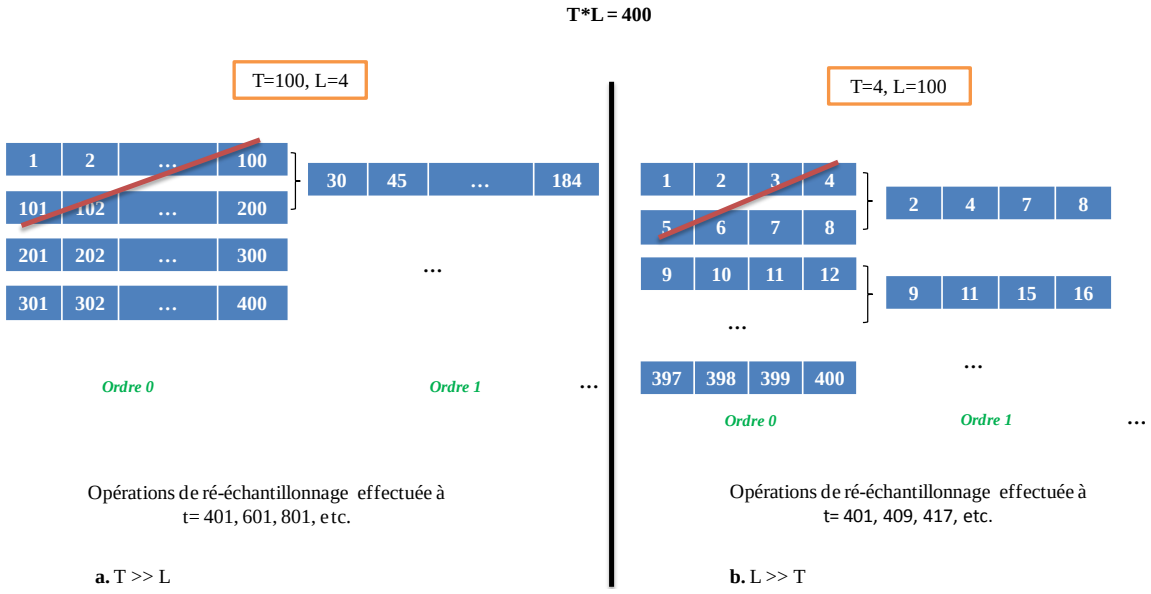

FIGURE 4.4 – Mise à jour du résumé cas du $T >> L$ et $L >> T$ à valeur $L * T$ constante. Nous supposons qu'on a un évènement par unité de temps.

Nous constatons que les deux paramètres ont un effet important sur le volume du résumé. Ils interviennent sous la forme de leur produit. Par conséquent, à valeur $L*T$ constante, ces deux paramètres ont des effets différents que nous décrivons ci-dessous.

2. **Puissance de calcul :** le paramètre T contribue à la régularité du processus de ré-échantillonnage. En effet, plus la taille d'un échantillon (T) est grande, moins souvent sera activée l'opération de ré-échantillonnage. Par conséquent, les opérations d'écriture sur le disque (qui sont assez coûteuses) seront moins fréquentes. Dans l'exemple (cf. Figure 4.4), pour $L = 4$ et $T = 100$, il y a une opération de mise à jour tous les 200 évènements du flux, aux instants $t = 401$, $t = 601$, $t = 801$, etc. Tandis que pour $L = 100$ et $T = 4$, des opérations de mise à jour sont prévues tous les 8 évènements du flux, aux instants $t = 401$, $t = 409$, $t = 417$, etc. Ainsi, plus T est petit, plus le nombre d'E/S sur disque est important.

3. **Qualité du résumé :** la qualité du résumé se traduit par deux notions. La première est la pondération associée aux échantillons et qui par conséquent reflète la qualité de ces derniers. La deuxième est le positionnement des fenêtres qui contribue dans la précision du résultat de la requête.

   (a) Pondération : nous avons indiqué précédemment que le paramètre T contrôle le nombre de mises à jour (opération de ré-échantillonnage) effectuées sur le résumé. Chaque opération de mises à jour double le poids d'un échantillon et produit par conséquent la dégradation de sa qualité.

   (b) Positionnement des fenêtres : le choix du paramètre T a un effet sur la taille de l'échantillon final (cf. section 4.2.3.1) sur lequel sera évaluée la requête. Plus la taille de ce paramètre est petite, meilleure est la précision étant donné qu'on conserve plus d'estampilles temporelles. Ainsi le résumé va contenir plusieurs échantillons (de petites tailles) et pour chaque échantillon, les dates de début et de fin de constitution sont sauvegardées. Par conséquent, meilleur est le positionnement des bornes de la requête sur la fenêtre inclinée, meilleurs seront les résultats retournées.

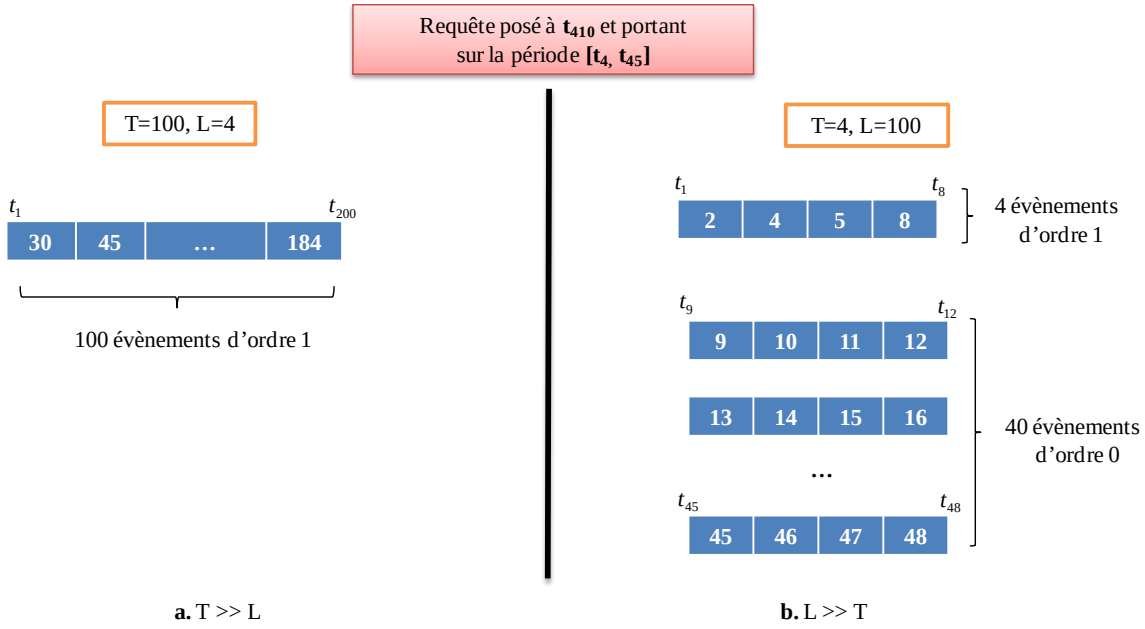

FIGURE 4.5 – Ensemble des échantillons du résumé couvrant la période interrogée $[t_4, t_{45}]$.

Le tableau 4.1 résume les effets des paramètres $L * T$ et T. Deux compromis distincts apparaissent. Ils permettent de régler respectivement les valeurs de $L * T$ et du paramètre T ; la valeur de L s'en déduit.

– Compromis entre le volume occupé et la qualité d'estimation : en augmentant le produit $L * T$, le résumé occupera un espace mémoire plus important (ce qui pose

TABLE 4.1 – Effet des paramètres T et L

Contrainte	Si $L * T \nearrow$	Si $T \nearrow$
Volume occupé	$\nearrow$	
E/S sur disque		$\searrow$
Qualité de l'échantillon	$\nearrow$	
Positionnement des fenêtres		$\searrow$

un problème vu qu'on est limité en ressources). En contre partie, ceci améliorera la qualité des échantillons ;

– Compromis entre la fréquence des opérations d'E/S sur disque et le positionnement des fenêtres : en augmentant la valeur de T, les opérations de lecture/écriture se feront moins souvent étant donné qu'il y aura moins d'opérations d'échantillonnage. Cependant, plus la taille des échantillons est grande, plus la précision sur le positionnement des fenêtres est mauvaise étant donné qu'on ne conserve pas les estampilles temporelles des évènements.

4.2.2 Persistance des résumés

L'interrogation a posteriori des résumés nécessite leur conservation sur disque. Le choix de l'approche adoptée pour le stockage du résumé doit tenir compte des coûts d'E/S sur disque. Nous détaillons dans ce qui suit les avantages et inconvénients de deux systèmes pouvant gérer la persistance des données : système de fichiers et système de gestion de base de données.

Système de fichiers. Dans cette modélisation, chaque échantillon est représenté par un fichier contenant les différents événements du flux. Pour accélérer l'opération de recherche des échantillons les plus anciens d'un ordre, chaque fichier est nommé selon la date de début de constitution de l'échantillon. Chaque dossier représente un ordre et contient ainsi L fichiers. Lors d'une opération de ré-échantillonnage, on peut facilement accéder au dossier de l'ordre concerné. Cependant, pour extraire les échantillons les plus anciens d'un ordre, il faut trier tous les fichiers du dossier selon leur nom. Suite à cette opération, on extrait aléatoirement $T/2$ événements de chaque fichier. Les évènements extraits sont ainsi enregistrés dans un nouveau fichier et les deux anciens fichiers sont supprimés.

Système de gestion de base de données. Une deuxième approche permet de conserver le résumé dans une base de données. Les événements du flux sont insérés dans des relations indexées afin de faciliter leur accès. Les échantillons sont triés en fonction de

leur date de début de constitution. Nous proposons dans ce qui suit un schéma relationnel possible.

Le schéma relationnel. La modélisation proposée permet de distinguer le flux de départ, le résumé conçu et, les paramètres utilisés pour la conception du résumé. Le schéma relationnel, illustré dans la Figure 4.6 a été conçu de façon à prendre en considération le traitement de plusieurs flux. Pour représenter cela, une relation **"Flux"** permet de conserver les informations relatives aux flux en entrée telles que son identifiant et son nom. A chacun des flux traités par StreamSamp correspond un enregistrement dans cette relation.
La relation **"Paramètres"** contient les différents paramètres utilisés par l'algorithme pour résumer les flux à savoir le taux d'échantillonnage (α), la taille des échantillons (T) et le nombre d'échantillons par ordre (L). Le résumé constitué pour un flux donné en fonction d'un lot de paramètres est accessible à travers la relation **"Résumé"**.
La relation **"Échantillon"** maintient les informations relatives aux échantillons conservés par l'algorithme. Chaque échantillon est caractérisé par son identifiant, son poids (2^{Ordre}) ainsi que sa date début et fin de constitution qui représentent les dates de la période échantillonnée. Chaque échantillon contient un ensemble d'évènements mémorisés dans la relation **"Évènement"**. Cette relation comporte l'identifiant de l'échantillon ainsi que les valeurs de ses attributs.

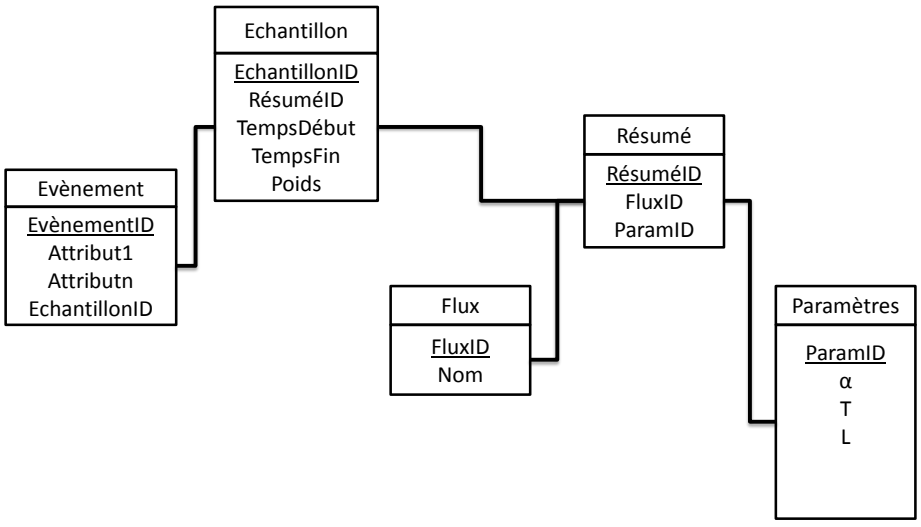

FIGURE 4.6 – Modèle relationnel du résumé conçu par StreamSamp.

Les avantages de cette représentation sont doubles : le premier consiste à permettre de séparer l'univers des flux de données de celui des résumés conçus et, le second consiste à permettre le résumé de plusieurs flux. Un même flux peut participer à l'élaboration de différents résumés ; différents résumés peuvent avoir les mêmes paramètres, etc.

Indexation des données. Un index est un élément de redondance que l'on va spécifier pour permettre au système de gestion de base de données (SGBD) d'optimiser certaines requêtes. Il est placé sur une relation, il permet un accès très rapide aux enregistrements, selon la valeur d'un ou plusieurs champs. Les index jouent un rôle discret mais important dans la gestion des bases de données. La décision de créer un index résulte de l'examen des avantages et inconvénients de cette opération. L'index ralentit les mises à jour et consomme un peu de place, mais il rend les tris et les recherches d'information plus rapides.

L'algorithme StreamSamp est basé sur un échantillonnage récursif. L'opération de ré-échantillonnage s'effectue entre les échantillons les plus anciens d'un même ordre. Ainsi, avant toute opération de fusion, les échantillons d'un même ordre doivent être ordonnés de façon à extraire les deux plus anciens échantillons. Pour ce faire, nous appliquons des index à la table "Résumé" afin d'assurer un accès direct aux échantillons. Deux possibilités d'indexation s'offrent à nous : (i) assurer que l'attribut "EchantillonID" soit incrémental, (ii) créer un index sur l'attribut "TempsDebut".

Dans le premier cas de figure, il suffit de conserver un identifiant incrémental pour les échantillons de façon à ce que chaque nouvelle entrée prend l'identifiant qui suit. L'utilisation d'un index groupant (appellé aussi cluster index) sur l'ensemble des attributs ("RésuméID, Poids"), permet d'extraire tous les échantillons d'un ordre donné et les deux plus anciens échantillons de l'ordre sont par la suite extraits en utilisant l'attribut "EchantillonID". L'utilisation des index groupants permet de spécifier que l'index créé correspond à un arrangement physique particulier où les enregistrements voisins sont stockés dans des blocs voisins.

Dans la deuxième approche, un index groupant sur l'attribut ("RésuméID, TempsDebut") permet d'extraire les deux plus anciens échantillons. Le résultat du ré-échantillonnage prendra par la suite la place de l'un des échantillons à supprimer de façon à conserver un ordre croissant dans les estampilles temporelles. Par ailleurs, afin d'accéder rapidement et extraire tous les évènements d'un même échantillon, on place un index groupant sur l'attribut "EchantillonID" de la relation "Évènement". Dans l'implantation de StreamSamp, nous avons opté pour cette approche étant donné qu'elle nous permet d'accéder rapidement aux échantillons les plus anciens d'un ordre.

Avantages et inconvénients. Nous discutons dans ce paragraphe les avantages et inconvénients des deux modélisations présentées précédemment.

- **Utilisation générale** : l'avantage d'une modélisation orientée système de fichiers est qu'elle est simple et facile à utiliser. De plus elle ne nécessite pas de pré-requis logiciels. Tandis qu'une solution orientée base de données, nécessite la mise en place d'un SGBD, création des index, etc.
- **Identification des plus anciens échantillons d'un ordre** : dans un système de fichiers, une opération de tri des fichiers est réalisée avant chaque fusion afin de localiser les deux échantillons les plus anciens. Cependant, en utilisant un SGBD, les

n-uplets représentant les échantillons sont déjà triés en fonction de la date de début des échantillons, ce qui facilite l'accès aux données des échantillons concernés.

- **Gestion de données** : les SGBD sont connus pour leur capacité à gérer les données, gérer les accès concurrents, etc. Ceci présente un avantage majeur lorsque le résumé doit être exploité par plusieurs entités (programmes, utilisateurs, etc).
- **Exploitation du résumé** : contrairement aux SGBD, les systèmes de fichiers n'ont pas de langage d'interrogation puissant tel que SQL. Ceci va nécessiter beaucoup plus d'effort dans l'implantation tels que le développement de méthodes pour parcourir les échantillons, accéder aux attributs, etc.

L'utilisation d'un SGBD offre plus d'avantages dans la construction et l'exploitation des résumés qu'un système de fichiers (e.g. indexation des données dans la base, optimisation des requêtes, etc). Plusieurs schémas relationnels peuvent être envisagés afin de modéliser le résumé de StreamSamp. Nous avons adopté un schéma permettant de séparer les informations sur les différents flux en entrée et les informations sur les résumés.

4.2.3 Tâches d'analyse

L'un des principaux objectifs d'un résumé généraliste de flux de données est la possibilité de fournir une réponse approchée à n'importe quelle requête pouvant porter sur les données d'origine. La structure du résumé doit par ailleurs permettre de calculer des bornes sur la précision des réponses approchées aux différentes requêtes posées.

En se basant sur le résumé StreamSamp, l'échantillon final E_f sur lequel va porter les tâches d'analyse est un échantillon pondéré. De ce fait, toutes les techniques classiques d'analyse de données peuvent lui être appliquées. Le tableau 4.2 illustre les notations utilisées dans la suite de cette section. Dans la suite de cette analyse, nous nous intéressons à des variables quantitatives et à un estampillage logique. La Figure 4.7 vient appuyer le tableau pour mieux illustrer les bornes de la requête.

Table 4.2 – Liste des notations utilisées pour StreamSamp

Symboles	Descriptions
$[t_a, t_b]$	bornes de la requête
E_f	échantillon final
$X_i, Y_i, Z_i, ...$	valeur de l'événement d'indice i pour les variable X, Y, Z, ...
w_i	poids de l'événement d'indice i

Nous nous intéressons dans ce qui suit aux analyses portées sur une période $[t_a, t_b]$. Pour cela, le système procède en deux étapes : (i) constitution de l'échantillon qui représente cette période (appelé *échantillon final* E_f) et, (ii) application de l'algorithme d'évaluation de la requête sur cet échantillon.

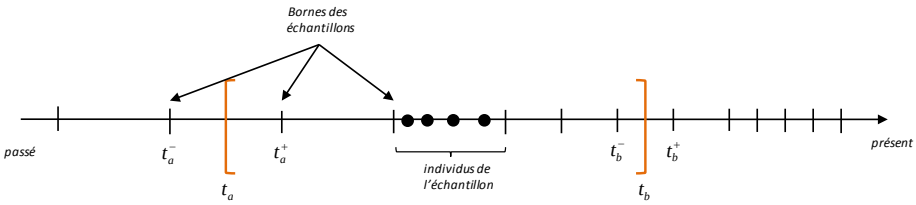

FIGURE 4.7 – Représentation des bornes des échantillons et des bornes de la période interrogée.

4.2.3.1 Constitution de l'échantillon final

Une fois la requête définie, le système constitue l'échantillon final sur la période évaluée en deux phases :

1. Le système positionne les bornes de la période évaluée sur la fenêtre inclinée. Deux cas sont possibles : soit les bornes de la requête coïncident avec les bornes des échantillons conservés (*cas particulier*) soit, les bornes ne coïncident pas (*cas général*). La Figure 4.7 illustre le second cas, les bornes de la requête ne coïncident pas avec les bornes des échantillons sauvegardés. Vu que notre résumé ne conserve pas les estampilles temporelles des évènements, les bornes de la requête doivent alors être décalées en fonction des bornes des échantillons.

2. Le système extrait l'échantillon final en fonction des échantillons avoisinant les bornes de cette période. En respectant les informations conservées dans notre résumé (i.e. bornes des échantillons maintenus), on peut retourner à l'utilisateur quatre estimations possibles (cf. Figure 4.7) : une première estimation sur l'intervalle $[t_a^+, t_b^-]$, une seconde estimation respectant l'intervalle $[t_a^+, t_b^+]$, une troisième suivant l'intervalle $[t_a^-, t_b^-]$ et enfin une dernière estimation suivant $[t_a^-, t_b^+]$.

   Une réponse possible à la requête portant sur l'intervalle $[t_a, t_b]$ consiste à retourner à l'utilisateur les quatres estimations possibles. L'utilisateur choisit ainsi la réponse qui lui convient. Cependant, dans le pire cas, une requête sera évaluée quatre fois. Une solution possible serait de calculer la réponse "optimale". Celle-ci se traduit par l'évaluation de la requête sur les échantillons dont les bornes sont les plus proches des bornes de la requête. Formellement, on cherche à déterminer les bornes t_1 et t_2 qui minimisent la formule ci-dessous :

$$\underbrace{Min}_{t_a \neq t_b}(|Log_2(t_a) - Log_2(t_2)| + |Log_2(t_b) - Log_2(t_1)|) \tag{4.13}$$

$$\text{avec } t_2 \in [t_a^+, t_a^-] \text{ et } t_1 \in [t_b^+, t_b^-]$$

L'emploi du log est argumenté par le fait que l'échelle de représentation qui est utilisée est la fenêtre inclinée. Comparons les résultats de la formule 4.13 avec ceux de la formule 4.14 (sans utilisation du Log).

$$\underbrace{Min}_{t_a \neq t_b}(|t_a - t_2| + |t_b - t_1|) \tag{4.14}$$

Suposons qu'on s'intéresse à une requête évaluée à l'intervalle $[400, 2]$ et qu'on conserve des échantillons aux dates suivantes : $10000, 1000, 100, 10, 1$.

Les quatres solutions possibles pour cette requête sont illustrées dans le tableau 4.2.3.1.

Solution	Coût avec Log	Coût sans Log
$[100, 1]$	2.07	302
$[100, 10]$	2.30	308
$[1000, 1]$	1.60	602
$[1000, 10]$	2.52	608
Solution de moindre coût	$[1000, 1]$	$[100, 1]$

TABLE 4.3 – Solutions possibles à la requête évaluée sur $[400, 2]$

On obtient bien des résultats différents, ce qui est cohérent avec le fait que sur une échelle logarithmique, 400 et plus près de 1000 (un facteur de 2) que de 100 (un facteur de 4).

D'autre part, dans l'équation 4.13, le traitement des deux bornes est réalisé en parallèle, conjointement afin d'éviter les intervalles vides.

L'échantillon E_f ainsi formé peut contenir des évènements appartenant à des ordres différents. Dans ce cas, les évènements sont pondérés de façon à conserver la même représentativité pour chaque événement dans l'échantillon final.

4.2.3.2 Tâches de requête

Dans cette section, nous nous intéressons à l'évaluation de requêtes d'agrégats sur le résumé StreamSamp telles que le la MOYENNE, la VARIANCE, et la MEDIANE. L'évaluation de ces requêtes est exécutée sur l'échantillon final E_f constitué à partir de la période interrogée $[t_a, t_b]$.

Soit X une variable aléatoire continue de moyenne m et de variance σ^2 et n la taille de l'échantillon final E_f (observations de X) sur lequel sera évaluée la requête.

Calcul de la moyenne et de la variance. Pour calculer l'estimateur sans biais de la moyenne $\overline{X}$, il faut prendre en considération le poids associé aux évènements. L'estimation de cet agrégat sur un échantillon extrait du résumé de StreamSamp est donnée par la formule suivante :

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i w_i}{\sum_{i=1}^{n} w_i} \tag{4.15}$$

Comme la variance de la population est inconnue, on estime la variance empirique comme suit :

$$S^2 = \frac{1}{(\sum_{i \in E_f} w_i - 1)} \sum_{i=1}^{n} w_i (X_i - \overline{X})^2 \tag{4.16}$$

L'intervalle de confiance à 95% est :

$$\overline{X} + t_{0.025} \frac{S}{\sum_{i=1}^{n} w_i} \leq m \leq \overline{X} + t_{0.0975} \frac{S}{\sum_{i=1}^{n} w_i} \tag{4.17}$$

où $t_{0.025}$ et $t_{0.975}$ sont respectivement les quantiles 2.5% et 97.5% de la loi de Student à $(n-1)$ degrés de liberté.

Pour le calcul de l'intervalle de confiance de la variance, nous avons :

$$P\left(\chi^2_{\frac{\alpha_1}{2}} \leq (\sum_{i=1}^{n} w_i - 1)\frac{S^2}{\sigma^2} \leq \chi^2_{1-\frac{\alpha_2}{2}}\right) = 1 - \alpha \tag{4.18}$$

où $\chi^2_{\alpha_1}$ est le fractile d'ordre α_1 de la loi $\chi^2(n-1)$
et $\chi^2_{1-\alpha_2}$ est le fractile d'ordre $1-\alpha_2$ de la loi $\chi^2(n-1)$
Etant donné que l'espérance est inconnue, l'intervalle de confiance bilateral pour la variance s'écrit donc au niveau $1-\alpha$ sous la forme suivante :

$$(\sum_{i=1}^{n} w_i - 1)\frac{S^2}{\chi^2_{1-\frac{\alpha_2}{2}}} \leq \sigma^2 \leq (\sum_{i=1}^{n} w_i - 1)\frac{S^2}{\chi^2_{\frac{\alpha_1}{2}}} \tag{4.19}$$

Calcul de la médiane. La médiane (appelée aussi deuxième quartile) est une statistique de l'ordre. Elle a pour rôle de diviser l'échantillon en deux parties de taille égale. Pour estimer la médiane sur E_f, il faut :

- Trier toutes les valeurs de l'échantillon final E_f, avec $Rang(i)$ = position de l'évènement i dans ce tri.
- La médiane est l'évènement p qui satisfait la formule suivante :

$$\underset{p}{^{argmin}} \left| \left(\sum_{rang(i) \leq rang(p)} w_i \right) - \frac{1}{2} \sum_i w_i \right| \tag{4.20}$$

Le calcul de l'intervalle de confiance de la médiane est donnée dans Scherrer[158].

4.2.3.3 Tâches de fouille

Dans cette section, nous nous intéressons à l'évaluation des tâches de classification automatique sur le résumé conçu par StreamSamp. La classification automatique peut être :

- supervisée (en anglais classification) : les classes sont connues à priori ;
- non-supervisée (en anglais clustering) : les classes sont fondées sur la structure des objets.

Classification non supervisée. L'objectif de cette analyse est de regrouper entre eux les individus qui sont semblables de façon à faire apparaître une structure dans les données traitées. Dans le cas des flux de données cette analyse doit pouvoir porter aussi bien sur l'intégralité des évènements observés que sur une partie de ceux-ci. Trois étapes sont nécessaires pour effectuer cette tâche de fouille sur l'échantillon final de StreamSamp :

1. Préparation de l'échantillon : la première étape consiste à prendre en considération chaque évènement de l'échantillon final et ceci en le dupliquant autant de fois que son poids. Certaines techniques permettent de prendre en considération l'attribut relatif aux poids des évènements ;
2. Construction des classes : après la préparation de l'échantillon final, l'ensemble des évènements sont traités par un algorithme de clustering (e.g., K-means, DBSCAN, EM, etc.) afin de construire l'ensemble des classes ;
3. Evaluation : après la construction des micro-classe, la phase d'évaluation consiste à calculer la distance intra-classe entre le jeu de données original et l'ensemble des classes construites à partir du résumé. Plusieurs passes sont réalisées sur l'échantillon E_f et la meilleure passe est par la suite sélectionnée afin d'évaluer la qualité de la classification obtenue.

Classification supervisée. L'objectif de cette analyse revient à construire un modèle en se basant sur le résumé conçu. Il est trivial d'appliquer la classification supervisée sur le résumé conçu par StreamSamp étant donné qu'il permet de conserver la même structure que le flux d'origine (i.e. des échantillons pondérés).

Comme pour la classification non supervisée, pour réaliser la tâche de classification supervisée sur l'échantillon final de StreamSamp, trois étapes sont nécessaires :

1. Préparation de l'échantillon : cette étape est identique à celle pour la classification non supervisée. Lorsque la méthode de classification utilisée ne permet pas la prise en compte de la pondération des évènements, on duplique chaque évènement par son poids ;

2. Construction du modèle : plusieurs approches peuvent être utilisées pour construire un modèle. On distingue des algorithmes basés sur les arbres de décision (C4.5, CART, etc), sur les réseaux de neurones, sur les K plus proches voisins, etc.
3. Évaluation du modèle : une fois le modèle construit, une méthode d'évaluation est appliquée afin de calculer l'erreur de classification (e.g. la validation croisée, apprentissage/test, etc.).

4.2.4 Limites de StreamSamp

En raison de sa simplicité, StreamSamp présente deux limites principales :

- **Dégradation de la qualité du résumé** : la qualité du résumé StreamSamp se dégrade au cours du temps. En effet, les échantillons ont des poids croissants pour une taille constamment fixe. Par conséquent, si l'échantillon final sur une période contient des évènements récents (poids beaucoup plus faible) et des évènements anciens, ces derniers vont considérablement augmenter la variance.
- **Incertitude sur le positionnement des fenêtres** : pour évaluer la requête, l'échantillon final est extrait du résumé en fonction des bornes des échantillons proches aux bornes de la requête. L'intervalle de la requête est généralement différent de celui de la réponse, à moins qu'il ne coïncide avec les bornes des échantillons conservés. Ainsi pour des périodes anciennes, l'écart entre les bornes de l'échantillon et ceux de la requête devient important ce qui peut engendrer une mauvaise estimation.

Nous présentons dans la section suivante une extension de l'algorithme StreamSamp. Cet algorithme (appelé StreamSamp+) vise une meilleure précision pour les requêtes d'agrégats traitées précedemment ainsi qu'un meilleur positionnement des bornes de la requête sur la fenêtre inclinée.

4.2.5 StreamSamp+

L'extension de StreamSamp concerne la structure de données utilisée ainsi que la démarche adoptée pour constituer l'échantillon final. L'objectif de StreamSamp+ est de fournir une réponse plus précise aux différentes requêtes d'agrégats pour lesquelles des statistiques sont maintenues et mises à jour avec l'arrivée des évènements du flux.

4.2.5.1 Structure de données

L'algorithme StreamSamp+ apporte deux extensions à la structure des échantillons conservés. Tout d'abord, chaque évènement dans l'échantillon est mémorisé avec sa date d'arrivée. La différence avec le résumé de StreamSamp réside dans la structure des évènements conservés e_i, définie comme suit : $< t_{ei}, v_1, v_2, ..., v_d >$ avec t_{ei} l'estampille temporelle associée à l'évènement, v_i la valeur de l'attribut et, d la dimension du flux.

De plus, pour chaque échantillon, un ensemble de mesures est maintenu au cours du temps. Une nouvelle structure nommée SFV (pour *Sample Feature Vector*) est sauvegardée pour chaque échantillon. On maintient dans cette structure le nombre total d'événements N dans la période représentée par l'échantillon, les estampilles temporelles début et fin de l'échantillon (t_D et t_F) et, la somme ainsi que la somme des carrés de chacune des variables numériques de la période : (N, t_D, t_F, $\sum x_i$, $\sum x_i^2$ avec x_i la valeur de l'attribut d'indice i).

4.2.5.2 Réponse aux requêtes

Nous nous intéressons dans ce qui suit aux mêmes requêtes d'agrégats définies précédemment (cf. section 4.2.3.2). L'évaluation de ces requêtes est réalisée sur le flux complet. Contrairement à StreamSamp, l'algorithme StreamSamp+ permet de respecter les bornes de la période évaluée pour répondre à la requête. Ainsi, la construction de l'échantillon final respectera les bornes de la période définie dans la requête. En effet, comme les estampilles temporelles des évènements sont maintenues dans l'échantillon, le système se base sur ces informations pour positionner la période évaluée $[t_a, t_b]$ sur la fenêtre inclinée.

Dans le cas général (cf. Figure 4.8), la requête est calculée en trois étapes :

1. estimation de l'agrégat sur l'intervalle $[t_a, t_a^+]$;
2. calcul exact en se basant sur les SFV associés aux échantillons inclus dans l'intervalle $[t_a^+, t_b^-]$;
3. estimation de l'agrégat sur l'intervalle $[t_b^-, t_b]$.

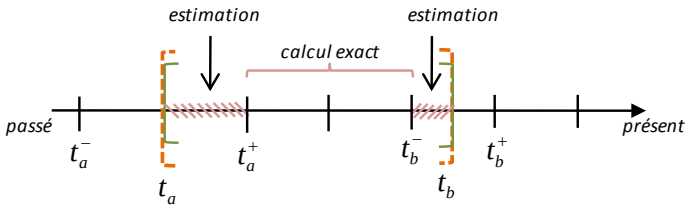

FIGURE 4.8 – Calcul de la requête sur StreamSamp+ : estimation et calcul exact.

De ce fait, le système est amené, dans la plupart des cas, à estimer les agrégats au niveau des extrémités de l'intervalle temporel. Nous détaillons dans ce qui suit le calcul de la moyenne aux bornes de la requête. La même technique est utilisée pour l'estimation de la variance ou de l'écart type. Nous nous basons sur la théorie des sondages [20] pour réaliser ces estimations.

Pour calculer l'estimateur de la moyenne $\overline{X}_{[t_a,t_b]}$, il faut tout d'abord calculer l'estimateur du nombre d'évènements de la période considérée $\overline{C}_{[t_a,t_b]}$ ainsi que l'estimateur de la somme $\overline{So}_{[t_a,t_b]}$.

$$\overline{X}_{[t_a,t_b]} = \frac{\overline{So}_{[t_a,t_b]}}{\overline{C}_{[t_a,t_b]}} \tag{4.21}$$

$$\overline{C}_{[t_a,t_b]} = \overline{C}_{[t_a,t_a^+]} + N_{[t_a^+,t_b^-]} + \overline{C}_{[t_b^-,t_b]} \tag{4.22}$$

$$\overline{So}_{[t_a,t_b]} = \overline{So}_{[t_a,t_a^+]} + \sum X_{i[t_a^+,t_b^-]} + \overline{So}_{[t_b^-,t_b]} \tag{4.23}$$

Dans les formules 4.22 et 4.23 $N_{[t_a^+,t_b^-]}$ et $\sum X_{i[t_a^+,t_b^-]}$ sont correctement calculés étant donné qu'ils sont conservés dans les SFV. Pour $\overline{C}_{[t_a,t_a^+]}$, $\overline{C}_{[t_b^-,t_b]}$, $\overline{So}_{[t_a,t_a^+]}$ et $\overline{So}_{[t_b^-,t_b]}$, il suffit d'appliquer les formules de la section 4.2.3.2 pour pouvoir les estimer.

Pour calculer un intervalle de confiance lié à ces estimations, on calcule la variance. Cependant, calculer la variance dans ce cas n'est pas une tâche simple du fait de la présence d'une estimation au dénominateur. Cependant, étant donné que $\overline{X}_{[t_a,t_a^+]}$ et $\overline{X}_{[t_b^-,t_b]}$ sont indépendants, on a :

$$Var(\overline{X}_{[t_a,t_a^+]}, m_{[t_a^+,t_b^-]}, \overline{X}_{[t_b^-,t_b]}) = Var(\overline{X}_{[t_a,t_a^+]}) + Var(\overline{X}_{[t_b^-,t_b]}) \tag{4.24}$$

avec $m_{[t_a^+,t_b^-]}$, la vraie moyenne sur la période $[t_a^+, t_b^-]$.

Pour approcher la variance, on applique la technique du Bootstrap [20] sur les intervalles $[t_a,\ t_a^+]$ et $[t_b^-,\ t_b]$:

1. Ré-échantillonnage avec remise de taille $|ES_a|$ (ou $|ES_b|$) (avec $|ES_a|$ le nombre d'évènements conservés sur $[t_a,\ t_a^+]$ et $|ES_b|$ le nombre d'évènements conservés sur $[t_b^-,\ t_b]$) ;
2. Calcul de la variance sur l'échantillon ;
3. Répétition des étapes (1) et (2) jusqu'à atteindre le niveau de précision souhaité ;
4. Calcul de la moyenne des valeurs obtenues.

4.3 Résumé de CluStream

Comme indiqué précédemment, CluStream [15] est un algorithme développé pour traiter des tâches de classification non supervisée, cependant, il fournit une structure particulièrement adaptée au résumé de flux de données. Cet algorithme permet de traiter des données quantitatives. Le tableau 4.4 résume les notations utilisées dans la suite de cette section.

4.3.1 Structure du résumé

Le résumé de CluStream est constitué à partir de l'utilisation conjointe d'un système de clichés et de la structure de CFV associée aux micro-classes. Nous rappelons que les CFV permettent de conserver des statistiques sur les évènements absorbés par les micro-classes. Dans chaque CFV on maintient un vecteur formé par ces mesures :

TABLE 4.4 – Liste des notations utilisées pour CluStream

Symboles	Descriptions
$[t_a, t_b]$	bornes de la requête
E_f	échantillon final
x_i	valeur de l'événement d'indice i
n_i	nombre d'événements absorbés par la micro-classe i
t_i	estampille temporelle de l'évènement d'indice i
α	raison de la suite dans la fenêtre pyramidale
$\alpha^{L_c} + 1$	nombre de clichés par ordre
k	nombre de micro-classes de départ
m	nombre d'événements utilisés pour créer le premier lot de micro-classes
N_c	nombre maximum de micro-classes

- L'effectif des évènements dans la micro-classe i : n_i
- Pour chaque dimension dans un évènement, la somme des valeurs : $\overline{CF1^x}=\sum_{j=1}^{n_i} X_j$
- Pour chaque dimension dans un évènement, la somme des carrées des valeurs : $\overline{CF2^x}=\sum_{j=1}^{n_i} X_j^2$
- La somme des estampilles temporelles : $CF1^t=\sum_{j=1}^{n_i} t_j$
- La somme des carrés des estampilles temporelles : $CF2^t=\sum_{j=1}^{n_i} t_j^2$

A intervalles de temps réguliers, le système photographie l'état des micro-classes en sauvegardant sur disque l'ensemble des CFV ainsi que l'historique des fusions des micro-classes. En se basant sur cette liste d'identifiants ainsi que les propriétés des CFV, il est possible de soustraire les clichés entre eux afin d'obtenir un résumé du contenu du flux entre deux instants et observer l'évolution des micro-classes (cf. la section 4.3.2.1).

4.3.1.1 Système de clichés dans CluStream

Le système de clichés utilisé dans CluStream permet de conserver des informations relatives à l'état des micro-classes à un instant donné. Chaque cliché est défini par son instant de capture et par l'ensemble des CFV des micro-classes. Ces clichés sont sauvegardés sur disque selon le principe de fenêtre logarithmique et sont classés dans différents ordres en fonction de leur instant de capture. Ces ordres varient de 1 à $Log(t)$ avec t le temps logique écoulé depuis le début du flux. Trois propriétés caractérisent le système de clichés :

- **Rythme de capture des clichés** : le rythme de prise des clichés est contrôlé par une horloge. Les clichés sont capturés lorsque la valeur de cette horloge est un multiple de α ($\alpha \in N^*$ étant la raison de la suite choisie pour cadencer les instants de captures).
- **Organisation des ordres** : chacune des valeurs de la suite correspond à un ordre donné. Ainsi, les clichés d'ordre i sont ceux capturés à un intervalle de temps α^i.
- **Nombre de clichés par ordre** : chaque ordre peut contenir au maximum $(\alpha^{L_c}+1)$ clichés (avec $L_c > 1$). Le paramètre L_c est déterminé en fonction du volume dédié au résumé. Lorsque le nombre maximum de clichés par ordre est atteint, le plus vieux cliché de cet ordre est supprimé pour libérer la place au plus récent. A l'instant courant t, le nombre total de clichés conservés est $(\alpha^{L_c}+1) * Log_\alpha(t)$ [15].

Ces propriétés engendrent une redondance dans la sauvegarde des clichés. En effet, soit un cliché capturé à l'instant 8. Suivant les définitions précédentes, ce cliché est sauvegardé à l'ordre 0, 1, 2 et 3 pour un $\alpha = 2$. Pour éviter la redondance, on ne garde que le cliché du plus grand ordre.

4.3.1.2 Paramétrage

L'inconvénient de CluStream est son nombre élevé de paramètres. Contrairement à StreamSamp qui requiert trois paramètres, l'algorithme CluStream demande des réglages minutieux de ses paramètres en se basant sur la nature des données. En effet, certains de ses paramètres dépendent de la nature des évènements ainsi que du débit du flux. Nous détaillons dans ce qui suit l'implication de chaque paramètre dans la construction du résumé.

1. **Initialisation des micro-classes :**
   - k : nombre de micro-classes de départ. Pour créer les micro-classes de départ, CluStream utilise un algorithme classique de classification (appelé algorithme d'initialisation). Il faut spécifier les paramètres d'exécution de cet algorithme. Dans le cas de k-means par exemple, il suffit de fixer le nombre de classes k.
   - m : nombre d'événement utilisés pour créer le premier lot de micro-classes.
2. **Maintenance des micro-classes :**
   - N_c : nombre maximum de micro-classes. Étant donné qu'on a une taille de stockage limitée, il est important de spécifier le nombre maximal de micro-classes. Si jamais N_c est atteint, CluStream fusionne deux micro-classes ou supprime la micro-classe la moins active.
   - $\alpha^{L_c}+1$: nombre maximal de clichés par ordre. Dans le système de fenêtre pyramidale, il est important de spécifier le nombre maximum de clichés par ordre. Ce paramètre permet de contrôler l'espace de stockage réservé pour les clichés. α est la raison de la suite qui permet de cadencer les instants de prise de clichés. La suite est choisie en fonction des besoins de l'application. Généralement, la raison de la suite vaut 2.

- Max_{Bound} : limite maximale d'une micro-classe. Lorsque la distance $D(e_i, M_c)$ entre un événement e_i et la micro-classe la plus proche M_c est supérieur à une distance Max_{Bound}, il faut créer une nouvelle micro-classe qui contiendra l'évènement e_i. Ce paramètre est important afin que les événements atypiques ne se mélangent pas avec les événements normaux. Dans la pratique, il est difficile de fixer ce paramètre. Dans [15] aucune indication n'est donnée sur la spécification de ce paramètre.
- S et P : ces paramètres sont utilisés afin de définir la micro-classe la moins active. Pour cela, le système tire aléatoirement un nombre S d'étiquettes temporelles suivant une distribution gaussienne. Si l'étiquette temporelle la plus récente se situe à un facteur P du temps courant alors la micro-classe correspondante peut être supprimée.

Nous étudions dans ce qui suit l'effet de ces paramètres sur le volume occupé, la puissance de calcul et la qualité des résultats obtenus. Le tableau 4.6 résume les effets de ces paramètres sur le résumé de CluStream.

1. **Volume occupé :** pour calculer l'espace occupé par le résumé de CluStream, il est nécessaire d'intégrer la taille des CFV ainsi que l'historique des fusions $Fusion(M_c)$ [31]. Cet historique permet d'identifier les micro-classes qui ont fusionné au cours du temps. Ainsi, le volume total occupé par CluStream à l'instant t se traduit par :

$$V(t) = n_{clichés} * Taille(cliché) \tag{4.25}$$

La taille d'un cliché fait intervenir les CFV de chaque micro-classe (M_c) dans le cliché ainsi que l'historique $Fusion(M_c)$. L'espace ocuupé par l'historique des fusions n'est pas borné et évolue linéairement.

$$Taille(cliché) = N_c * [Taille(CFV(M_c)) + Fusion(M_c)] \tag{4.26}$$

La taille de la structure CFV dépend de la dimensionalité des évènements du flux, elle correspond à la formule suivante :

$$Taille(CFV(M_c)) = (2 * d) + 3 \tag{4.27}$$

avec d : le nombre d'attributs d'un évènement du flux

Ainsi nous avons au final la formule du volume ci-dessous :

$$V(t) = [(\alpha^{L_c} + 1) * log_\alpha t] * (N_c * [Taille(CFV(M_c)) + Fusion(M_c)]) \tag{4.28}$$

Nous constatons que les paramètres qui ont un effet sur le volume occupé par le résumé sont N_c et α^{L_c}. En effet, le choix de α a un effet sur le volume occupé.

31. Chaque micro-classe possède une liste des fusions réalisées.

Comme illustré dans le tableau 4.5, pour un nombre constant de clichés par ordre, calculé à l'instant $t = 10000$, le volume occupé par tous les clichés pour $\alpha = 2$ est beaucoup plus important que celui occupé pour un $\alpha = 5$.

TABLE 4.5 – Effet de alpha sur le volume occupé

α	L_c	nb clichés par ordre	nb clichés à $t = 10000$
2	7	128	1700
5	3	125	715

2. **Puissance de calcul :** le nombre maximal de micro-classes N_c a une grande importance sur la charge du CPU. En effet, pour chaque nouvel évènement du flux, le calcul de distance pour localiser la micro-classe la plus proche est effectué autant de fois que de micro-classes. Le coût est évalué à $O(N_c * d)$. Par ailleurs, la dimensionnalité (d) des évènements du flux joue un rôle important sur la charge du système. En effet, le barycentre d'une micro-classe est un vecteur de taille d. Ainsi, plus d est petit, plus rapide est le calcul de la distance entre un point du flux et les barycentres du système.

3. **Qualité du résumé :** Dans le cadre de CluStream, la qualité des résumés se traduit par les réponses retournées à l'utilisateur en fonction des clichés conservés et par le positionnement des fenêtres.

   (a) Qualité des résultats : la précision des résultats dépend fortement du nombre de clichés conservés. Plus le nombre de clichés est grand, plus précis sont les résultats des requêtes. Par ailleurs, le nombre de micro-classes contribue lui aussi dans cette performance. En effet, plus il y a de micro-classes, meilleure est la représentation temporelle des données vu qu'on aura plus de CFV.

   (b) Positionnement des fenêtres : le choix du paramètre α^{L_c} a un effet sur le nombre total de clichés conservés et par conséquent sur la précision de la période temporelle interrogée. D'autre part, le choix de α a une conséquence sur la dispersion des clichés. En effet, plus la valeur de α est petite, meilleure est la dispersion des clichés sur les ordres.

Nous remarquons suite à cette analyse qu'il existe un compromis entre le volume occupé par le résumé et la qualité des réponses retournées. Ce compromis permet ainsi de régler le choix de α^{L_c} et de N_c. En effet, en augmentant α^{L_c}, on augmente le nombre de clichés capturés, ce qui améliorera la qualité du résumé et provoque un meilleur positionnement des fenêtres.

TABLE 4.6 – Effet des paramètres sur le résumé CluStream

Contrainte	Si α^{L_c} ↗	Si N_c ↗	Si α ↗
Volume occupé	↗	↗	↘
Charge CPU		↗	
Qualité des résultats	↗	↗	↘
Positionnement des fenêtres	↗		↘

4.3.1.3 Persistance des résumés

Comme ce fut le cas pour le résumé de StreamSamp, pour toute analyse portant sur l'historique des flux, il est nécessaire de conserver en mémoire auxiliaire le résumé conçu par CluStream.

La persistance du résumé de CluStream peut être réalisée au moyen de fichiers ou d'une base de données. Cependant, la deuxième approche est à privilégier pour les mêmes raisons définies précédemment dans la section 4.2.2. Cette modélisation doit prendre en considération les contraintes d'intégrité fonctionnelle liées au système [32]. Cependant, quelque soit la modélisation utilisée, il est important de n'enregistrer le cliché qu'à la fin de sa constitution. La mise à jour des CFV doit être effectuée en mémoire centrale étant donné le coût de cette opération. Ainsi, l'état des micro-classes est temporairement conservé en mémoire afin de faciliter toutes les opérations de calcul des distances entre l'événement et les barycentres des micro-classes.

Le schéma relationnel de la Figure 4.9 se caractérise par la présence des relations **"Cliché"**, **"Micro-classe"** et **"Fusions"**. La relation **"Cliché"** enregistre tous les clichés ainsi que l'ordre auxquels ils appartiennent. Cette relation est identifiée par l'estampille temporelle du cliché ("ClichéID") étant donné qu'on est sûr de son unicité.
La relation **"Micro-classe"** contient l'ensemble des micro-classes enregistrées. Chaque enregistrement symbolise l'identifiant de la micro-classe ainsi que le contenu de son CFV. La liste des fusions de chaque micro-classe est enregistrée dans la relation **"Fusions"**. La clé primaire de cette relation est composée de deux attributs. Le premier attribut est l'identifiant de la micro-classe et le second est l'identifiant des classes fusionnées.

Contrairement à StreamSamp, étant donné que les opérations de mise à jour des CFV sont réalisées en mémoire, il n'est pas nécessaire de spécifier d'autres index. En effet, seules les opérations d'insertion et de suppression seront appliquées à la base de données.

32. Exemple de contrainte : vérifier que le nombre de clichés par ordre enregistré dans la relation "Paramètres" est respecté dans la relation "Cliché"

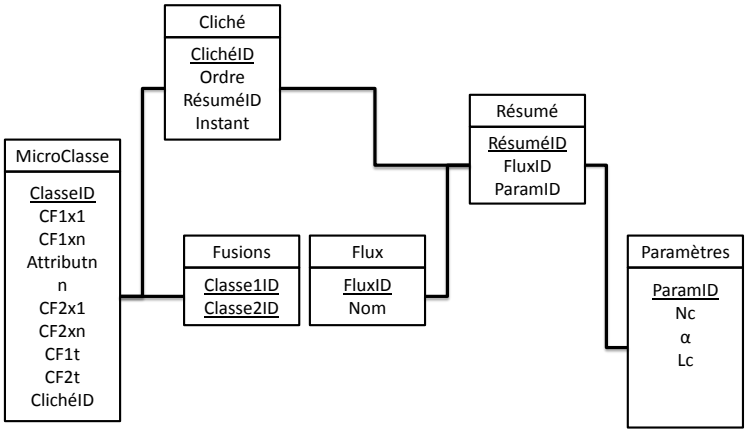

FIGURE 4.9 – Modèle relationnel du résumé conçu par CluStream.

4.3.2 Tâches d'analyses

Contrairement à StreamSamp où les différentes tâches d'analyses sont réalisées sur un échantillon pondéré, dans le résumé de CluStream on conserve la densité des données de détail dans des vecteurs appelés CFV. Les tâches d'analyse sont alors évaluées en utilisant ces structures.

4.3.2.1 Reconstitution de l'horizon temporel

L'avantage de CluStream est qu'il utilise une version évoluée du système de clichés qui permet des opérations de soustraction. Le système est alors flexible il permet, dans la limite du nombre de clichés pris, d'extraire le résumé d'une période quelconque dans l'histoire du flux. Ainsi, pour toute analyse sur la période temporelle $[t_a, t_b]$, le système procède en deux temps. La première étape consiste à localiser les clichés les plus proches des bornes t_a et t_b. La seconde étape consiste à soustraire ces clichés afin de recréer l'état qu'avait le système entre l'instant t_a et l'instant t_b.

1. Pour localiser le cliché le plus proche de la borne t_a (Vs. t_b), la même stratégie définie dans la section 4.2.3.1 est utilisée. La Figure 4.10 illustre le positionnement des bornes de la requête de l'utilisateur par rapport aux instants de capture de clichés. Les clichés choisis pour l'évaluation de la requête doivent être les plus proches de t_a et t_b. Ils doivent alors respecter l'équation suivante :

$$Min(|Log(t_a) - Log(t_1)| + |Log(t_b) - Log(t_2)|) \tag{4.29}$$
$$\text{avec } t_1 \in [t_a^+, t_a^-] \text{ et } t_2 \in [t_b^+, t_b^-]$$

2. Soient C_{t_1} le cliché le plus proche de la borne t_a capturé à l'instant t_1 et, C_{t_2} le cliché le plus proche de la borne t_b capturé à l'instant t_2. La soustraction de ces deux clichés

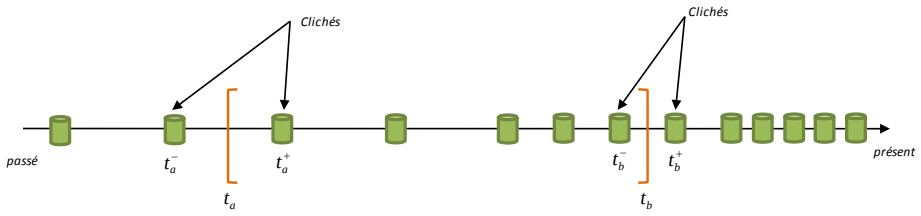

FIGURE 4.10 – Positionnement des bornes de la requête dans un résumé de CluStream.

résume l'évolution du système entre les instants t_1 et t_2. Pour réaliser cette opération de soustraction, le système se base sur les informations conservées dans les clichés, à savoir les CFV et l'historique des fusions des micro-classes. Généralement, la stratégie de soustraction de deux clichés est la suivante : le CFV de chaque micro-classe MC_i du cliché C_{t_1} est soustrait du CFV de la micro-classe correspondante $MC_{i'}$ dans le cliché C_{t_2} (i.e. l'identifiant de MC_i doit être localisé dans la liste d'identifiant de $MC_{i'}$). Cependant, quatre scénarios de base (liste non exhaustive) peuvent être présentés :

- **Scénario a) :** il s'agit du cas le plus simple où une micro-classe est créée à l'instant t_1 et a absorbé de nouveaux événements jusqu'à atteindre un nouvel état (nouveau CFV) à l'instant t_2 (cf. Figure 4.11 (a)). Le CFV de la micro-classe MC_{t1} est soustrait du CFV de la micro-classe MC_{t2}. Soit $CFV_{t_1} = (n_1, \overline{CF1^x}(t1), \overline{CF2^x}(t1), CF1t(t1), CF2t(t1))$ le vecteur associé à la micro-classe MC_{t1} à t_1 et $CFV_{t_2} = (n_1, \overline{CF1^x}(t2), \overline{CF2^x}(t2), CF1t(t2), CF2t(t2))$ celui associé à MC_{t2} à l'instant t_2. La soustraction de CFV_{t_1} et de CFV_{t_2} est la suivante : $n_2 - n_1, \overline{CF1^x}(t2) - \overline{CF1^x}(t1), \overline{CF2^x}(t2) - \overline{CF2^x}(t1), CF1t(t2) - CF1t(t1), CF2t(t2) - CF2t(t1)$.
- **Scénario b) :** il s'agit du cas le plus général où plusieurs micro-classes ont été créées à t_1. Celles-ci évoluent et, pour libérer de l'espace elles fusionnent en une seule micro-classe à l'instant t_2 (cf. Figure 4.11 (b)). Le résultat de l'opération de soustraction à t_2 est une nouvelle micro-classe définie en soustrayant le vecteur CFV_{t_2} du vecteur CFV_{t_1} qui est constitué à partir de la somme des $CFV_{t_1}^{(i)}$ de toutes les micro-classes qui ont fusionné dans la micro-classe considérée à l'instant t_1. Une nouvelle information s'ajoute à la micro-classe issue de la soustraction. Il s'agit d'une liste permettant de conserver les identifiants des micro-classes mères. Ces identifiants sont conservés dans la relation **"Fusions"** du schéma relationnel.
- **Scénario c) :** il s'agit du premier cas particulier du système où la micro-classe créée figure dans le cliché pris à l'instant t_1 mais ne figure pas dans celui pris à l'instant t_2 (cf. Figure 4.11 (c)). En effet, pour libérer de l'espace une des stratégies du système CluStream est de supprimer la micro-classe qui n'a pas été active pendant une certaine période. Ainsi, cette micro-classe ne fera pas partie du résultat de la suppression. Ceci présente un problème étant donné qu'il y a eu une perte

d'évènements entre les instants t_1 et t_2. Le nombre total d'évènements absorbés par le système à l'instant t_1 est supérieur au nombre d'évènements absorbés à l'instant t_2, ce qui est problématique.

– **Scénario d) :** il s'agit du second cas particulier de CluStream où la micro-classe est présente dans le cliché pris à l'instant t_2 mais ne figure pas dans celui pris à l'instant t_1 (cf. Figure 4.11 (d)). En effet, cette micro-classe a été constituée après la capture du cliché de C_{t_1} et supprimée avant la capture du cliché de C_{t_2}.

Ainsi, hormis les deux derniers scénarios qui ne sont pas fréquents, le résumé de CluStream permet de conserver l'état réel du système à différents instants temporels.

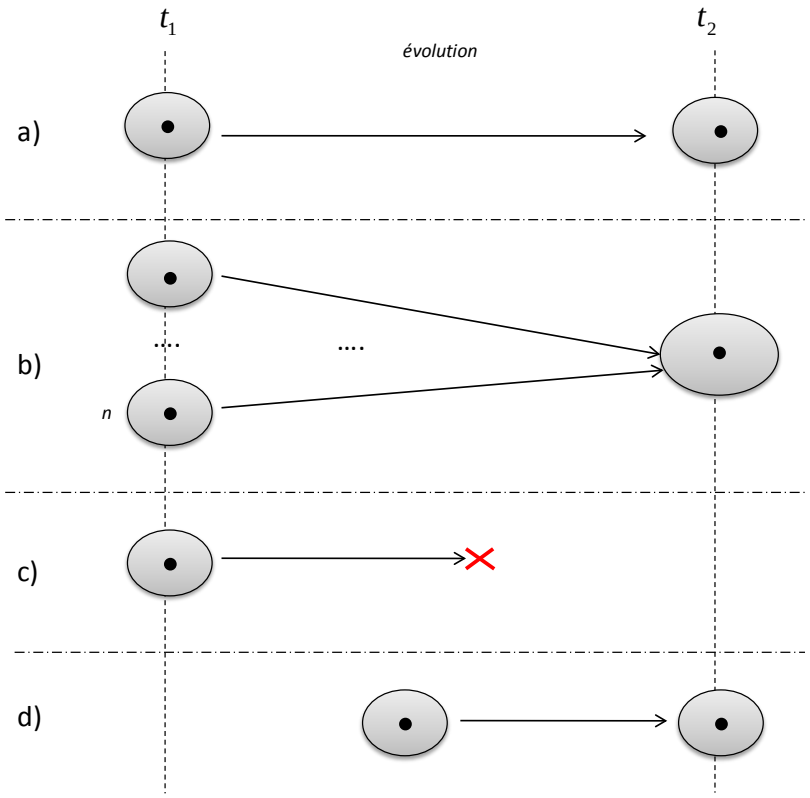

FIGURE 4.11 – Scénarios possibles lors de la soustraction des micro-classes dans les clichés.

4.3.2.2 Requêtes d'agrégats

Dans cette section, notre intérêt porte sur l'évaluation des requêtes d'agrégats sur le résumé de CluStream au niveau de la période $[t_a, t_b]$. Nous supposons qu'on a conservé deux clichés aux bornes de cet intervalle. L'évaluation des requêtes est réalisée sur l'état du système obtenu par soustraction de deux clichés sur les bornes de la période temporelle de la requête $[t_a, t_b]$. Ainsi, en nous basant sur les statistiques conservées dans les CFV, il est trivial de calculer de façon exacte les agrégats suivants : effectif, moyenne et variance

sur la période temporelle [t_a, t_b]. Pour cela, comme on a conservé un cliché C_{t_a} à l'instant t_a et un autre C_{t_b} à t_b. La soustraction de ces deux clichés (C_{t_a}-C_{t_b}) permet d'extraire les CFVs des N_c micro-classes CFV_1, CVF_2, ..., CFV_{N_c} et de calculer par conséquent ces agrégats :

- L'effectif total sur cette période est déterminé par : $n_t = \sum_{i=1}^{N_c} n_i$
- La moyenne sur [t_a, t_b] est donnée par : $\overline{X}_{[t_a,t_b]} = \frac{1}{n_t} \sum_{i=1}^{N_c} \overline{CF1}^{(i)}_{[t_a,t_b]}$
- La variance est calculée par : $\sum_{i=1}^{N_c} (\overline{CF2}^{(i)}_{[t_a,t_b]}) - \overline{X}^2_{[t_a,t_b]}$

Cependant, comme défini précédemment, si des clichés ne sont pas conservés aux instants t_a et t_b alors, on se base sur les clichés les plus proches de ces instants pour calculer les agrégats.

Calcul de la médiane. Dans le cadre du résumé de CluStream, l'estimation de cet agrégat suit trois étapes :

1. Préparation des données. L'idée consiste à calculer à partir des CFV inclus dans la période [t_a, t_b] les barycentres des micro-classes. Ces barycentres sont les moyennes $\overline{X}$ calculés au niveau de chaque micro-classe. Pour le calcul de la médiane, nous utilisons l'effectif (noté n_i) observé au sein de la micro-classe (MC_i) dans le résumé de [t_a, t_b] comme poids associés au barycentres.
2. Tri des barycentres. Il s'agit de trier les barycentres inclus dans la période [t_a, t_b] sur la variable choisie ;
3. Calcul de la médiane. En nous basant sur le poids associé à chaque barycentre, le calcul de la médiane est l'évènement p qui satisfait la formule suivante :

$$\underset{p}{argmin} \left| \sum_{\substack{i=1..N_c \\ rang(i) \leq rang(p)}} n_i - \frac{1}{2} \sum_{i}^{N_c} n_i \right| \tag{4.30}$$

4.3.2.3 Tâches de fouille

Classification non supervisée. CluStream a été initialement développé pour traiter des tâches de classification non supervisée. En effet, sa fonctionnalité principale consiste à effectuer un clustering des micro-classes en considérant, comme poids, le nombre d'évènements absorbés par la micro-classe.

Classification supervisée. Pour pouvoir réaliser une classification supervisée sur le résumé de CluStream, il est nécessaire de procéder à une étape de préparation des données dans laquelle on reconstitue, en fonction des statistiques conservées dans chaque micro-classe, les événements qui représentent la partie du flux entre [t_a, t_b].

1. Préparation des données. CluStream ne permet de traiter que les données quantitatives ($\in R^p$). Ainsi, si on des données qualitatives, il faut les coder en disjonctifs complets. Cette étape de préparation des données est très importante, elle rend possible la construction d'un modèle sur le résumé généré par CluStream. Pour cela, la méthode du tableau disjonctif est utilisée pour représenter les données. L'idée consiste à représenter chaque label (variable à prédire) par un attribut. Pour chaque évènement, la valeur de l'attribut à prédire sera remplacée par une suite binaire dont la valeur 1 correspond à l'attribut concerné et 0 pour les autres. La Figure 4.12 illustre un exemple de transformation de l'attribut Label pouvant intégrer trois valeurs distinctes (A, B ou C).

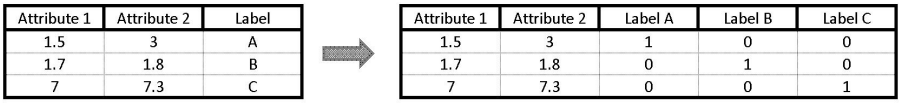

FIGURE 4.12 – Exemple de tableau disjonctif.

Par la suite, on génére aléatoirement des évènements suivant la distribution des caractéristiques des micro-classes. Pour chaque micro-classe, on génère n_i points avec n_i le nombre de points observés dans la micro-classe. (MC_i) Les données générées (pour chaque attribut) suivent une distribution de loi normale ayant comme paramètres la moyenne et la variance de la micro-classe (loi multi-normale à variables indépendantes). Cette façon d'utiliser CluStream comme modèle génératif de données, exploite toute l'information contenue dans le résumé : on ne pourrait pas faire plus "fin".

2. Constitution du modèle. Une fois les données régénérées, un algorithme de classification supervisée est appliqué afin de construire le modèle.
3. Évaluation du modèle. la dernière étape consiste à évaluer le modèle construit sur le résumé de CluStream. Pour cela, une méthode classique d'évaluation est utilisée, par exemple pour calculer l'erreur de classification sur un échantillon test.

4.4 Expérimentations

Nous présentons dans cette section une série d'évaluations permettant de comparer d'une part StreamSamp et CluStream et d'autre part StreamSamp avec son extension StreamSamp+. Ces évaluations comparent StreamSamp et CluStream sur différentes portions du flux, afin d'étudier la robustesse de chacun face aux variations des données au cours du temps. Notre intérêt porte sur l'évaluation des résultats obtenus pour une fenêtre fixe évaluée à différents instants. Notre objectif est d'étudier les performances des algorithmes dans l'estimation de requêtes sur une période qui vieillit au cours du temps. Pour l'expérimentation sur la classification supervisée, le jeu de données utilisée est ***Cover Type***

(cf. Annexe A.2). Cependant, pour le reste des expérimentations, le jeu de données utilisé est ***KDD99*** (cf. Annexe A.1). Les tableaux 4.7 et 4.8 illustrent les paramètres utilisés pour l'algorithme CluStream et les algorithmes d'échantillonnage. Ces paramètres ont été définis en vue d'obtenir un même volume de résumé pour tous les algorithmes utilisés.

CluStream
N_c : nombre de micro-classes = 50
$\alpha^{L_c} + 1$: nombre de clichés par ordre = 17 ($\alpha = 2$, $L_c = 4$)
m : nombre d'évènements pour l'initialisation = 2000
Algorithme intial de classification = $K - means$
Nombre d'itérations = 10

TABLE 4.7 – Paramètres de l'algorithme CluStream.

StreamSamp / StreamSamp+
$\alpha = 1$
$T = 500$ évènements par échantillon
$L = 8$ échantillons par ordre

TABLE 4.8 – Paramètres des algorithmes d'échantillonnage progressif.

4.4.1 Vitesse d'exécution et volume occupé

Dans le contexte des flux de données, le temps d'exécution nécessaire pour la construction des résumés est un critère important. Nous nous intéressons au temps global écoulé pour le traitement de la totalité du flux de données par les algorithmes.

La Figure 4.13 montre les performances en termes de temps d'exécution (échelle logarithmique). Il est clair que StreamSamp fournit de meilleurs résultats comparé à CluStream. En effet, l'algorithme StreamSamp ne se base que sur le processus de ré-échantillonnage récursif tandis que CluStream est fortement ralenti par le calcul des distances entre les évènements et les barycentres ainsi que les mises à jour des CFV.

StreamSamp est de même plus rapide que son extension (l'algorithme StreamSamp+). En effet, les mises à jour des structures SFV nécessitent plus de temps et ralentissent par ailleurs le traitement de l'intégralité du flux. Cependant, les deux algorithmes d'échantillonnage restent beaucoup plus rapides que CluStream.

Nous étudions sur le même jeu de données l'évolution des volumes pour les résumés de StreamSamp et CluStream. Les Figures 4.14 et 4.15 illustrent l'évolution du volume du résumé de StreamSamp. Les volumes minimum et maximum sont représentés en fonction des équations de la section 4.2.1.2. Nous observons que le volume occupé par StreamSamp

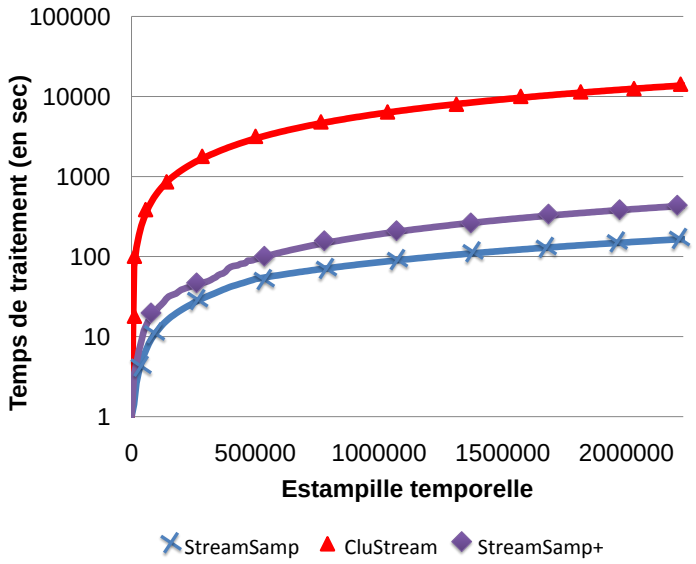

FIGURE 4.13 – Temps nécessaire aux trois algorithmes pour le traitement de l'intégralité du flux (échelle logarithmique).

croît linéairement en fonction de l'échelle logarithmique du temps. De même, l'écart entre le volume minimum et maximum croît linéairement avec le volume.

$$V_{max} - V_{min} \approx (L - 1/2) * (V_{max} + V_{min})/2 \tag{4.31}$$

Le choix de l'organisation temporelle des données joue un rôle important dans l'évolution du volume occupé. De plus, nous observons un comportement oscillatoire du volume occupé par StreamSamp. Ceci est causé par le processus de ré-échantillonnage qui permet : (i) de vider les ordres récents afin de remplir les ordres les plus anciens, (ii) de propager le résumé sur le passé.

D'autre part, la Figure 4.14 illustre le volume occupé par le résumé de CluStream au cours du temps. Nous observons que l'évolution du volume croît logarithmiquement en fonction en t. Cette évolution est dûe à la structure utilisée pour représenter les données. Nous avons de même calculé le volume occupé en prenant en considération l'historique des fusions des micro-classes (cf. Figure 4.14 courbe rouge). Le volume occupé par l'historique croit lui aussi de manière logarithmique. Avec l'ancienneté du résumé, cet historique prend de l'importance.

Les volumes occupés par les deux résumés croient logarithmiquement cependant, la vitesse de croissance n'est pas équivalente. En effet, le résumé de StreamSamp se propage rapidement et occupe le volume dédié alors que celui de CluStream met plus de temps pour occuper le même volume que StreamSamp. Ceci est dû aux processus qu'utilisent les deux techniques pour répartir les évènements du flux : alors que le processus de ré-

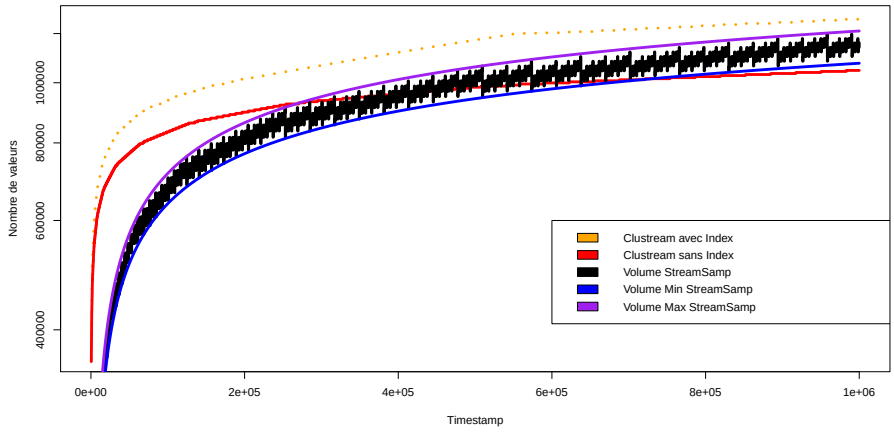

FIGURE 4.14 – Volume occupé par les algorithmes StreamSamp et CluStream.

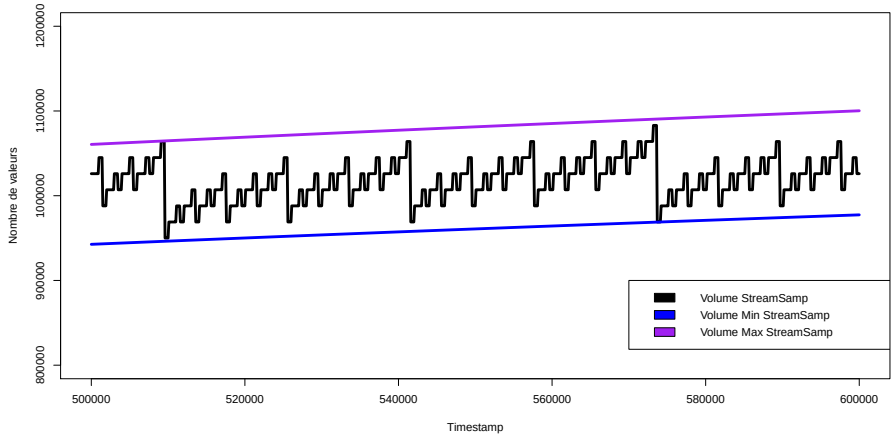

FIGURE 4.15 – Volume occupé par StreamSamp (Zoom sur la période [5e+05, 6e+05]).

échantillonnage de StreamSamp est très rapide, CluStream utilise un processus beaucoup plus lent (i.e. processus de classification).

4.4.2 Évaluation des requêtes d'agrégat

Contrairement aux évaluations précédentes où notre objectif était le traitement de la totalité du flux, nous étudions dans ce qui suit les performances des résumés face à l'évaluation d'une même période dont le résumé vieillit du fait de l'arrivée de nouveaux évènements. Nous étudions la qualité des réponses retournées pour l'évaluation des requêtes d'agrégats sur la période [0,50000]. Cette évaluation est réalisée à différents instants temporels. Pour cela, nous évaluons les performances des algorithmes à estimer la moyenne et la médiane sur

une fenêtre fixe qui vieillit au cours du temps. Étant donné que les algorithmes StreamSamp et StreamSamp+ sont basés sur des techniques d'échantillonnage, les estimations sont répétées 100 fois et, la moyenne des résultats est prise en compte. Pour CluStream, nous présentons deux types de résultat. Un résultat obtenu en gardant un cliché au début du flux et un cliché à l'instant 50000. Un autre résultat est obtenu sans conservation préalable des clichés. Dans tel cas, on se base sur les clichés les plus proches de la période $[0, 50000]$ pour évaluer la requête.

4.4.2.1 Évaluation de la moyenne

Cette évaluation consiste à étudier l'évolution de l'estimation de la moyenne pour une période temporelle de plus en plus ancienne. Cet agrégat est calculé pour chaque variable d'intérêt à partir de l'échantillon final représentant la période [0,50000]. Afin d'évaluer le résultat obtenu nous utilisons l'erreur moyenne relative [33]. La valeur réelle de la moyenne sur la période [0,50000] est calculée à partir des données d'origine (i.e. les données incluses sur la période [0,50000]).

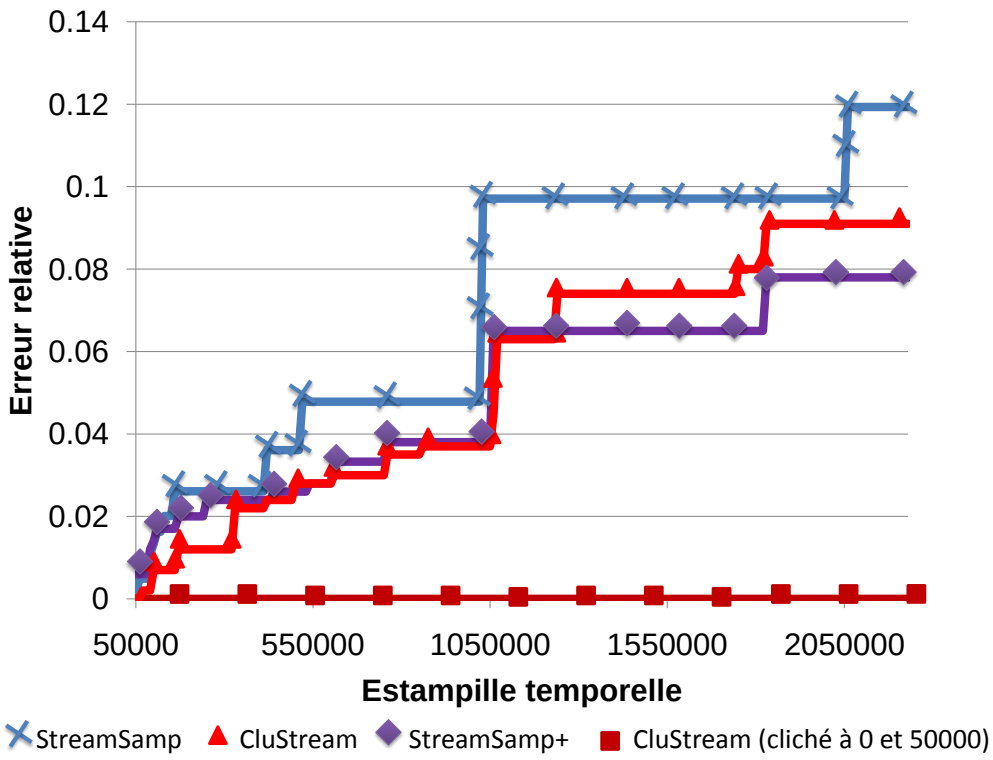

FIGURE 4.16 – Erreur relative sur la moyenne sur la période [0, 50000] évaluée à différents instants.

33. L'erreur relative d'un estimateur est $|valeur_{approchée} - valeur_{réelle}| / valeur_{réelle}$.

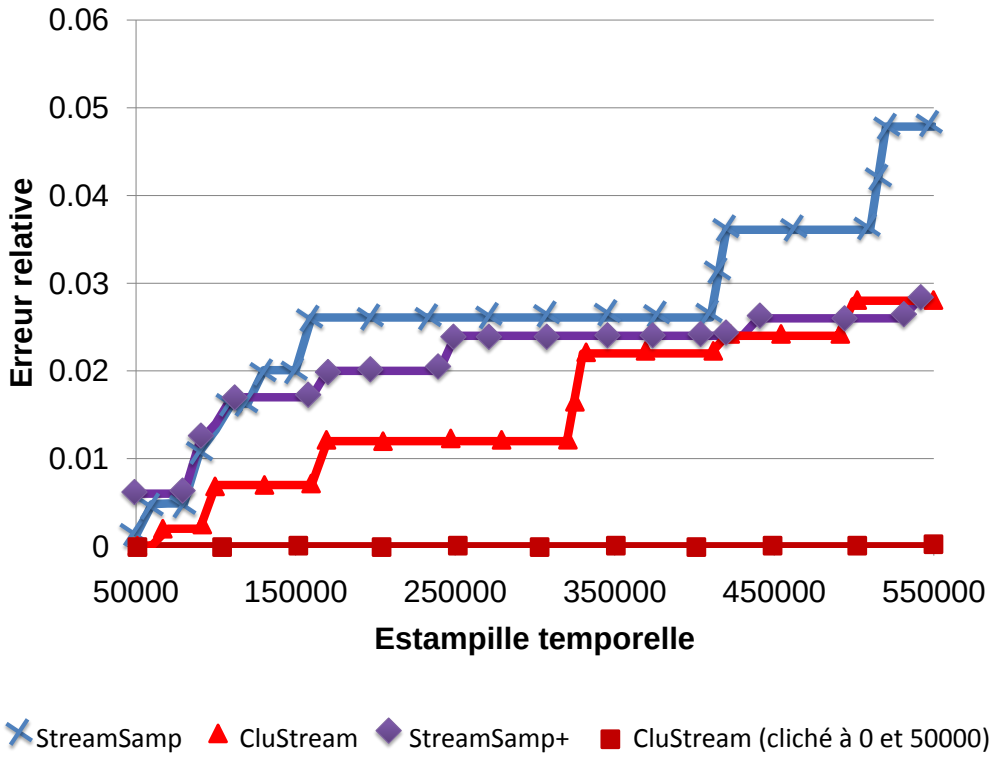

FIGURE 4.17 – Erreur relative sur la moyenne sur la période [0, 50000] évaluée à différents instants (Zoom sur la période [50000,550000]).

Les Figures 4.16 et 4.17 illustrent les résultats obtenus pour l'évaluation de la moyenne. La période a été évaluée de l'instant t=60000 à l'instant t=2300000. Les résultats obtenus confirment la dégradation des performances de StreamSamp et StreamSamp+ au cours du temps. En effet, nous constatons que l'erreur relative moyenne augmente avec l'ancienneté de la période évaluée. L'estimation de l'agrégat devient difficile pour le passé lointain étant donné que peu d'évènements portant sur la période interrogée sont conservés dans les échantillons (i.e. conséquence logique du processus de ré-échantillonnage). Cependant, malgré la dégradation des performances de StreamSamp+ au cours du temps, celui-ci présente de meilleurs résultats que son prédécesseur. Ces performances s'expliquent par l'utilisation d'une structure SFV permettant la mise à jour des statistiques du flux 4.2.5.

Les performances fournies par l'algorithme CluStream sont comparables à celles des techniques d'échantillonnage pour le passé proche. Toutefois, les performances se dégradent au cours du temps pour les trois algorithmes (i.e. suppression d'échantillons pour les techniques d'échantillonnage et suppression de clichés pour CluStream). Malgré cette dégradation, CluStream présente de meilleurs résultats pour le passé lointain. Ceci s'explique par le caractère spatio-temporel des micro-classes donc leur sélection permet de conserver une moyenne sur les périodes.

4.4.2.2 Évaluation de la médiane

Nous étudions dans cette évaluation les performances des algorithmes lors de l'estimation de la médiane. Nous calculons l'erreur d'estimation de la médiane comme suit :

$$erreur = \frac{\left| \left(\sum_{\substack{i \in N \\ i|rang(i) \leq rang(p^*)}} w_i \right) - \frac{1}{2} \sum_{i \in N} w_i \right|}{\sum_{i \in N} w_i} \tag{4.32}$$

avec N : la population (dans StreamSamp $N = 50000$, dans CluStream $N = N_c$ le nombre de micro-classes). w_i le poids d'un évènement d'indice i dans StreamSamp. Pour Clustream, $w_i = n_i$) (l'effectif total de la micro-classe i), P la position de la valeur réelle de la médiane, P^* la position de la valeur estimée de la médiane.

$$p^* = \overset{argmin}{p} \left| \left(\sum_{\substack{i \in N \\ rang(i) \leq rang(p)}} w_i \right) - \frac{1}{2} \sum_{i \in N} w_i \right| \tag{4.33}$$

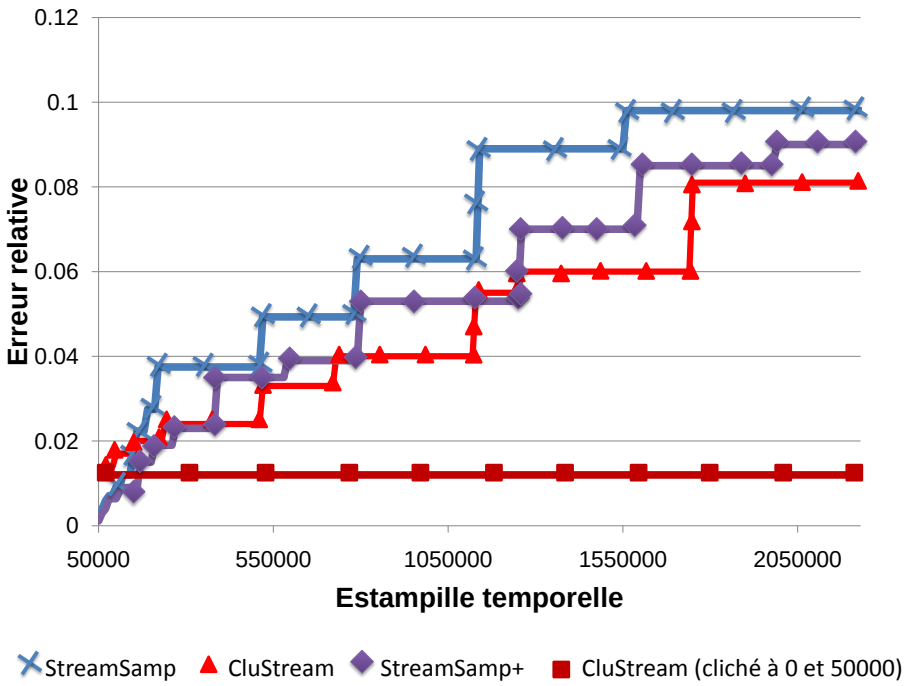

FIGURE 4.18 – Évaluation de la médiane sur la période [0, 50000] à différents instants temporels.

Les Figures 4.18 et 4.19(zoom sur la période [50000,550000]) décrivent les résultats obtenus pour l'évaluation de la médiane. StreamSamp et StreamSamp+ présentent des

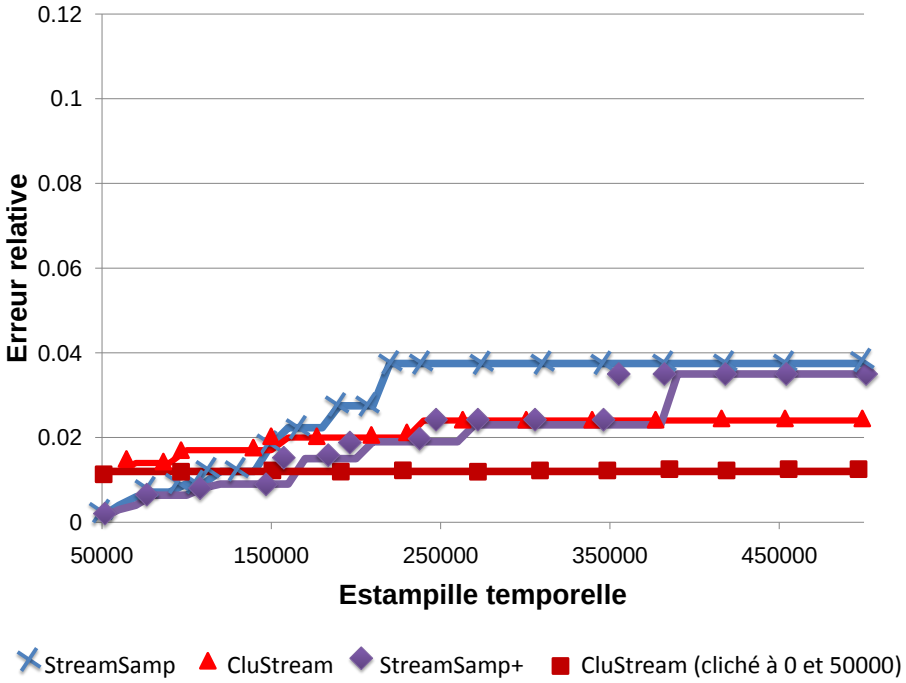

FIGURE 4.19 – Évaluation de la médiane (Zoom sur la période [50000, 500000]).

comportements similaires pour l'évaluation de cet agrégat. En effet, ces deux algorithmes présentent une bonne estimation comparé à CluStream pour le passé proche mais se dégradent avec l'ancienneté de la période. Par ailleurs, nous observons que StreamSamp+ présente des performances légèrement meilleures que StreamSamp étant donné qu'il offre plus de précision dans le positionnement des fenêtres.

La médiane est un agrégat robuste à l'échantillonnage, ce qui explique les résultats de StreamSamp et StreamSamp+ pour le passé proche. Toutefois, l'échantillonnage récursif finit par dégrader leurs performances. Étant donné que l'algorithme CluStream ne conserve aucune information sur la médiane, les résultats de celui-ci sont médiocres (comparés à ceux de StreamSamp et StreamSamp+) pour le passé proche. Cependant, avec l'ancienneté de la période, la dégradation des algorithmes d'échantillonnage est beaucoup plus rapide que celle de CluStream. Ce qui explique que, pour le passé lointain, les performances de ce dernier sont meilleures.

4.4.3 Évaluation sur les tâches de fouille

Nous étudions dans cette partie les performances des algorithmes de résumé pour les tâches de classification sur la période [0, 50 000] évaluée à des instants différents.

4.4.3.1 Évaluation de la classification non supervisée

Pour réaliser la classification non supervisée sur les résumés, l'algorithme k-means est utilisé. Pour évaluer la qualité de la classification, nous comparons le clustering réalisé sur les différents résumés, au clustering réalisé en utilisant tous les évènements de la période $[0, 50000]$. Pour cela, nous avons utilisé l'inertie intra-classe moyenne. Cette mesure est la somme des carrés des distances de tous les points au barycentre. La division de cette somme par l'effectif total présent dans l'ensemble de test permet d'obtenir l'inertie intra-classe moyenne.

Dans le cadre de la classification automatique, l'algorithme k-means est utilisé en fixant le même nombre de classes pour les algorithmes. Les trois algorithmes sont lancés avec avec 15 micro-classes. L'inertie intra-classe moyenne est obtenue en utilisant comme ensemble d'apprentissage l'échantillon final sur la période $[0, 50000]$ pour les résumés de StreamSamp et StreamSamp+ et, l'ensemble des barycentres conservés sur cette période pour CluStream.

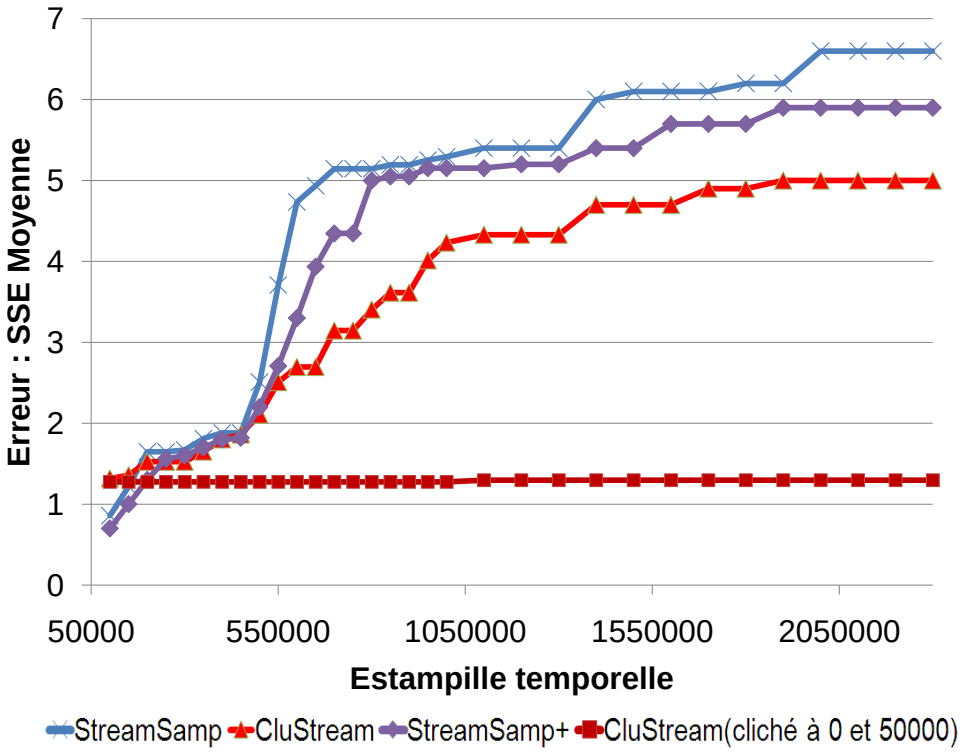

FIGURE 4.20 – Étude de l'évolution de l'inertie intra-classe moyenne pour la période [0, 50000] évaluée à différents instants temporels.

Comme le montre la Figure 4.20, les trois algorithmes présentent des résultats assez similaires sur les données du passé proche avec un léger avantage pour les algorithmes d'échantillonnage. Cependant, avec l'ancienneté, les performances se dégradent. La dé-

gradation causée par le processus du ré-échantillonnage récursif est plus rapide que la dégradation causée par la suppression de clichés pour CluStream. En effet, concernant les algorithmes StreamSamp et StreamSamp+, l'évolution des performances au cours du temps s'explique par la diminution du nombre d'évènements sur la période [0 - 50000] utilisés pour réaliser les 15 classes du k-means. En revanche, la dégradation des performances de CluStream s'expliquent d'une part par la perte de précision dûe à la suppression de certains clichés mais aussi par une perte de précision lors du positionnement des bornes de la requête. En effet, la perte de cliché provoque le recalibrage des bornes de la requêtes en fonction des clichés conservés. Plus on perd de clichés, plus on s'éloigne des bornes de la requête.

4.4.3.2 Évaluation de la classification supervisée

Dans cette section nous comparons les performances des modèles conçus à partir des résumés de StremSamp, StreamSamp+ et CluStream.
Tout d'abord, ce jeu de données ne contient pas d'attributs temporels. Pour le transformer en un flux de données, nous avons ajouté aléatoirement des estampilles temporelles incrémentales pour chaque évènement. Cette information est uniquement ajoutée lors du processus de création des résumés cependant, avant la création des modèles, l'attribut temporel est supprimé des résumés.
D'autre part, le jeu de données Cover Type (cf. Annexe A.2) contient 7 labels, prédire ces labels est une tâche assez difficile c'est pourquoi nous avons transformé le jeu à deux labels : la classe majoritaire (classe A) et toutes les autres (classe B). Les modèles sont par la suite construits sur les résumés de la période [0, 50000] comme indiqué précédemment (dans la section 4.3.2.3) en prenant en considération les structures des résumés (i.e. échantillons pondérés pour StreamSamp et l'ensemble de données généré à partir des statistiques conservées dans les micro-classes pour l'algorithme CluStream)(cf. section 4.3.2.3). Pour la construction des arbres de décision, la méthode C4.5 a été utilisée et, pour évaluer les modèles, la méthode de validation croisée [30].

Comme le montre la Figure 4.21, nous distinguons trois périodes pour lesquelles les performances des algorithmes varient. La période 1 est définie sur l'intervalle [50000, 100000]. La période 2 est définie sur l'intervalle [100000, 350000]. La période 3 est définie sur l'intervalle [350000, 500000].

Tout d'abord, nous observons que sur la période 1, StreamSamp et StreamSamp+ présentent les meilleures performances. Ces résultats sont prévisibles étant donné que les résumés conservent : (i) la structure des données d'origine et, (ii) des évènements "réels" du flux. Toutefois, avec le vieillissement de la période, le processus de ré-échantillonnage détruit la précision des échantillons d'où une dégradation des performances observée sur les périodes 2 et 3. Contrairement aux algorithmes d'échantillonnage, CluStream utilise des données générées calculées sur la base des statistiques conservées pour représenter

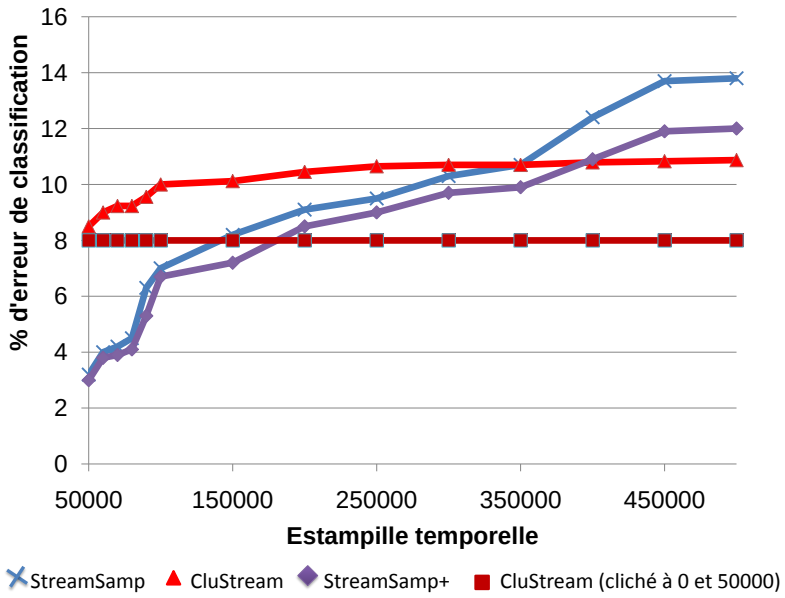

FIGURE 4.21 – Résultats pour la tâche de classification supervisée sur la période [0, 50000].

les évènements du flux sur la période [0, 50000], ce qui explique une dégradation de ses performances dès les premières évaluations (sans conservation de clichés).

Concernant la période 2, CluStream présente les meilleurs résultats s'il conserve des clichés au début et à la fin de la période.

Durant la troisième période, les performances de CluStream (sans conservation de clichés) se stabilisent alors que le processus de ré-échantillonnage de StreamSamp et StreamSamp+ continue d'altérer la qualité des échantillons causant ainsi une dégradation rapide dans les performances des algorithmes d'échantillonnage.

4.4.4 Discussion

Les expérimentations réalisées ont révélé un comportement reproductible des performances des algorithmes quelque soit l'analyse effectuée. Nous observons trois phases distinctes au cours desquelles les algorithmes adoptent des comportements différents. La première phase représente les évaluations sur le passé proche pour lesquelles les algorithmes d'échantillonnage progressif occupent la première place en terme de performance. La troisième phase regroupe les instants du passé lointain pour lesquels CluStream prend la main sur les performances réalisées. Sur ces deux phases, nous distinguons clairement un algorithme par rapport à un autre ce qui n'est pas le cas des évaluations sur la période intermédiaire où parfois StreamSamp est meilleur (e.g. cas de la classification supervisée), d'autres fois CluStream (e.g. cas de la médiane). En effet les performances des algorithmes sur cette période dépendent fortement de l'analyse réalisée. Nous interprétons dans ce qui

suit les résultats observés au cours de ces expérimentations. Pour cela, nous étudions les facteurs qui influencent la qualité des résumés.

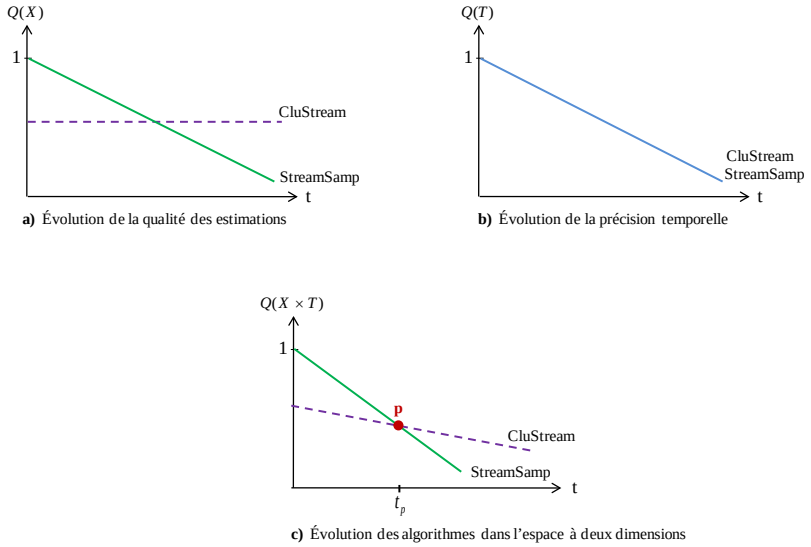

FIGURE 4.22 – Évolution des performances des algorithmes au cours du temps.

Les performances d'un algorithme de résumé dépendent de deux facteurs : (i) la disposition temporelle des données et, (ii) leur disposition dans l'espace de représentation. On parle ainsi d'un espace à deux dimensions : l'espace du temps ($\mathbb{X}$) et, l'espace des valeurs ($\mathbb{T}$). Le premier concerne le positionnement des fenêtres et, le second concerne la technique utilisée pour représenter les données. La qualité des résumés diffère en fonction de ces facteurs.

- Sur l'espace du temps ($\mathbb{T}$) : les algorithmes CluStream et StreamSamp adoptent un système de fenêtre logarithmique (fenêtres inclinées). Ce système génère une erreur exponentielle [34] sur $\mathbb{T}$ (cf. Figure 4.22 b). Cette erreur engendre par conséquent une dégradation rapide des performances au cours du temps.
- Sur l'espace des données ($\mathbb{X}$) : contrairement à l'espace temporel où les algorithmes adoptent la même stratégie pour le traitement du temps, la représentation des données diffère d'un algorithme à un autre.

  Dans le cas de CluStream, il y a une dégradation initiale des estimations mais qui reste constante au cours du temps (cf. Figure 4.22 a). En effet, lorsqu'un cliché est sauvegardé, il représente l'état du système à un instant donné et, tant que ce cliché n'est pas supprimé du résumé, il reste toujours représentatif (i.e. les clichés ne se dégradent pas avec l'ancienneté de la période). Cependant, la dégradation

34. Du fait que les fenêtres inclinées suivent une échelle logarithmique, la précision retournée pour une période temporelle évaluée décroît exponentiellement.

de CluStream observée pour toute analyse sur le passé proche s'explique par une dégradation liée au volume dédié. Plus on augmente le nombre de micro-classes N_c, meilleurs sont les résultats retournés.

Dans le cadre de StreamSamp, la dégradation de la qualité des résultats est progressive, il s'agit de la conséquence logique liée à l'effet du ré-échantillonnage. En dépit de cette dégradation, les premières estimations sont de bonne qualité (erreur initiale nul pour un $\alpha = 1$). Nous entendons par premières évaluations, l'exécution de l'algorithme de la requête sur les échantillons d'ordre 0.

D'un point de vue général, CluStream est contraint par un seul facteur (le positionnement des fenêtres de la requête sur le système de fenêtres inclinées), ce qui explique sa dégradation au cours du temps. En revanche, la dégradation de StreamSamp est influencée par les deux facteurs (le positionnement des fenêtres et la représentation des données). La combinaison des deux espaces fournit le résultat suivant (cf. Figure 4.22 **c)**) : les performances de StreamSamp se dégradent progressivement, elles se caractérisent par une précision initiale de bonne qualité mais, une vitesse de dégradation importante. Tandis que CluStream se caractérise par une précision moindre à celle de StreamSamp mais, une vitesse de dégradation plus lente. Par conséquent, les performances des algorithmes se dégradent à vitesse différente provoquant un croisement des courbes au point p. Il s'agit du point à partir duquel les performances de CluStream vont être meilleures que celles de StreamSamp. Ce point de transit se situe dans la période intermédiaire. Il dépend du jeu de données et des tâches à réaliser.

Ainsi, chacun de ces deux algorithmes présente des avantages et des inconvénients. Dans le chapitre suivant, nous proposons de combiner le meilleur des deux afin de produire une approche hybride.

4.5 Synthèse

L'objectif de ce chapitre consistait à étudier en profondeur les caractéristiques des algorithmes StreamSamp et CluStream (e.g. paramètres, évolution des résumés, etc.). Les expérimentations ont révélé la robustesse de l'échantillonnage progressif lors de l'évaluation des requêtes et des tâches de fouille sur le passé proche. Cependant, ces performances se dégradent en fonction de l'ancienneté de la période. En revanche CluStream montre un comportement assez stable au cours du temps, il ne présente pas des performances meilleures que StreamSamp pour le passé proche mais reste plus performant pour le passé lointain. Un résumé plus stable serait de tirer profit des avantages de ces approches. L'approche hybride, décrite dans le chapitre suivant, vise à combiner StreamSamp et CluStream de façon

à obtenir le meilleur des deux algorithmes. Elle permet par ailleurs de réduire le temps de calcul et de traitement des évènements du flux observé avec l'algorithme CluStream.

Par ailleurs, nous avons étudié dans ce chapitre l'exploitation statique des résumés aux différentes tâches d'analyses. Adapter l'algorithme de la requête à la structure de résumé n'est pas toujours trivial. Nous discutons dans le chapitre 6 une approche plus dynamique permettant de pallier à ces difficultés d'adaptation.

Chapitre 5

Résumé Hybride de flux de données

Sommaire

5.1 Introduction

Dans le chapitre précédant, nous avons discuté la conception et la réalisation de deux algorithmes généralistes permettant de résumer les flux de données : StreamSamp [61] et CluStream [15]. L'étude réalisée a permis de les comparer et de conclure par le fait qu'ils présentent des comportements opposés : alors que StreamSamp est performant pour les évaluations réalisées sur le passé proche, CluStream est meilleur pour le passé lointain. Ces

deux processus semblent être complémentaires, les combiner revient à concevoir un résumé qui fournit des résultats de bonne qualité pour le passé proche ainsi que le passé lointain.

Nous nous intéressons dans ce chapitre à la conception d'une nouvelle approche de résumé, appelée "*approche hybride*". L'intention sur laquelle se base cette technique consiste à coupler les avantages de StreamSamp et ceux de CluStream afin de produire un résumé généraliste. Nous rappelons dans un premier temps le comportement de ces algorithmes au cours du temps. Nous étudions par la suite le principe de l'approche hybride et nous présentons la structure du résumé conçu par ce processus ainsi que l'effet des paramètres sur le résumé. Enfin, nous établissons une étude comparative entre les performances de l'approche hybride et ceux des algorithmes StreamSamp et CluStream.

5.2 Performances de StreamSamp et CluStream

L'étude réalisée dans le chapitre 4 a révélé les forces et les faiblesses des algorithmes StreamSamp et CluStream (synthétisés dans le tableau 5.1). Comme nous l'avons discuté dans la section 4.4.4, la précision de CluStream sur l'espace ($\mathbb{T} \times \mathbb{X}$) est influencée par la précision sur le positionnement des fenêtres et par le volume dédié. En revanche, la précision de StreamSamp sur cet espace est affectée par la précision sur le positionnement des fenêtres mais aussi par la précision des estimations sur l'espace des données. Les performances de ces deux algorithmes adoptent des comportements totalement opposés et à un instant donné t_p, ils finissent par se croiser. Il s'agit de l'instant de transition à partir duquel les performances de CluStream dépassent celles de StreamSamp. Pour profiter des avantages de chaque algorithme, il est intéressant d'étudier le processus de passage entre les deux. Cet instant de passage dépend de plusieurs facteurs (i.e. nature du jeu de données, tâches d'analyse). Un critère de passage indépendant de ces facteurs doit alors être défini afin de concevoir un résumé généraliste de meilleure qualité que StreamSamp et CluStream. Nous étudions tout au long de ce chapitre le principe de cette approche (appelée approche hybride) ainsi que les critères qui assurent le passage des données du résumé StreamSamp vers CluStream.

5.3 Principe de l'approche hybride

La Figure 5.1 décrit la structure générale de l'approche hybride : le flux est tout d'abord traité par StreamSamp puis, les échantillons conservés sont envoyés vers le processus CluStream. Trois principales caractéristiques permettent de spécifier cette approche :

– **Enchaînement des processus.** La conception de l'approche hybride est guidée par les avantages de StreamSamp et de CluStream. L'enchaînement de ces deux processus (StreamSamp puis CluStream) est réalisé suite aux observations suivantes : (i) la qualité des résultats de StreamSamp pour le passé proche et les performances de CluStream sur le passé lointain ; (ii) la rapidité de StreamSamp face à un algorithme

	StreamSamp	CluStream
Données dans le résumé	données de détail	données agrégées
Structure du résumé	échantillons	clichés
Nombre de paramètres	faible	important
Vitesse de constitution du résumé	rapide	lent
Qualité du résumé	passé proche (+) passé lointain (-)	passé proche (-) passé lointain (+)
Dimensionnalité des données	pas d'impact	impact important

TABLE 5.1 – Caractéristiques des processus StreamSamp et CluStream

lent qui dépend de la dimensionnalité des données. Il est de ce fait naturel de tirer profit des avantages de StreamSamp dès le début du traitement du flux puis enchaîner par Clustream.

- **Instant de transition.** Nous avons constaté dans le chapitre précédent que l'instant de transition dépend de la tâche à réaliser. Les critères de passage qui assurent la transition de StreamSamp vers CluStream ne doivent pas dépendre de l'analyse effectuée mais des caractéristiques des résumés. Il s'agit alors d'organiser ce passage en fixant des critères objectifs qui dépendent de la nature des deux systèmes.
- **Traitement du flux.** Dans l'approche hybride, les évènements du flux suivent deux cycles de vie différents : (i) ils sont tout d'abord traités par StreamSamp qui permet de concevoir un résumé sous forme d'échantillons, (ii) ils évoluent par la suite dans le système d'agrégation de CluStream. L'algorithme StreamSamp permet de conserver la structure des évènements et échantillonner le flux de façon à envoyer vers CluStream un flux plus lent que le flux d'origine. Ainsi, à l'instant de transition, StreamSamp est bloqué à un ordre donné i. Un flux constitué par des évènements pondérés par 2^i est envoyé vers CluStream. L'algorithme StreamSamp joue ainsi un rôle important au début de l'approche hybride, il sert d'échantillonneur pour CluStream.

5.3.1 Critères de passage

Étant donné que l'approche hybride fait intervenir deux processus différents, il est important de contrôler la qualité du résumé généré par chacun. Pour ce faire, nous contrôlons la représentativité des échantillons dans StreamSamp et la position des barycentres des micro-classes dans CluStream. Contrôler ces deux processus revient à définir deux critères de passage. Ces derniers assurent l'arrêt du processus d'échantillonnage récursif de StreamSamp avant que les échantillons ne se dégradent trop. Le choix de ces critères est basé sur les propriétés intrinsèques des deux techniques : StreamSamp est guidé par le réechantillonnage aléatoire et CluStream par la mise à jour de micro-classes évolutives. Nous

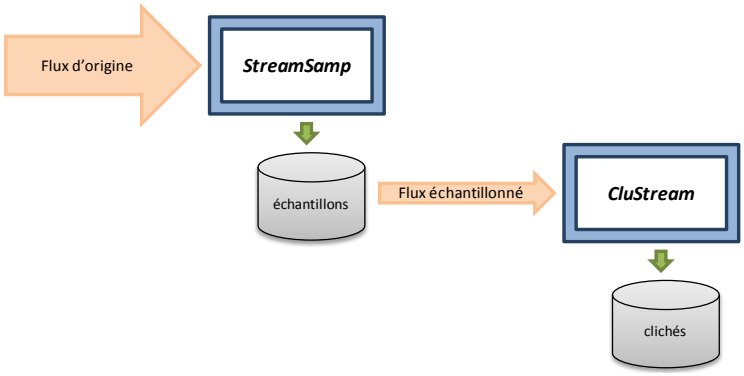

FIGURE 5.1 – Principe de base de l'approche hybride.

spécifions pour cela un seuil pour chaque critère. Ces seuils sont fixés par l'utilisateur, ils dépendent de la nature des évènements du flux.

5.3.1.1 Critère de variance

L'opération de ré-échantillonnage récursif de StreamSamp est appliquée aux échantillons afin de libérer de l'espace. Au cours de cette opération, deux échantillons de même ordre (E_1 et E_2) sont concaténés et ré-échantillonnés en un nouvel échantillon E_3. Les échantillons E_1 et E_2 sont de même taille T et couvrant respectivement les périodes contigües $[t_1,t_2]$ et $[t_2,t_3]$. L'échantillon E_3 est lui aussi de taille T et couvre la période $[t_1,t_3]$. Ce processus de ré-échantillonnage aléatoire contribue ainsi à la dégradation de la qualité du résumé conservé.

Pour contrôler la qualité du résumé de StreamSamp, il est important de contrôler le processus de ré-échantillonnage. Pour cela, nous mesurons la précision des estimateurs des agrégats sur les échantillons à fusionner. Il s'agit de mesurer l'erreur d'échantillonnage. Cette erreur survient lorsqu'un échantillon n'est pas représentatif de ce que l'on veut mesurer ; le résultat dépend alors de la manière dont on choisit l'échantillon (du tirage). On cherche ainsi à définir un encadrement de la précision de l'estimateur. Il existe plusieurs techniques permettant le calcul de cette précision : la variance, l'erreur standard, le coefficient de variation ainsi que le demi intervalle de confiance. Nous nous référons pour cela à la théorie des sondages [20]. Nous essayons de borner dans ce qui suit le demi intervalle de confiance calculé au niveau de l'agrégat moyenne. Un intervalle de confiance indique la précision d'une estimation. En effet, pour un risque donné, l'intervalle est d'autant plus grand que la précision est faible.

Un seuil β est défini par l'utilisateur afin d'assurer une qualité minimale et éviter une dégradation importante des résultats. Ainsi, avant toute opération de ré-échantillonnage, l'erreur relative est calculée pour chaque attribut. Si cette erreur dépasse ce seuil, alors la précision minimale exigée n'est plus respectée et les échantillons ne sont plus représentatifs.

Étant donné que notre sondage est aléatoire simple, nous obtenons avec un seuil de risque a l'inéquation suivante :

$$\left|m - \overline{X}(E_3)\right| \leq t.\sqrt{Var(\overline{X}(E_1 \cup E_2))} \tag{5.1}$$

avec m : moyenne exacte, $\overline{X}(E_3)$: moyenne estimée sur l'échantillon E_3 résultant de la fusion, $Var(\overline{X}(E_1 \cup E_2)$: la variance de l'estimateur de la moyenne calculée au niveau de $E_1 \cup E_2$. t un nombre tel que $\prod(t) = 1 - \frac{a}{2}$ ($\prod(t)$ est donnée par la table de la loi de Student).

Pour calculer la précision de l'estimateur de la moyenne, nous devons calculer la variance de l'estimateur de la moyenne au niveau de $(E_1 \cup E_2)$. Cette quantité est estimée comme suit :

$$Var(\overline{X}(E_1 \cup E_2)) = \frac{1}{|E_1 \cup E_2|}\left[\sum_{i \in E_1 \cup E_2} w_i(x_i - \overline{X}(E_1 \cup E_2))^2\right] \tag{5.2}$$

A partir de l'équation 5.1, nous déduisons l'inégalité ci-dessous afin de valider la représentativité de l'échantillon résultant de la fusion. Nous vérifions cette condition de précision sur E_3 étant donné qu'on n'a aucune garantie sur cet échantillon.

$$\frac{t\sqrt{Var(\overline{X}(E_1 \cup E_2))}}{\overline{X}(E_3)} \leq \beta \tag{5.3}$$

L'inégalité 5.3 doit être vérifiée pour chaque attribut des évènements du flux. L'opération de fusion des échantillons est annulée si cette inégalité n'est pas vérifiée pour au moins un des attributs. Toutefois, si ce critère est respecté, la fusion des échantillons n'est possible qu'après la vérification du second critère.

Choix de la statistique. Dans ce qui a précédé, nous avons mesuré la précision de l'estimation au niveau de l'agrégat moyenne. Cependant, il est possible de rendre ce critère plus difficile à satisfaire en ajoutant la précision sur plusieurs autres agrégats en parallèle. Il est possible que ce critère soit vérifié pour les agrégats moyenne et somme. Il suffit, pour cela, de borner les deux demi intervalles de confiance et d'effectuer une opération de "ET" logique entre les différentes précisions.

5.3.1.2 Critère de position des barycentres

Dans l'approche hybride, le cycle de vie d'un échantillon est composé de deux phases complémentaires. La première est l'évolution de l'échantillon dans StreamSamp. La seconde est l'intégration des évènements de cet échantillon dans le processus CluStream. Nous avons étudié dans le premier critère la contrainte pour StreamSamp. En suivant la même logique, nous étudions la contrainte liée au processus CluStream.

Le principe de CluStream consiste à regrouper les évènements "proches" dans une même micro-classe. Or dans l'approche hybride, les évènements passent par le processus de ré-échantillonnage avant leur insertion dans CluStream. Les barycentres sont alors calculés

sur la base de ces échantillons. Les opérations de ré-échantillonnage récursif contribuent au déplacement des barycentres des micro-classes. Ainsi, les statistiques conservées au sein des CFVs des micro-classes peuvent être altérées par le processus de ré-échantillonnage de StreamSamp entraînant ainsi la dégradation de la position des barycentres. Un second critère est alors ajouté afin de contrôler l'évolution de la position des barycentres.

Une précision minimale sur la position des barycentres est requise. Pour l'assurer, nous fixons un seuil ϵ qui nous permettra de contrôler le déplacement des barycentres. Ce seuil est calculé une seule fois pour tous les attributs des événénements. Comme illustré dans la Figure 5.2, nous calculons le barycentre $G_{E_1 \cup E_2}$ sur l'échantillon $E_1 \cup E_2$ et, le barycentre $\overline{G}_{E_3}$ sur l'échantillon E_3 issu de la fusion. La distance entre G et $\overline{G}$ doit être inférieure au seuil fixé, autrement, la précision minimale requise n'est plus respectée.

$$\frac{d^2(G_{E_1 \cup E_2}, G_{E_3})}{\sum\limits_{x_i \in E_1 \cup E_2} d^2(G_{E_1 \cup E_2}, x_i)} \leq \epsilon \tag{5.4}$$

avec $\sum\limits_{x_i \in E_1 \cup E_2} d^2(G_{E_1 \cup E_2}, x_i)$: l'inertie intra-classe de l'échantillon constitué à partir de $(E_1 \cup E_2)$ et $d^2(G_{E_1 \cup E_2}, G_{E_3})$ l'inertie inter-classe entre $E_1 \cup E_2$ et E_3.

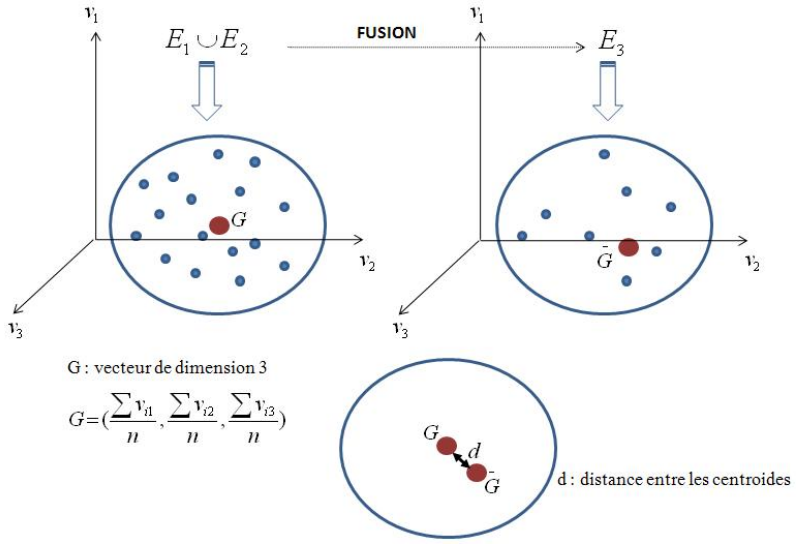

FIGURE 5.2 – Contrôle de la position des barycentres.

5.3.2 Processus de passage

Les critères énoncés ci-dessus permettent de contrôler la représentativité des échantillons conservés au sein du processus StreamSamp. Si au moins l'un de ces deux critères n'est plus respecté, le processus de ré-échantillonnage récursif est arrêté et, les évènements des échantillons sont envoyés vers CluStream. L'envoi de ces évènements dépend de deux

facteurs : le premier est le poids associé à ces évènements et, le second est l'ordre chronologique qui guide leur envoi. Nous étudions dans ce qui suit ces deux facteurs.

5.3.2.1 Poids des évènements

Les évènements envoyés au cours du temps à CluStream n'appartiennent pas nécessairement au même ordre. Pour maintenir la représentativité de ces évènements, leur poids (w_i observé au moment de l'envoi) doit être pris en considération. L'algorithme CluStream se base sur la distance euclidienne pour déterminer l'appartenance d'un évènement à une micro-classe. Cette distance est calculée entre l'évènement et le barycentre des micro-classes. Une fois, la classe la plus proche déterminée, l'évènement e_i est alors envoyé une seule fois en le multipliant par son poids w_i. En effet, l'insertion d'un évènement dans une micro-classe rapproche la valeur du barycentre de cette micro-classe vers l'évènement. C'est pourquoi, le calcul des distances est réalisé une seule fois. L'évènement e_i est ajouté à cette structure comme suit : mise à jour du nombre d'éléments ($n = n + w_i$), mise à jour de $\overline{CF1^x}$ en additionnant, w_i fois, l'évènement e_i à cette valeur, mise à jour de $\overline{CF2^x}$ en ajoutant, w_i fois, le carré de la valeur pour e_i.

5.3.2.2 Ordre de passage des événénements

Dans CluStream, l'information temporelle a une grande importance dans la construction du résumé. Tout d'abord, la construction des micro-classes dépend de l'extension temporelle apportée à la structure des CFV. De plus, chaque cliché pris à un instant donné reflète l'état du système à cet instant. Il est indispensable de conserver l'ordre chronologique des évènements lors de leur envoi du résumé StreamSamp vers le processus CluStream.

Pour respecter cette contrainte, il faut toujours considérer la règle suivante : envoyer les évènements du passé lointain avant ceux du passé proche. Comme illustré dans la Figure 5.3, s'il y a un problème de fusion entre les deux échantillons E_1 et E_2 à l'ordre 2, alors l'envoi vers CluStream est comme suit :

1. envoyer les évènements d'ordre 4 ;
2. envoyer les évènements d'ordre 3 ;
3. envoyer les deux échantillons E_1 et E_2 d'ordre 2 ;

L'envoi prématuré de ces évènements d'ordre 3 et 4 de StreamSamp vers CluStream conduit à des inconvénients importants :

- Une perte de précision en particulier pour les tâches où les performances de StreamSamp restent meilleures que celles de CluStream pour une longue période et, ce n'est que sur le passé très lointain que CluStream devient meilleur (e.g. classification supervisée) ;
- Une perte de temps inutile car le processus de CluStream va calculer les distances entre les évènements de ces échantillons et les micro-classes, opération qui pourrait

encore être retardée. En effet, CluStream est beaucoup plus lent que le processus StreamSamp ;

– Le résumé de StreamSamp perd rapidement la totalité de ses échantillons (envoyés à CluStream). Ceci engendre la perte des avantages de ce processus à savoir la rapidité de traitement des évèements du flux, la rapidité de conception du résumé, les bonne performances pour le passé proche, etc.

L'algorithme 1 illustre le fonctionnement de l'approche hybride.

Algorithm 1 Approche hybride

Require: E_1 et E_2 : deux échantillons d'ordre w à fusionner à l'instant t.
Require: $ordre_{max}$: l'ordre maximum atteint dans le résumé StreamSamp à l'instant t.
{On vérifie si les deux échantillons respectent les critères de passage.}
if (testVariance(E_1,E_2) = faux OU testBarycentres(E_1,E_2) = faux) **then**
for $i = ordre_{max}$; $i > ordre(E_1)$; $i--$ **do**
EnvoyerCluStream(E,i) {Envoyer à CluStream les échantillons d'ordre i}
end for
EnvoyerCluStream(E_1,w) {Envoyer à CluStream l'échantillon E_1}
EnvoyerCluStream(E_2,w) {Envoyer à CluStream l'échantillon E_2}
else
Fusionner(E_1,E_2) {Fusionner les deux échantillons.}
end if

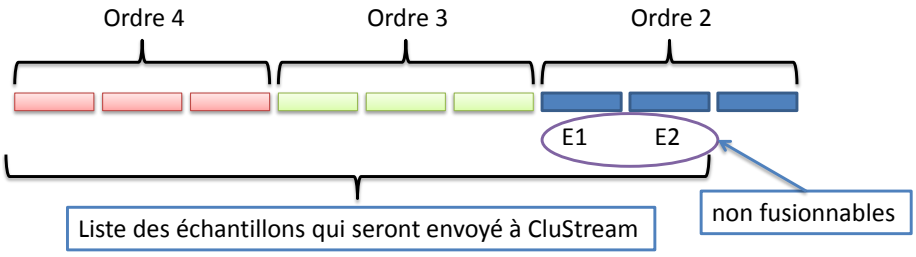

FIGURE 5.3 – Règle d'envoi des évènements lors d'un problème de fusion

5.4 Utilisation d'une réserve

Le respect de l'ordre chronologique dans CluStream nous contraint à toujours envoyer les échantillons suivant leur ordre d'arrivée sans tenir compte de ceux qui sont encore représentatifs. Plusieurs échantillons sont alors contraints de passer à CluStream bien qu'ils respectent encore les deux critères de passage. Pour pallier cette limite, nous avons mis en place une structure "appelée réserve" illustrée dans la Figure 5.4. Cette structure a pour rôle de conserver, temporairement, les échantillons ne pouvant plus fusionner. Ces échantillons

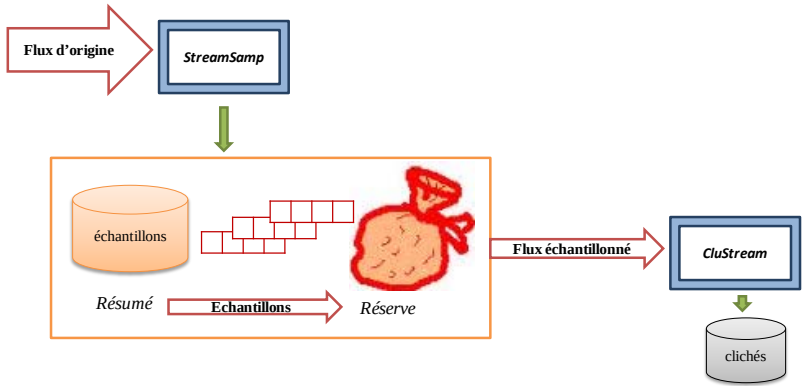

FIGURE 5.4 – Fonctionnement de l'approche hybride avec réserve.

passeront naturellement vers CluStream lorsque les échantillons plus anciens qu'eux seront envoyés. Nous illustrons dans ce qui suit les détails sur la taille allouée à cette réserve ainsi que les règles de passage des évènements de StreamSamp vers la réserve.

5.4.1 Taille de la réserve

La réserve contient les échantillons qui n'ont pas pu fusionné dans StreamSamp. Elle présente ainsi la même structure que le résumé StreamSamp : un ensemble d'échantillons regroupés selon leurs poids. Il s'agit ainsi d'une structure auxiliaire à StreamSamp.

L'ajout d'une nouvelle structure de stockage ne doit pas être contraignant au niveau de l'espace disque. Ainsi, nous n'avons pas alloué un espace spécifique pour la réserve. Nous avons défini un espace maximum global partagé entre le résumé de StreamSamp et la réserve. Lorsque cet espace est atteint, il est nécessaire de libérer des échantillons de la réserve et de les transférer vers CluStream.

$$Espace_{max} = Taille_{résuméStreamSamp} + Taille_{réserve} \tag{5.5}$$

Un tel mécanisme de partage permet une gestion souple de l'espace de stockage. C'est un atout important de l'approche hybride étant donné que l'espace de stockage requis par StreamSamp et la réserve dépend fortement de la qualité du résumé. En effet, lorsque les critères de passage sont fréquemment satisfaits, StreamSamp nécessite plus d'espace que la réserve étant donné que peu d'échantillons lui sont envoyés. Dans le cas opposé, la réserve a besoin de plus d'espace que StreamSamp. Nous étudions dans ce qui suit les deux règles de remplissage (StreamSamp vers réserve) et de libération (réserve vers CluStream) de la réserve.

5.4.2 Règles de passage

Les évènements du flux évoluent dans un premier temps dans le résumé de StreamSamp, ils passent par la suite par la réserve pour finir dans le résumé de CluStream. Le passage des échantillons entre les différentes structures dépend de deux règles. La première définit le transfert des échantillons du résumé de StreamSamp vers la réserve. Tandis que la seconde règle définit leur passage de la réserve vers le processus CluStream.

Règle 1 : transfert des échantillons vers la réserve. Lorsque deux échantillons d'un ordre ne respectent plus les critères de passage, soit ces deux échantillons sont envoyés vers la réserve soit l'un des deux seulement. Prenons l'exemple des échantillons E_1 et E_2 appartenant à l'ordre i. Supposons que ces deux échantillons ne peuvent pas fusionner alors, nous vérifions si la fusion de E_2 et E_3 est possible (avec E_3 l'échantillon de l'ordre i qui suit E_2). Ceci suppose que $L \geq 3$. Si cette fusion est possible alors seul l'échantillon E_1 est envoyé à la réserve autrement, les deux échantillons E_1 et E_2 y seront envoyés. Cette règle assure une occupation efficace de l'espace mémoire étant donné que seul les échantillons qui ne respectent pas les critères de passage seront transférés à la réserve. L'algorithme 2 illustre le processus d'envoi des échantillons vers la réserve.

Règle 2 : transfert des échantillons vers le processus CluStream. Une fois l'espace maximum global est atteint, δ échantillons sont envoyés vers CluStream (δ est un paramètre utilisateur). Ces δ échantillons sont conjointement extraits du résumé StreamSamp et de la réserve. Ils sont envoyés, par paquets de δ, dans leur ordre chronologique. L'algorithme 3 illustre la fonctionnalité de libérer la réserve et d'envoyer les évènements vers l'algorithme CluStream.

5.5 Résumé hybride

Le résumé résultant de l'approche hybride est constitué de deux entités différentes :

- Les échantillons : constitués lors du passage des évènements du flux dans le processus StreamSamp ;
- Les clichés : composés à partir des échantillons envoyés du résumé StreamSamp et traités par CluStream.

Ainsi, le cycle de vie des évènements du flux appartenant à une période temporelle $[t_a, t_b]$ suit trois évolutions différentes. Comme illustré dans la Figure 5.5, les évènements de la période sont tout d'abord représentés sous forme d'échantillons et évoluent uniquement dans le résumé StreamSamp (*état 1*). A un moment donné, quelques échantillons deviennent non représentatifs et passent alors à l'algorithme CluStream. Ces évènements seront sous forme de micro-classes. Le résumé hybride sera composé d'un ensemble d'échantillons et un ensemble de clichés (*état 2*). Lorsque la totalité des échantillons de la période $[t_a, t_b]$

Algorithm 2 Test de fusionnabilité et envoi à la réserve

Require: E_1 et E_2 : deux échantillons d'ordre w à fusionner à l'instant t.
Require: $ordre_{max}$: l'ordre maximum atteint dans le résumé StreamSamp à l'instant t.
{On teste si les deux échantillons respectent les critères de passage.}
if (testVariance(E_1,E_2) = faux OU testBarycentres(E_1,E_2) = faux) **then**
Soit $E3$: Echantillon plus récent que E_1, E_2 et de même ordre
if (testVariance(E_2,E_3) = faux OU testBarycentres(E_2,E_3) = faux) **then**
EnvoyerRéserve(E_1,w) {Envoyer à la réserve l'échantillon E_1}
EnvoyerRéserve(E_2,w) {Envoyer à la réserve l'échantillon E_2}
else
EnvoyerRéserve(E_1,w) {Envoyer à la réserve l'échantillon E_1}
Fusionner (E_2,E_3)
end if
else
Fusionner(E_1,E_2) {Fusionner les deux échantillons.}
end if

Algorithm 3 Fonction Gestion de la réserve

Require: *Res* : la réserve
EspaceHybride : Espace global requis par StreamSamp et la réserve
δ : Nombre d'échantillons à envoyer vers CluStream
if Taille(Res) + Taille(StreamSamp) $\geq$ EspaceHybride **then**
Liste F = ExtraireAnciensEchantillons(Res,SS,δ) {F contient les δ plus anciens échantillons extraits de la réserve et de StreamSamp}
for $i = 0$ to Taille(F) **do**
EnvoyerClustream(F[i], w_i) {Envoyer l'échantillon d'ordre w_i vers CluStream}
end for
end if

passent à CluStream, on se retrouve avec un résumé uniquement sous forme de clichés (*état 3*).

5.5.1 Paramétrage de l'approche

L'approche hybride ajoute trois nouveaux paramètres aux algorithmes StreamSamp et CluStream à savoir β (paramètre du critère de variance), ϵ (paramètre du critère de la position des barycentres) et δ (paramètre de la réserve). Il est vrai que cette approche fait intervenir plus de paramètres que les algorithmes StreamSamp et CluStream cependant, l'utilisation de ces paramètres présente les avantages suivants : (i) réduction de la charge et, (ii) qualité du résumé. Pour le premier avantage, ces paramètres font que CluStream fonctionne sur un flux échantillonné (plus léger que le flux d'origine), ce qui lui assure un

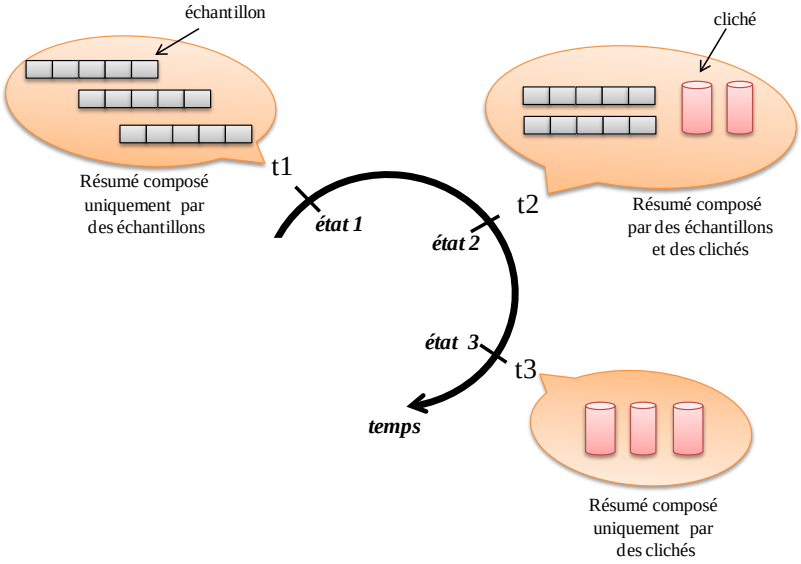

FIGURE 5.5 – Cycle de vie des évènements du flux dans l'approche hybride.

traitement plus rapide. De plus, ces paramètres garantissent un résumé de meilleure qualité sur toute la période temporelle étant donné qu'ils contrôlent le processus d'échantillonnage de StreamSamp.

Nous notons que dans le cadre d'un flux monodimensionnel, les critères de passage seront évalués deux fois, étant donné que les contraintes sur β et ϵ donnent le même résultat dans le cas d'une seule variable. Cependant, dans le cadre d'un flux multidimmensionnel, les contraintes β impliquent sur chaque variable une contrainte sur ϵ qui n'est pas assez forte. Inversement, une contrainte sur ϵ implique une contrainte sur les β mais qui n'est pas assez forte. D'où l'utilité de spécifier ces deux contraintes.

Paramétrage de δ. Le choix de la valeur de δ n'a pas d'effet sur le processus de CluStream. La seule différence entre une grande et une petite valeur de δ est le temps nécessaire à CluStream pour traiter le lot d'échantillons en entrée. Nous distinguons entre les évènements initiaux (évènements avant leur passage à StreamSamp) et, les évènements échantillonnés (évènements après leur passage à StreamSamp). Ainsi, ce qui a un effet sur CluStream est le débit avec lequel les évènements éhantillonnés lui sont envoyés. Ces évènements sont envoyés avec un débit inférieur au débit avec lequel les évènements initiaux entrent au système. Néanmoins, si malgré la phase de ré-échantillonnage de StreamSamp, l'algorithme CluStream ne peut faire face au débit des évènements échantillonnés, un processus de délestage doit alors être appliqué engendrant ainsi la perte de certains évènements. Ce système de délestage peut être appliqué soit sur les évènements initiaux (c'est à dire avant leur entrée dans StreamSamp), soit sur les évènements échantillonnés (c'est à dire avant leur passage à CluStream). Etant donné que StreamSamp est rapide, on conserve le

plus possible ces échantillons et, leur perte sera planifiée au niveau de leur passage vers CluStream.

5.5.2 Tâches d'analyse

Comme indiqué dans le cas des résumés StreamSamp et CluStream, afin d'analyser des portions précises (e.g. la période $[t_a, t_b]$) de l'histoire du flux, il est nécessaire d'extraire la partie du résumé qui respecte la période en question. Cependant, l'approche hybride combine deux structures de données différentes. A cet effet, en fonction de la période évaluée $[t_a, t_b]$, trois scénarios sont possibles :

- Tous les évènements inclus dans la période appartiennent au résumé StreamSamp.
- Tous les évènements de la période sont inclus dans le résumé CluStream.
- Les évènements de $[t_a, t_b]$ sont partagés entre le résumé StreamSamp et celui de CluStream.

Nous nous intéressons dans ce qui suit au troisième cas de figure étant donné que le traitement des deux premiers scénarios revient à ce qui a été présenté dans les sections 4.2.3.1 et 4.3.2.1.

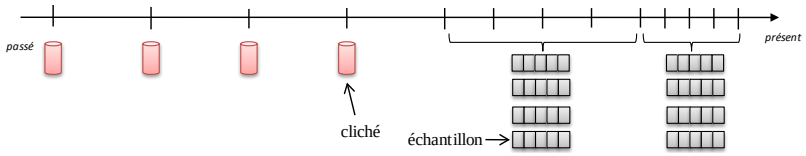

FIGURE 5.6 – Composition du résumé hybride sur la période interrogée $[t_a, t_b]$.

5.5.2.1 Réponse aux requêtes

Dans le cadre de l'approche hybride l'ensemble des analyses sont réalisées sur le résumé de la période $[t_a, t_b]$. Ce résumé peut être constitué d'un ensemble d'échantillons et d'un ensemble de micro-classes. Les évènements des échantillons sont pondérés et, les micro-classes seront représentées par leurs barycentres pondérés par l'effectif total observé dans les micro-classes. Nous notons $E_{f_{[T,t_b]}}$ la partie du résumé hybride extraite du résumé StreamSamp et, $E_{f_{[t_a,T]}}$ la partie récupérée du résumé CluStream. Dans la plupart des cas, il est possible d'estimer la requête indépendamment sur $E_{f_{[t_a,T]}}$ et $E_{f_{[T,t_b]}}$ et combiner par la suite les deux résultats. Cependant, cette règle ne peut être généralisée (e.g. cas de la médiane) et dépend de l'analyse à effectuer.

Calcul de la moyenne et de la variance. Pour calculer ces agrégats sur le résumé hybride de la période $[t_a, t_b]$, il suffit de calculer l'agrégat séparément sur le résumé fourni par StreamSamp ainsi que celui fourni par CluStream.

Ainsi dans la cas de la moyenne :

$$\overline{X}_{[t_a,t_b]} = \frac{\sum\limits_{i \in E_{f_{[T,t_b]}}} x_i w_i + \sum\limits_{i=1}^{N_c} \overline{CF1}^{(i)}_{[T_a,T]}}{\sum\limits_{i \in E_{f_{[T,t_b]}}} w_i + \sum\limits_{i=1}^{N_c} n_i} \tag{5.6}$$

De même pour la variance :

$$S^2{}_{[t_a,t_b]} = S^2_{[t_a,T]} + S^2_{[T,t_b]} \tag{5.7}$$

avec $S^2{}_{[t_a,t_b]}$ l'estimateur de la variance calculé au niveau de la période $[t_a, t_b]$. De même pour les périodes $[t_a, T]$ et $[T, t_b]$. Nous estimons leurs variances comme suit :

$$S^2_{[t_a,T]} = \frac{\sum\limits_{i=1}^{N_c} \overline{CF2}^{(i)}_{[t_a,T]} - \overline{X}^2}{\sum\limits_{i=1}^{N_c} n_i - 1} \tag{5.8}$$

$$S^2_{[T,t_b]} = \frac{1}{|E_{f_{[T,t_b]}}| - 1} \left[\sum_{i \in E_{f_{[T,t_b]}}} w_i (x_i - \overline{X})^2 \right] \tag{5.9}$$

Calcul de la médiane. Le calcul de la médiane pour une variable x ne peut être réalisé en deux parties. En effet, cette statistique est basée sur l'ordre des évènements. Il est ainsi nécessaire de concaténer les évènement résultants du résumé StreamSamp ainsi que les barycentres issus du résumé CluStream. La médiane est calculée sur l'ensemble des évènements. Pour cela le système procède comme suit :

- Concaténation des deux résumés : $E_{f_{[t_a,t_b]}} = E_{f_{[t_a,T]}} \cup E_{f_{[T,t_b]}}$;
- Tri des valeurs dans le résumé $E_{f_{[t_a,t_b]}}$;
- Calcul du poids cumulé. Notons que sur la partie du résumé extraite de CluStream, la valeur w_i dans la formule est égale à n_i (i.e. effectif total de chaque micro-classe).

$$Poids = \sum_{i \in E_{f_{[t_a,t_b]}}} w_i$$

- Calcul de la médiane p :

$$\underset{p}{argmin} \left| \sum_{rang(i) \leq rang(p)} w_i - \frac{1}{2} \sum_{i}^{\left|E_{f_{[t_a,t_b]}}\right|} w_i \right|$$

Classification supervisée et non supervisée. Pour les tâches de classification supervisée et non supervisée, nous utiliserons la même démarche utilisées spéraément pour les algorithmes StreamSamp et CluStream. Il s'agit de faire l'union entre les évènements générés à partir du résumé CluStream ainsi que les échantillons du résumé StreamSamp. Pour la classification non supervisée, les classes sont construites à partir de l'union de ces évènements. De même, pour la construction du modèle dans le cas de la classification supervisée. La phase d'évaluation reste identique à celle appliquée pour les algorithmes StreamSamp et CluStream.

5.6 Expérimentations

Nous comparons dans cette section les performances de notre approche avec les algorithmes StreamSamp et CluStream. Des comparaisons faisant intervenir l'approche hybride sans réserve et l'approche hybride avec réserve permettent de mettre en avant l'utilité de la réserve dans l'étude des performances. Comme précédemment, nous nous intéressons à l'évolution des résultats sur une période temporelle qui vieillit au cours du temps. Les tableaux 5.2, 5.4 et 5.3 illustrent les paramètres des différents algorithmes lors des expérimentations. Ces paramètres ont été définis de façon à obtenir un même volume de résumé pour tous les algorithmes. Le jeu de données utilisé pour les expérimentations est KDD99 (cf. Annexe A.1). Cependant, pour la classification supervisée le jeu de données utilisé est Cover Type (cf. Annexe A.2).

Table 5.2 – Paramètres de l'algorithme CluStream.

CluStream
N_c : nombre de micro-classes = 50
Nombre de clichés par ordre = 17 ($\alpha = 2$, $L_c = 4$)
m : nombre d'évènements pour l'initialisation = 2000

Table 5.3 – Paramètres de l'algorithme StreamSamp.

StreamSamp
$\alpha = 1$
$T = 500$ évènements par échantillon
$L = 8$ échantillons par ordre

5.6.1 Vitesse d'exécution

Nous nous intéressons dans un premier lieu au temps global écoulé pour le traitement de la totalité du flux de données par les algorithmes.

TABLE 5.4 – Paramètres de l'approche hybride.

StreamSamp
$\alpha = 1$ $T = 250$ évènements par échantillon $L = 8$ échantillons par ordre
CluStream
N_c : nombre de micro-classes = 50 Nombre de clichés par ordre = 9 ($\alpha = 2$, $L_c = 3$)
Approche hybride avec réserve
$\beta = 4.10^{-2}$ $\epsilon = 10^{-4}$ δ : nombre d'échantillons transférés vers CluStream = 2

La Figure 5.7 illustre les performances en termes de temps d'exécution (échelle logarithmique). Comme prévu StreamSamp reste l'algorithme le plus rapide et CluStream, l'algorithme le plus lent. Tandis que les performances de l'approche hybride sont entre celles observées pour StreamSamp et CluStream. Elles se rapprochent des performances de StreamSamp et restent 10 fois meilleures que CluStream. L'utilisation de l'échantillonnage progressif comme premier mécanisme de l'approche hybride a allégé le traitement des données du flux.

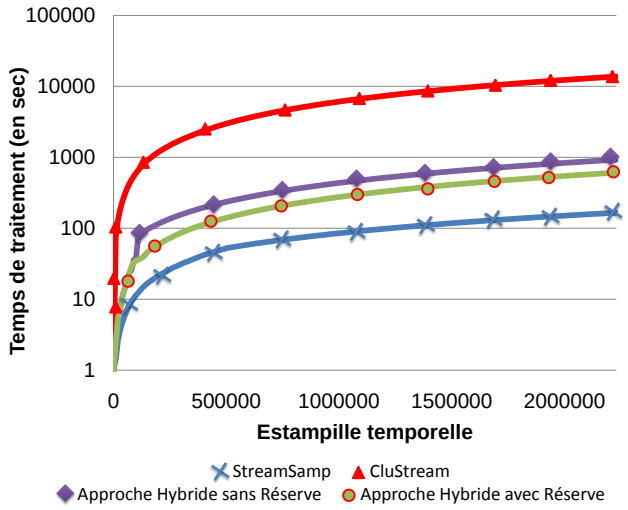

FIGURE 5.7 – Comparaison des algorithmes sur la vitesse de traitement de l'intégralité du flux (échelle logarithmique).

Par ailleurs, l'utilisation de la réserve rend encore plus rapide l'approche hybride étant donné que l'envoi des évènements vers CluStream est différé par leur entrée dans la réserve.

De plus, la phase d'initialisation de CluStream est devenue une tâche légère et rapide dans l'approche hybride avec réserve étant donné que les échantillons sont envoyés à CluStream par lots de δ.

5.6.2 Évaluation des requêtes d'agrégat

Nous étudions dans ce qui suit l'évolution des erreurs observées pour l'évaluation des requêtes d'agrégats sur la période [0, 50000] à différents instants temporels. Les estimations pour les algorithmes StreamSamp et les approches hybrides (sans et avec réserve) sont répétées 100 fois. Nous étudions la moyenne des résultats obtenus. Pour CluStream, nous calculons ces agrégats sur la base des clichés les plus proches des bornes de l'intervalle de la requête.

5.6.2.1 Évaluation de la moyenne

Dans le cadre de l'approche hybride, l'agrégat moyenne est calculé pour chaque variable d'intérêt à partir de l'échantillon final représentant la période [0,50000]. Pour l'évaluation des résultats nous utilisons comme précédemment (cf. section 4.4.2.1) l'erreur moyenne relative. La valeur réelle de la moyenne sur la période [0,50000] est calculée à partir des données d'origine.

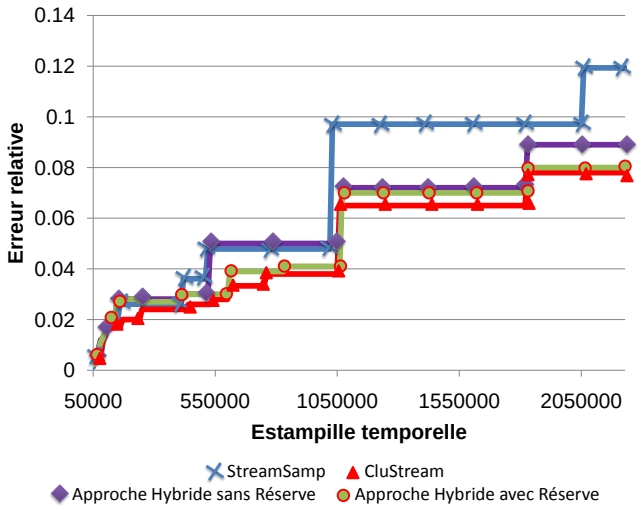

FIGURE 5.8 – Comparaisons des algorithmes pour l'évaluation de la moyenne sur la période [0, 50000].

La Figure 5.8 illustre les résultats obtenus pour l'évaluation de la moyenne. Sur le passé proche, les quatre algorithmes adoptent des comportements similaires. C'est à partir de l'instant 500000 que les performances de ces approches se différencient. Nous avons discuté dans la section 4.4.2.1 les performances de StreamSamp et CluStream. Nous discutons dans

ce qui suit les performances de l'approche hybride. Les résultat ont montré que l'introduction d'une réserve a contribué à un gain en précision qui est constaté sur une bonne partie de la période évaluée et reste observé sur le passé lointain de la période.

5.6.2.2 Évaluation de la médiane

Comme indiqué dans la section 4.4.2.2, nous calculons l'erreur d'estimation en utilisant le rang de la valeur médiane estimée. La Figure 5.9 illustre les résultats obtenus pour l'évaluation de la médiane. Nous comparons ici les performances des approches hybrides avec celles de StreamSamp et CluStream. Sur le passé proche, les approches hybrides suivent le même comportement que StreamSamp. Cependant, lorsque les performances de StreamSamp se dégradent, les approches hybrides se rapprochent du comportement de CluStream et finissent par suivre un comportement semblable à ce dernier. Par ailleurs, nous obtenons de meilleurs résultats sur le passé lointain pour l'approche hybride avec réserve par rapport à l'approche hybride sans réserve. Cependant, ils finissent tous les deux par rejoindre les performances de Clustream.

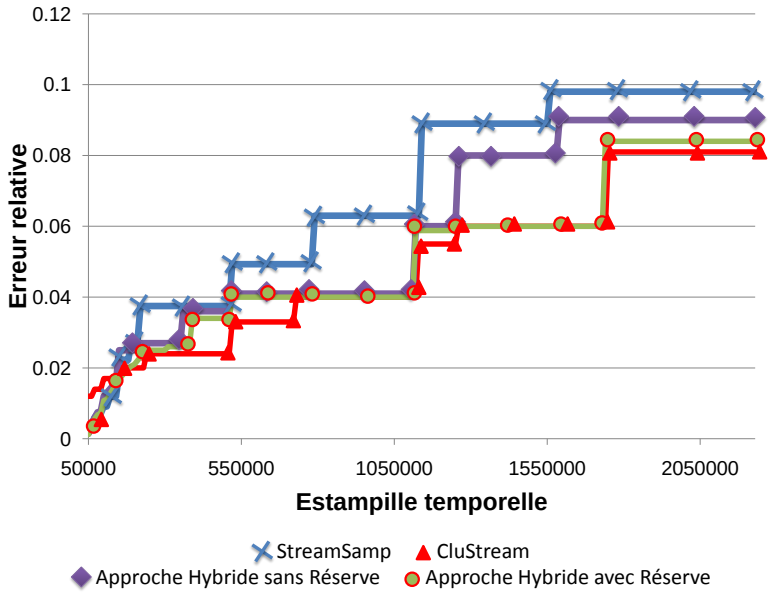

FIGURE 5.9 – Comparaison des performances pour l'évaluation de la médiane sur la période [0, 50000].

5.6.3 Évaluation sur des tâches de fouille

Nous discutons dans ce qui suit les quatre algorithmes pour les tâches de classification. Les performances de StreamSamp et CluStream ont été étudiées pour ces tâches dans la section 4.4.3.

5.6.3.1 Évaluation de la classification non supervisée

Les Figures 5.10 et 5.11 présentent les résultats de l'évolution de l'inertie intra-classe moyenne sur la période [0,50000] évaluée à différents instants temporels. Sur les périodes du passé proche, les quatre algorithmes adoptent des comportement similaires. Cependant, sur le passé lointain, les écarts se creusent entre ces algorithmes. CluStream devient rapidement l'algorithme le plus performant et les approches hybrides suivent un comportement similaire à CluStream. Les deux approches hybrides présentent des performances similaires mais, elles se différencient à partir de l'instant 1050 000 où l'approche avec réserve finit par se stabiliser et rester proche des performances de CluStream.

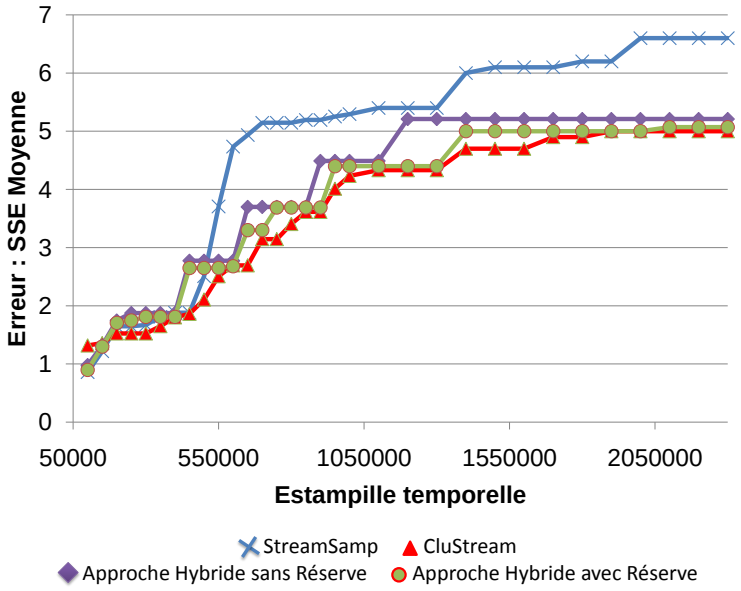

FIGURE 5.10 – Comparaison des performances pour la tâche de la classification non supervisée sur la période [0, 50000].

5.6.3.2 Évaluation de la classification supervisée

Comme dans le chapitre 4, pour évaluer les performances des algorithmes sur la classification supervisée, les modèles sont construits sur les résumés de la période [0,50000]. Les résultats des quatre algorithmes sont comparés entre eux.

La Figure 5.12 illustre les performances des quatre algorithmes étudiés. Nous avons observé dans la section 4.4.3.2 que StreamSamp reste l'algorithme le plus performant pour la classification supervisée et ce pour une période assez importante. Ce n'est qu'à partir de l'instant $t = 350000$ que ses performances sont dépassées par les performances de CluStream et des approches hybrides.

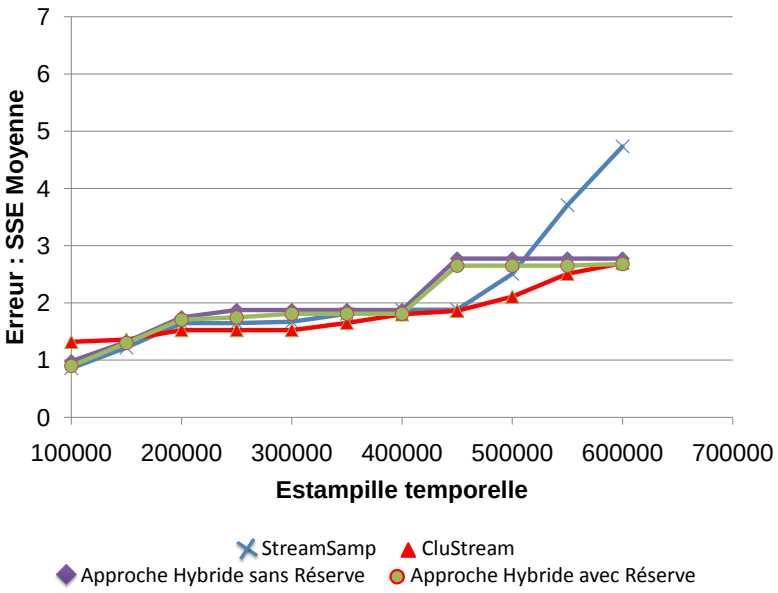

FIGURE 5.11 – [Zoom]Comparaison des performances pour la tâche de la classification non supervisée sur la période [0, 50000].

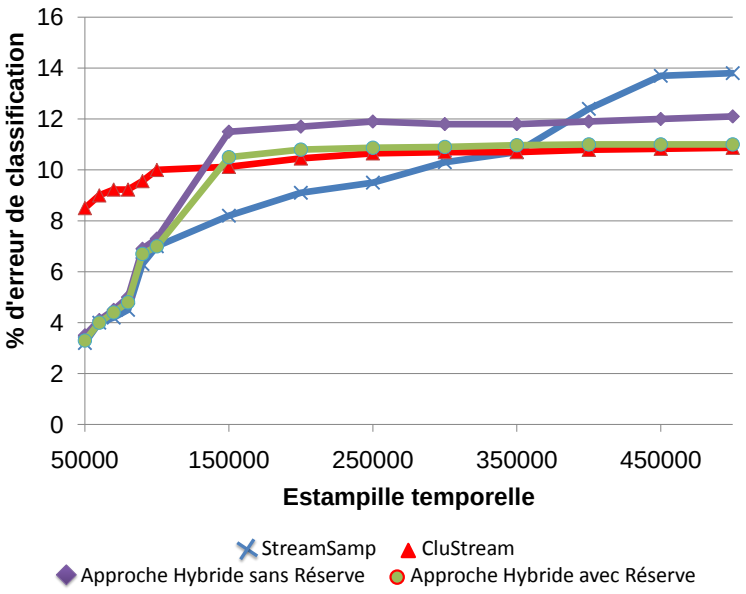

FIGURE 5.12 – Comparaison des performances pour la tâche de la classification supervisée sur la période [0, 50000].

L'apport des approches hybrides est ressenti sur les périodes du passé proche où elles adoptent un comportement similaire à celui de StreamSamp. A partir de l'instant 100000, l'écart entre les deux approches hybrides commence à se creuser. En effet, l'approche hy-

bride avec réserve présente des performances semblables à celles de CluStream tandis que l'approche hybride sans réserve présente un taux d'erreur plus élevé.

5.7 Synthèse

Nous avons présenté dans ce chapitre une nouvelle approche de résumé généraliste de flux de données. Cette technique combine une méthode d'échantillonnage couplé à une méthode de micro-clustering. Les avantages de ces deux techniques sont combinés en vue de produire un résumé stable qui permet de fournir des approximations précises pour des analyses évaluées à différents instants temporels.

Les expérimentations ont tout d'abord porté sur le temps de traitement des différents algorithmes. Les résultats ont montré que l'approche hybride est 10 fois plus rapide par rapport à CluStream qui reste l'algorithme le plus lent.

D'autres séries d'expérimentations ont été menées sur des tâches de requête et de fouille. Quelque soit la tâche étudiée (hormis la classification supervisée), l'approche hybride a présenté de bons résultats pour les périodes du passé proche et du passé lointain. Cependant, dans le cas de la classification supervisée, les résultats des expérimentations montrent que l'algorithme StreamSamp reste le plus performant. En effet, malgré la dégradation des performances de StreamSamp sur le passé lointain, l'écart entre qui sépare ses performances à celles des autres approches n'est pas très important. On peut conclure qu'il est avantageux d'utiliser StreamSamp dans la tâche de classification supervisée. Pour toutes les autres tâches (moyenne, médiane et classification non supervisée), l'approche hybride présente de meilleures performances par rapport aux autres approches. L'utilisation d'une réserve a amélioré les résultats sur la totalité de la période temporelle.

Après avoir construit un résumé généraliste du flux de données, il est intéressant d'étudier son exploitation d'une façon naturelle à partir d'une requête sur le SGFD. C'est l'objet du chapitre suivant dans lequel nous allons nous focaliser sur la conception et l'implémentations de techniques permettant d'intégrer la gestion de résumé dans les SGFD.

Chapitre 6

Exploitation de résumés de flux de données

Sommaire

6.1 Introduction

Nous nous sommes intéressés dans les précédents chapitres à l'évaluation de différentes tâches d'analyses sur le passé lointain des flux. Ces tâches sont définies a posteriori et sont

appliquées sur les résumés qui ont été conservés à partir des données des flux. L'évaluation de ces tâches nécessite une étude approfondie de la structure des résumés et l'adaptation des algorithmes de requêtes et tâches de fouille à ces structures. Les requêtes sont directement appliquées sur les résumés, ces derniers sont **isolés** c'est à dire qu'ils ne s'intègrent pas dans l'architecture des SGFD. A cet effet, l'utilisateur doit connaître au préalable le schéma des résumés pour pouvoir appliquer la requête et l'évaluer. Cette démarche est simple si toutes les requêtes sont définies sur le même résumé or, dans le cas où nous avons plusieurs résumés de structures différentes, une telle interrogation devient de plus en plus difficile. Ceci rend cette approche de requêtage peu avantageuse étant donné qu'elle n'est pas transparente à l'utilisateur et ne s'intègre pas de façon naturelle dans un SGFD. Il faut donc penser à une approche d'interrogation plus générique qui peut être appliquée quelque soit l'analyse réalisée et quelque soit la structure de résumé adoptée. C'est dans ce contexte que s'introduit ce dernier chapitre. Notre objectif consiste à proposer une extension des SGFD de façon à pouvoir traiter, de façon transparente pour l'utilisateur, toutes les requêtes évaluées sur le passé lointain des flux. En effet, la principale fonctionnalité pour laquelle les SGFD ont été développés est l'évaluation de requêtes continues qui portent sur le passé proche du flux [35]. Cependant, il est nécessaire dans plusieurs systèmes d'aide à la décision et dans des applications de détection de fraudes d'évaluer des requêtes qui portent sur des données du passé lointain. Dans les applications de contrôle de cartes bancaires par exemple, une des méthodes utilisées pour détecter les fraudes est de construire un modèle sur les activités effectuées (e.g. opérations de retrait) et de comparer par la suite ce modèle à la dernière activité du client. De nouveaux besoins sont alors apparus. Malgré l'enrichissement des SGFD par des systèmes d'archivage, les requêtes exprimées sur les données du passé lointain restent impossibles à traiter si elles font appel à des connaissances qui n'ont pas été conservées à priori. Le champ de requêtes que le SGFD est capable d'évaluer reste ainsi limité.

Nous présentons dans ce chapitre une extension de l'architecture des SGFD permettant de pallier ces limites en proposant une solution à l'évaluation des requêtes qui portent sur les données du passé lointain. Le système qu'on introduit se veut générique et complet (i.e. inclut les données du passé proche et les données résumées). Notre approche bénéficie d'un traitement rapide et donne la possibilité de retourner des réponses accompagnées d'intervalles de confiance. Contrairement aux applications des tâches de fouille et d'analyses présentées dans le chapitre précédent, nous présentons ici des techniques permettant à l'utilisateur de poser sa requête au SGFD sans se soucier de la structure du résumé. Le traitement de la requête est complètement transparent à l'utilisateur et **s'intègre** de façon naturelle aux SGFD.

35. Une requête continue sur un flux de données ne peut fournir de réponse que sur la fenêtre temporelle posée sur le flux.

6.2 Formulation du problème

Comme défini dans la conclusion de l'état de l'art, nous distinguons le passé proche (i.e. évènements conservées dans la mémoire du SGFD) et le passé lointain (i.e. évènements expirées du SGFD) d'un flux. Les fonctionnalités des systèmes actuels permettent de traiter toute requête évaluée sur le passé proche, en revanche, ils sont incapables de traiter les interrogations sur le passé lointain. Deux types de requêtes peuvent être appliquées sur les SGFD : les requêtes ponctuelles et, les requêtes continues. Nous évoquons deux problématiques liées à ces requêtes : (i) la disponibilité des données pour les requêtes du passé lointain et, (ii) le temps nécessaire pour accéder aux évènements conservés. Généralement les requêtes continues sur le passé lointain sont continues du fait qu'elles sont couplées par une requête sur les évènements du passé proche (cf. requête 1). En revanche, les requêtes ponctuelles sur le passé lointain ont un sens indépendamment des évènements sur le passé proche (cf. requête 2).

- `Requête 1`. Pour étudier la gestion des forfaits, l'opérateur téléphonique décide de comparer le temps de communication moyen des appels émis, du mois courant, par rapport au temps moyen des appels du même mois de l'année précédente. A chaque évaluation de la requête, les deux fenêtres doivent être synchrones.
- `Requête 2`. Pour étudier l'efficacité de sa dernière campagne publicitaire, un opérateur téléphonique compare le nombre de clients recensés en 2008 et en 2009.

Disponibilité des données. Quelque soit la nature de la requête, lorsque celle-ci invoque des évènements du passé lointain, elle affronte le problème de l'indisponibilité des évènements. La disponibilité des évènements du flux dépend de la mise en place d'un système de résumé pour les conserver ainsi que de la fréquence de la requête (dans le cas de requêtes continues). Différentes structures de résumés existent permettant de conserver une trace de l'histoire du flux (cf. section 2.3), cependant, il est nécessaire de faire un choix sur la structure à utiliser. Nous nous intéressons dans ce qui suit aux résumés généralistes. L'utilisation de ces structures de résumé s'avère nécessaire pour assurer une réponse approchée aux requêtes. Nous proposons de répondre à la problématique de l'indisponibilité des données par l'intégration d'une structure de résumé généraliste dans l'architecture des SGFD. En effet, les systèmes actuels n'étant pas équipés par des mécanismes capables de réaliser ces traitements, ils doivent être enrichis afin de permettre l'expression et l'évaluation des requêtes sur le passé lointain.

Temps d'accès aux évènements conservées. La disponibilité et le temps d'accès aux évènements sont deux notions fortement liées. En effet, lorsque les évènements du passé lointain d'un flux sont disponibles dans une structure de résumé, il est important d'étudier leur temps d'accès. Nous constatons cette relation dans le cas des requêtes continues où l'accès à ces évènements doit être rapide afin d'assurer leur disponibilité avant la pro-

chaine réévaluation (traduite par la fréquence de la requête). Lorsque les évènements sont disponibles, ils sont alors conservés en base. Un compromis doit être fait entre le temps nécessaire pour extraire les évènements de la base et la précision des résultats. Il faut en effet s'assurer que la fréquence d'évaluation de la requête continue soit supportable par le système. L'accès aux évènements des flux est illustré, dans ce chapitre, via deux approches permettant d'intégrer le résumé dans l'architecture des SGFD : **l'approche statique** et, **l'approche dynamique**. Le choix de ces appellations est guidé par le rôle que prend le résumé dans l'architecture du SGFD. La première approche présente une forte dépendance entre le SGFD et la structure du résumé. Le résumé joue alors un **rôle passif**. En revanche, dans la deuxième approche, le résumé joue un **rôle actif**. Ces rôles sont détaillés dans la section 6.3. Néanmoins, malgré cette différence, ces deux approches partagent les points suivants :

- Intégration du résumé dans l'architecture du SGFD : nous envisageons que la construction des résumés soit réalisée au sein du moteur des SGFD ce qui revient à créer les résumés dès l'arrivée des flux et à procéder à leur mise à jour en temps réel. Cette construction se fait à la volée en profitant des avantages de l'infrastructure des SGFD ;
- Évaluation des requêtes sur le passé : nous utilisons les résumés afin de répondre aux requêtes sur le passé lointain au même titre que les requêtes continues définies sur les évènements du passé proche. Elles sont inscrites dans le registre des requêtes associé au SGFD. Le système évalue la requête de façon transparente à l'utilisateur.

La Figure 6.1, décrit l'architecture globale que nous proposons pour étendre les systèmes existants. Cette architecture fait intervenir deux composantes principales : le SGFD et le résumé. Le premier module regroupe les outils standards d'un système de gestion de flux de données à savoir les traitements des flux de données et les traitements de requêtes. Le second module réunit la conception et la conservation des résumés réalisés à partir des évènements des flux. Ces deux composantes sont complémentaires et, leur combinaison nous permet de répondre aux requêtes sur le passé lointain. Cependant, la question qui soulève notre intérêt est comment se fait la communication entre ces deux composantes ainsi que l'interaction entre les différents processus appartenant à ces modules. Les deux approches présentées dans ce qui suit permettent cette communication. Nous détaillons dans les sections suivantes le fonctionnement et la particularité de chacune des approches.

Dans la suite de ce chapitre, nous considérerons un flux F produit à partir d'une source S. Chaque événement e_i du flux est une séquence $< t_i; v_{i_1}, ..., v_{i_d} >$ avec $< v_{i_1}, ..., v_{i_d} >$ l'ensemble des valeurs de chaque attribut du flux et t_i est l'instant d'observation de l'événement e_i.

Nous nous intéressons dans le reste de ce chapitre aux requêtes référençant des périodes du passé proche et du passé lointain, cela permet d'aborder la difficulté qu'engendre la

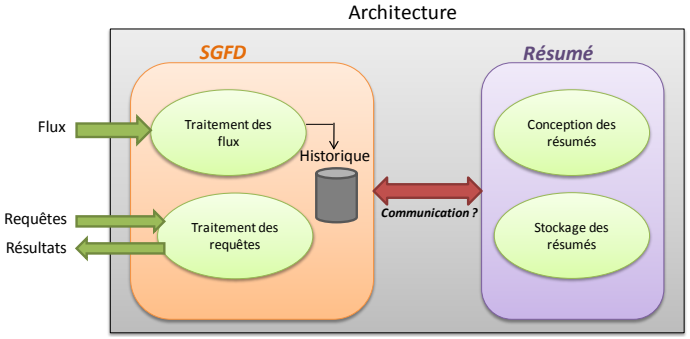

FIGURE 6.1 – Schéma de l'architecture générale proposée.

synchronisation des deux périodes. Nous considérons deux fenêtres formalisées comme suit (cf. Figure 6.2) :

- F_C (*C pour fenêtre "Courante"*), une fenêtre temporelle définie sur le passé proche et portant sur la période $[t_c - f_c + 1, t_c]$ avec f_c la taille de la fenêtre F_C et t_c l'instant courant ;
- F_P (*P pour fenêtre du "Passé"*), une fenêtre temporelle définie sur le passé lointain et portant sur la période $[t_c - D - f_p + 1, t_c - D]$ avec f_p la taille de la fenêtre F_P et D le délai séparant l'instant courant et le début de la fenêtre F_P.

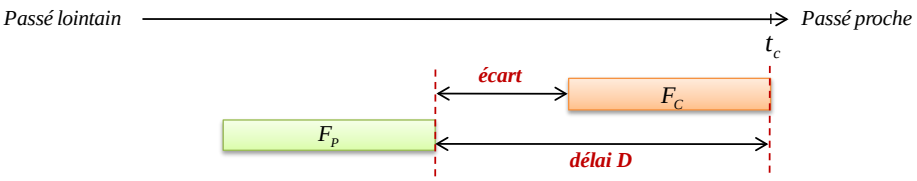

FIGURE 6.2 – Représentation des fenêtres sur le passé proche F_C et le passé lointain F_P.

L'utilisateur doit spécifier un ensemble de paramètres permettant de définir la requête : f_c, f_p, D et Δ (le taux de rafraîchissement de la fenêtre F_C [36]). Le type de ces paramètres dépend de la nature de la fenêtre. Dans le cas des fenêtres physiques, t_c est une date, f_c, f_p et Δ sont exprimés en fonction d'unités temporelles. Dans le cas de fenêtres logiques, tous ces paramètres sont exprimés en utilisant des entiers.

Le tableau 6.2 illustre la liste des notations utilisées pour l'approche statique et dynamique.

Nous présentons deux approches différentes comme solution à la problématique énoncée ci-dessus. La première ("approche statique") se caractérise par l'extraction des évènements à partir de la base. En effet, le SGFD se charge d'extraire du résumé les évènements nécessaires à la requête et, de l'évaluation de celle-ci. La seconde ("approche dynamique")

36. Δ est la fréquence de déplacement de la fenêtre F_C vers le présent. La requête est réévaluée à chaque déplacement.

Paramètre	Symbole
i^{eme} évènement du flux	e_i
Poids du i^{eme} évènement	w_i
Temps courant	t_c
Fenêtre sur les évènements du passé proche (Fenêtre Courante)	F_C
Fenêtre sur les évènements du passé lointain (Fenêtre du Passé)	F_P
Taille de F_C	f_c
Taille de F_P	f_p
Délai	D
Flux retardé de D	S_{-D}
Taux de rafraîchissement : fréquence d'évaluation de la requête	Δ

TABLE 6.1 – Liste des notations

se veut plus générique avec une autonomie et indépendance du rôle du résumé. Elle repose sur la régénération d'un flux à partir des évènements résumés.

6.3 Gestion des résumés

Notre système se veut générique, capable de répondre à un éventail large de requêtes posées sur le passé (proche et lointain) d'un flux. Ce sont des résumés généralistes qui seront alors intégrés dans l'architecture du système et conservés sur disque dans la base de données. L'utilisation de ces structures permet de :

- Récupérer une représentation des évènements du passé lointain tout en respectant l'espace mémoire alloué ;
- Fournir des informations agrégées ;
- Traiter plusieurs requêtes et tâches de fouille sur le passé lointain d'un flux.

Les approches proposées utilisent un module de résumé permettant de sauvegarder une trace sur les flux. Les résumés peuvent être de structures différentes. Nous avons proposé dans le chapitre 4 une modélisation des résumés conçus à partir de l'algorithme StreamSamp (cf. section 4.2.2) et de l'algorithme CluStream (cf. section 4.3.1.3). Une base de données est associée à ce module pour la conservation des résumés conçus.

Dans le cas des algorithmes d'échantillonnage, la structure du résumé conservé respecte la structure des évènements du flux. Dans ce cas, chaque n-uplet de la base de données a ainsi la structure suivante (i.e. certains algorithmes d'échantillonnage tel que StreamSamp ne conservent pas l'estampille temporelle des évènements) :

Evènement (IdEvènement, Poids, Timestamp, Attribut1, ..., Attributd)

Une nouvelle fenêtre temporelle notée F_R de taille f_r est spécifiée dans le moteur du SGFD pour la construction des résumés. Elle permet de contenir les évènements du flux qui,

suite à leur entrée dans le système, sont envoyés à un algorithme de résumé. Les résumés sont mis à jour au fil de l'eau avec l'arrivée des nouveaux évènements. Les mises à jour sont alors propagées sur la base de données. Dans l'exemple de StreamSamp, la mise à jour du résumé se traduit par le ré-échantillonnage des plus anciens évènements afin d'insérer de nouveaux événements. L'opération consiste à intégrer les événements récents au détriment des anciens événements.

Le traitement des résumés est différent de celui des requêtes. En effet lorsque les requêtes sont supprimées du registre des requêtes, le traitement des évènements est stoppé. En revanche, la conception des résumés est un processus qui ne s'arrête qu'avec l'arrêt des flux. L'approche statique et l'approche dynamique partagent la phase de construction et de conservation du résumé. Cependant, le rôle du résumé diffère d'une approche à l'autre :

- Rôle passif pour l'approche statique : dans cette approche, le rôle du module résumé se limite à la conception et le stockage du résumé dans une base de données. Le module SGFD se charge d'extraire les événements du résumé pour pouvoir évaluer les requêtes.
- Rôle actif pour l'approche dynamique : le module résumé joue le rôle d'une entité autonome. Il permet de re-créer, à partir du résumé conservé, un second flux qui contribuera à l'évaluation de la requête au sein du module SGFD. Cette seconde approche nécessite beaucoup plus de communication entre le module du SGFD et les résumés.

Nous détaillons dans les sections suivantes ces deux approches, notamment le fonctionnement des modules SGFD et résumé pour l'évaluation de requêtes définies sur le passé lointain d'un flux.

6.4 Exploitation statique des résumés

Nous proposons une première approche pour évaluer les requêtes faisant appel à des évènements du passé lointain. Cette méthode est caractérisée par une interrogation statique des résumés. Ces derniers sont conservés en base et de nouveaux mécanismes sont implémentés afin d'assurer l'accès et l'extraction des évènements : ces outils sont appelés opérateurs (i.e. opérateur de sélection, de fenêtrage, d'agrégat, etc.). Ces opérateurs sont directement appliqués sur les résumés.

Dans notre approche, nous avons défini de nouveaux opérateurs de fenêtrage [37] au niveau du sous module "traitement des flux" (cf. Figure 6.1). Ils sont chargés d'extraire à partir du résumé les évènements nécessaires au traitement de la requête. Notre système doit être capable de gérer les requêtes ponctuelles et les requêtes continues sur le passé lointain. Le traitement de ces dernières revient à évaluer des requêtes continues sur une base de données vu que les résumés sont stockés en base.

37. Un opérateur de fenêtrage permet de créer, modifier, nommer, sélectionner et supprimer des fenêtres.

Pour chaque résumé, il faut définir un nouvel opérateur. Il faut de même spécifier à chaque opérateur la stratégie d'accès au résumé. En effet, ce dernier doit connaître au préalable la structure du résumé et prélever les événements en fonction de cette structure.

Le rôle du nouvel opérateur consiste à remplir la fenêtre F_P sur le passé lointain avec des évènements pondérés extraits du résumé. La pondération est la conséquence de l'application des techniques de résumé sur les évènements du flux. Ceci permet de respecter la représentativité des évènements dans le résumé.

Nous présentons dans ce qui suit le fonctionnement et la conception d'un opérateur de fenêtrage. Comme évoqué précédemment, nous nous plaçons dans le cas d'une fenêtre glissante faisant intervenir une période du passé proche (représentée par F_C) et une période du passé lointain (représentée par F_P). La Figure 6.3 schématise le contenu des deux fenêtres F_P et F_C.

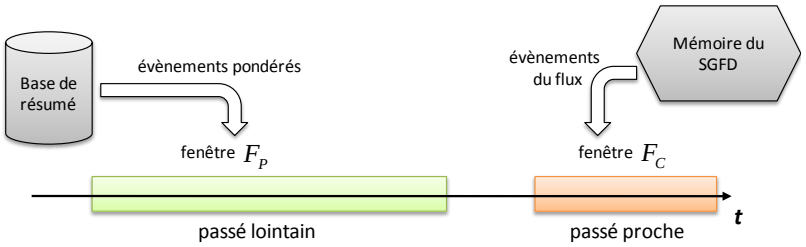

FIGURE 6.3 – Composition des fenêtres sur le passé proche et sur le passé lointain.

Dans le cas d'un algorithme basé sur l'échantillonnage, la fenêtre F_P contient des évènements extraits des échantillons conservés dans les résumés.

La mise à jour du contenu de la fenêtre F_C est automatiquement réalisée via le SGFD (outils standards disponibles). En revanche, la mise à jour de la fenêtre F_P doit se faire spécifiquement par le nouvel opérateur.

Le fonctionnement général de cette approche est décrit comme suit (cf. Figure 6.4) :

- Traitement des flux. Lorsque un flux S entre dans le système, une fenêtre F_R est définie sur ses évènements (étape 1). Chaque fois que cette fenêtre est remplie, son contenu est envoyé vers le module résumé où un algorithme de résumé prendra en charge ces évènements (étape 2). L'algorithme de résumé traite ces évènements en vue de produire un résumé généraliste du flux de données. Ce résumé est conservé sur disque et mis à jour à jour en temps réel (étape 3).
- Traitement des requêtes. Dès l'arrivée d'une requête, le système vérifie s'il s'agit d'une requête standard, ou d'une requête nécessitant des évènements du passé lointain. Dans ce dernier cas, le nouvel opérateur de fenêtrage est invoqué (étape 4). Nous développons l'exemple de la requête 1 [38] présentée dans la section 6.2. Le sous module

38. `Requête 1`. Pour étudier la gestion des forfaits, l'opérateur téléphonique décide de comparer le temps de communication moyen des appels émis, du mois courant, par rapport au temps moyen des appels du même mois de l'année précédente.

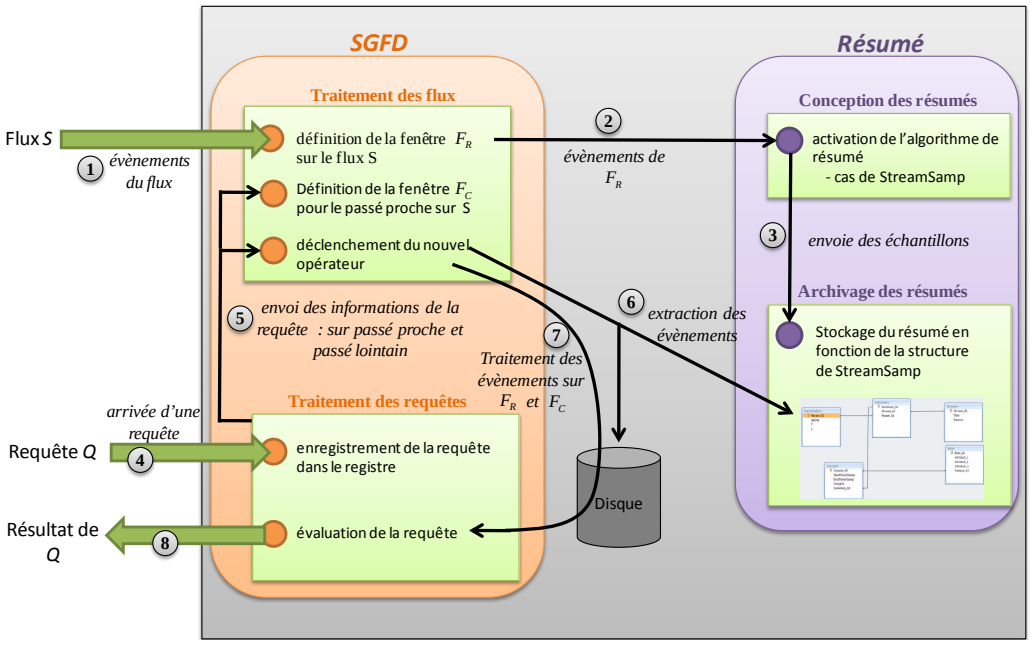

FIGURE 6.4 – Fonctionnement de l'approche statique.

traitement des requêtes envoie les informations suivantes sur les requêtes au module traitement des flux : la taille de la fenêtre F_C (le mois courant dans la requête 1) et l'identifiant du flux. Les opérateurs classiques du SGFD se chargent de créer une fenêtre glissante sur le flux des communications S. Le nouvel opérateur est déclenché. Il reçoit comme information la taille de la fenêtre sur le passé lointain ainsi que le délai (la taille de la fenêtre F_P est de un mois et le délai est de un an dans la requête 1) (étape 5). Le nouvel opérateur de fenêtrage se charge d'extraire du résumé les évènements et de remplir sur disque la fenêtre F_P (étape 6). De même, il a pour rôle de mettre à jour la fenêtre F_P chaque fois que la fenêtre F_C est rafraîchie. Cette mise à jour consiste à extraire les nouveaux évènements du résumé pour gérer le glissement de la fenêtre F_C, tout en tenant compte de la dégradation du résumé en fonction du temps. Le système évalue la requêtes sur le contenu de F_C et F_P (étape 7). Comme les outils standards du SGFD ne sont pas capables de prendre en considération les évènements de la fenêtre F_P, de nouveaux opérateurs sont alors définis au niveau du module traitement de requête. Leur rôle consiste : à évaluer la requête sur les évènements pondérés de la fenêtre F_P et sur les évènements de F_C et, à fournir à l'utilisateur un résultat approché avec un calcul de l'erreur d'estimation. A cet effet, l'opérateur doit être capable d'appliquer les formules d'estimation sur le contenu de la fenêtre F_P.

Nous étudions dans ce qui suit le cas des fenêtres logiques et des fenêtres physiques.

6.4.1 Cas des fenêtres physiques

Dans le cas des fenêtres physiques, tous les paramètres du système doivent être définis en terme d'unités temporelles (e.g. secondes, minutes, etc.). Lorsque la fenêtre sur le passé proche (F_C) glisse de Δ unités de temps, la fenêtre F_P doit être rafraîchie par Δ unités. Le problème consiste à synchroniser le décalage de ces deux fenêtres. Dans le cas des fenêtres physiques, la mise à jour de la fenêtre F_P (cf. Figure 6.5) est réalisée à travers deux opérations :

- Suppression des plus anciens évènements couvrants Δ unités de temps : les événements ayant expiré de la fenêtre F_C (évènements couvrant la période $[t_c - D - f_p, t_c - D - f_p + \Delta]$) sont supprimés. Si par exemple la fenêtre F_C est rafraîchie chaque 30 secondes, il faut supprimer 30 secondes d'événements successifs à partir de F_P en commençant par la borne la plus ancienne de la fenêtre ;
- Insertion de nouveaux évènements : en parallèle à l'opération de suppression, le nouvel opérateur de fenêtrage envoie une requête à la base de données afin d'extraire les nouveaux évènements du résumé et les insérer dans F_P. Formellement, il s'agit d'extraire du résumé les événements qui couvrent la période temporelle $[t_c\text{-}D, t_c\text{-}D+\Delta]$.

Ces deux opérations (s'appliquent / sont appliquées) à des structures de résumé par échantillonnage pour lesquelles les évènements sont datés. Cependant, si la structure du résumé n'inclut pas les estampilles temporelles des évènements (i.e. tel est le cas pour StreamSamp), une solution consiste à dater l'ensemble des évènements inclus dans l'échantillon avec l'estampille temporelle du début de l'échantillon. Cette solution n'est pas optimale, mais permet de résoudre le problème. Une étude équivalente peut être réalisée en utilisant d'autres algorithmes de résumé généralistes. Par exemple, pour l'algorithme CluStream, nous nous basons sur la suppression des clichés pour déterminer les densités d'une période afin de générer les évènements en utilisant l'attribut temporel. Nous fournissons, dans les perspectives de ce document, plus de précisions sur l'intégration de CluStream dans l'architecture proposée.

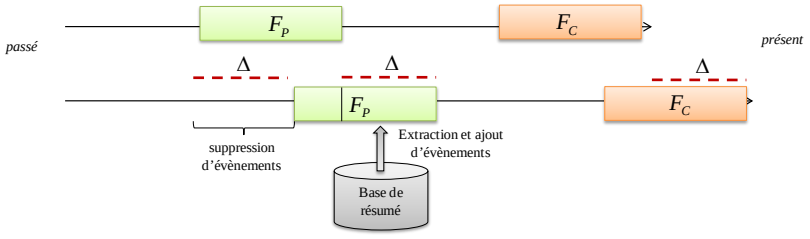

FIGURE 6.5 – Rafraîchissement de la fenêtre sur le passé d'un flux (F_P).

L'algorithme 4 illustre le processus de mise à jours des fenêtres physiques.

Algorithm 4 Mise à jour des deux fenêtres physiques F_P et F_C

Require: Fenêtres : F_C, F_P, f_p : taille de F_P, f_c : taille de F_C, D : délai,
Δ : Taux de rafraîchissement de la fenêtre
$Mettreájour(F_C)$ {Mise à jour automatique par le SGFD}
$i \leftarrow 0$
while i $< F_P$.taille() & $t_c - D - f_p < F_P[i]$.Timestamp $< t_c - D - f_p + \Delta$ **do**
F_P.Supprimer(i) {Supprimer l'événement d'indice i de la fenêtre F_P}
$i \leftarrow i + 1$
end while
Req =SELECT * FROM Table_Résumé WHERE Timestamp BETWEEN $t_c - D$ AND $t_c - D + \Delta$
Résultat $\leftarrow$ Exécuter_requête(Req)
F_P.Ajouter(Résultat) {Insertion des nouveaux évènements avec leurs poids dans F_P}

6.4.2 Cas des fenêtres logiques

Dans le cas de fenêtres logiques, on ne considère plus des fenêtres en termes d'unités temporelles mais des fenêtres en nombres d'évènements. Comme pour les fenêtres physiques, l'opérateur de fenêtrage doit procéder à la synchronisation des fenêtres F_C et F_P. Ainsi, lors de chaque rafraîchissement de F_C, la mise à jour de la fenêtre F_P doit respecter deux phases :

- Suppression des évènements de la fenêtre F_P : cela revient à supprimer les évènements de la fenêtre couvrant Δ. Pour rafraîchir la fenêtre du passé, le système supprime l'équivalent de Δ évènements.
- Extraction des nouveaux évènements et remplissage de la fenêtre F_P : extraction de Δ nouveaux évènements à partir de la base de données pour mettre à jour la fenêtre F_P. Afin de respecter la représentativité des évènements, chaque évènement est considéré avec son poids. Deux solutions s'offrent à nous : (1) la solution implémentée qui consiste à dupliquer l'évènement e_i autant de fois que son poids et de remplir la fenêtre F_P par lot de Δ évènements. Les évènements de la fenêtre auront tous un poids égale à 1. (2) une solution plus optimale consiste à remplir la fenêtre F_P avec des évènements pondérés. On envoie alors l'évènement avec son poids (i.e. l'équivalent de Δ évènements par envoi). Le remplissage de la fenêtre du passé dépend ainsi de cette pondération et de Δ. Quelque soit la solution adoptée, la requête est évaluée sur l'ensemble des évènements qui constituent F_P ainsi que sur F_C.

L'algorithme 5 illustre le processus de mise à jour des fenêtres logiques.

6.4.3 Implémentation de l'approche

Il est à noter que l'approche proposée peut être implémentée avec la plupart des SGFD et SGBD, académiques ou commerciaux. Nous discutons dans cette section l'implantation

Algorithm 5 Mise à jour des deux fenêtres logiques F_P et F_C

Require: Fenêtres : F_C, F_P, f_p : taille de F_P, f_c : taille de F_C, D : délai,
Δ : Taux de rafraîchissement de la fenêtre
$Mettreàjour(F_C)$ {Mise à jour automatique par le SGFD}
$i \leftarrow 0$
while i < F_P.taille() & $t_c - D - f_p < F_P[i]$.Timestamp < $t_c - D - f_p + \Delta$ **do**
F_P.Supprimer(i) {Supprimer l'évènement d'indice i de la fenêtre F_P}
$i \leftarrow i + 1$
end while
Req =SELECT evenement, poids FROM Table_Résumé WHERE Timestamp BETWEEN $t_c - D$ AND $t_c - D + \Delta$
Résultat ← Exécuter_requête(Req)
$D^{'} \leftarrow D$ { $D^{'}$ est une variable qui récupère la nouvelle valeur du délai}
while $D^{'} == D$ **do**
$e \leftarrow RécupérerEvénènement(Résultat, D)$ {Récupérer un évènement à partir du résultat de la requête}
Duppliquer (e, $poids(e)$) {Duppliquer l'évènement par son poids}
Envoyer_Dupliquats (e,F_P, Δ) {Envoyer les évènements dupliqués à F_P par lot de Δ}
$D^{'} \leftarrow CalculerDélai()$ {Calcul de la nouvelle valeur du délai}
end while

de l'opérateur de fenêtrage sur le SGFD Esper [1]. Pour le stockage des résumés, nous avons opté pour un SGBD open-source, il s'agit de PostgreSQL.

Esper est un ensemble de bibliothèques Java permettant à l'utilisateur de créer une application de gestion de flux de données. Ces bibliothèques permettent de déclarer un flux et son schéma, ainsi que de créer des requêtes en EPL (Event Processing Language, le langage d'interrogation d'Esper). Le langage EPL permet l'expression de requêtes complexes qui incluent des fenêtres temporelles, des opérations de jointure sur les flux d'événements, etc. Une requête EPL reçoit en entrée des événements individuels et fournit en sortie soit des événements individuels soit un lot de plusieurs événements selon le type de la fenêtre (fenêtre sautante, une fenêtre bandissante, etc.). Les évènements en entrée sont encapsulés dans des objets qu'on peut facilement exploiter. Considérons la structure suivante d'un flux de données illustrant l'ensemble des communications émises par les abonnés d'un opérateur téléphonique :

```
Communications (timestamp, type_usage, type_appel, nom_client, sexe, durée)
```

La requête illustrée ci-dessous interroge une fenêtre glissante logique de taille 3000 évènements sur le flux Communication.

```
SELECT * FROM Communication.WIN:LENGTH{3000}
```

L'API d'Esper fournit plusieurs outils pour l'extension du langage d'interrogation, non seulement par des fonctions définies par l'utilisateur, mais également par de nouvelles méthodes de fenêtrage et de pattern matching. Cet avantage nous facilite l'extension du SGFD et l'implantation des nouveaux opérateurs pour le traitement des requêtes sur le passé lointain d'un flux. En effet, pour ce type de requête, le SGFD doit être en mesure de communiquer avec une base de données et d'interroger ses relations.

Les requêtes qui sont évaluées sur le passé lointain d'un flux sont des requêtes d'agrégats. Il est nécessaire de ré-implémenter la plupart des fonctions du SGFD afin qu'elles puissent opérer sur des données mémorisées sur disque. L'objectif est que l'utilisateur envoie sa requête au SGFD et que celui-ci lui fournisse le résultat de façon totalement transparente. Nous détaillons dans ce qui suit les grandes phases pour l'implémentation d'une fonction d'agrégat et d'un opérateur de fenêtrage dans le système de gestion de flux de données Esper.

Implémentation des fonctions d'agrégats dans Esper. Le SGFD Esper fournit les outils nécessaires pour l'implémentation de fonctions définies par l'utilisateur (les UDF pour *User Defined Functions*). Ces fonctions auront comme paramètre en entrée la fenêtre sur laquelle sera évaluée la fonction. Lorsqu'il s'agit de requêtes faisant uniquement appel aux données du passé proche, ces fenêtres sont définies dans la mémoire du SGFD. En revanche, dans le cas de requêtes sur passé lointain, les traitements sont réalisés sur disque. Comme illustré dans la figure 6.6, pour chaque fonction d'agrégat, il est nécessaire de définir l'état d'initialisation, l'état de la fonction après la lecture d'une valeur, l'état de la fonction après la lecture de toutes les valeurs et le résultat final de la fonction.

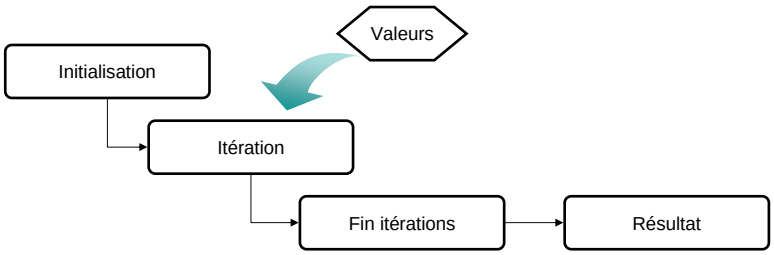

FIGURE 6.6 – Processus de création d'une fonction d'agrégat.

Nous souhaitons que notre système puisse fournir à l'utilisateur un résultat à la requête assorti d'un intervalle de confiance. Pour cela, nous devons mettre en place des fonctions d'agrégats capables de fournir des estimations par intervalle de confiance.

Soit une requête[39] utilisant l'agrégat "AVG_IC". Cet agrégat permet de fournir les bornes de l'intervalle de confiance pour l'estimation de la moyenne. On cherche à déterminer la moyenne de la durée des communications téléphonique entre le 01/10/2003 et 20/10/2003. On suppose que la période interrogée ne fait partie que du passé lointain. Les données de cette période ont été résumées et mémorisées sur disque. Comme illustré dans la Figure 6.7, les données sont pondérées via l'attribut (poids). Il s'agit des évènements nécessaires pour répondre à la requête de l'utilisateur. Cette réponse est réalisée au niveau du nouvel opérateur en exécutant les étapes suivantes :

1. Filtrer les données de la table résumé "Communications" afin d'extraire les événements compris entre le 01/10/2003 et 20/10/2003 ;
2. Calculer l'estimateur ainsi que l'intervalle de confiance de la moyenne en utilisant les événements pondérés ;
3. Retourner les bornes de l'intervalle de confiance à l'utilisateur. Dans cet exemple, les évènements ne font partie que de la fenêtre F_P. Cependant, si la requête inclus des évènements du passé proche (fenêtre F_C), ces derniers doivent êtres comptabilisés dans le calcul de l'agrégat. La requête s'évalue sur les évènements de la fenêtre F_C (i.e. evènements auxquels on associe un poids égal à 1) et sur les évènements de F_P qui sont déjà pondérés. Le calcul des bornes de l'intervalle de confiance suit les règles illustrées dans la section 4.2.3.2.

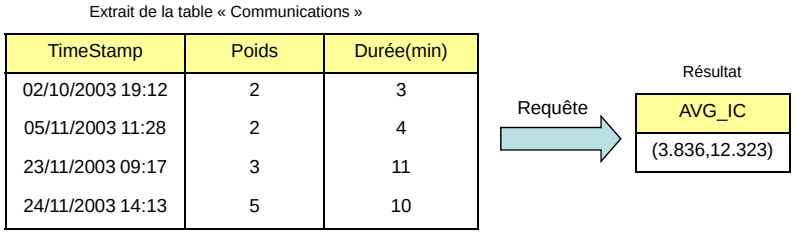

FIGURE 6.7 – Utilisation de la fonction d'agrégat pour l'évaluation d'une requêtes du passé sur le résumé du flux Communication

Implémentation d'un opérateur de fenêtrage. Les opérateurs de fenêtrage fournis avec le SGFD ESPER ne permettent pas de remplir des fenêtres avec des évènements du passé lointain. Il est ainsi nécessaire de développer un nouvel opérateur de fenêtrage permettant de prendre en considération lors de l'évaluation de la requête, les évènements extraits de la base des résumés et les évènements du passé proche présents dans la mémoire du SGFD. Pour cela, nous avons développé un nouvel opérateur intitulé "ns :HistWindow"

39. SELECT AVG_IC(Durée) FROM Communications WHERE timestamp between 01/10/2003 AND 20/10/2003

pour le traitement des requêtes sur le passé lointain d'un flux. Il s'agit d'une extension de l'implémentation classique des fenêtres glissantes dont la particularité est la possibilité de se connecter et d'accéder aux données d'une base de données. Cette fenêtre peut être définie en terme de nombre d'événements (fenêtre logique) ou en terme d'unités de temps (fenêtre physique). Un exemple d'utilisation de cet opérateur de fenêtrage est présenté dans l'annexe C.2.

6.4.4 Limites de l'approche

L'apport de l'approche statique consiste à développer de nouveaux opérateurs de fenêtrage permettant la gestion de fenêtres sur les évènements du passé lointain. Plusieurs inconvénients découlent de cette méthode, principalement la nécessité de définir tous les outils essentiels au traitement des requêtes faisant appel à des évènements du passé lointain à savoir :

- Définir toutes les fonctions d'agrégat que le système va par la suite utiliser ;
- Définir les nouveaux opérateurs de fenêtrage ainsi que la stratégie à employer pour l'évaluation de la requête ;
- Le comportement passif du résumé engendre une dépendance entre la structure du résumé utilisé et les nouveaux opérateurs à implémenter ;
- La mise à jour de la fenêtre F_P consiste à exécuter une requête continue sur une base de données, ce qui est coûteux et difficile à réaliser notamment pour les flux à haut débit.

Ces extensions rendent ainsi difficile l'utilisation de cette approche. Une approche plus naturelle, qui exploite au maximum les outils standards du SGFD est proposée dans la suite de cette section. Elle permet à la fois de répondre aux requêtes définies sur le passé lointain d'un flux et, de profiter par la même occasion des outils déjà présents dans les SGFDs.

6.5 Exploitation dynamique des résumés

Nous présentons dans ce qui suit une nouvelle approche intitulée "DSi" (pour *Delay System infrastructure*). L'objectif de cette approche consiste à traiter les requêtes sur le passé lointain des flux de façon naturelle, au même titre que les requêtes standard appliquées sur les flux de données. En effet, cette approche se base sur la réutilisation des composants existants dans les SGFD et l'intégration d'un nouveau module (module de résumé) qui permet de conserver l'histoire des flux. En plus de sa fonction principale (i.e. conception et gestion des résumés), le module de résumé est actif et indépendant. Il doit assurer deux rôles : construction des résumés et régénération des flux à partir de l'historique conservé. Les traitements sont ainsi divisés entre le module du SGFD et le module du résumé.

– Construction du résumé : cette phase est commune à l'approche statique et dynamique, elle permet de maintenir une version compacte du flux de données afin de pouvoir interroger les données du passé distant tout en respectant l'espace mémoire disponible ;
– Régénération du résumé : le résumé doit par ailleurs être capable de rejouer, sous forme de flux de données, une partie du résumé. Le SGFD lance alors une requête au module de résumé pour activer son mécanisme de ré-émission de données à partir d'un évènement dans le résumé. L'idée consiste à rendre le module du résumé autonome et capable de ré-émettre sous forme d'un flux les évènements dont le système a besoin pour évaluer les requêtes sur le passé lointain. Ce nouveau flux sera traité par le SGFD comme un flux ordinaire alors qu'en effet, il s'agit d'un flux retardé ayant la même structure que le flux d'origine. L'évaluation de la requête est effectuée sur les flux d'origine et les flux retardés, et non sur les résumés.

Le système DSi doit être capable d'évaluer les requêtes ponctuelles ainsi que les requêtes continues. Le traitement de ces requêtes est réalisé en deux temps : (i) régénération du flux (requête continue posée sur la base de données)(ii) évaluation de la requête posée via les outils standard du SGFD.

Nous désignons par **"flux retardé"** (noté S_{-D}) l'ensemble des évènements qui sont extraits du résumé et ré-émis par une source interne (i.e. module de régénération) dans le SGFD et, par **"flux d'origine"** (noté S), l'ensemble des évènements qui entrent dans le système et qui sont envoyés par une source externe (e.g. capteur).

6.5.1 Architecture du système DSi

Comme pour la première approche, le système que nous proposons se compose de deux principaux modules : le module du SGFD et, le module du résumé. La différence réside dans le module du résumé qui se compose de deux sous-modules : le sous-module de construction et, le sous-module de régénération. La Figure 6.8 illustre l'architecture du système. Nous décrivons dans les sections suivantes, les différents composants du système et notamment le cas des fenêtres logiques et physiques.

– Le SGFD : ce composant a pour entrée les évènements du flux ainsi que les requêtes posées par l'utilisateur. Son rôle consiste à gérer les fenêtres sur les différents flux (flux d'évènements et flux retardés), évaluer les requêtes et gérer la coordination entre les différents modules ;
– Le module du résumé : permet de concevoir et gérer les résumés conservés en base de données. Ce module comporte deux processus : le sous-module de construction (appelé aussi *Builder*) et le sous-module de régénération (ou *Replayer*). Le premier permet la construction et la gestion des résumés tandis que le second est chargé de ré-émettre les évènements du résumé sous forme de flux retardés et, assurer la synchronisation des fenêtres sur le passé proche et sur le passé lointain.

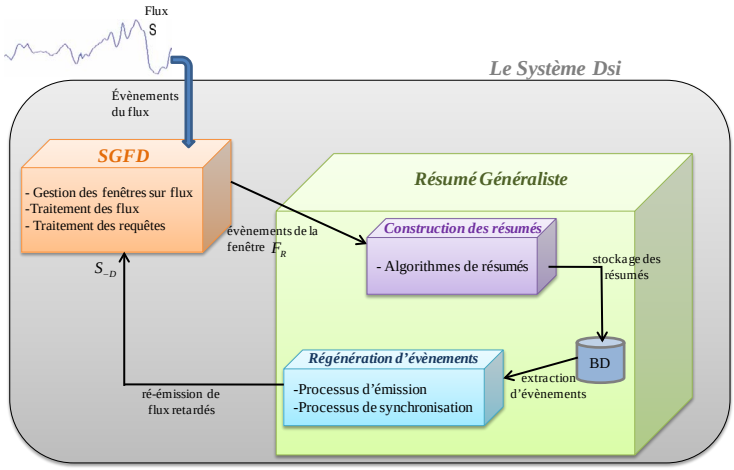

FIGURE 6.8 – Architecture du système DSi.

Parmi ses différentes tâches, le système DSi doit établir un mécanisme de communication entre l'algorithme de résumé et le SGFD. Cette tâche n'est pas simple. Elle nécessite une coordination entre le SGFD et le module de gestion du résumé. En effet, un agencement entre l'évolution de la fenêtre sur le passé proche et le remplissage de la fenêtre sur le passé lointain après extraction des données de la base est nécessaire. La solution que propose DSi pour faciliter cette communication est d'intégrer un nouveau composant permettant la relecture des données du résumé et leur ré-injection dans le SGFD sous l'étiquette d'un flux retardé.

Fonctionnement des modules dans le composant résumé. Le système DSi est organisé de façon à ce que chaque module soit indépendant. Les modules communiquent entre eux par envoi de messages internes. Les rôles sont séparés et définis en vue d'étendre le système au traitement des requêtes sur passé lointain. La Figure 6.9 décrit le mécanisme entre les différents composants de cette architecture.

- **Rôle du SGFD.** Dès l'arrivée des flux S et S_{-D}, le SGFD les traite sans différencier les flux originaux des flux retardés. Le traitement est classique, les outils standard sont réutilisés. La seule différence réside dans la définition même de la requête, qui au départ est définie sur un seul flux mais au final est évaluée sur deux flux. A cet effet, un module de traduction de requêtes doit être introduit, il permet de redéfinir les requêtes en intégrant les flux retardés. Concrètement, le SGFD doit traiter les évènements de la fenêtre F_C (i.e. fenêtre définie et gérée par le SGFD sur le flux S) simultanément avec les évènements de la fenêtres F_P (i.e. fenêtre définie et gérée par le SGFD sur le flux S_{-D}).
- **Rôle du sous-module construction des résumés**. Grâce à des algorithmes de résumé embarqués dans ce module, le système DSi permet de concevoir des résumés

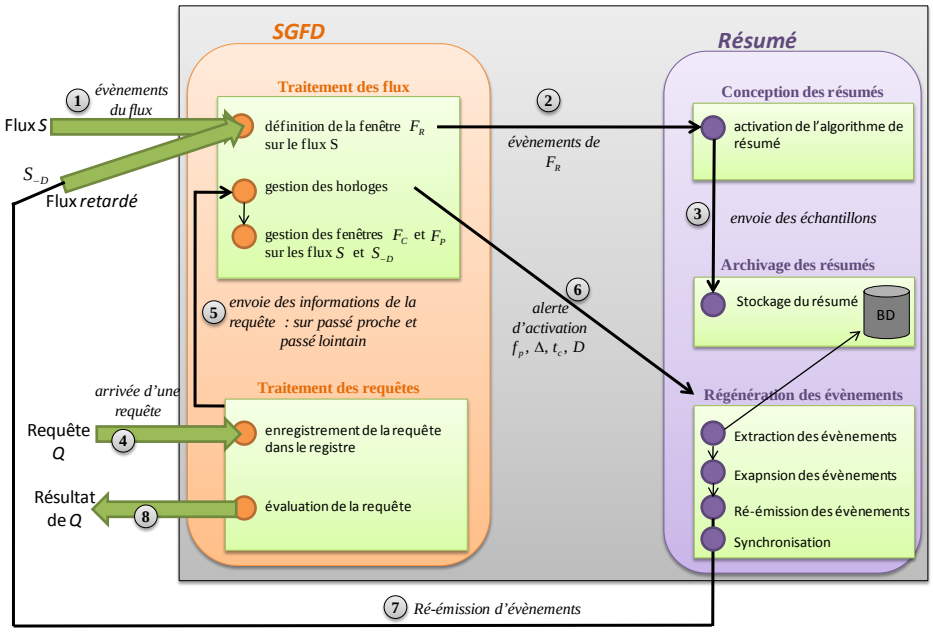

FIGURE 6.9 – Fonctionnement de l'approche DSi.

généralistes de flux de données et les conserver sur disque. Ce module est activé dès l'arrivée du premier évènement du flux S par alerte du SGFD. Il a pour entrée les évènements inclus dans la fenêtre F_R.

– **Rôle du sous-module régénération**. Ce module permet de jouer un ou plusieurs flux retardés en ré-émettant les évènements archivés du résumé vers le SGFD. Dès qu'une requête portant sur des données du passé lointain est enregistrée, ce module est activé par appel du SGFD.

Contrairement à l'approche statique dans laquelle toutes les requêtes se font sur le résumé conservé en base, le sous-module de régénération offre l'avantage d'exécuter une seule requête continue sur la base de données. Il s'agit de la requête permettant la régénération du flux. Toutes les autres requêtes (calcul de l'agrégat, les filtres, etc.) sont à la charge du SGFD. Ainsi dans l'approche dynamique, le traitement d'une requête sur le passé lointain est partagé entre le module résumé et le SGFD. L'application d'une seule requête continue sur une base de données, facilite l'optimisation de l'opération d'archivage dans la base. Tandis que, dans l'approche statique, le mécanisme d'optimisation doit prendre en considération toutes les requêtes continues qui peuvent être évaluées sur le résumé.

Le module de régénération intègre deux processus permettant de respecter le délai D et, assurer la ré-émission des évènements conservés en base :

– Un processus d'émission chargé de l'extraction, la préparation et l'envoi des évènements sous forme de flux retardés ;

– Un processus de synchronisation qui maintient le délai D entre la fenêtre F_P définie sur le flux retardé et le temps courant t_c. Il synchronise ainsi les événements retardés de F_P avec les événements du passé proche dans F_C.

Nous présentons dans ce qui suit le module de régénération en détail tout en distinguant son fonctionnement dans le cas des fenêtres logiques et des fenêtres physiques.

6.5.2 Module de régénération

Nous avons évoqué précédemment le rôle de ce module qui est de ré-émettre les évènements du flux ayant été résumés et archivés. Pour atteindre cet objectif, deux processus sont utilisés. Le premier concerne la phase de reconstitution des flux retardés tandis que le second processus vise à synchroniser les fenêtres sur le passé proche avec celles définies sur le passé lointain. Comme illustré dans la Figure 6.10, ces deux processus s'enchaînent. Le processus de synchronisation est tout d'abord activé, il reçoit une alerte de la part du SGFD afin de notifier le processus d'émission. Ce dernier reçoit comme paramètres : f_p la taille de la fenêtre sur le passé lointain, D le délai qui sépare le temps courant de la fenêtre F_P, t_c l'instant courant et Δ le taux de rafraîchissement de F_C. A la réception de ce message, le processus de ré-émission lance trois actions successives : extraction, expansion et émission.

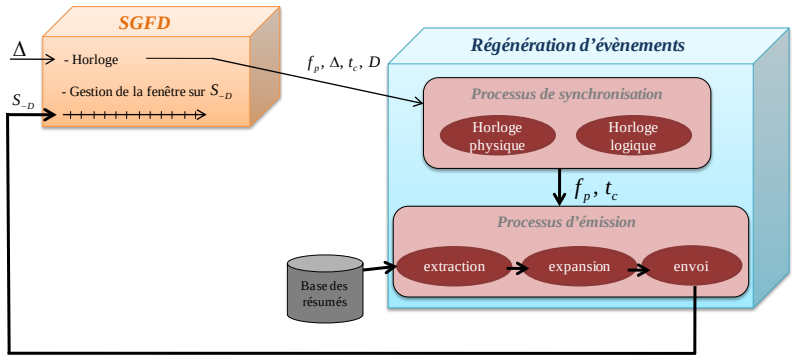

FIGURE 6.10 – Composition du module de régénération.

– La phase d'extraction requiert l'information sur le taux de rafraîchissement de la fenêtre F_C qui lui permet de prendre connaissance du nombre d'évènements (dans le cas logique) ou de l'intervalle temporel (dans le cas physique) à extraire de la base. Il nécessite par ailleurs le délai et le temps courant afin de délimiter l'ensemble des évènements à extraire à partir de la base. Une fois les évènements prélevés du résumé, ceux-ci sont envoyés à la phase d'expansion qui se charge de les préparer avant leur ré-émission dans le SGFD.
– La phase d'expansion requiert comme information en entrée l'ensemble des évènements extraits du résumé dans la phase précédente. Ce processus se charge de les

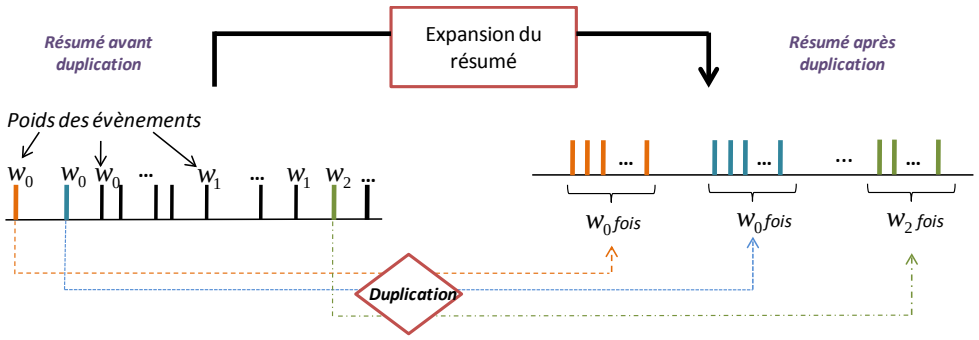

FIGURE 6.11 – Expansion du résumé.

préparer afin de reconstituer la structure du flux d'origine et, recréer la représentativité des évènements. Les évènements sont alors dupliqués en fonction de leur poids. Étant donné que le résumé évolue en fonction du temps, le dernier poids observé est pris en considération lors de l'opération d'extraction. Comme illustré dans la Figure 6.11, lorsque un événement e_i ayant un poids w_i est sélectionné, il est dupliqué w_i fois.

Dans le cas de résumés conçus par échantillonnage, la structure du flux d'origine reste inchangée et, les évènements conservés sont pondérés.

– La phase d'envoi a pour entrée la liste des évènements traités ainsi que le paramètre Δ. Ce processus se charge de la ré-injection des évènements dans le flux S_{-D}. La politique de ré-émission dépend du type de la fenêtre.

6.5.2.1 Cas des fenêtres physiques

Chaque fois que la fenêtre F_C glisse de Δ unités de temps, la fenêtre F_P définie sur le flux S_{-D} doit elle aussi glisser de Δ unités. Contrairement à l'approche statique, l'opération de mise à jour de F_P est réalisée par le SGFD de façon naturelle étant donné que le flux S_{-D} est considéré comme un flux indépendamment du fait qu'il soit régénéré. Cette opération est transparente pour le SGFD qui ne se charge pas de la provenance du flux S_{-D}. Par conséquent, le SGFD gère les flux originaux et retardés au même titre. Parallèlement, les évènements couvrant la période $[t_c - D - f_p,\ t_c - D - f_p + \Delta]$ sont extraits du résumé pour alimenter le flux S_{-D}.

Le cas des fenêtres physiques est simple, l'évènement ainsi que l'ensemble de ses répliques sont envoyées, en une seule fois, à la fenêtre F_P. Contrairement aux fenêtres logiques, il n'y pas de contrainte sur la taille maximale de la fenêtre physique.

6.5.2.2 Cas des fenêtres logiques

Le traitement des fenêtres logiques est plus complexe étant donné que pour tout glissement de la fenêtre F_C de Δ évènements, la fenêtre F_P doit elle aussi être déplacée du même nombre d'évènements. Comme le cas des fenêtres physiques, le SGFD est chargé de la mise à jour de la fenêtre F_P. Cependant, le processus d'envoi des évènements est plus pointilleux étant donné qu'on est amené à envoyer les évènements par lots de Δ :

- Lorsqu'un évènement appartenant à la période $[t_c - D,\ t_c - D + \Delta]$ est extrait du résumé, il est dupliqué en fonction de son poids et envoyé par lots de Δ évènements vers le SGFD jusqu'à l'envoi de la totalité des répliques.
- Lorsqu'un évènement e_i est ré-émis, l'ensemble de ses répliques doivent intégrer le SGFD. Les répliques d'un évènement sont des clones de celui-ci. Elles ont alors la même estampille temporelle que e_i. L'injection dans le SGFD d'un lot d'évènements ayant la même estampille peut engendrer un décalage entre le temps courant t_c et le début de la fenêtre F_P (i.e. le délai n'est plus respecté). Pour régulariser ce décalage, une fois que la totalité des répliques est envoyée, le module de régénération extrait à partir de la base de données l'évènement qui permet de respecter le plus possible le délai D. Certains évènements du résumé peuvent par conséquent être écartés étant donné qu'ils ne respectent pas le délai initial.

6.5.3 Processus de synchronisation

Le processus de synchronisation est chargé de coordonner l'expiration des évènements sur F_C avec l'envoi des évènements vers F_P. Ce processus gère des alarmes qui ont pour rôle d'envoyer des alertes au processus d'émission chaque fois qu'une mise à jour s'opère sur la fenêtre F_C.

Comme nous l'avons indiqué précédemment, le processus de régénération diffère en fonction du type de la fenêtre. Ainsi, pour assurer la notification des alertes en respectant les types des fenêtres, deux horloges sont mises en place : une horloge physique et une horloge logique. Ces horloges sont de nature différente mais partagent le même rôle. Nous illustrons dans ce qui suit le fonctionnement du processus de synchronisation dans le cas de fenêtres logiques et dans le cas de fenêtres physiques.

6.5.3.1 Cas des fenêtres physiques

Comme illustré dans la Figure 6.12, le processus de synchronisation dispose d'une horloge physique pour gérer la ré-émission des évènements. A chaque mise à jour de F_C, l'horloge physique envoie une notification au processus d'émission indiquant que les fenêtres F_C et F_P ne sont pas synchronisées (i.e. le délai initial n'est plus respecté). Suite à cette notification, le processus d'émission est ainsi activé afin de régner les nouveaux évènements nécessaires à l'évaluation de la requête. Les algorithmes 6 et 7 illustrent la mise à jour des fenêtres physiques. Le principe est le suivant : le gestionnaire de F_C envoie une notification

au gestionnaire F_P qu'il a été mis à jour de Δ unités de temps. A la réception de cette alerte, le processus de synchronisation envoie un ensemble de paramètres (l'instant courant t_c, le délai D et le taux de rafraîchissement Δ) au processus d'émission. Ce dernier envoie les nouveaux évènements au SGFD. Les outils standards du SGFD sont alors appliqués après réécriture de la requête [40] en intégrant les flux retardés dans sa définition.

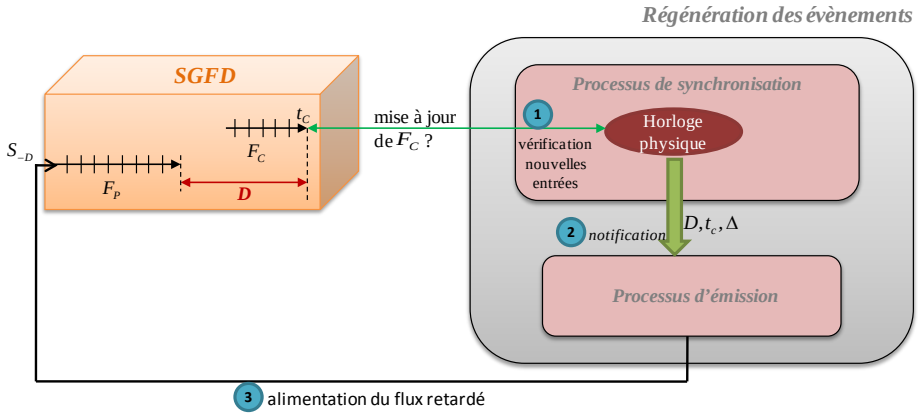

FIGURE 6.12 – Processus de synchronisation en utilisant des fenêtres physiques

Algorithm 6 Gestionnaire de la fenêtre F_C (Cas des fenêtres physiques)

Require: F_C, Δ

1: **while** F_C est mis à jour par Δ unités de temps **do**
2: t_c : temps courant
3: $EnvoyerAlarme(ProcessusSynchronisation, t_c, \Delta)$ {L'horloge physique envoie une alarme au processus de synchronisation}
4: **end while**

Algorithm 7 Gestionnaire de la fenêtre F_P (Cas des fenêtres physiques)

Require: F_P, D, Δ, t_c

1: AttendreSignal(GestionnaireF_C)
2: **for** $i = t_c - D$ to $t_c - D + \Delta$ **do**
3: Supprimer(e_i, F_P) {Supprimer les évènements expirés de F_P}
4: **end for**
5: **for** $i = t_c - D$ to $t_c - D + \Delta$ **do**
6: Envoyer$F_P(e_i, w_i)$ {Envoyer l'évènement e_i, w_i fois à F_P}
7: **end for**

40. Lorsque la requête fait appel à des évènements du passé lointain, une phase de réécriture est nécessaire afin que celle-ci puisse faire appel aux évènements du flux retardé et du flux d'origine lors de son évaluation.

6.5.3.2 Cas des fenêtres logiques

Dans le cas de fenêtre logiques, le processus de synchronisation fait appel à une horloge logique afin de coordonner les fenêtres F_P et F_C (cf. Figure 6.13). Suite à une mise à jour de Δ évènements sur F_C, une alerte est envoyée au processus de synchronisation. Ce dernier active le processus d'émission en lui envoyant les informations suivantes : t_c, D et Δ. Dès la réception de ces paramètres, le processus d'émission procède à l'alimentation du flux retardé $S - D$. Lorsque toutes les répliques d'un évènement sont envoyées au SGFD, le processus d'émission envoie une notification au processus de synchronisation. Suite à cela, le processus de synchronisation lui retourne le nouveau t_c qui lui permettra d'extraire l'évènement qui rétablit le délai D (cf. section 6.5.2).

Les algorithmes 8 et 9 illustrent le principe des mises à jour dans le cas des fenêtres logiques.

Algorithm 8 Gestionnaire de la fenêtre F_C (Cas des fenêtres logiques)

Require: F_C, Δ

1: **while** F_C est mis à jour par Δ évènements **do**
2: t_c : temps courant
3: $EnvoyerAlarme(ProcessusSynchronisation, t_c, \Delta)$ {L'horloge du SGFD envoie une alarme au processus de synchronisation}
4: **end while**

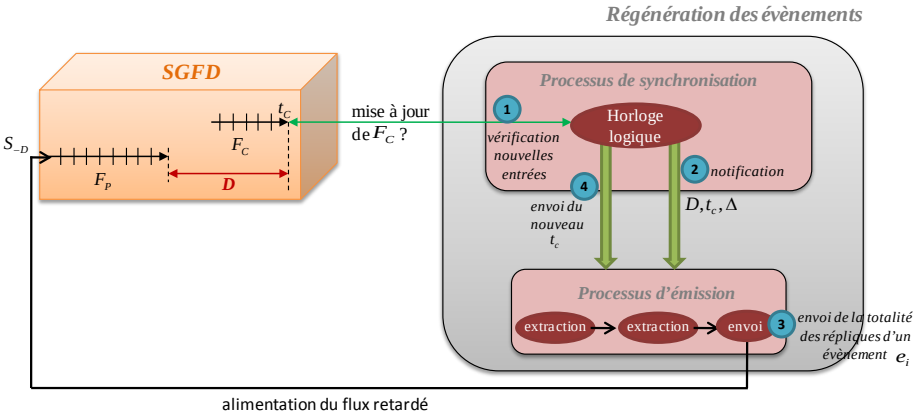

FIGURE 6.13 – Processus de synchronisation en utilisant des fenêtres logiques

La mise en place d'un processus de synchronisation offre plusieurs avantages :

– Point de vue architectural : il est possible de fusionner les processus d'émission et de synchronisation en un seul processus. Cependant, en ingénierie des systèmes, il est important de moduler le plus possible les différentes structures de l'architecture. Ceci permet de détecter plus facilement les pannes et d'agir en conséquence ;

Algorithm 9 Gestionnaire de la fenêtre F_P (Cas des fenêtres logiques)

Require: F_P, D, Δ, t_c

1: **for** $i = t_c - D + 1$ to $t_c - D + \Delta$ **do**
2: Supprimer(e_i, F_P) {Supprimer les évènements expirés de F_P}
3: **end for**
{Extraction}
4: WAIT(alarm from synchroProcess)
5: List [ ] evts = Extraction($i = t_c - D + 1$, $t_c - D + \Delta$) {Extraire de la Base de données la liste des évènements de la période $[i = t_c - D + 1, t_c - D + \Delta]$ avec leurs poids.}
{Expansion des évènements}
6: $i \leftarrow 0$
7: $D^{'} \leftarrow D$ { $D^{'}$ est une variable qui récupère la nouvelle valeur du délai}
8: **while** $i < evts.taille()$ ET $D^{'} == D$ **do**
9: Liste [ ] dupEvts = duplicate(elems[i].value, elems[i].weight) {Liste des répliques d'un évènement.}
10: Supprimer(dupEvts, Δ) {Supprimer de la liste, les répliques envoyés.}
11: EnvoyerF_P(dupEvts, Δ) {Envoyer les répliques à F_P par lots de Δ évènements.}
12: $D^{'} \leftarrow CalculerDélai()$ {Calcul de la nouvelle valeur du délai}
13: **end while**
14: **if** $D! = D^{'}$ **then**
15: Extraire(e,D) {Extraire l'évènement e qui permet de respect le délai initial D.}
16: Envoyer(e)
17: **end if**

– Mutualisation des requêtes : il est important de mettre en place un processus de synchronisation lorsque le système gère plusieurs requêtes et fenêtres simultanément. Si plusieurs requêtes font intervenir des données du passé lointain d'un même flux, le système devra alors gérer plusieurs flux retardés. Ces requêtes peuvent de même partager des périodes temporelles en commun (partage des événements). Ainsi, il serait inutile que le module de régénération ré-émette plusieurs fois les mêmes évènements. Le processus de synchronisation jouera alors le rôle d'ordonnanceur afin de gérer ces ré-émissions. Ce point relève de l'optimisation des requêtes et du partage des ressources. Il n'est pas développé dans ce mémoire.

6.6 Contraintes de fonctionnement des approches

Pour que les deux approches décrites ci-dessus puissent fonctionner correctement, il faut s'assurer qu'à l'instant d'évaluation de la requête :

– Les deux fenêtres F_C et F_P sont remplies et disponibles pour l'évaluation de la requête ;

– Les évènements ayant une estampille temporelle inférieure ou égale à $t_c - f_c$ (i.e. évènements ayant expiré de la fenêtre F_C) ont déjà été sauvegardés dans le résumé.

Soit τ le délai (e.g. en ms) nécessaire pour traiter, écrire et valider en base un évènement du flux. Un évènement e_t est disponible en base à partir de l'instant $t + \tau$. Soit d le débit d'arrivée des évènements du flux (e.g. en nombre d'évènements par ms). Si ce débit est constant, la fenêtre F_C couvre une période égale à f_c/d. Afin d'assurer le bon fonctionnement de l'approche, il faut que la contrainte $\tau < (f_c/d)$ soit respectée quelque soit le type de la fenêtre (physique ou logique).
On peut dans la pratique supposer qu'il est difficile d'agir sur le temps de latence τ et le considérer par conséquent comme constant. De même, il est difficile d'agir sur le débit d'un flux. Par conséquent, la contrainte doit porter sur f_c qui doit être supérieure à $\tau * d$. Dans le cas où les évènements du flux arrivent avec un débit variable, cette contrainte doit être appliquée sur le débit maximum du flux (d_{max}). Toutefois, si cette condition ne peut pas être satisfaite, nous pouvons envisager une technique d'échantillonnage sur la fenêtre glissante F_C. Ainsi, cette fenêtre ne contient plus les f_c évènements du flux les plus récents mais plutôt un échantillon aléatoire de taille inférieure à f_c.

6.7 Requêtes de sélection

Nous avons précédemment présenté le fonctionnement de l'architecture statique et l'architecture dynamique pour une requête d'agrégat sur le flux global. Cependant, ces approches peuvent de même intégrer des opérateurs de sélection. Nous notons deux types de sélection :

– Sélection temporelle : cette sélection existe dans les SGFD actuels. A ce titre, son utilisation dans nos deux architectures se fait de façon naturelle. Le traitement de ce type de requête est simple. Il s'agit de sélectionner les événements suivant leurs estampilles temporelles et les intervalles de la requête.
– Sélection par attribut : il s'agit de sélectionner les événements qui respectent une condition généralement liée à un ou plusieurs attributs de l'évènement (i.e. sélectionner les clients ayant plus de 60 ans et de sexe féminin : sélection suivant les attributs âge et sexe).

Les approches statiques et dynamiques permettent l'utilisation des opérateurs de sélection. Cependant, chaque approche traite les requêtes de sélection en suivant ses contraintes architecturales.

– Avec l'approche statique, le SGFD doit extraire les évènements, à partir du résumé, qui respectent la sélection posée dans la requête. Pour assurer cette extraction, le système doit être équipé de tous les outils nécessaires, il faut alors spécifier pour chaque opérateur de sélection et pour chaque type de résumé la stratégie de sélection des évènements. Un tel traitement est complexe, il requiert l'implémentation et l'intégration de toutes les stratégies.

– Avec l'approche dynamique, plusieurs stratégies peuvent être appliquées. On peut soit régénérer le flux en entier, soit régénérer le flux déjà filtré. Dans la première technique, les évènements extraits du résumé passent par un opérateur de filtrage avant qu'ils soient réémis vers le SGFD. Cet opérateur (implémenté dans le module résumé) se charge d'appliquer une sélection sur ces évènements et de ne ré-émettre sous forme de flux retardés que ceux qui respectent la contrainte imposée par la requête. Dans la seconde approche, le module de résumé régénère le flux en entier et le filtre est appliqué au niveau du SGFD.
Contrairement à l'approche statique, l'évaluation des requêtes de sélection dans l'approche dynamique est une tâche "simple".

6.8 Expérimentations

Nous présentons dans cette section une série d'évaluations permettant de mesurer les performances des approches statique et dynamique lors du traitement de requêtes faisant appel à des données du passé lointain.

Les deux approches décrites dans ce chapitre font appel à une structure de résumé. L'utilisation d'une structure de résumé évolutive associée à un environnement de flux de données implique plusieurs contraintes (i.e. temps de construction du résumé, espace occupé, résistance au débit, etc.) au niveau de l'évaluation des requêtes. L'ensemble des expérimentations illustrées dans cette section sont liées à ces contraintes.

Le premier volet des expérimentations consiste à calculer, pour chaque approche, le taux d'erreur dans la réponse à une requête. Cette étude permet de comparer la résistance des approches face à des flux à débits variables.

Le second volet des expérimentations concernera le module du résumé étant donné qu'il s'agit d'un module commun aux deux approches. Nous allons tout d'abord calculer le temps nécessaire pour la construction du résumé en utilisant trois algorithmes d'échantillonnage (StreamSamp, approche réservoir et échantillonnage uniforme). Le choix de ces algorithmes n'est pas trivial, nous avons choisi des algorithmes qui présentent des mécanismes de pondération différents. Nous étudions par la suite le comportement de ces trois algorithmes par rapport à des variations au niveau du délai. L'idée consiste à analyser la sensibilité de chaque algorithme de résumé.

6.8.1 Environnement de développement

Nous illustrons dans ce qui suit les différents paramètres et configuration utilisées pour la réalisation des expérimentations.

– ***SGFD.*** Les expérimentations ont été réalisées en utilisant SGFD ESPER. Une description plus détaillée de ce système est illustrée dans la section 1.7.4.2.

– ***Configuration.*** Toutes les expérimentations ont été réalisées sur une machine dotée d'un processur ayant une fréquence de 2 Ghz et d'une mémoire vive de 2.0 GO. Les deux approches ont été développé avec le langage Java en utilisant la biobliothèque Esper et une distribution Ubuntu de Linux.
– ***Jeu de données.*** Pour l'évaluation des deux approches, nous avons utilisé le jeu de données des communications téléphoniques France Télécom décrit dans l'annexe A.3.
– ***Liste des paramètres.*** Le tableau 6.2 présente les paramètres utilisés dans les expérimentations. Ces paramètres ont été définis afin que le système puisse interroger une longue période faisant appel à un passé lointain. La requête utilisée dans toutes les expérimentations est la suivante : ("*Requête 3 : calculer la durée moyenne des appels émis par les clients de l'opérateur téléphonique sur une fenêtre glissante de 1100k évènements. Nous supposons que la mémoire du SGFD ne peut contenir que 100k évènements.*")

TABLE 6.2 – Liste des paramètres

Paramètre	Valeur
f_c	10^5évènements
f_p	10^6évènements
Retard D	4.10^5
Taux de rafraîchissement Δ	10^3
Flux lent	100 evt/sec
Flux rapide	1000 evt/sec

6.8.2 Étude des performances des deux architectures

Dans cette expérimentation, nous étudions l'impact du débit du flux sur la qualité des réponses retournées pour chaque approche. Il s'agit ici d'une simulation réalisée sur les approches que nous avons développées. L'objectif consiste à comparer la robustesse de l'approche dynamique et celle de l'approche statique face à des flux avec des débits très variables. Les deux approches utilisent une base de résumé pour conserver les résumés du flux.

La qualité des réponses est calculée en utilisant l'erreur moyenne relative [41] sur la durée moyenne des appel. Nous avons simulé l'arrivée du flux avec deux débits différents : un flux "lent" avec un débit de 100 événements par seconde et un flux "rapide" avec un débit de 1000 événements par seconde.

41. L'erreur relative d'un estimateur $\widehat{X_0}$ est $\left|\widehat{X_0} - X_0\right| / X_0$.

Le débit du flux influence le résultat des requêtes. En effet, si le temps de traitement d'un événement est lent par rapport au débit du flux, le système risque de perdre certains événements qu'il ne pourra pas traiter. En effet, un retard dans la mise à jour des fenêtres F_C et F_P peut être observé, il est dû à la phase d'extraction des évènements du résumé conservé en base. Plus le débit est important plus le système risque de perdre des évènements s'il n'arrive pas à extraire rapidement la partie du résumé qui l'intéresse.

La Figure 6.14 présente les résultats obtenus pour cette évaluation. Nous observons qu'en fonction du débit des flux, les performances des deux approches varient. En effet, pour le flux à débit faible, les performances des deux approches semblent similaires. En revanche pour un flux à débit élevé, nous constatons une baisse du régime pour l'approche statique. Tandis que l'approche dynamique a conservé des performances à peu près proches de celles observées pour le flux lent. Cette dernière présente des résultats plus réguliers.

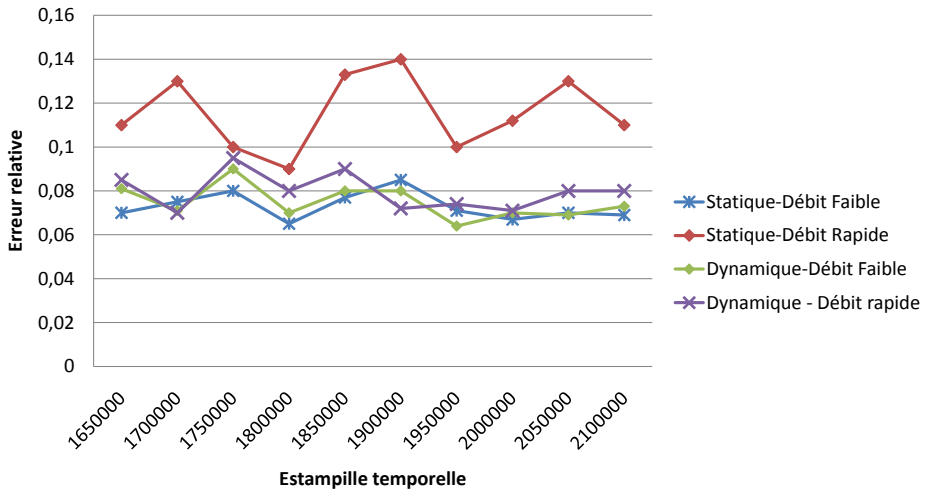

FIGURE 6.14 – Résistance des approche aux variations du débit du flux

Ces performances s'expliquent par la stratégie adoptée pour accéder aux résumés, extraire les évènements et répondre à la requête posée. Dans la première approche, le SGFD se charge d'extraire les évènements, remplir la fenêtre sur le passé lointain et évaluer la requête alors que dans l'approche dynamique, ces traitements sont partagés entre le résumé et le SGFD. Le résumé ré-émet les évènements tandis que le SGFD se charge du traitement de la requête.

Pour l'approche dynamique, le système risque de perdre des évènements sur la fenêtre F_C étant donné qu'elle est mise à jour beaucoup plus rapidement que F_P si le système fait face à des flux ayant un débit très important. Par ailleurs, dans l'approche statique, le système risque de perdre des évènements sur la fenêtre F_P (dû à la phase d'extraction à partir de la base).

Les performances de l'approche dynamique ne sont pas sensibles au débit avec lequel les évènements arrivent. En revanche, les performances de l'approche statique se dégradent avec débit des évènements en entrée.

6.8.3 Expérimentation sur le module du résumé

Le module de résumé est commun à l'approche statique et à l'approche dynamique. Nous consacrons la suite de ces expérimentations à étudier le comportement de ce composant en utilisant différents algorithmes de résumé. Nous nous sommes intéressés à trois algorithmes d'échantillonnage : StreamSamp, échantillonnage réservoir et échantillonnage uniforme. Ces algorithmes présentent des mécanismes de pondération différents.

Dans l'approche statique, le module du résumé a un rôle passif, son rôle se limite à construire et à mettre à jour un résumé du flux de données. En revanche, le module de résumé dans l'approche dynamique est actif. Il est chargé d'émettre des évènements en fonction de leur pondération. Notre objectif consiste à étudier le comportement du module de résumé face à différentes stratégies de pondération.

Nous présentons dans un premier lieu les différents algorithmes utilisés lors des expérimentations. Nous nous intéressons en particulier à leur fonctionnement ainsi que leurs techniques de pondération.

- Échantillonnage uniforme : il s'agit d'un algorithme d'échantillonnage dans lequel les événements sont choisis avec une probabilité p et rejetés avec une probabilité $(1-p)$. Cet algorithme se caractérise par une taille de résumé qui ne fait que croître d'une manière linéaire au cours du temps. Chaque événement de l'échantillon a un poids de $1/p$. Le poids d'un événement ne change pas au cours du temps étant donné qu'aucune opération de vieillissement ou de remplacement n'est mise en place.
- Échantillonnage Réservoir : cet algorithme d'échantillonnage permet de concevoir un échantillon de taille fixe n d'une population N. Dès l'arrivée d'un nouvel évènement, la phase de mise à jour est lancée, elle permet de remplacer un événement du réservoir avec une probabilité n/i avec i l'indice de l'événement dans le flux. Le poids associé à un évènement e_i du réservoir est N_t/n, avec N_t le nombre d'événements passés à l'instant t. Le mécanisme de cet algorithme est détaillé dans la section 2.3.1.1
- StreamSamp : il s'agit d'un algorithme basé sur l'échantillonnage récursif des évènements. Le poids associé à un évènement croit de façon exponentielle, il est calculé en fonction de l'ordre auquel appartient l'évènement $2^{Ordre} \times \frac{1}{\alpha}$. Le mécanisme de cet algorithme est détaillé dans la section 2.3.2.2

Nous présentons dans ce qui suit des expérimentations sur le temps de construction du résumé ainsi que sur l'espace occupé par les différents algorithmes. Le temps de construction du résumé traduit le moment à partir duquel le résumé est disponible afin de pouvoir évaluer les requêtes. L'expérimentation sur l'espace occupé par le résumé est aussi importante étant donné que nous sommes dans le contexte des flux de données et que les ressources sont toujours restreintes.

6.8.3.1 Calcul de l'espace disque utilisé pour un taux maximum d'erreur

Nous étudions dans ce qui suit l'espace disque occupé par un résumé pour un taux d'erreur inférieur à un seuil fixé. La Figure 6.15, illustre les performances observées par les trois algorithmes d'échantillonnage.

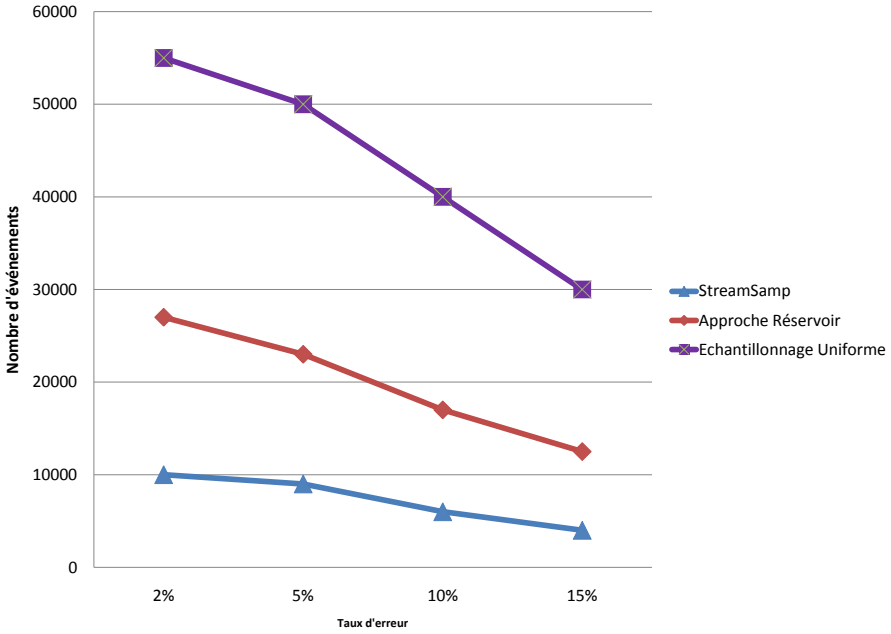

FIGURE 6.15 – Espace mémoire occupé par les algorithmes de résumé.

Nous avons évalué les trois algorithmes sur la requête définie précédemment [42]. Nous observons que pour atteindre un taux d'erreur de 2%, StreamSamp a besoin de moins d'espace disque que les autres algorithmes (e.g. 10000 événements pour un taux d'erreur égal à 2%). Ce taux d'erreur est un exemple qui a été fixé afin d'étudier la taille du réservoir nécessaire à chaque algorithme pour atteindre ce taux. Pour le même taux d'erreur, l'échantillonnage réservoir a besoin de 27000 événements (soit presque trois fois plus que StreamSamp) et, l'échantillonnage uniforme a besoin de 55000 événements (soit plus de cinq fois plus d'espace que StreamSamp). Ces performances sont expliquées par une bonne gestion des données de la part de l'algorithme StreamSamp. Les performances de chaque algorithme dépendent de la requête évaluée. Mais généralement, les tendances observés restent les mêmes.

Le choix de l'algorithme de résumé reste un compromis entre des objectifs antagonistes (les ressources disponibles Vs. la qualité des résultats).

42. calculer la durée moyenne des appels émis par les clients de l'opérateur téléphonique sur une fenêtre glissante de 1100k évènements. Nous supposons que la mémoire du SGFD ne peut contenir que 100k évènements.

6.8.3.2 Etude de la sensibilité des algorithmes à la variation du délai

Dans cette expérimentation, nous étudions la sensibilité des différents algorithmes de résumé à la variation du délai entre la fenêtre courante (F_C) et la fenêtre passée (F_P). Il s'agit de calculer le taux d'erreur de la réponse à la requête tout en faisant varier le délai qui est un paramètre fixé par l'utilisateur. Ces résultats n'incluent pas l'algorithme d'échantillonnage uniforme. En effet, cet algorithme n'utilisant aucune procédure de remplacement (seules les opérations d'insertion sont permises), le taux d'erreur reste fixe au cours du temps. Nous avons fixé l'espace disque disponible pour la construction du résumé à 40000 événements. Dans cette expérimentation, nous avons calculé l'erreur relative moyenne sur la requête 3 tout en utilisant trois valeurs différentes du délai ($D = 10000$, $D = 20000$, et $D = 30000$). La Figure 6.16 illustre les résultats de ces expérimentations .

Nous observons que l'algorithme réservoir présente des résultats presque similaires quelque soit la valeur du délai. Ceci est expliqué par le caractère uniforme de l'algorithme et sa pondération uniforme. Contrairement à l'algorithme réservoir, StreamSamp est sensible à ce paramètre. En effet, plus la valeur du délai est importante, plus le taux d'erreur augmente. Les performances de StreamSamp sont le résultat de l'échantillonnage récursif. Plus le délai est important, plus vieux sont les évènements à extraire (i.e. on puise dans un passé de plus en plus lointain). Ces événements sont caractérisés par un poids très important dans le résumé de StreamSamp. Quelque soit la valeur du délai, le poids relatif des évènements reste quasi-constant, c'est ce qui explique la non-sensibilité de cet algorithme de résumé au délai.

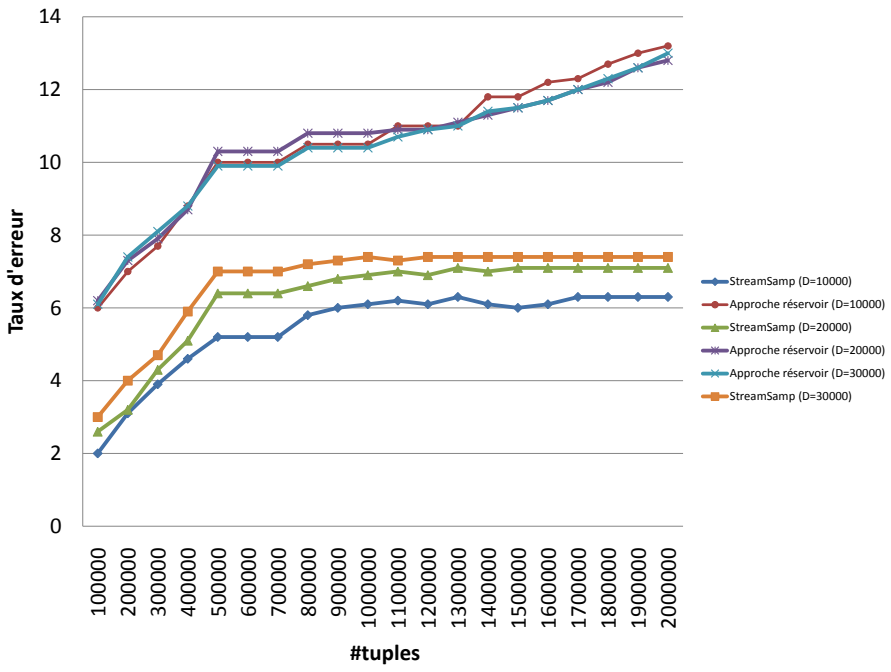

FIGURE 6.16 – Performance des algorithmes face à une variation du délai

TABLE 6.3 – Complexité des algorithmes de résumé

	Échantillonnage réservoir	Échantillonnage uniforme	StreamSamp
Taille de l'échantillon à t_c	++ O(1)	- - O(t)	+ O(Log(t))
Poids de l'échantillon à t_c pour un retard D	- - O(t)	++ O(1)	+ O(D)

6.8.4 Etude de complexité

Nous étudions dans cette section la complexité de l'approche statique et l'approche dynamique.

- La complexité de l'approche dynamique dépend fortement de l'algorithme de résumé utilisé.Comme illustré dans le tableau 6.3, l'échantillonnage uniforme présente un usage mémoire important, ce qui le rend peu utilisé dans le cadre des flux de données. Cependant, tous les événements de l'échantillon ont le même poids. Ce qui le rend avantageux (i.e. insensible au délai) par rapport aux deux autres algorithmes.
  L'algorithme réservoir est l'algorithme qui occupe le moins d'espace mémoire. Cependant, le poids d'un évènement de l'échantillon à l'instant t est de l'ordre de $O(t)$ ce qui le défavorise par rapport aux autres approches.
  L'algorithme StreamSamp présente une croissance en O(log(t)) pour la taille du résumé à l'instant t étant donné qu'il utilise une organisation basée sur les fenêtres inclinées pour la gestion des évènements. Par ailleurs, il présente une croissance en $O(D)$, indépendante de t, pour le poids des événements d'un retard D. En effet, en supposant que l'estampille temporelle d'un évènement correspond à son rang dans le flux, le temps de remplissage de l'ordre o correspond à $L \times T \times (2^{o+1} - 1)$. Etant donné que seuls les évènements du résumé évoluent et que la valeur du délai reste fixe, D est toujours dans le même ordre $o - 1 < D \leq o$.
- La complexité de l'approche statique dépend elle aussi de la structure du résumé utilisée (même étude que l'approche dynamique). Cependant, d'autres paramètres doivent être pris en considération tels que le coût d'évaluation de la requête continue sur la base de données.

6.9 Synthèse

Quelque soit le type de la requête (ponctuelle ou continue), lorsque celle ci fait appel à des évènements du passé lointain d'un flux, le système doit être en mesure de récupérer les évènements adéquats. On est ainsi face à un problème d'indisponibilité des données.

Nous avons illustré dans ce chapitre deux approches (statique et dynamique) dans le but de conserver une trace de l'histoire du flux et d'évaluer des requêtes sur celle-ci.

Dans l'approche statique le module de résumé a une tâche unique, celle de concevoir et gérer les résumés sur disque. L'évaluation des requêtes et l'accès aux évènements du passé lointain est une tâche qui doit être affectée au moteur du SGFD. Plusieurs extensions doivent être intégrées dans cette architecture. En effet, pour chaque opérateur, il faut redéfinir l'accès à la structure du résumé. Le système peut utiliser plusieurs résumés de structures différentes. Cette implémentation s'avère une tâche lourde.

En revanche, dans l'approche dynamique, le module du résumé est actif. Il est indépendant du SGFD et les traitements sont partagés. En plus de concevoir et gérer les résumés, le module du résumé est chargé de ré-émettre des flux retardés à partir des évènements conservés dans les résumés et à partir de la requête de l'utilisateur. Ces seconds flux sont considérés par les SGFD comme des flux ordinaires. Ils contribuent à l'évaluation de la requête au même titre que les flux d'origine. Pour pouvoir réaliser cela, une phase de traduction de la requête est nécessaire permettant de ré-écrire la requête en fonction des flux retardés. Cette seconde approche s'avère plus efficace que la première. Elle réutilise les outils standard des SGFD. D'une part, elle permet l'évaluation des requêtes sur un passé lointain. D'autre part, elle permet l'intégration de différentes structures de résumés généralistes. Nous avons évoqué dans ce chapitre les résumés par échantillonnage. Nous abordons le cas de l'utilisation de CluStream avec les deux approches dans la partie perspective de ce manuscrit.

Chapitre 7

Conclusion

Nous nous sommes intéressés dans ce manuscrit au contexte des flux de données. Une des caractéristiques de ce modèle de données est le caractère éphémère des évènements. Afin de conserver une trace sur ces évènements, les algorithmes de résumé de flux de données ont fait leur apparition. Nous avons proposé en première partie (cf. chapitre 5) une nouvelle technique de résumé généraliste permettant de tirer profit des avantages des techniques existantes. Il s'agit de l'approche hybride combinant à la fois deux approches existantes : un algorithme d'échantillonnage progressif (StreamSamp) et un algorithme de classification non supervisée (CluStream). Nous nous sommes intéressés par la suite à l'intégration de ces résumés dans un système de gestion de flux de données (SGFD).

Notre première contribution consiste en la mise en place d'un algorithme de résumé généraliste. La conception de cet algorithme est le fruit d'une étude approfondie de deux méthodes existantes (StreamSamp et CluStream). L'étude analytique de chacune de ces techniques nous a permis de déceler leurs avantages ainsi que leurs inconvénients. Nous avons ainsi proposé une nouvelle approche intitulée "approche hybride". Cet algorithme débute par le traitement des évènements du flux avec une opération d'échantillonnage récursif. Cette étape permet d'alléger le flux et de le résumer de façon très rapide. Les échantillons résultants sont par la suite traités par un algorithme de classification non supervisée. Un bon paramétrage de cet algorithme nous permet d'obtenir un résumé généraliste de bonne qualité sur toute la période temporelle. Le passage d'un algorithme à un autre se fait en fonction de la détérioration de qualité des échantillons dans le processus de ré-échantillonnage.

Nous avons par ailleurs mis en place une extension à cette approche. Il s'agit d'une réserve permettant une meilleure gestion de l'espace et qui permet de pallier les problèmes liés à l'envoi prématuré d'échantillons vers CluStream.

L'approche hybride présente de multiples avantages :

- Constitution d'un résumé généraliste de la totalité du flux de données ;
- Création d'un compromis entre la vitesse de création du résumé et sa qualité ;

- Possibilité de répondre à des requêtes qui portent sur des données du passé lointain ;
- Possibilité de répondre aux requêtes en fournissant des intervalles de confiance lors de l'estimation des requêtes.

Notre seconde contribution concerne l'exploitation et l'interrogation des résumés généralistes. Nous avons étudié dans les chapitres 4 et 5 l'application des tâches de fouille (classification supervisée et non supervisée) et de requêtes (requêtes d'agrégats : Moyenne, Variance, Médiane avec estimation de l'intervalle de confiance) sur les résumés conçus par StreamSamp, Clustream et l'approche hybride. L'application de ces analyses sur les résumés reste une tâche complexe quand elle est dédiée à l'utilisateur. Ce dernier doit en effet connaître au préalable la structure du résumé afin d'appliquer l'analyse.

Nous avons abordé en dernière partie de ce manuscrit (cf. chapitre 6) l'intégration des résumés généralistes à l'architecture des SGFD existante. L'objectif consiste à étendre les SGFD de façon à pouvoir traiter, sans se soucier de la structure des résumés utilisés, une large gamme de requêtes sur le passé lointain des flux. Nous avons pour cela présenté deux approches : l'approche statique basée sur un comportement passif du module de résumé, et l'approche dynamique fondée sur un comportement actif du module de résumé. L'approche statique est basée sur l'extraction des évènements du passé lointain à partir de la base de données, et le SGFD se charge de récupérer ces évènements pour répondre à la requête, ce qui crée une forte dépendance entre le module de résumé et le SGFD. L'approche dynamique se distingue de l'approche statique par sa capacité de réutilisation des fonctionnalités standard déjà définies dans les SGFD en régénérant les évènements du passé lointain sous forme de flux retardé.

Ces deux approches permettent : (1) d'intégrer un module de résumé dans l'architecture du SGFD, (2) d'évaluer des requêtes sur le passé. L'utilisation des modules de résumés permet de répondre aux requêtes sur le passé lointain au même titre que les requêtes continues définies sur les données du passé proche. Le système évalue la requête de façon transparente à l'utilisateur.

7.1 Perspectives

Nous présentons dans ce qui suit l'ensemble des perspectives liées aux deux contributions décrites dans ce manuscrit.

7.1.1 Perspectives liées à la construction de résumés généralistes

Une perspective possible liée à l'approche hybride peut être une meilleure gestion de l'opération de fusion des fenêtres. Dans le résumé StreamSamp, les échantillons

sont fusionnés selon le principe des fenêtres inclinées où le présent est privilégié par rapport au passé. Le passé est ainsi traité de façon déterministe. Afin d'améliorer cette opération, il est possible de fusionner les fenêtres "semblables". Cette opération est effectuée en utilisant une fonction de similarité entre les fenêtres. Le principe consiste à repérer les fenêtres les plus proches selon l'algorithme des plus proches voisins par exemple.

Nous avons fait l'hypothèse de travailler sur des flux de données mono-dimensionnelles. Ce point ne reflète pas toujours le monde réel. En effet, dans plusieurs applications de flux, les données arrivent à partir de plusieurs sources. Une perspective de notre travail est d'intégrer la gestion des flux de données multiples dans l'approche hybride. Une approche naïve serait de créer une instance de l'algorithme de résumé pour chaque flux de données. Une telle approche peut être coûteuse en terme de temps et d'espace. Une seconde perspective serait de résumer les flux de données en considérant l'ensemble des relations entre les variables de chaque flux et ainsi mutualiser la construction du résumé. Cette piste reste intéressante dans le cas où plusieurs évènements du flux présentent des similarités au niveau de plusieurs attributs.

Détection de rupture. Une perspective possible liée à l'optimisation des résumés consiste à conserver dans ces structures les instants de ruptures qu'on peut identifier dans les flux de données. Cette optimisation permet de garder dans le résumé les moments les plus importants dans la vie d'un flux. Les différentes techniques de résumé utilisent généralement des fonctions d'oublies qui permettent de conserver le plus possibles d'évènements. Cependant, il serait plus intéressant de ne conserver que les données qui correspondent aux instants de rupture (de changement). On favorise alors les données apparues aux instants de rupture plutôt qu'aux données classique apparus entre deux instants de ruptures. Il est ainsi nécessaire d'inclure dans l'algorithme de résumé un mécanisme de détection de rupture pour détecter les instants "critiques". Pour ce faire, il est ainsi possible de s'imprégner de plusieurs travaux existants [33][21] afin de déterminer le meilleur modèle de rupture pour le type de données utilisés.

7.1.2 Perspectives liées à l'intégration des résumés dans les SGFD

Gestion de CluStream et de l'approche hybride avec l'approche statique et dynamique. Les approches statiques et dynamiques ont été présentées dans ce manuscrit dans le cadre des résumés de flux de données basés sur une technique d'échantillonnage. Cependant, il existe d'autre formes de résumé (résumé par micro-classes évolutives, approche hybride, etc.). Une perspective naturelle serait d'étudier l'intégration de ces structures de résumé dans les approches statique et dynamique (cf. chapitre 6).

L'intégration de l'algorithme CluStream à ces approches revient à sélectionner les deux clichés regroupant la fenêtre de la requête. En se basant sur les statistiques conservées

au niveau des micro-classes ainsi que les informations temporelles sauvegardés dans les CFVs, nous régénérons les évènements du flux. Les évènements régénérés possèdent chacun une estampille temporelle définie de façon aléatoire suivant la distribution de l'information temporelles conservée. Nous déterminions par la suite les évènements qui sont inclus dans la fenêtre de la requête. ces évènements sont alors utilisés pour évaluer la requête posée sur le passé d'un flux.

Pour l'approche hybride, la fenêtre de la requête peut inclure à la fois des évènements mémorisés dans des échantillons et des évènements absorbés dans des micros-classes. A cet effet, les évènements utilisés dans la réponse à la requête sont l'union des évènements générés à partir de la distribution des micro-classes et ceux issus des échantillons. La génération des évènements reste une solution pour répondre aux requêtes sur des résumés construits à partir de micro-classes, cependant, cette solution reste coûteuse.

Gestion de la densité par le SGFD. Dans le cadre de l'exploitation des résumés de flux de données, l'approche dynamique proposée consiste à générer un flux de données à partir des résumés conservés. Etant donné que les SGFD actuels ne peuvent traiter que des évènements (données de détails), le flux retardé généré doit être sous la forme d'évènements. Cependant, plusieurs algorithmes de résumé, tels que CluStream et DenStream, constituent un résumé sous forme de densité. Une phase de transformation est ainsi nécessaire pour générer un flux d'évènements. Il s'agit de passer d'une densuré à un ensemble d'évènements. Une perspective de cette approche consiste à intégrer au sein des SGFD des primitives qui permettent de traiter ces densités. Ainsi, les outils standards de ces systèmes telles que les fenêtres seront capables d'interpréter les densités et de traiter les requêtes sur la base de cette représentation. Nous avons vu qu'il était simple d'intégrer un résumé généraliste dans un SGFD, cependant, la tâche d'intégrer les estimations de densité dans ces systèmes s'avère complexe.

Sélection sur un flux. Une perspective possible liée au requêtage des résumés de flux de données serait d'appliquer des requêtes de sélection fine sur les résumés conçus par StreamSamp, Clustream et l'approche hybride. En effet, comme expliquée dans la section 6.7, cette sélection peut porter sur un attribut ou sur une sélection temporelle. Les requêtes de sélection peuvent aussi être testées avec les approches statiques et dynamiques afin d'étudier les performances des deux approches face à des requêtes de sélection.

Annexe A

Jeux de données

A.1 KDD CUP 99

KDD CUP 99 découle du jeu de données DARPA IDS [128]. Ce jeu de données a été présenté dans le cadre d'une compétition à la conférence internationale *KDD 99* (*International Conference on Knowledge Discovery and Data Mining*). Le but de la compétition était de construire un détecteur d'intrusion réseau, un modèle prédictif permettant de distinguer entre les connexions dites "mauvaises", appelées intrusions ou attaques, et les connexions dites "normales". Ce jeu contient un ensemble de données à vérifier. Il inclut une grande variété d'intrusions simulées dans un environnement réseau militaire.

Le jeu de données contient près de 5 millions de n-uplets avec 42 attributs qualitatifs et quantitatifs. Dans l'enseomble des analyses, nous avons utilisé l'attribut (dst_byte) qui représente le nombre d'octets envoyés de la machine source. Chaque n-uplet représente une connexion TCP/IP au réseau. La totalité du jeu contient 39 types d'attaques regroupées en 4 catégories [177] :

- Deny of Service : dans ce type d'attaques, un pirate fait en sorte que plusieurs ressources de calcul et de mémoire soientt trop occupées pour gérer des requêtes standards privant ainsi l'accès aux utilisateurs légitimes à la machine.
- Probe attack : dans ces attaques, un pirate scanne un réseau pour recueillir des informations ou trouver des vulnérabilités connues.
- Remote-to-Local (R2L) : dans ce type d'ataques, un pirate envoie des paquets à une machine sur un réseau, puis exploite la vulnérabilité de la machine illégalement en accèdant localement en tant qu'utilisateur.
- User-to-Root (U2R) : dans ce type d'attaques, le pirate se connecte à la machine en tant qu'utilisateur normal. Cependant, il exploite la vulnérabilité de la machine en accédant avec les droits root sur le système.

Les tableaux A.1 et A.2 illustent un échantillon de ce jeu de données.

a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21
0	tcp	http	SF	217	2075	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	tcp	http	SF	200	42747	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	tcp	http	SF	206	1377	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	tcp	http	SF	208	1289	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	tcp	http	SF	205	1198	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	tcp	http	SF	205	54886	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	tcp	http	SF	207	1030	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

FIGURE A.1 – Echantillon du jeu de données KDD CUP 99 (attribut 1 à 21)

a22	a23	a24	a25	a26	a27	a28	a29	a30	a31	a32	a33	a34	a35	a36	a37	a38	a39	a40	a41	a42
0	4	4	0	0	0	0	1	0	0	48	255	1	0	0.02	0.05	0	0	0	0	normal
0	4	4	0	0	0	0	1	0	0	58	255	1	0	0.02	0.05	0	0	0	0	normal
0	8	8	0	0	0	0	1	0	0	8	255	1	0	0.12	0.05	0	0	0	0	normal
0	18	18	0	0	0	0	1	0	0	18	255	1	0	0.06	0.05	0	0	0	0	normal
0	28	28	0	0	0	0	1	0	0	28	255	1	0	0.04	0.05	0	0	0	0	normal
0	10	10	0	0	0	0	1	0	0	38	255	1	0	0.03	0.05	0	0	0	0	normal
0	20	20	0	0	0	0	1	0	0	48	255	1	0	0.02	0.05	0	0	0	0	normal

FIGURE A.2 – Echantillon du jeu de données KDD CUP 99 (attribut 22 à 42)

A.2 Cover Type

Ce jeu de données présente les prévision du couvert forestier à partir seulement de variables de type cartographique(pas de données de télédétection). Les variables ont été calculées à partir des données obtenues de US Geological Survey (USGS) et des données US Forest Service (USFS). Les données sont à l'état brut (non réduites), et les variables quantitatives (aires de nature sauvage et les types de sol) sont binaires.

La zone d'étude comprend quatre zones de nature sauvage située dans le Roosevelt National Forest du nord du Colorado. Ces domaines représentent les forêts avec un minimum de perturbations causées par l'Homme afin que les types de couverts forestier existant soient davantage le résultat de processus écologiques plutôt que des pratiques de gestion forestière.

Nous illustrons dans ce qui suit certaines informations concenrant ces quatre zones :

- Neota (zone 2) : cette zone a probablement l'élévation moyenne la plus élevée parmi les quatre zones.
- Rawa (zone 1) et Comanche Peak (zone 3) : ces deux zones auraient une élévation moyenne plus faible par rapport à la zone 2.
- Cache la Poudre (zone 4) : cette zone a probablement l'élévation moyenne la plus faible.

Les zones Rawa et Comanche Peak auraient tendance à être plus typique de l'ensemble de données globale en comparison avec la Neota ou Cache la Poudre. Ceci est dû à leur assortiment d'espèces d'arbres et la gamme des valeurs des variables prédictives (altitude, etc). Cache la Poudre serait probablement unique en raison de ses altitudes relativement faibles et de la composition de ses espèces.

Le jeux de données comporte 581 012 n-uplets avec 54 attributs. Ces attributs représentent 12 mesures. Nous présentons dans le tableau A.3, l'ensemble des informations relatives aux attributs de ce jeu de données.

Nom de l'attribut	Type	Mesure	Description
Altitude	quantitatif	mètre	Altitude en mètres
Aspect	quantitatif	azimuth	Aspect en degrés d'azimut
Pente	quantitatif	degré	La pente en degrés
Distance horizontale à l'hydrologie	quantitatif	mètre	Distance horizontale au plus proche point d'eau de surface
Distance verticale à l'hydrologie	quantitatif	mètre	Distance verticale au plus proche point d'eau de surface
Distance horizontale sur les routes	quantitatif	mètre	Distance horizontale à route la plus proche
Estompage à 9h	quantitatif	0 à 255	Index d'estompage à 9h, le solstice d'été
Estompage à midi	quantitatif	0 à 255	Index d'estompage à midi, soltice été
Estompage à 15h	quantitatif	0 à 255	Index d'estompage à 15 heures, le solstice d'été
Distance horizontale à feu Points	quantitatif	mètre	Distance horizontale la plus proche des points d'allumage feux de forêt
Zone de nature sauvage (4 colonnes binaires)	quantitatif	0/1	Désignation de la zone Wilderness
Type de sol (40 colonnes binaires)	quantitatif	0/1	Type de sol (désignation)
Type de couverture (7 types)	entier	1 à 7	Type de couvert forestier (désignation)

FIGURE A.3 – Liste des mesures du jeu de données Cover Type

Les tableaux A.4 et A.5 illustrent un échantillon de ce jeu de données.

a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25	a26	a27
2596	51	3	258	0	510	221	232	148	6279	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2590	56	2	212	-6	390	220	235	151	6225	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2804	139	9	268	65	3180	234	238	135	6121	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
2785	155	18	242	118	3090	238	238	122	6211	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2595	45	2	153	-1	391	220	234	150	6172	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2579	132	6	300	-15	67	230	237	140	6031	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2606	45	7	270	5	633	222	225	138	6256	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIGURE A.4 – Echantillon du jeu de données Cover Type (attribut 1 à 27)

a28	a29	a30	a31	a32	a33	a34	a35	a36	a37	a38	a39	a40	a41	a42	a43	a44	a45	a46	a47	a48	a49	a50	a51	a52	a53	a54	a55
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	5

FIGURE A.5 – Echantillon du jeu de données Cover Type (attribut 28 à 55). L'attribut 55 représente la classe à prédire.

A.3 France Télécom

Le jeu de données France Télécom présente des informations sur les communications téléphoniques des abonnés France Télécom durant le mois de Janvier 2005. Le jeu est composé de plusieurs fichiers :

- un fichier par semaine de communication (entre 38 523 962 et 42 451 220 n-uplets) ;
- un fichier très volumineux représentant un mois de communications (158 254 668 n-uplets) ;
- un ensemble de fichiers contenant des échantillons de 100 ou 1000 n-uplets.

Le jeu de données est aussi accompagné de la liste des clients (2 094 196 clients) ainsi que les détails sur les communications.

Le tableau A.6 illustre les 15 attributs du jeu de données.

Attribut	Description
Timestamp	Nombre indiquant le temps d'enregistrement de l'appel par le système
Usage_type	Indique s'il s'agit d'un appel entrant ou d'un appel sortant
geographic_zone	Zone géographique à partir delaquelle l'appel a été émis
call_type	Type d'appel (national ou international)
client_postal_code	Code postal du client
client_department	Département du client
client_status	Indique si le client est encore actif ou pas
client_name	Prénom du client
client_title	Status du client (célibataire, marié, etc.)
client_sex	Sexe du client
client_date_creation	Age du client
client_anciennete	Nombre d'année d'ancienneté du client
date	Date de la communication
call_duration	Durée de l'appel en seconde

FIGURE A.6 – Liste des attributs du jeu de données France Télécom

Le tableau A.7 illustre un échantillon de ce jeu de données.

timestamp	usage_type	geographic_zone	call_type	client_postal_code	client_department	client_status	client_name	client_title	client_sex	client_date_creation	client_age	client_anciennete	date	call_duration
820154237	7	GSM SFR	CTS			Actif	Laure	MME	F	16/08/2001	40	3	05/01/2005 12:01:56	17
820154238	7	PAYS-BAS	CTI	73210	73	Actif	Frédéric	M	M	28/04/1986	5	19	05/01/2005 09:01:11	226
820154238	5	SAVOIE MOUTIERS	CTN	73210	73	Actif	Frédéric	M	M	28/04/1986	5	19	05/01/2005 09:01:11	226
820154239	7	SAVOIE MOUTIERS	CTN	73210	73	Actif	Frédéric	M	M	28/04/1986	5	19	05/01/2005 12:01:47	323
820154239	5	SAVOIE MOUTIERS	CTN	73210	73	Actif	Frédéric	M	M	28/04/1986	5	19	05/01/2005 12:01:47	323
820154240	5	SAVOIE MOUTIERS	CTN	73210	73	Actif	Frédéric	M	M	28/04/1986	5	19	05/01/2005 12:01:16	256
820154240	7	SAVOIE MOUTIERS	CTN	73210	73	Actif	Frédéric	M	M	28/04/1986	5	19	05/01/2005 12:01:16	256
820154241	7	YONNE TOUCY	CTN	89130	89	Actif	Jacqueline	MME	F	27/02/1969	5	36	05/01/2005 12:01:12	177
820154241	5	YONNE TOUCY	CTN	89130	89	Actif	Jacqueline	MME	F	27/02/1969	5	36	05/01/2005 12:01:12	177
820154242	5	COTE D'OR DIJON	CTN	21160	21	Actif	Pascale	MME	F	03/12/1997	5	7	05/01/2005 13:01:04	92
820154242	7	TOP MESSAGE	CTS	21160	21	Actif	Pascale	MME	F	03/12/1997	5	7	05/01/2005 13:01:04	92
820154243	7	TOP MESSAGE	CTS	21160	21	Actif	Pascale	MME	F	03/12/1997	5	7	05/01/2005 14:01:02	16
820154243	5	COTE D'OR DIJON	CTN	21160	21	Actif	Pascale	MME	F	03/12/1997	5	7	05/01/2005 14:01:02	16
820154244	5	COTE D'OR DIJON	CTN	21160	21	Actif	Pascale	MME	F	03/12/1997	5	7	05/01/2005 14:01:30	420
820154244	7	COTE D'OR DIJON	CTN	21160	21	Actif	Pascale	MME	F	03/12/1997	5	7	05/01/2005 14:01:30	420

FIGURE A.7 – Echantillon du jeu de données France Télécom

Annexe B

Algorithmes de résumé

Nous représentons dans cette annexe les pseudo-codes de plusieurs algorithmes de résumé illustrés dans le chapitre 2.

B.1 Échantillonnage

B.1.1 Échantillonnage réservoir

```
1. Stocker les k premiers éléments dans le réservoir R.
2. Pour i de k+1 à n (n : taille du flux)
    * i ∈ R avec une probabilité p = k/i;
    * Si i est accepté alors,
        supprimer aléatoirement un élément de R et le
            remplacer par i.
      Fin Si
   Fin Pour
```

B.1.2 Échantillonnage réservoir biaisé

```
Soit R un réservoir vide d'une capacité k = 1/λ
Soit F(i) la fraction remplie de R juste avant l'arrivée du i^ème
    élément (F(i) ∈ [0,1])
1. L'élément (i+1) est inséré dans le réservoir
2. x = Random()
3. Si succès (x = F(i))
    - un élément de R est aléatoirement sélectionné et
        remplacé par le (i+1)^ème élément
   Sinon
        - le (i+1)^ème élément est ajouté à R
```

```
        - la taille de R augmente de 1}
    Fin Si
```

B.1.3 Échantillonnage périodique

```
1. Stocker les k premiers éléments dans le réservoir R.
2. Soit x un élément expiré et apprtenant au réserveoir R
3. Soit y l'élément le plus récent du flux
4. Remplacer x par y
```

B.1.4 Échantillonnage avec réserve

```
1. Soit R un échantillon de réserve.
2. Pour chaque élément s_i du flux
            p = Random()
            Si (p ≥ (2ck log n)/n)
                    ajouter s_i dans R
            Fin Si

    Fin Pour
```

B.1.5 Échantillonnage chaîné

```
1. Soit R un échantillon de réserve.
2. Pour chaque élément s_i du flux
            p = Random()
            Si (p ≥ (2ck log n)/n)
                    ajouter s_i dans R
            Fin Si

    Fin Pour
```

B.2 Sketchs

B.2.1 Flajolet Martin

```
Soit A un tableau de taille log(N) initialisé à 0
Soit k fonctions de hachage h[1..k]
```

Lorsqu'un nouvel élément s_i arrive
$A[h[s_i]] = 1$
Soit $R = minj|A[j] = 0$
Le nombre d'éléments distincts est : $D^{'} = 1.29.2^R$

B.2.2 Count Min

Soit A une matrice de taille wxd intialisée à 0
Soit d fonctions de hachage indépendantes 2 à 2.
Pour chaque Mise à jour (i, c)
Toutes les lignes de la matrice sont incrémentées de c
$count[j, h_j(i)] = count[j, h_j(i)] + c$
Fin Pour

B.2.3 Count

Pour chaque élément s_i du flux F
    Pour j de 1 à w
        $h_j[s_i] = h_j[s_i] + p_j[s_i]$
    Fin Pour
    Insérer s_i dans FI s'il n'existe pas
    $Freq = median_j\{h_j[s_i].p_j[s_i]\}$
    S_m : élément le moins fréquent de FI
    C_m : fréquence de S_m
    Si $Freq > C_m$
        {éliminer S_m}
    Fin Si
Fin Pour

Annexe C

Interrogation d'une requête : approche statique et approche dynamique

C.1 Fichier de configuration

Pour connecter Esper à une base de données, il faut créer un fichier de confiuration. Il s'agit d'un fichier XML qui contient toutes les informations relatives à la connections à la base de données (url, nom de la base, mot de passe, etc.)

```
<?xml version="1.0" encoding="UTF-8"?>
<esper-configuration xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance"
     xmlns="http://www.espertech.com/schema/esper"
     xsi:noNamespaceSchemaLocation="esper-configuration-2-0.xsd
        ">
     <database-reference name="dbstream">
         <drivermanager-connection class-name="org.postgresql.
             Driver" url="jdbc:postgresql://localhost:5432/
             dbstream?user=demo&amp;password=demo"
                                    user="demo"
                                    password="demo">
          <connection-arg name="user" value ="demo"/>
          <connection-arg name="password" value ="demo"/>
         </drivermanager-connection>
         <connection-lifecycle value="retain"/>
         <expiry-time-cache max-age-seconds="60" purge-interval
            -seconds="120" ref-type="weak"/>
```

```
    </database-reference>
</esper-configuration>
```

Dans le code source de l'application, on fait appel à cette configuration tout en instantiant les différents opérateurs qu'on a implémenté. Dans le code ci-dessous, on fait appel à l'opérateur StaticWin de type HistLengthBatchViewFactory définit dans la section C.2.

```
Configuration myConfig = new Configuration();
myConfig.configure("esper.cfg.xml");
myConfig.addPlugInView("ns", "StaticWin",
    HistLengthBatchViewFactory.class.getName());
```

Le système se charge par la suite de parser le fichier XML ainsi que les attributs de l'évènement :

```
        MyXmlParser myXmlParser = new MyXmlParser();
        //Analyse et enregistrement de la structure de l'
            événement contenu dans le fichier XML
        myXmlParser.parseAndRegister("edfEventSchema.xml",
            myConfig);
        //Récupération des paires (nomAttribut,typeAttribut)
        Map<String,Object> eventSchema = myXmlParser.
            getEventSchema();
        //Récupération de l'ordre des attributs dans un
            événement
        String [] propertyOrder = myXmlParser.getPropertyOrder
            ();
        //Récupération du nom de l'événement
        String eventName = myXmlParser.getEventName();
        //Récupération du nom de l'attribut contenant l'
            estampille de l'événement
        String timestampProperty = myXmlParser.
            getTimestampProperty();
        //Récupération des noms des attributs
        String properties = myXmlParser.getProperties();
```

La dernière étape consiste à créer une instance du serveur Esper et de spécifier le fichier qui sera la source du flux de données ainsi que le parseur qui lui correspond.

```
// Instanciation d'un objet de type serveur Esper.
    myServer = EPServiceProviderManager.getDefaultProvider(
        myConfig);
```

```
//Instanciation et Paramétrage de l'adaptateur CSV pour le
   flux eventName
    /* Instanciation d'un objet mySpec de type
       CSVInputAdapterSpec. Cet
       objet est utilisé pour indiquer à l'adaptateur les
          informations qui lui
       sont nécessaires pour effectuer la conversion des
          données du fichier
      CSV en des objets de type eventName.*/
    String fileName = "fluFT.csv";
    AdapterInputSource sourceCSV = new AdapterInputSource(
       fileName);
    //AdapterInputSource sourceCSV = new AdapterInputSource("
       oldenburg.csv");
    /* Association des données de la source externe à la
       classe définissant
    la structure des objets qui devront les contenir.*/
    CSVInputAdapterSpec mySpec = new CSVInputAdapterSpec(
       sourceCSV, eventName);
    //Optionnel, Vitesse de Lecture : ici 30 evts/s
    mySpec.setEventsPerSec(1000);
```

C.2 Approche statique

L'approche statique est caractérisé par un comprtement actif du module de résumé. Dans cette approche, nous avons développé un nouvel opérateur permettant d'extraire les évènements du passé lointain à partir de la base de données. Nous présentons dans ce qui suit le code source permettant la création et l'instanciation de ce nouvel opérateur.

Pour créer cet opérateur, nous avons utilisé la classe ViewFactorySupport qui permet d'étendre Esper en lui ajoutant de nouveaux opérateurs. Nous présentons dans ce qui suit l'implémentation de la méthode la plus importante de cette classe et qui permet d'extraire les évènements de la base de données.

```
    public final void updateHist()
    {
        // add data points to the current batch
        try {
            Statement stmt = myConnection.createStatement();
            ResultSet rs1 = stmt.executeQuery("SELECT * FROM
               pg_tables WHERE tablename like '"+tableName
```

```
                +"'");                  if (!rs1.next()){System.err
                .println(tableName+ " n'existe pas");}
                 ResultSet rs = myConnection.prepareStatement("
                    SELECT * FROM "+ tableName+
                     " ORDER BY timestamp DESC LIMIT "+sizeHist
                        ).executeQuery();

            EventType mapType = getEventType();
            String [] properties = mapType.getPropertyNames();
            while(rs.next())
            {
                Map<String,Object> event = new HashMap<String,
                    Object>();
                for(String property : properties)
                {
                    InputStream is = rs.getAsciiStream(
                        property);
                    event.put(property, convertStreamToString(
                        is));
                }
                EventBean evtBean = new MapEventBean(event,
                    mapType);
                histBatch.add(evtBean);
            }
        }
        catch (SQLException e)
            {
            e.printStackTrace();
            }

    }
```

Cette approche nécessite aussi la redéfinition des différentes fonctions d'agrégats. Pour cela, nous avons utilisé la classe AggregationSupport qui permet de créer de nouvelles fonctions d'agrégation. Pour pouvoir utiliser la nouvelle fonction, il faut la définir dans le fichier de configuration comme suit :

```
<esper-configuration
  <plugin-aggregation-function name="concat"
    function-class="com.espertech.esper.regression.client.
       MaFonctionMoyenne" />
```

```
</esper-configuration>
```

C.3 Approche dynamique

L'approche dynamique est basée sur la régénération des évènements du passé lointain. Ainsi une des fonctionnalité principale du module résumé dans cette approche est l'envoi des évènements nécessaires afin qu'ils puissent être traiter par le SGFD. Nous présentons dans ce qui suit, le code source de la méthode de régénration. La méthode Send() permet d'envoyer un évènement vers le moteur Esper. La méthode de régénration utilise plusieurs classes prédéfinies dans Esper. La documentation de Esper [1] offre plus de précisions sur ces méthodes.

```
    public final void replayTimeWindow(int delta, int delai )
     {
         // add data points to the current batch
         try {
             Statement stmt = myConnection.createStatement();
             ResultSet rs1 = stmt.executeQuery("SELECT * FROM
                pg_tables WHERE tablename like '"+tableName
                +"'");              if (!rs1.next()){System.err
                .println(tableName+ " n'existe pas");}
                 ResultSet rs = myConnection.prepareStatement("
                    SELECT * FROM "+ tableName+
                     "WHERE TIMESTAMP BETWEEN SYSDATE-delai AND
                         SYSDATE-delai+delta ORDER BY timestamp
                        ).executeQuery();

             EventType mapType = getEventType();
             String [] properties = mapType.getPropertyNames();
             while(rs.next())
             {
                 Map<String,Object> event = new HashMap<String,
                    Object>();
                 for(String property : properties)
                 {
                     InputStream is = rs.getAsciiStream(
                        property);
                     event.put(property, convertStreamToString(
                        is));
                 }
```

```
                    EventBean evtBean = new MapEventBean(event,
                        mapType);
                    send(evtBean);
                }
            }
            catch (SQLException e)
                {
                e.printStackTrace();
                }

        }
```

Publications

Articles dans des conférences internationales avec comité de sélection

[**1**] N. Gabsi, F. Clérot and G. Hébrail Efficient trade-off between speed processing and accuracy in summarizing data stream PAKDD '10 : Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining, June 2010, Hyderabad, India

Articles dans des conférences nationales avec comité de sélection

[**2**] N. Gabsi, F. Clérot and G. Hébrail Interrogation des résumés de flux de données EGC '10 : Conference Internationale Francophone sur l'Extraction et la Gestion des Conaissances, January 2010, Hammamet, Tunisia.

[**3**] N. Gabsi, F. Clérot and G. Hébrail Résumé hybride de flux de données par échantillonnage et classification automatique EGC '09 : Conference Internationale Francophone sur l'Extraction et la Gestion des Conaissances, January 2009, Strasbourg, France.

Chapitres de livres

[**4**] N. Gabsi, F. Clérot and G. Hébrail An Hybrid Data Stream Summarizing Approach by Sampling and Clustering in : Advances in Knowledge Discovery and Management, AKDM 2009.

Bibliographie

[1] Esper. http ://esper.codehaus.org/.

[2] http ://tradebot.fewin.com/.

[3] http ://www.aleri.com/cep/aleri-streaming-platform.

[4] http ://www.progress.com/apama/index.ssp.

[5] Krach de 1929. http ://fr.wikipedia.org/wiki/krach_de_1929.

[6] Option price reporting authority.

[7] Oracle bea. http ://fr.bea.com/.

[8] Streambase home page : http ://www.streambase.com.

[9] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora : a new model and architecture for data stream management. *The VLDB Journal*, 12(2) :120–139, 2003.

[10] T. Abdessalem, R. Chiky, G. Hébrail, and J. L. Vitti. Traitement de données de consommation électrique par un système de gestion de flux de données. In *EGC*, pages 521–532, 2007.

[11] C. Aggarwal, editor. *Data Streams – Models and Algorithms*. Springer, 2007.

[12] C. C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 575–586, New York, NY, USA, 2003. ACM.

[13] C. C. Aggarwal. On abnormality detection in spuriously populated data streams. In *SDM*, 2005.

[14] C. C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *VLDB '06 : Proceedings of the 32nd international conference on Very large data bases*, pages 607–618. VLDB Endowment, 2006.

[15] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '2003, pages 81–92. VLDB Endowment, 2003.

[16] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. On demand classification of data streams. In *KDD '04 : Proceedings of the tenth ACM SIGKDD international confe-*

rence on Knowledge discovery and data mining, pages 503–508, New York, NY, USA, 2004. ACM.

[17] P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison. Scalable bloom filters. *Inf. Process. Lett.*, 101(6) :255–261, 2007.

[18] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. Stream : the stanford stream data manager (demonstration description). In *SIGMOD '03 : Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 665–665, New York, NY, USA, 2003. ACM.

[19] A. Arasu, S. Babu, and J. Widom. The cql continuous query language : semantic foundations and query execution. *The VLDB Journal*, 15(2) :121–142, 2006.

[20] P. Ardilly. *Les techniques de sondage*. 2006.

[21] S. Arlot and A. Celisse. Segmentation of the mean of heteroscedastic data via cross-validation. *Statistics and Computing*, 2010.

[22] R. Avnur and J. M. Hellerstein. Eddies : continuously adaptive query processing. *SIGMOD Rec.*, 29(2) :261–272, 2000.

[23] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS '02 : Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16, New York, NY, USA, 2002. ACM.

[24] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA '02 : Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 633–634, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[25] B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *ICDE '04 : Proceedings of the 20th International Conference on Data Engineering*, page 350, Washington, DC, USA, 2004. IEEE Computer Society.

[26] H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. Zdonik. Retrospective on aurora. *The VLDB Journal*, 13(4) :370–383, 2004.

[27] M. Balazinska, Y. Kwon, N. Kuchta, and D. Lee. Moirae : History-enhanced monitoring. In *CIDR*, pages 375–386, 2007.

[28] D. Barbará. The characterization of continuous queries. *Int. J. Cooperative Inf. Syst.*, 8(4) :295–, 1999.

[29] V. Barnett and T. Lewis. Outliers in statistical data. *International Journal of Forecasting*, 12(1) :175–176, March 1996.

[30] H. Blockeel and J. Struyf. Efficient algorithms for decision tree cross-validation. In *ICML*, pages 11–18, 2001.

[31] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7) :422–426, 1970.

[32] F. Bodon. A survey on frequent itemset mining. Technical report, Budapest University of Technology and Economic, 2006.

[33] A. Bondu and M. Boullé. Détection de changements de distribution dans un flux de données : une approche supervisée. In *EGC*, pages 191–196, 2011.

[34] G. Brettlecker and H. Schuldt. The osiris-se (stream-enabled) infrastructure for reliable data stream management on mobile devices. In *SIGMOD '07 : Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1097–1099, New York, NY, USA, 2007. ACM.

[35] T. Calders, N. Dexters, and B. Goethals. Mining frequent itemsets in a stream. In *ICDM*, pages 83–92, 2007.

[36] C. Carnes, J. B. Park, and A. Vernon. Scalable trigger processing. In *ICDE '99 : Proceedings of the 15th International Conference on Data Engineering*, page 266, Washington, DC, USA, 1999. IEEE Computer Society.

[37] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Linear classification and selective sampling under low noise conditions. In *NIPS*, pages 249–256, 2008.

[38] A. Chakraborty and A. Singh. Processing exact results for sliding window joins over time-sequence, streaming data using a disk archive. In *ACIIDS*, pages 196–201, 2009.

[39] J. Q. Chakravarthy, Sharma. *Stream Data Processing : A Quality of Service Perspective*, chapter Overview of Data Stream Processing, pages 10–11. Springer, 2009.

[40] C.-Y. Chan and Y. E. Ioannidis. An efficient bitmap encoding scheme for selection queries. In *SIGMOD*, 1999.

[41] S. Chandrasekaran. *Query processing over live and archived data streams*. PhD thesis, Berkeley, CA, USA, 2005. Adviser-Franklin, Michael J.

[42] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq : continuous dataflow processing. In *SIGMOD '03 : Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668, New York, NY, USA, 2003. ACM.

[43] S. Chandrasekaran and M. Franklin. Remembrance of streams past : overload-sensitive management of archived streams. In *VLDB '04 : Proceedings of the Thirtieth international conference on Very large data bases*, pages 348–359. VLDB Endowment, 2004.

[44] S. Chandrasekaran and M. J. Franklin. Streaming queries over streaming data. In *VLDB '02 : Proceedings of the 28th international conference on Very Large Data Bases*, pages 203–214. VLDB Endowment, 2002.

[45] S. Chandrasekaran and M. J. Franklin. Psoup : a system for streaming queries over streaming data. *The VLDB Journal*, 12(2) :140–156, 2003.

[46] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction : how much is enough ? In *SIGMOD '98 : Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 436–447, New York, NY, USA, 1998. ACM.

[47] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. *SIGMOD Rec.*, 28(2) :263–274, 1999.

[48] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagaracq : a scalable continuous query system for internet databases. In *SIGMOD '00 : Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 379–390, New York, NY, USA, 2000. ACM.

[49] K. Chen and L. Liu. Detecting the change of clustering structure in categorical data streams. In *SDM*, 2006.

[50] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 323–334. VLDB Endowment, 2002.

[51] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. 2003.

[52] R. Chiky and G. Hébrail. Summarizing distributed data streams for storage in data warehouses. In *DaWaK*, pages 65–74, 2008.

[53] K. Coleman. Communication intelligence and geographic information in homeland security. Directions Magazine, 2004.

[54] G. Cormode and M. N. Garofalakis. Approximate continuous querying over distributed streams. *ACM Trans. Database Syst.*, 33(2), 2008.

[55] G. Cormode and S. Muthukrishnan. An improved data stream summary : The count-min sketch and its applications. *Journal of Algorithms*, 55 :29–38, 2004.

[56] G. Cormode and S. Muthukrishnan. What's hot and what's not : tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1) :249–278, 2005.

[57] G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different : Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In *ICDE*, page 57, 2006.

[58] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith. Hancock : A language for analyzing transactional data streams. *ACM Trans. Program. Lang. Syst.*, 26(2) :301–338, 2004.

[59] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope : a stream database for network applications. In *SIGMOD '03 : Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 647–651, New York, NY, USA, 2003. ACM.

[60] B. Csernel. *Résumé généraliste de flux de données.* PhD thesis, Ecole Nationale Supérieur des Télécommunications, Février 2008.

[61] B. Csernel, F. Clerot, and G. Hébrail. Streamsamp : Datastream clustering over tilted windows through sampling. In *ECML PKDD 2006 Workshop on Knowledge Discovery from Data Streams*, 2006.

[62] H. Cui. Online outlier detection over data streams. Master thesis, 2005.

[63] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD '03 : Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 40–51, New York, NY, USA, 2003. ACM.

[64] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*, 2006.

[65] C. d'auteurs issus du projet MIDAS. Résumé généraliste de flux de données. In *EGC*, pages 255–261, 2010.

[66] U. Dayal. *Active Database Systems : Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[67] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *ESA '02 : Proceedings of the 10th Annual European Symposium on Algorithms*, pages 348–360, London, UK, 2002. Springer-Verlag.

[68] A. Demers, J. Gehrke, and B. P. Cayuga : A general purpose event monitoring system. In *In CIDR*, pages 412–422, 2007.

[69] A. Deutsch, D. Florescu, M. Fernandez, A. Levy, and D. Suciu. Xml-ql : A query language for xml.

[70] N. Dindar, B. Güç, P. Lau, A. Ozal, M. Soner, and N. Tatbul. Dejavu : declarative pattern matching over live and archived streams of events. In *SIGMOD*, 2009.

[71] P. Domingos and G. Hulten. Mining high-speed data streams. In *KDD '00 : Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, New York, NY, USA, 2000. ACM.

[72] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In G. Di Battista and U. Zwick, editors, *Annual European Symposium on Algorithms (ESA03)*, volume 2832 of *Lecture Notes in Computer Science*, pages 605–617, September 2003.

[73] P. S. Efraimidis and P. G. Spirakis. Weighted random sampling with a reservoir. *Inf. Process. Lett.*, 97 :181–185, March 2006.

[74] C. Estan and G. Varghese. New directions in traffic measurement and accounting : Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3) :270–313, 2003.

[75] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache : a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3) :281–293, 2000.

[76] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *VLDB '98 : Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 299–310, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[77] S. Ferrandiz. Note sur la gestion de flux de données. Novembre 2007.

[78] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2) :182–209, 1985.

[79] G. P. Féraud Raphaël, Clérot Fabrice. Sampling the join of streams. In *IFCS'2009 Int. Conf. on Int. Federation of Classification Societies, Dresden*, 2009.

[80] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams : you only get one look a tutorial. In *SIGMOD '02 : Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 635–635, New York, NY, USA, 2002. ACM.

[81] M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *SIGMOD '02 : Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 476–487, New York, NY, USA, 2002. ACM.

[82] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. *SIGMOD Rec.*, 30(2) :13–24, 2001.

[83] R. Gemulla and W. Lehner. Sampling time-based sliding windows in bounded space. In *SIGMOD '08 : Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 379–392, New York, NY, USA, 2008. ACM.

[84] R. Gemulla, W. Lehner, and P. J. Haas. A dip in the reservoir : Maintaining sample synopses of evolving datasets. In *VLDB*, pages 595–606, 2006.

[85] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining frequent patterns in data streams at multiple time granularities, 2002.

[86] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC '02 : Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 389–398, New York, NY, USA, 2002. ACM.

[87] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC '02 : Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 389–398, New York, NY, USA, 2002. ACM.

[88] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams : One-pass summaries for approximate aggregate queries. In *VLDB '01 : Proceedings of the 27th International Conference on Very Large Data Bases*, pages 79–88, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[89] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Optimal and approximate computation of summary statistics for range aggregates. In *PODS '01 : Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 227–236, New York, NY, USA, 2001. ACM.

[90] L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2) :5–14, 2003.

[91] T. J. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu. Processing xml streams with deterministic automata and stream indexes. volume 29, pages 752–788, New York, NY, USA, 2004. ACM.

[92] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD '01 : Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 58–66, New York, NY, USA, 2001. ACM.

[93] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *STOC '01 : Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 471–475, New York, NY, USA, 2001. ACM.

[94] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. Database Syst.*, 31(1) :396–438, 2006.

[95] S. Guha, N. Koudas, and D. Srivastava. Fast algorithms for hierarchical range histogram construction. In *PODS '02 : Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 180–187, New York, NY, USA, 2002. ACM.

[96] S. Guha and A. McGregor. Stream order and order statistics : Quantile estimation in random-order streams. *SIAM J. Comput.*, 38(5) :2044–2059, 2008.

[97] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams : Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15 :515–528, 2003.

[98] S. Guha, N. Mishra, and R. Motwani. Clustering data streams. volume IEEE FOCS Conference, 2000.

[99] S. Guha, K. Shim, and J. Woo. Rehist : relative error histogram construction algorithms. In *VLDB '04 : Proceedings of the Thirtieth international conference on Very large data bases*, pages 300–311. VLDB Endowment, 2004.

[100] L. GÜRGEN. *Gestion à grande échelle de données de capteurs hétérogènes*. PhD thesis, Institut National Polytechnique de Grenoble, 2007.

[101] D. Gyllstrom, E. W. 0002, H.-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson. Sase : Complex event processing over streams (demo). In *CIDR*, pages 407–411, 2007.

[102] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. *SIGMOD Rec.*, 28(2) :287–298, 1999.

[103] M. Halkidi, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Resilient and energy efficient tracking in sensor networks. *Int. J. Wire. Mob. Comput.*, 1(2) :87–100, 2006.

[104] A. W. F. M. M. M. Hammad, M. and A. K. Elmagarmid. Efficient execution of sliding window queries over data streams. Technical Report CSD TR 03-035, Purdue University Department of Computer Sciences, December 2003.

[105] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Y. Eltabakh, M. G. Elfeky, T. M. Ghanem, R. Gwadera, I. F. Ilyas, M. S. Marzouk, and X. Xiong. Nile : A query processing engine for data streams. In *ICDE*, page 851, 2004.

[106] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29 :1–12, May 2000.

[107] A. C. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. 1991.

[108] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM.

[109] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB '98 : Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 275–286, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[110] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering : A review. *ACM Comput. Surv.*, 31(3) :264–323, 1999.

[111] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. In *PODS '07 : Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 243–252, New York, NY, USA, 2007. ACM.

[112] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. *ACM Trans. Database Syst.*, 33(4), 2008.

[113] T. S. Jayram and S. Muthukrishan. Estimating statistical aggregates on probabilistic data streams. In *In ACM Symposium on Principles of Database Systems*, pages 243–252, 2007.

[114] Y. C. G. D. J. P. B. W. W. Jiawei Han, Y. Dora Cai and J. Wang. *Multi-Dimensional Analysis of Data Streams Using Stream Cubes*, volume Data Streams Models and Algorithms of *Advances in Database Systems*, pages 103–125. Springer, 2007.

[115] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *CIKM '03 : Proceedings of the twelfth international conference on Information and knowledge management*, pages 287–294, New York, NY, USA, 2003. ACM.

[116] H. Jin, M.-L. Wong, and K.-S. Leung. Scalable model-based clustering by working on data summaries. *Data Mining, IEEE International Conference on*, 0 :91, 2003.

[117] R. A. Johnson and D. W. Wichern, editors. *Applied multivariate statistical analysis*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[118] K. Jouini. Spécifications et choix du système de gestion de flux de données. Technical report, Télécom ParisTech, 2009.

[119] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1) :51–55, 2003.

[120] H. Kawashima, R. Sato, and H. Kitagawa. Models and issues on probabilistic data streams with bayesian networks. In *SAINT*, pages 157–160, 2008.

[121] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *VLDB '04 : Proceedings of the Thirtieth international conference on Very large data bases*, pages 180–191. VLDB Endowment, 2004.

[122] N. Koudas, S. Muthukrishnan, and D. Srivastava. Optimal histograms for hierarchical range queries (extended abstract). In *PODS '00 : Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 196–204, New York, NY, USA, 2000. ACM.

[123] B. Kovalerchuk and E. Vityaev. *Data mining in finance : advances in relational and hybrid methods*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[124] S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Madden, F. Reiss, and M. A. Shah. Telegraphcq : An architectural status report. *IEEE Data Eng. Bull.*, 26(1) :11–18, 2003.

[125] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statistics*, 22 :79–86, 1951.

[126] K. leong Ong, W. Li, W. keong Ng, and E. peng Lim. Sclope : An algorithm for clustering data streams of categorical attributes. Technical report, 2004.

[127] X. Li and G. Agrawal. Efficient evaluation of xquery over streaming data. In *VLDB '05 : Proceedings of the 31st international conference on Very large data bases*, pages 265–276. VLDB Endowment, 2005.

[128] R. P. Lippmann and et al. Evaluating intrusion detection systems - the 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition (DISCEX) 2000*, volume 2, pages 12–26. IEEE Computer Society Press, Los Alamitos, CA, 2000.

[129] H. Liu, S. Shah, and W. Jiang. On-line outlier detection and data cleaning. *Computers & Chemical Engineering*, 28(9) :1635–1647, 2004.

[130] J. Liu, J. Liu, J. Reich, P. Cheung, and F. Zhao. Distributed group management for track initiation and maintenance in target localization applications. In *In Proc. of 2nd workshop on Information Processing in Sensor Networks (IPSN*, pages 113–128, 2003.

[131] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Trans. on Knowl. and Data Eng.*, 11(4) :610–628, 1999.

[132] L. Liu, C. Pu, and W. Tang. Webcq-detecting and delivering information changes on the web. In *CIKM '00 : Proceedings of the ninth international conference on Information and knowledge management*, pages 512–519, New York, NY, USA, 2000. ACM.

[133] S. Mahadevan and P. Tadepalli. Quantifying prior determination knowledge using the pac learning model. *Mach. Learn.*, 17 :69–105, October 1994.

[134] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB '02 : Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

[135] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB '02 : Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

[136] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *SIGMOD '98 : Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 426–435, New York, NY, USA, 1998. ACM.

[137] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *SIGMOD '99 : Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 251–262, New York, NY, USA, 1999. ACM.

[138] A. S. Maskey and M. Cherniack. Replay-based approaches to revision processing in stream query engines. In *SSPS*, 2008.

[139] A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, pages 398–412, 2005.

[140] N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings of IEEE International Conference on Data Engineering*, page 685, 2002.

[141] S. Muthukrishnan. Data streams : algorithms and applications. In *SODA '03 : Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 413–413, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[142] S. Muthukrishnan. Data streams : algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2) :117–236, 2005.

[143] S. Muthukrishnan, E. v. d. Berg, and Y. Wu. Sequential change detection on data streams. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, ICDMW '07, pages 551–550, Washington, DC, USA, 2007. IEEE Computer Society.

[144] K. V. Nesbitt and S. Barrass. Finding trading patterns in stock market data. *IEEE Comput. Graph. Appl.*, 24(5) :45–55, 2004.

[145] P. O'Neil and D. Quass. Improved query performance with variant indexes. In *SIGMOD '97 : Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 38–49, New York, NY, USA, 1997. ACM.

[146] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, unsupervised stream mining. *The VLDB Journal*, 13 :222–239, September 2004.

[147] F. Peng and S. S. Chawathe. Xpath queries on streaming data. In *SIGMOD '03 : Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 431–442, New York, NY, USA, 2003. ACM.

[148] N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, and N.-E. Frantzell. Supporting streaming updates in an active data warehouse. In *ICDE*, pages 476–485, 2007.

[149] C. Raïssi, T. Calders, and P. Poncelet. Mining conjunctive sequential patterns. In *ECML/PKDD (1)*, page 19, 2008.

[150] C. Raïssi and P. Poncelet. Échantillonnage pour lextraction de motifs séquentiels : des basesde données statiques aux flots de données. In *EGC*, pages 145–156, 2008.

[151] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, and J. M. Hellerstein. Enabling real-time querying of live and historical stream data. In *SSDBM*, 2007.

[152] D. M. Rocke and D. L. Woodruff. Identification of Outliers in Multivariate Data. *Journal of the American Statistical Association*, 91(435) :1047–1061, 1996.

[153] P. P. Rodrigues, a. Gama, Jo and J. Pedroso. Hierarchical clustering of time-series data streams. *IEEE Trans. on Knowl. and Data Eng.*, 20(5) :615–627, 2008.

[154] A. Roginska, E. Childs, and M. K. Johnson. Monitoring real-time data : A sonification approach. In *Proceedings of the 12th International Conference on Auditory Display*, pages 176–181, 2006.

[155] X. Rong and J. Wang. A robust approach to find effective items in distributed data streams. In *Proceedings of the Life system modeling and simulation 2007 international conference on Bio-Inspired computational intelligence and applications*, LSMS'07, pages 688–696, Berlin, Heidelberg, 2007. Springer-Verlag.

[156] D. Rotem, K. Stockinger, and K. Wu. Optimizing candidate check costs for bitmap indices. In *CIKM*, 2005.

[157] I. Sachpazidis. @home : A modular telemedicine system. In *Mobile Computing in Medicine*, pages 87–95, 2002.

[158] B. Scherrer. *Biostatistique*. 2007.

[159] U. Schreier, H. Pirahesh, R. Agrawal, and C. Mohan. Alert : An architecture for transforming a passive dbms into an active dbms. In *VLDB '91 : Proceedings of the 17th International Conference on Very Large Data Bases*, pages 469–478, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[160] L. Segoufin and V. Vianu. Validating streaming xml documents. In *PODS '02 : Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 53–64, New York, NY, USA, 2002. ACM.

[161] M. G. V. Stoa, S. Lindeberg. Online analysis of myocardial ischemia from medical sensor data streams with esper. *Applied Sciences on Biomedical and Communication Technologies, 2008. ISABEL '08. First International Symposium on*, 2008.

[162] M. Stonebraker, U. Çetintemel, and S. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Rec.*, 34(4) :42–47, 2005.

[163] J.-E. Symphor, A. Mancheron, L. Vinceslas, and P. Poncelet. Le fia : un nouvel automate permettant lextraction efficace ditemsets fréquents dans les flots de données. In *EGC*, pages 157–168, 2008.

[164] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB '2003 : Proceedings of the 29th international conference on Very large data bases*, pages 309–320. VLDB Endowment, 2003.

[165] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *SIGMOD '92 : Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 321–330, New York, NY, USA, 1992. ACM.

[166] K. Towne, Q. Zhu, C. Zuzarte, and W.-C. Hou. Window query processing for joining data streams with relations. In *CASCON*, 2007.

[167] L. Vinceslas, J.-E. Symphor, A. Mancheron, and P. Poncelet. Spams : Une nouvelle approche incrémentale pour l'extraction de motifs séquentiels fréquents dans les data streams. In *EGC*, pages 205–216, 2009.

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

[169] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang. Density-based clustering of data streams at multiple resolutions. *ACM Trans. Knowl. Discov. Data*, 3 :14 :1–14 :28, July 2009.

[170] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *KDD '03 : Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235, New York, NY, USA, 2003. ACM.

[171] G. Weiss. Data mining in telecommunications. In *Data Mining and Knowledge Discovery Handbook : A Complete Guide for Practitioners and Researchers, Kluwer Academic, 2005*, pages 1189–1201. kluwer, 2004.

[172] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1) :69–101, 1996.

[173] G. J. Williams, R. A. Baxter, H. He, S. Hawkins, and L. Gu. A comparative study of rnn for outlier detection in data mining. In *ICDM*, pages 709–712, 2002.

[174] K. Wu, E. Otoo, and A. Shoshani. On the performance of bitmap indices for high cardinality attributes. In *VLDB '04 : Proceedings of the Thirtieth international conference on Very large data bases*, pages 24–35. VLDB Endowment, 2004.

[175] K. Wu, E. J. Otoo, and A. Shoshani. An efficient compression scheme for bitmap indices. Technical report, ACM Transactions on Database Systems, 2004.

[176] K. Wu, A. Shoshani, and K. Stockinger. Analyses of multi-level and multi-component compressed bitmap indexes. *ACM Trans. Database Syst.*, 2010.

[177] C. Xiang, P. C. Yong, and L. S. Meng. Design of multiple-level hybrid classifier for intrusion detection system using bayesian clustering and decision trees. *Pattern Recognition Letters*, 29(7) :918 – 924, 2008.

[178] L. Xiaoyan. *Data Stream Mining in Financial Securities Databases.* PhD thesis, University of Hong Kong, 2007.

[179] B. Xu, S. Tirthapura, and C. Busch. Sketching asynchronous data streams over sliding windows. *Distributed Computing*, 20(5) :359–374, 2008.

[180] C. Yang and J. Zhou. Hclustream : A novel approach for clustering evolving heterogeneous data stream. In *ICDMW '06 : Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, pages 682–688, Washington, DC, USA, 2006. IEEE Computer Society.

[181] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31 :2002, 2002.

[182] T. Zhang, R. Ramakrishnan, and M. Livny. Birch : an efficient data clustering method for very large databases. *SIGMOD Rec.*, 25(2) :103–114, 1996.

[183] S. Zhou, A. Zhou, J. Cao, W. Jin, Y. Fan, and Y. Hu. Combining sampling technique with dbscan algorithm for clustering large spatial databases. In *PAKDD*, pages 169–172, 2000.

[184] Y. Zhu and D. Shasha. Statstream : Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.