Traitement de données de consommation électrique par un Système de Gestion de Flux de Données

Talel Abdessalem*, Raja Chiky*, Georges Hébrail*,**, Jean Louis Vitti*

* GET-ENST Paris
Laboratoire LTCI - UMR 5141 CNRS - Département Informatique et Réseaux
46 rue Barrault, 75634 Paris Cedex 13
Email: prenom.nom@enst.fr
** EDF R&D - Département ICAME
1, Avenue du Général de Gaulle, 92140 Clamart
Email: georges.hebrail@edf.fr

Résumé. Avec le développement de compteurs communicants, les consommations d'énergie électrique pourront à terme être télérelevées par les fournisseurs d'électricité à des pas de temps pouvant aller jusqu'à la seconde. Ceci générera des informations en continu, à un rythme rapide et en quantité importante. Les Systèmes de Gestion de Flux de Données (SGFD), aujourd'hui disponibles sous forme de prototypes, ont vocation à faciliter la gestion de tels flux. Cette communication décrit une étude expérimentale pour analyser les avantages et limites de l'utilisation de deux prototypes de SGFD (STREAM et TelegraphCQ) pour la gestion de données de consommation électrique.

1 Introduction

Il est envisagé de déployer en France dans les prochaines années des compteurs *communicants* chez un grand nombre de clients des compagnies d'électricité, à l'image de ce qui a été déjà réalisé en Italie par la société ENEL. Ces nouveaux compteurs, reliés aux Systèmes d'Information (SI) des fournisseurs d'électricité, permettront de relever à distance les consommations d'électricité afin d'effectuer des opérations tels que la facturation, l'agrégation, le contrôle,... L'approche standard pour le traitement de ce type de données dans les SI est d'utiliser des Systèmes de Gestion de Bases de Données (SGBD) qui assurent leur stockage et permettent de les consulter par des requêtes. Cette approche trouve ses limites lorsque les flux de données sont importants (en débit et/ou en nombre). C'est pour cela que de nouvelles approches ont été développées dans le domaine des bases de données pour traiter de façon performante des flux de données, sans mémoriser l'ensemble des informations : il s'agit des Systèmes de Gestion de Flux de Données - SGFD - ou encore Data Stream Management Systems - DSMS - en anglais.
Plusieurs prototypes de SGFD ont été développés ces dernières années pour répondre à ces besoins. Le rôle de ces systèmes est de traiter en temps réel un ou plusieurs flux de données par des requêtes dites *continues*.
Le travail présenté ici décrit l'expérimentation de deux prototypes de SGFD du domaine public

(STREAM (Arasu et al., 2004) et TelegraphCQ (Chandrasekaran et al., 2003)) pour la réalisation de fonctions de traitement de données de consommation électrique, disponibles sous forme de flux à partir de compteurs communicants.
La communication est organisée comme suit : la section 2 présente en général les SGFD et en particulier les systèmes STREAM et TelegraphCQ. La section 3 présente notre expérimentation, la synthèse de celle-ci est exposée à la section 4. Enfin, nous concluons à la section 5 et donnons nos perspectives de recherche.

2 Systèmes de Gestion de Flux de Données

Les Systèmes de Gestion de Bases de Données (SGBD) sont des logiciels permettant de définir des structures de données, de stocker des données dans ces structures, de modifier ces données, et de les consulter à l'aide de requêtes. Les SGBD les plus utilisés sont ceux dits relationnels, où les données sont structurées sous forme de tables (aussi appelées relations). Le langage standard pour spécifier les requêtes aux bases de données relationnelles est le langage SQL. Les Systèmes de Gestion de Flux de Données (SGFD) préfigurent des prochaines générations de SGBD. En effet, devant la volumétrie sans cesse plus importante des données produites par les systèmes de mesure et les systèmes informatiques, certaines applications nécessitent de lever l'hypothèse que toutes les données produites sont stockées dans la base de données avant d'être consultées par des requêtes. Pour les SGFD, la structure de données est celle d'une collection de flux de données entrants, qui sont traités au fil de l'eau pour répondre aux requêtes des utilisateurs, elles-mêmes définies pour produire des flux sortants. A titre d'exemple, si les flux entrants de données correspondent à des mesures de consommation d'électricité à différents points du réseau électrique, une requête utilisateur peut consister en la restitution des consommations moyennes et maximales pour chaque département dans les trois dernières heures.
Les travaux de recherche menés aux Etats-Unis sur le thème des flux de données se divisent principalement en deux grandes directions (voir (Babcock et al., 2002) et (Golab et al., 2003)) : (1) la conception de systèmes de gestion de flux de données, (2) la conception d'algorithmes de fouille de données (data mining) pouvant fonctionner sur des flux de données, c'est-à-dire sans mémoriser l'ensemble des informations apparaissant dans les flux. Nous nous intéressons ici uniquement à la gestion des flux de données.
Les SGFD permettent de formuler des requêtes dites «continues» qui s'évaluent au fur et à mesure que les différents flux sur lesquels elles portent sont alimentés. Afin que ces requêtes aient un sens et ne nécessitent pas le stockage de l'ensemble du flux, des techniques de fenêtrage sont utilisées : une fenêtre définit un intervalle temporel exprimé soit en terme de durée (par exemple les 5 dernières minutes), ou soit sous forme logique exprimé en nombre de tuples (par exemple les 20 derniers éléments). Ces fenêtres peuvent être délimitées par des bornes fixes ou glissantes dans le temps.
La figure 1 présente une architecture abstraite de SGFD, extraite de (Golab et al., 2003). Un moniteur d'entrée sert à réguler le débit d'arrivée des données et peut être amené à supprimer quelques tuples en cas de surcharge du système. Les données des flux sont stockées temporairement dans des buffers (working storage) pour permettre le traitement des requêtes fenêtrées. Des résumés des flux sont éventuellement constitués (summary storage) lorsque la capacité de stockage ne permet pas de gérer tous les flux entrants : le résultat des requêtes est alors ap-

proché. Un stockage statique (static storage) assure la mémorisation des meta-données (structure et provenance des flux) ainsi que des données présentes de façon permanente dans la base. Lorsque les requêtes des utilisateurs sont définies, celles-ci sont stockées dans le système (query repository) et traitées par le processeur de requêtes (query processor). Ce dernier a la charge de définir un plan optimisé d'exécution de l'ensemble de requêtes (en mutualisant les ressources entre plusieurs requêtes) et de produire les résultats des requêtes soit directement sous forme de flux, soit sous forme de données stockées temporairement (output buffer) et remises à jour périodiquement. Il faut noter que les plans d'exécution des requêtes ne doivent pas être figés car ils peuvent être remis en question soit par l'ajout de nouvelles requêtes, soit par des variations dans le contenu et le débit des flux.

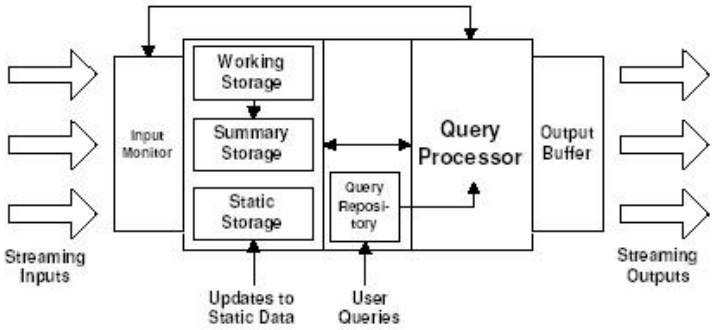

FIG. 1 – *Architecture globale d'un SGFD*

Si les applications qui ont motivé le développement des SGFD portaient sur la supervision des réseaux informatiques, de nombreuses autres applications se sont développées : le traitement des logs de sites web, des tickets de communications fixes ou mobiles, des données de capteurs (trafic routier, météorologie par exemple), mais aussi de données boursières. Plusieurs systèmes prototypes ont été développés, certains avec une vocation généraliste (tels que STREAM (Arasu et al., 2004), TelegraphCQ (Chandrasekaran et al., 2003), Borealis (Abadi et al., 2005), Aurora/Medusa (Zdonik et al., 2003)), alors que d'autres sont destinés à un type particulier d'application (tels que Gigascope (Cranor et al., 2003), Hancock (Cortes et al., 2000), NiagaraCQ (Chen et al., 2000)).

2.1 STREAM

STREAM (pour STanford stREam datA Management) est un système prototype de gestion de flux de données généraliste développé à l'Université de Stanford (Arasu et al., 2004). Ce système permet d'appliquer un grand nombre de requêtes déclaratives et continues à des données statiques (tables) et dynamiques (flux de données).
Un langage déclaratif CQL (Continuous Query Language) (Arasu et al., 2003), dérivé de SQL, a été implémenté pour pouvoir traiter des requêtes continues. STREAM considère deux structures de données :

- Flux : Un flux S est un ensemble d'éléments (s,t), où s est un tuple appartenant au flux, et t ∈T est l'estampille temporelle (timestamp) du tuple, t croissant de façon monotone.
- Relation : Une relation R(t) est un ensemble de tuples qui dépend du temps. A chaque instant une relation est susceptible d'avoir un nouveau contenu.

Le language CQL spécifie des opérations de sélection, d'agrégation, de jointure, de fenêtrage sur les flux, ainsi que trois opérateurs transformant des relations en flux :

- Istream(R(t)) : envoie sous forme de flux les nouveaux tuples apparus dans R à la période t ;
- Dstream(R(t)) : envoie sous forme de flux les tuples supprimés de R à la période t ;
- Rstream(R(t)) : envoie sous forme de flux tous les tuples appartenant à R à la période t.

CQL supporte des fenêtres glissantes sur un flux S, qu'on peut décliner en 3 groupes : (1) fenêtre basée sur le temps T (S[Range T]), (2) fenêtre basée sur le nombre de tuples N (S[Rows N]), et (3) fenêtre partitionnée (S[Partition By A_1, ..., A_k Rows N]) similaire au Group By en SQL.
L'utilisateur saisit ses requêtes grâce à une interface graphique ou en utilisant un fichier XML. Un plan d'exécution de la requête est créé par le système. Il est composé d'opérateurs, de buffers (pour garder temporairement les données non encore traitées) et de résumés de certaines données (pour donner des réponses approximatives en cas de surcharge du système).

2.2 TelegraphCQ

TelegraphCQ est un Système de Gestion de Flux de Données réalisé à l'Université de Berkeley (Chandrasekaran et al., 2003). Ce système est construit comme une extension du SGBD relationnel PostGreSQL pour supporter des requêtes continues sur des flux de données. Le format d'un flux de données est défini comme toute table PostGreSQL dans le langage de définition de données de PostGreSQL, et est créé à l'aide de la clause CREATE STREAM, avant d'être utilisé dans des requêtes continues. A chaque source de flux est affecté un traducteur (wrapper) qui transforme les données en un format compatible avec les types de données PostGreSQL. TelegraphCQ est un mode d'exécution de PostGreSQL, l'utilisateur pouvant aussi lancer un serveur PostGreSQL en mode normal.
Les requêtes continues peuvent inclure une clause de fenêtrage sur les flux. Une fenêtre est définie sur un intervalle de temps, le timestamp de chaque n-uplet du flux étant assigné par le wrapper si celui-ci ne possède pas un attribut déclaré comme timestamp. La sémantique de fenêtrage a évolué au cours des différentes versions de TelegraphCQ. La version la plus récente définit la clause de fenêtrage par [RANGE BY 'interval' SLIDE BY 'interval' START AT 'date et heure de début'] :

- RANGE BY : définit la taille de la fenêtre en terme d'intervalle de temps ;
- SLIDE BY : définit l'intervalle après lequel la fenêtre est recalculée, spécifié en intervalle de temps ;
- START AT : définit l'instant (date et heure) auquel la première fenêtre commence, dans un format de date standard.

Les flux interrogés par les requêtes peuvent être archivés dans la base de données PostgreSQL à la demande de l'utilisateur (clause ARCHIVED). Dans le cas contraire, ils sont supprimés du système lorsque les requêtes n'ont plus à les conserver dans des espaces temporaires pour leur exécution.

3 Expérimentation du traitement de flux de consommations électriques

Nous avons installé les deux systèmes STREAM et TelegraphCQ version 2.1 et les avons testés sur un jeu de données composé d'index de consommations relevés à pas de 2 secondes pendant 150 jours. Ces relevés correspondent à trois compteurs électriques se trouvant dans des villes différentes.

3.1 Spécification des traitements

L'analyse des consommations électriques peut être utile pour le fournisseur d'électricité comme pour le client. Ce dernier pourra s'en servir, par exemple, pour détecter les causes d'une consommation excessive ou anormale.

Un compteur « communicant » est capable de transmettre toutes les 2 secondes un enregistrement (tuple) de données contenant le numéro de compteur, l'heure du relevé, la date, l'index de consommation, et d'autres informations annexes. La différence entre deux index à deux instants différents représente la consommation d'électricité en energie entre ces deux instants. Les tuples provenant de plusieurs compteurs forment un flux de données de consommation.

Pour notre expérimentation, le jeu de données que nous avons utilisé est composé de tuples dont la structure (simplifiée) est la suivante :

```
Flux_index(annee INT, mois INT, jour INT, h INT, m INT, sec INT,
compteur char(12), index INT)
```

Flux_index est le nom identifiant le flux de données. La date d'un relevé est spécifiée à l'aide de trois attributs : 'mois', 'jour', 'année' ; l'heure à l'aide de trois autres attributs : 'h', 'm', 'sec' (heure, minute, seconde). L'attribut 'compteur' désigne l'identifiant du compteur et 'index' désigne le relevé de l'index du compteur (en kWh).

L'analyse des consommations s'appuie sur des requêtes continues sur le flux. Parmi les requêtes possibles nous avons choisi trois requêtes qui nous semblent assez représentatives de ce type d'application.

1. Consommation des 5 dernières minutes, minute par minute, par compteur, ou par ville ;
2. Consommation historique minute par minute, par compteur, à partir d'une date de départ ;
3. Alarme de dépassement de consommation heure par heure par rapport à une consommation dite « normale » dépendant de la température.

Nous avons essayé d'implanter ces requêtes sur les deux SGFD TelegraphCQ et STREAM. Les sections suivantes, 3.2 et 3.3 présentent le résultat de ce travail. Elles précisent pour chaque SGFD les requêtes qui ont pu être implantées, et expliquent comment cette implantation a pu se faire.

3.2 Implantation sur Stream

Une des spécificités de STREAM est qu'il ne supporte pas les expressions de requêtes imbriquées. Cependant, il offre la possibilité d'associer un nom à une expression de requête (comme une vue) et de s'en servir dans l'expression d'une requête plus complexe.

Traitement de données de consommation électrique par un SGFD

Q1 - Consommation des cinq dernières minutes, minute par minute, par compteur ou par ville. Le flux est une suite infinie de tuples qui associent des relevés d'index à des dates de prélèvements (année, mois, jour, heure, minute, seconde). Dans ce cas, le calcul de la consomation sur une durée donnée (dans notre exemple une minute) peut se faire en calculant la différence entre les valeurs minimales de l'index observées pendant deux périodes (minutes) successives. L'implantation de la requête $Q1$ peut donc se faire sur STREAM en suivant les étapes suivantes :

1. Min_index : calcul du minimum des index par compteur et par minute.

```
SELECT    annee, mois, jour, h, m, compteur, min(index) as minindex
FROM      Flux_index
GROUP BY  annee, mois, jour, h, m, compteur ;
```

2. Conso_Minute_60 : calcul de la consommation de la dernière minute de chaque heure, entre hh :59 et hh+1 :00 (exemple : entre 07 :59 et 08 :00).

```
SELECT  c1.annee as annee_deb, c1.mois as mois_deb, c1.jour as jour_deb,
        c1.h as h_deb, c1.m as m_deb, c1.compteur, c1.minindex as minindex_deb,
        c2.annee as annee_fin, c2.mois as mois_fin, c2.jour as jour_fin,
        c2.h as h_fin, c2.m as m_fin, c2.minindex as minindex_fin,
        c2.minindex-c1.minindex as conso
FROM    Min_index as c1, Min_index as c2
WHERE   c1.mois=c2.mois AND c1.jour=c2.jour AND c1.annee=c2.annee
AND     c1.h=(c2.h-1) AND c1.m=59 AND c1.compteur=c2.compteur AND c2.m=0
```

3. Conso_Minute_01_59 : Calcul de la consommation minute par minute entre la première minute et la minute 59 de chaque heure (exemple : entre 07 :00 et 07 :59 minute par minute).

```
SELECT  (identique à Conso_Minute_60)
FROM    Min_index as c1, Min_index as c2
WHERE   c1.mois=c2.mois AND c1.jour=c2.jour AND c1.annee=c2.annee
AND     c1.h=c2.h AND c1.m=(c2.m-1) AND c2.m>0 AND c1.compteur=c2.compteur
```

4. Flux_Conso : Union des consomations calculées par les deux requêtes précédentes en un flux unique.

```
ISTREAM(Conso_Minute_60 UNION Conso_Minute_01_59) ;
```

5. Conso_par_CptV : Calcul de la consommation des 5 dernières minutes, minute par minute et par compteur.
   Nous faisons une jointure avec TableVille (table de correspondance compteur-ville) pour afficher la ville de chaque compteur. Nous avons pris une fenêtre de 150 secondes car un tuple arrive normalement toutes les 2 secondes.

```
SELECT    annee_deb, mois_deb, jour_deb, h_deb, m_deb, Flux_Conso.compteur,
          Tableville.ville, minindex_deb, annee_fin, mois_fin, jour_fin, h_fin,
          m_fin, minindex_fin, conso
FROM      Flux_Conso[range 150 seconds], Tableville
WHERE     Flux_Conso.compteur = Tableville.compteur ;
```

6. Calcul de la somme des consommations par ville.

```
SELECT      annee_deb, mois_deb, jour_deb, h_deb, m_deb, ville, sum(conso)
FROM        Conso_par_CptV
GROUP BY    annee_deb, mois_deb, jour_deb, h_deb, m_deb, ville ;
```

Q2 - Consommation historique minute par minute, par compteur, à partir d'un point de départ. Nous appliquons les mêmes requêtes préliminaires que précédemment sans les clauses de fenêtrage. A la dernière requête (Conso_par_CptV), nous filtrons pour ne garder que les dates et heures dépassant le point de départ fixé. Par exemple, pour un point de départ le 03/12/2003 à 07h56 :

```
SELECT    annee_deb, mois_deb, jour_deb, h_deb, m_deb, Fluxconso.compteur,
          Tableville.ville, minindex_deb, annee_fin, mois_fin, jour_fin,
          h_fin, m_fin, minindex_fin, conso
FROM      Flux_Conso, Tableville
WHERE     Fluxconso.compteur = Tableville.compteur
AND       annee_fin >= 2003 AND mois_fin >=12 AND jour_fin >=03
AND       h_fin >= 07 AND m_fin > 56
```

Q3 - Alarme de dépassement de consommation heure par heure de minuit à l'heure courante. Cette requête donne une alarme quand une consommation dépasse de 10 % la consommation normale sur une période de minuit à l'heure courante, et cela à partir de midi. La requête $Q3$ n'a pas été traitée dans STREAM. Ceci est dû d'une part au manque des opérateurs de gestion des dates, et d'autre part, au fait que STREAM ne gère que les timestamps de type entier naturel. En effet, pour réaliser cette requête, il aurait fallu traiter plusieurs cas : le passage d'une journée à une autre (exemple : 12/02/2004 23 :00 et 13/02/2004 00 :00), le passage d'un mois à un autre (exemple : 31/01/2004 23 :00 et 01/02/2004 00 :00), le passage d'une année à une autre,...

3.3 Implantation sur TelegraphCQ

Nous avons créé un flux Elec.flux et lui avons assigné un wrapper de la manière suivante :

```
CREATE STREAM Elec.flux (    compteur varchar(12), index INTEGER,
                             tcqtime TIMESTAMP TIMESTAMPCOLUMN )
TYPE ARCHIVED ;
ALTER STREAM Elec.flux       ADD WRAPPER csvwrapper ;
```

csvwrapper est un wrapper fourni dans le package d'installation de TelegraphCQ. Il permet de manipuler des fichiers de valeurs séparées par des virgules.
Ce fichier de données contient des timestamps. Pour un même compteur, l'intervalle entre deux

mesures d'index envoyées est de 2 secondes. 'compteur' désigne l'identifiant du compteur, 'index' désigne le relevé de l'index du compteur (en kWh), et 'tcqtime' est la date des tuples, de type date au format 'AAAA-MM-JJ hh :mm :ss'. Grâce aux fonctionnalités offertes par PostGreSQL, TelegraphCQ manipule les dates sous leur format standard .

Q1 - Consommation des cinq dernières minutes, minute par minute, par compteur ou par ville. Pour résoudre cette requête, nous avons calculé le minimum des index pour chaque minute. Nous avons ensuite effectué une autojointure du résultat obtenu sur un décalage d'une minute. Grâce aux opérateurs de gestion du format date standard, nous avons pu calculer facilement la consommation entre deux instants décalés d'une minute.
Les requêtes imbriquées ne sont pas supportées mais ce problème peut être détourné grâce à la construction WITH qui permet l'utilisation de requêtes intermédiaires nommées, pour produire un résultat final. Les autojointures ne sont pas permises dans TelegraphCQ, nous avons donc été contraints de créer deux flux identiques pour les joindre. Nous avons créé un premier flux Elec.minflux1 qui calcule le minimum d'index par minute sur une fenêtre de 6 minutes, et un deuxième flux Elec.minflux2 identique mais sur une fenêtre de 5 minutes. Ceci nous permet de faire la différence entre ces deux flux avec un décalage d'une minute pour obtenir la consommation par minute pendant les 5 dernières minutes.

```
CREATE STREAM   Elec.minflux1 (   compteur varchar(12), minindex INTEGER,
                                  tcqtime TIMESTAMP TIMESTAMPCOLUMN )
                                  TYPE UNARCHIVED ;
CREATE STREAM   Elec.minflux2 (   identique à Elec.minflux1 ) ;

WITH
Elec.minflux1 AS (  SELECT      compteur,min(index),DATE_TRUNC('minute', tcqtime)
                    FROM        Elec.flux [RANGE BY '6 minutes' SLIDE BY '1 minute'
                                START AT '2003-12-04 07 :50 :00']
                    GROUP BY    compteur, DATE_TRUNC('minute', tcqtime) )
Elec.minflux2 AS (  SELECT ...  même définition que Elec.minflux1
                                avec RANGE BY à 5 minutes )
(
SELECT  f1.compteur, f1.minindex, f1.tcqtime, f2.minindex, f2.minindex-f1.minindex, f2.tcqtime
FROM    Elec.minflux1 as f1 [RANGE BY '1 minute' SLIDE BY '1 minute'
        START AT '2003-12-04 07 :50 :00'],
        Elec.minflux2 as f2 [RANGE BY '1 minutes' SLIDE BY '1 minute'
        START AT '2003-12-04 07 :50 :00']
WHERE   f1.compteur = f2.compteur AND f1.tcqtime = (f2.tcqtime - interval '1 minute')) ;
```

Pour afficher les consommations par ville, nous transformons la dernière requête en flux que nous appelons Elec.fluxconso. Les flux Elec.minflux1 et Elec.minflux2 sont calculés sur des fenêtres de 1 minute. Ensuite, nous réalisons une jointure entre la table Tableville et Elec.fluxconso fenêtré à 5 minutes.

Q2 - Consommation historique minute par minute, par compteur, à partir d'un point de départ. La réalisation de cette requête est facilitée par la clause START AT permettant de filtrer les tuples à partir d'un point de départ fixé.

Q3 - Alarme de dépassement de consommation heure par heure de minuit à l'heure courante. La consommation normale est fournie dans une table consoNormale, pour chaque heure et pour une température de 20°C. Nous disposons d'un flux Elec.temperature qui donne la température relevée chaque heure pendant 150 jours. Ce flux est affecté à un deuxième wrapper csvwrapper2. Pour s'approcher des conditions réelles, nous construisons un flux de consommation normale Elec.fluxconsonormale qui dépend de la température. L'implantation de la requête Q3 a été faite en suivant les étapes suivantes :

1. Nous calculons les consommations cumulées de 24 heures heure par heure, et nous sélectionnons celles au-delà de 12h. Le flux Elec.fluxconsoParHeure a été construit en suivant la même démarche que pour la consommation minute par minute en prenant une fenêtre de 1 heure. wtime(*) donne le timestamp du dernier tuple de la fenêtre en cours.

```
WITH
Elec.fluxconsoJour AS(      SELECT      compteur,sum(consommation),
                                        DATE_TRUNC('day',tcqtime),wtime(*)
                            FROM        Elec.fluxconsoParHeure [RANGE BY '24 hours'
                                        SLIDE BY '1 hour'START AT '2003-12-03 00 :00 :00']
                            GROUP BY    compteur, DATE_TRUNC('day',tcqtime))
Elec.consoCumul1224 AS(     SELECT      compteur, consoCumul, jour, tcqtime
                            FROM        Elec.fluxconsoJour
                            WHERE       EXTRACT('day' from jour)=EXTRACT('day' from tcqtime)
                                        AND EXTRACT('hour' from tcqtime)>11)
```

2. Nous calculons les consommations normales cumulées de 24 heures heure par heure, et nous sélectionnons celles au-delà de 12h. Nous utilisons une approche de gradient de température dans le calcul de la consommation normale.

```
Elec.fluxconsonormale AS(   SELECT      Conso*(1+(f1.temperature-20)/20)
                                        as ConsoFdeT,f1.tcqtime
                            FROM        Elec.temperature as f1 [RANGE BY '1 hour'
                                        SLIDE BY '1 hour'START AT '2003-12-04 00 :00 :00'],
                                        ConsoNormale
                            WHERE       EXTRACT(hour from f1.tcqtime)=ConsoNormale.HeureJournee)

Elec.fluxconsoNormaleJour AS...         (même raisonnement que Elec.fluxconsojour)
Elec.consoNormaleCumul1224 AS...        (même raisonnement que Elec.consoCumul1224)
```

3. Nous comparons la consommation journalière avec la consommation normale.

```
(
SELECT      f1.compteur, f1.consocumul as Conso,
            f2.consocumul as ConsoNormale,f1.jour,f1.tcqtime
FROM        Elec.consoCumul1224 as f1 [RANGE BY '13 hours' SLIDE BY '1 hour'
            START AT '2003-12-03 00 :00 :00'], Elec.consoNormaleCumul1224 as f2
            [RANGE BY '13 hours' SLIDE BY '1 hour' START AT '2003-12-03 00 :00 :00']
WHERE       f1.tcqtime=f2.tcqtime AND f1.consocumul > 1.10*f2.consocumul) ;
```

4 Synthèse des expérimentations

Adéquation des langages

Au vue des implantations des requêtes à la section 3, nous remarquons que la logique de résolution des requêtes par SGFD est assez éloignée de la logique SQL. Dans le cas des bases de données, les requêtes manipulent des données persistentes qui sont sélectionnées selon leurs valeurs dans le contenu d'une ou de plusieurs relations. Dans le cas des flux, les données sont volatiles et le contenu est non borné ce qui rend plus complexe la sélection des données et leur traitement. Il faut (1) spécifier par le mécanisme des fenêtres temporelles un réservoir de tuples à constituer et sur lequel s'appliquera la requête, et (2) spécifier les critères de sélection et le traitement à faire sur ce réservoir. Sachant que le réservoir de tuple peut évoluer dans le temps (de nouveaux tuples y entrent et d'autres en sortent) et par conséquent la requête doit être re-calculée et son résultat mis à jour par une série d'insertions et de suppressions de tuples. Nous remarquons également que les langages proposés sur les SGFD que nous avons testés sont assez limités, ce qui rend certaines requêtes plus complexes qu'elles ne le devraient et nous oblige à contourner le manque d'expressivité de ces langages par des solutions assez lourdes. A titre d'exemple, les SGFD STREAM ET TelegraphCQ ne traitent que des données estampillées par une date et ne sont pas adaptés au traitement des données estampillées par un intervalle de temps, ce qui explique en partie la complexité des solutions proposées pour résoudre les requêtes. Il aurait été intéressant à titre d'exemple d'avoir la possibilité d'obtenir la première et la dernière valeur d'une expression pour une fenêtre donnée. Ceci nous aurait permis de calculer directement l'énergie entre deux instants t1 et t2 sans passer par des requêtes intermédiaires.
Malgré ces limites, nous remarquons que les raisonnements et les formulations des requêtes dans TelegraphCQ sont plus intuitives que dans STREAM grâce à la gestion des fenêtres et du type de timestamp utilisé (format date).

Performances

L'équipe STREAM s'est intéressée à améliorer les performances de son système. Pour une bonne utilisation des ressources, STREAM propose des mécanismes permettant un partage des requêtes entre les flux et un partage des flux entre les requêtes. Les développeurs ont implanté une interface graphique pour visualiser les plans d'exécution des requêtes ainsi que le taux d'utilisation du système grâce à des jauges disponibles sur les différents composants du plan de la requête. L'utilisateur peut surveiller l'état du système et ré-optimiser l'exécution des requêtes dynamiquement en manipulant les tailles des buffers pour mieux s'adapter aux conditions du système ou des flux. Cependant, STREAM tel qu'il est offert au public demeure un prototype utile pour effectuer des démonstrations sur les possibilités des SGFD, mais n'est pas adapté à un besoin réel tel que le nôtre pour y répondre de manière complète et robuste. En effet, nous avons été confrontés à des arrêts intempestifs du serveur à plusieurs reprises lors de l'exécution des requêtes.
TelegraphCQ est un système robuste, en terme de charge et de complexité des requêtes acceptées. Il est possible d'ajouter des requêtes même si d'autres requêtes sont en cours d'exécution. Nous avons implanté toutes les requêtes présentées dans cet article sans le moindre souci. Néanmoins, nous ne nous sommes intéressés dans le cadre de cette étude qu'à l'as-

pect fonctionnel de ces deux SGFD. Nous envisageons dans nos prochaines expérimentations d'augmenter le nombre de compteurs électriques de notre jeu de données pour tester la charge de TelegraphCQ.

Divers

Les données d'un flux de consommation électrique arrivent constamment et une requête continue ne connait pas à priori l'extrémité d'un flux (contrairement à une table dans les SGBD). Les fenêtres nécessaires à des opérations sur les flux telle que l'agrégation sont implémentées dans les deux SGFD testés. Toutefois, il était impossible d'utiliser la fonction *min* avec un opérateur de fenêtrage lors de l'exécution de la requête Q1 dans STREAM, pour une raison inconnue. Ceci nous aurait permis d'améliorer les performances en ne gardant que les données dont nous avons besoin pendant une période donnée.
TelegraphCQ possède une syntaxe claire, une simplicité d'utilisation et de formulation des requêtes bien meilleure que STREAM. Ceci est dû aux fonctionnalités fournies par PostgreSQL telle que la gestion des formats de date et les méthodes associées. De plus, TelegraphCQ offre la possibilité de réutiliser les résultats des requêtes sous forme de flux, ou de les stocker dans un fichier résultat en spécifiant le format de sortie.
Quant à STREAM, il se distingue par son interface graphique permettant de se connecter au serveur, de lui envoyer des requêtes et de visualiser le plan des requêtes. Par exemple, il a été possible de voir dans STREAM que les opérations d'agrégation (sans fenêtrage) de la requête $Q2$ s'effectuent de façon incrémentale sans garder en mémoire tout le flux pour recalculer le résultat de l'agrégation. Une autre caractéristique de STREAM est de pouvoir spécifier des fenêtres logiques exprimées en nombre de tuples, ces fenêtres peuvent être utiles dans certaines applications.

5 Conclusion et perspectives

Nous avons montré que deux Systèmes de Gestion de Flux de Données (SGFD) permettent de traiter des données de consommation électrique arrivant en rythme continu et rapide. D'après nos expérimentations, il apparaît que TelegraphCQ est mieux adapté à nos besoins que STREAM. En effet, il offre une simplicité d'utilisation, une syntaxe claire et des fonctionnalités intéressantes pour analyser les résultats. De plus, l'équipe de Berkeley travaillant sur ce projet est toujours active dans ce domaine alors que l'équipe STREAM a suspendu ses travaux depuis janvier 2006.
Cependant, la comparaison entre ces deux SGFD est principalement fonctionnelle et n'a pas pris en compte les performances en temps d'exécution et en consommation de ressources système. Ce sera un objectif de nos prochaines expérimentations. Une des perspectives des nombreuses équipes qui ont implémenté des SGFD est de proposer une architecture distribuée de leur système. Une première proposition est faite par l'équipe de Aurora qui propose le système Boréalis (Abadi et al., 2005). Boréalis distribue le traitement des flux de données sur plusieurs sites d'Aurora pour des raisons de répartition de charge et de tolérance aux pannes. Nous testerons dans nos prochains travaux ce système qui a le mérite d'être en libre téléchargement. Nous surveillerons aussi les éventuelles versions futures de TelegraphCQ.

Références

Abadi D.J., Ahmad Y., Balazinska M., Cetintemel U., Cherniack M., Hwang J.H., Lindner W., Maskey A.S., Rasin A., Ryvkina E., Tatbul N., Xing Y., and Zdonik S. (2005). The Design of the Borealis Stream Processing Engine. 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05), Asilomar, CA, January 2005.

Arasu A., Babu S., Widom J. (2003). The CQL Continuous Query Language : Semantic Foundations and Query Execution, Technical Report, Oct. 2003

Arasu A., Babcock B., Babu S., Cieslewicz J., Datar M., Ito K., Motwani R. , Srivastava U., Widom J.(2004). STREAM : The Stanford Data Stream Management System, Book chapter.

Babcock B., Babu S., Datar M., Motwani R., et Widom J. (2002). Models and issues in data stream systems. In Symposium on Principles of Database Systems, pages 1-16. ACM SIGACT-SIGMOD.

Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin,Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Vijayshankar Raman, Fred Reiss, Mehul A. Shah (2003). TelegraphCQ : Continuous Dataflow Processing for an Uncertain World. CIDR 2003.

Chen J., DeWitt D. J., Tian F., Wang Y. (2000). NiagaraCQ : A Scalable Continuous Query System for Internet Databases. SIGMOD Conference 2000 : 379-390.

Cortes C., Fisher K., Pregibon D., and Rogers A. (2000). Hancock : A Language for Extracting Signatures from Data Streams. In Proceedings of the Association for Computing Machinery Sixth International Conference on Knowledge Discovery and Data Mining, pages 9–17.

Cranor C. D., Johnson T., Spatscheck O., Shkapenyuk V. (2003). Gigascope : A Stream Database for Network Applications. SIGMOD Conference 2003 : 647-651.

Golab L. and Özsu M.T. (2003). Data stream management issues - a survey. Technical report CS 2003-08, University of Waterloo, Waterloo, Canada, April 2003.

Stan Zdonik, Michael Stonebraker, Mitch Cherniack, Ugur Cetintemel, Magdalena Balazinska, and Hari Balakrishnan (2003). The Aurora and Medusa Projects. Bulletin of the Technical Committe on Data Engineering. IEEE Computer Society. March 2003. p.3-10. (Invited Paper).

Summary

With the development of AMM (Automatic Metering management), it will be possible for electric power suppliers to acquire from their customers their electric power consumption up to every one second. This will generate data arriving in multiple, continuous, rapid, and time-varying data streams. Data Stream Management Systems (DSMS) - currently available as prototypes - aim at facilitating the management of such data streams. This paper describes an experimental study which analyzes advantages and limitations of using two DSMSs (STREAM and TelegraphCQ) for the management of electric power consumption data.