Extension de C-SPARQL pour l'échantillonnage de flux de graphes RDF

Amadou Fall Dia*, Zakia Kazi-Aoul*, Aliou Boly**, Yousra Chabchoub*

*ISEP, 10 rue de Vanves 92130 ISSY LES MOULINEAUX, France
prenom.nom@isep.fr,
**Université Cheikh Anta Diop de Dakar, BP 5005 Dakar-Fann, Sénégal
prenom.nom@ucad.edu.sn

Résumé. Les technologies du web sémantique sont de plus en plus utilisées pour la gestion de flux de données. Plusieurs systèmes de traitement de flux RDF ont été proposés : C-SPARQL, CQELS, SPARQL$_{stream}$, EP-SPARQL, SPARKWAVE, etc. Ces derniers étendent tous à la base, le langage d'interrogation sémantique SPARQL. Les données à l'entrée du système sont volumineuses et générées en continu à un rythme rapide et variable. De ce fait, le stockage et le traitement de la totalité du flux deviennent coûteux et le raisonnement presque impossible. Par conséquent, le recours à des techniques permettant de réduire la charge tout en conservant la sémantique des données, permet d'optimiser les traitements voire le raisonnement. Cependant, aucune des extensions de SPARQL n'inclut cette fonctionnalité. Ainsi, dans cet article, nous proposons d'étendre le système C-SPARQL pour générer des échantillons à la volée sur flux de graphes RDF. Nous ajoutons trois opérateurs d'échantillonnage (UNIFORM, RESERVOIR et CHAIN) à la syntaxe de C-SPARQL. Les expérimentations montrent la performance de notre extension en terme de temps d'exécution, et de la préservation de la sémantique des données.

1 Introduction

Notre usage quotidien des réseaux sociaux (Facebook, Twitter, Linkedin, etc.), des plateformes de contenus multimédias (YouTube, Flickr, iTunes, etc.), des réseaux de capteurs (observation, télé-relève, etc.), produit en continu des flux de données. Plusieurs groupes de recherches ont proposés des systèmes appliquant les technologiques du web sémantique aux flux de données.

Étant donné la vitesse de génération des données et leur volume, le traitement de la totalité du contenu d'un flux est difficile et le raisonnement presque impossible. Ces systèmes nécessitent dès lors (i) des techniques d'allocation dynamique de ressources au système ou (ii) de réduction de la charge des données en entrée (résumé). Concernant cette dernière technique, aucune des extensions de SPARQL proposées (Streaming SPARQL Bolles et al. (2008), C-SPARQL Barbieri et al. (2010), CQELS Le-Phuoc et al. (2011), SPARQL$_{stream}$ Calbimonte

et al. (2010), EP-SPARQL Anicic et al. (2011), Sparkwave Komazec et al. (2012), etc.) n'implémente des mécanismes permettant de se délester d'une partie des données par la construction à la volée de résumés.

Dans cet article, nous proposons d'étendre le système C-SPARQL par l'ajout de fonctionnalités permettant d'échantillonner en continu les flux en entrée. Notre objectif est de réduire à la volée la charge des données en entrée afin de gagner en performance (réduction du temps de traitement) tout en préservant les liens sémantiques. Pour ce faire, nous avons dû travailler sur deux axes. Le premier concerne le format d'entrée des données. Toutes les extensions proposées prennent en entrée des successions de triplets RDF assortis d'estampilles temporelles. Suivant cette forme de représentation, l'application des techniques actuelles d'échantillonnage conduit à la disparition de liens entre les données, impliquant de ce fait une perte d'une partie de la sémantique. Ainsi, en lieu et place du format (<sujets, predicat, objet>, estampille), nous adoptons une approche orientée graphe (succession de graphes estampillés). Le deuxième axe consiste à ajouter de nouveaux opérateurs à la syntaxe de C-SPARQL et implémentés dans le module continu de C-SPARQL.

Cet article est organisé comme suit. La section 2 introduit la notion de flux de graphes par déduction de la représentation standard de données sémantiques (RDF). Nous faisons un bref état de l'art sur les extensions existantes de SPARQL dans la section 3. La section 4 présente les trois (3) algorithmes d'échantillonnage implémentés. La section 5 est consacrée à la principale contribution de notre travail avec la syntaxe et l'architecture étendue de C-SPARQL. La section 6 porte sur l'évaluation des performances de la solution implémentée sur des flux de données réelles issues de capteurs déployés dans un réseau de distribution d'eau potable. Enfin, nous donnons la conclusion et les perspectives à nos travaux.

2 Flux de graphes RDF

Dans cette section, nous présentons la notion de flux de graphes RDF que nous utilisons pour l'échantillonnage dans C-SPARQL.

2.1 Flux RDF : approche orientée triplet

RDF (Resource Description Framework) est la recommandation formelle du W3C pour la représentation sémantique de données. Les données sont représentées sous forme de triplet $\langle s, p, o\rangle \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ où $\mathcal{I}$ est l'ensemble des IRIs, $\mathcal{B}$ l'ensemble des nœds anonymes et $\mathcal{I}$ l'ensemble des littéraux. s, p, o représentent l'information et sont respectivement appelés le sujet, le prédicat et l'objet.

Golab et Özsu (2003) définissent les flux de données comme une séquence d'items continue, ordonnée (implicitement par temps d'arrivée dans le Système de Gestion de Flux de Données, ou explicitement par timestamp de production à la source), arrivant en temps réel. L'adoption des technologies du web sémantique dans le monde des données dynamiques et des capteurs a donné naissance à la notion de flux de données RDF. Ainsi, les flux RDF ont été introduits comme extension naturelle du modèle RDF dans l'environnement des flux. Un

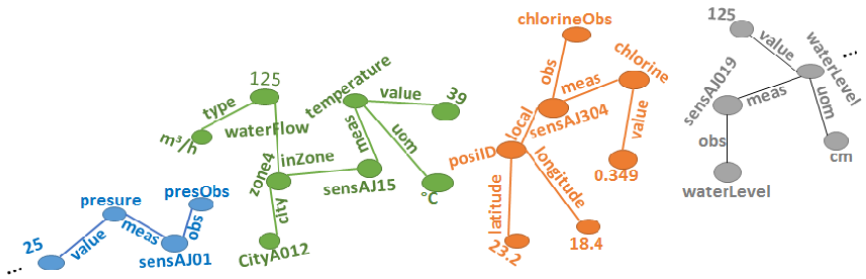

FIG. 1 – *Exemple de flux de graphes RDF.*

flux de données RDF $\mathcal{S}$ est une séquence ordonnée de paires, où chaque paire est constituée d'un triplet RDF et d'une estampille temporelle : $\mathcal{S} = (< t_r, \tau_i >)$, où t_r est un triplet RDF observé ou arrivé à l'instant τ_i. τ_i est monotone, non décroissante et non strictement croissante ($\forall i, \tau_i \leq \tau_{i+1}$).

$$(< subject_i, predicate_i, object_i >, \tau_i)$$
$$(< subject_{i+1}, predicate_{i+1}, object_{i+1} >, \tau_{i+1})$$
$$...$$

Étant donné la vitesse de génération de ces flux, une portion du flux dite fenêtre est traitées en utilisant le langage CQL Arasu et al. (2004). Cette forme de représentation permet un traitement à la volée d'un large flux de données RDF mais ignore complètement leur structuration en graphes.

2.2 Flux RDF : approche orientée graphe

Étant donné un ensemble de triplets t_r, nous définissons $G(t_r)$ comme un graphe orienté et labellisé sur les arrêtes où chaque nœud représente le sujet (s) ou l'objet (o) d'un triplet et chaque arrête, le prédicat (p) d'un triplet. La notion de connectivité dans un graphe RDF est très importante pour tous les triplets devant constituer un événement. En effet, les triplets formant un événement partagent entre eux un ou plusieurs chemins. Cette connectivité est définie par une séquence d'arrêtes $p_1, ..., p_n$ tel que $\forall 1 \leq i < n$, p_i partage un nœud (sujet ou objet) avec p_{i+1}.
De manière similaire, un graphe RDF peut être défini comme une séquence de paires $(G(t_r)_i, \tau_i)$, où $G(t_r)_i$ est un graphe RDF représentant un événement complet et τ_i son instant occurrent. La figure 1 montre l'exemple d'un flux de graphes RDF issus de capteurs déployés dans un réseau de distribution d'eau potable. Les données prélevées concernent la pression, le débit, le taux de chlore, la température, la zone de déploiement, etc.

$$(G(t_r)_i, \tau_i)$$
$$(G(t_r)_{i+1}, \tau_{i+1})$$
$$...$$

En particulier, l'échantillonnage de flux de triplets RDF peut casser les liens entre triplets formant un même événement. Nous adoptons dans nos travaux d'extension, une approche orientée graphe pour garantir la préservation de la sémantique entre données dans l'échantillon.

3 État de l'art sur les extensions de SPARQL

Tous les systèmes de traitement de flux RDF proposés partagent les mêmes problématiques en terme d'hétérogénéité (données multisources) et d'absence de sémantique explicite permettant de satisfaire des requêtes complexes et le raisonnement. Dans cette section, nous faisons un bref état de l'art sur les travaux d'extensions de SPARQL.

Streaming SPARQL Bolles et al. (2008) étend simplement la grammaire de SPARQL en ajoutant la possibilité de définir une fenêtre physique ou logique qui représente une portion du flux RDF. Ce système n'inclut pas d'opérateurs d'agrégation et reste jusqu'ici théorique (non implémenté).

C-SPARQL Barbieri et al. (2010) ajoute la définition spécifique de la notion de fenêtre (à l'image de CQL pour les flux relationnels). Il inclut, en plus des fenêtres physique et logique, des opérateurs d'agrégation, des fonctions temporelles et permet également la combinaison de données RDF statiques avec un ou plusieurs flux RDF (flux multiples).

CQELS Le-Phuoc et al. (2011) utilise une approche native, c'est-à-dire, implémente un système de traitement de flux RDF en ne réutilisant pas des technologies existantes comme le fait C-SPARQL (SGFD). Tout comme C-SPARQL, CQELS inclut également en plus des opérateurs de streaming (register query, windowing, etc), le traitement des données statiques.

SPARQL$_{stream}$ Calbimonte et al. (2010), comme C-SPARQL et CQELS étend également SPARQL 1.1 mais inclut également des solutions additionnelles. Il prend en entrée un flux RDF virtuel identifié par un IRI. Côté fenêtrage, sa syntaxe n'inclut pas les fenêtres logiques (un nombre de triplets donné).

EP-SPARQL Anicic et al. (2011) s'oriente vers le traitement d'événements complexes (CEP) et inclut des opérateurs de séquence et de simultanéité (**SEQ**, **EQUALS**, **OPTIONAL-SEQ** et **EQUALSOPTIONAL**). EP-SPARQL n'a pas d'opérateurs de fenêtrage.

Sparkwave Komazec et al. (2012), tout comme EP-SPARQL, garde une approche basée sur le traitement d'événements complexes. Il implémente une variante de l'algorithme Rete (Rete (1982)). La portion de données à traiter (fenêtre) est extraite en utilisant les mécanismes de fenêtrage similaires aux techniques traditionnelles utilisées par les DSMS et C-SPARQL.

Le tableau 3 fournit quelques éléments de comparaison entre ces différents langages d'extension. Streaming SPARQL est le plus limité en terme d'opérateurs comparé à C-SPARQL, CQELS et SPARQL$_{stream}$ qui étendent SPARQL 1.1. EP-SPARQL et Sparkwave sont différents des autres propositions et introduisent la notion de traitement d'événements. Les systèmes implémentés adoptent des approches différentes dans leur architecture et il n'est pas toujours évident de dire de façon générale quelle approche demeure la meilleure. Étant donné que CQELS et Sparkwave utilisent des approches natives, ils peuvent apporter des optimisations adaptables contrairement aux autres systèmes qui, ne peuvent faire qu'une optimisation algébrique. Cependant, coté fonctionnalité, C-SPARQL est le seul système qui fournit à la fois

System	DIF	TiW	TrW	U, J, O, F	TF	A	S/S	CQ
Streaming SPARQL	RS	✓	✓	✓	×	×	×	✓ (P)
C-SPARQL	RS&S	✓	✓	✓	✓	✓	×	✓ (P)
CQELS	RS&S	✓	✓	✓	×	✓	×	✓ (T)
SPARQL$_{stream}$	(V) RS	✓	×	✓	×	✓	×	✓ (P)
EP-SPARQL	RS&S	×	×	✓	✓	✓	✓	✓ (P)
Sparkwave	RS&S	✓	×	✓	×	×	×	✓ (T)

DIF : Data Input Format **TiW :** Time Window **TrW :** Triple Window **U :** UNION **J :** JOIN **O :** OPTIONAL **F :** FILTER **TF :** Temporal Function **A :** Aggregate **S/S :** Sequence/Simultaneity **CQ :** Continuous Query **RS :** RDF streams **RS&S :** RDF stream & Statics **(V) :** (Virtual) **(P) :** Periodic **T :** Trigger

TAB. 1 – *Quelques éléments de comparaison des extensions.*

les opérateurs Union, Join, Optional et Filter, les fenêtres logiques et physiques, les opérateurs d'agrégation, l'exécution continue, les flux multiples, l'interrogation à la fois de données statiques et dynamiques, les fonctions temporelles, etc. De plus, son architecture est modulaire et utilise Jena ou Sesame pour le traitement sémantique et Esper pour le traitement continu.

4 Algorithmes d'échantillonnage

Les systèmes de traitement de flux de données RDF nécessitent de plus en plus un traitement rapide, continu et intelligent des données. Il est nécessaire d'extraire un échantillon représentatif du flux en entrée. Plusieurs techniques d'échantillonnage sont proposées dans la littérature parmi lesquelles on peut citer l'échantillonnage aléatoire, l'échantillonnage réservoir et l'échantillonnage chain.

4.1 Échantillonnage aléatoire uniforme sans remise

Comme défini par Cochran (2007), l'échantillonnage aléatoire peut être avec ou sans remise. Ici, il consiste à sélectionner aléatoirement (avec la même probabilité p et sans remise) un échantillon de taille n parmi un ensemble d'indexes présents dans la fenêtre W. L'index d'un élément présent dans W ne peut être sélectionné qu'une seule fois.
Cette méthode est très basique et a l'avantage d'être simple et facile à implémenter. Cependant, cette technique donne à tous les éléments la même chance d'être inclus dans l'échantillon. Ce qui constitue un inconvénient car dans le contexte des flux de données, on s'intéresse souvent aux données récentes. Dans la constitution de l'échantillon, il est donc préférable de favoriser les données récentes par rapport aux autres données.

4.2 Échantillonnage réservoir

L'idée principale de tout type d'échantillonnage réservoir Vitter (1985) est de maintenir un échantillon d'une taille n fixe dans le "réservoir". A la fin de chaque processus de fenêtrage sur les flux, un échantillon valide de taille n peut être extrait du réservoir. Initialement, nous chargeons les n premiers indexes dans le "réservoir". Ensuite, à chaque arrivée d'un nouveau élément d'indice i, nous le remplaçons avec la probabilité $\frac{n}{i}$ avec un élément depuis le réservoir.
Cette méthode favorise clairement les éléments anciens. En effet, plus on avance dans la fenêtre, plus la probabilité d'inclusion de l'élément est réduite : $lim_{i\to\infty}\frac{n}{i} = 0$. Ainsi, les éléments les plus anciens ont plus de chances d'être inclus dans l'échantillon.

4.3 Échantillonnage chain sans redondance

L'échantillonnage chain Babcock et al. (2002) consiste à construire un échantillon de taille n sur une fenêtre glissante de taille $\omega > n$. Comme le montre l'algorithme 1, sur la première fenêtre, on ajoute les indexes i dans l'échantillon avec la probabilité $p = \frac{min(\omega,i)}{\omega}$. L'indice de remplacement r_i de l'index i est aléatoirement choisi dans l'intervalle $[i+1, i+\omega]$ et le remplace dans l'échantillon à son expiration (i sort de la fenêtre). Le remplaçant de r est choisi de la même manière aléatoirement dans l'intervalle $[r+1, r+\omega]$. Ce processus est ainsi répété de façon indépendante. Un échantillon S et un remplaçant R sont construits et l'échantillon final est la fusion des échantillons de chaque fenêtre. Pour garantir l'obtention d'éléments distincts, le processus du choix du remplaçant est répété jusqu'à l'obtention d'un nouvel élément non présent dans l'échantillon.

Algorithm 1 Échantillonnage chain sans redondance

1: **function** CHAINSAMP(ω, p)
2: $R_{epl} \leftarrow \varnothing$
3: S$\leftarrow$ put indexes (i) from the first window (with size ω) with probability $\frac{Min(\omega,i)}{\omega}$
4: **for** each index i in S **do**
5: Select a random replacement r_i with probability p between $i+1$ and $j+\omega$
6: $R_{epl} \leftarrow r_i$
7: **end for**
8: **while** each new index is added **do**
9: $i \leftarrow i+1$ ▷ move window index by one step
10: Replace each expired index j in S by its replacement r_j in R without redondancy
11: Choose a random replacement for r_j between r_j+1 and $r_j+\omega$ without redondancy
12: **end while**
13: **end function**
14: Return S

Cette méthode est particulièrement adaptée aux types de fenêtres glissantes mais néanmoins a une utilisation de mémoire non négligeable du fait du critère de choix non redondant de remplaçants.

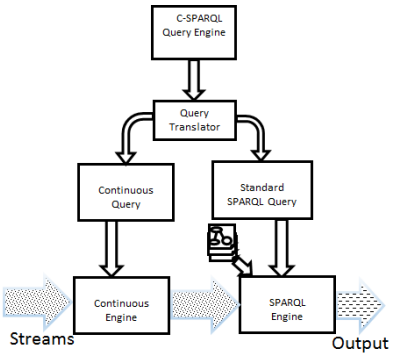

FIG. 2 – *Architecture de C-SPARQL.*

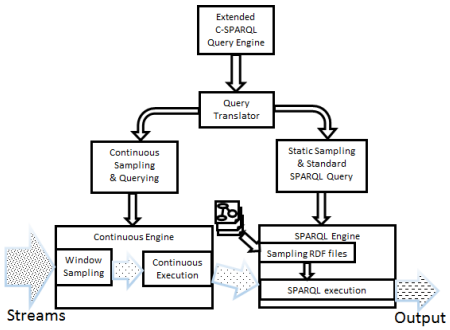

FIG. 3 – *Architecture étendue.*

5 Notre extension de C-SPARQL

Barbieri et al. (2010) proposent C-SPARQL comme langage et framework d'interrogation de flux de données sémantiques. Les auteurs introduisent d'abord les flux RDF et ajoutent de nouveaux éléments à la syntaxe de SPARQL (voir Barbieri et al. (2009). Ils proposent ensuite une architecture constituée de deux modules principaux : un DSMS tel que ESPER ou STREAM et un moteur SPARQL tel que Jena [1] ou Sesame [2].

La figure 2 présente l'architecture de C-SPARQL. *Query Translator* est le composant qui réalise la configuration, l'initialisation et le dispatching. Cette dernière tâche consiste d'abord à considérer en entrée une requête respectant la syntaxe de C-SPARQL afin de produire deux (2) instances destinées aux modules continu et statique.

- ***ContinuousEngine*** est constitué d'un DSMS qui traite à la volée les triplets appliquant ainsi la notion de fenêtre aux flux à travers une requête CQL Arasu et al (2004). Ce module produit en sortie un ensemble de quadruplets $(sujet, prdicat, objet, estampille)$ destiné au moteur SPARQL.
- ***SparqlEngine*** est le composant dit "sémantique". Il exécute la partie SPARQL qui correspond à la requête C-SPARQL à chaque fois que le résultat produit par le module de traitement continu est mis à jour.

La principale contribution de notre travail est l'extension de C-SPARQL au niveau syntaxique et architectural pour l'échantillonnage en continu de flux de graphes RDF. Pour réduire la charge des flux à traiter et stocker uniquement un échantillon du flux destiné à d'éventuels analyses ou raisonnements, notre implémentation est située en trois principales étapes :

- Considération d'un format graphe des flux au lieu d'une suite ordonnée de triplets RDF pour la préservation des liens sémantiques
- Ajout d'opérateurs d'échantillonnage à la syntaxe de requête C-SPARQL
- Implémentation des méthodes correspondant à ces opérateurs dans le module de traitement continu interne de C-SPARQL (Esper).

1. https ://jena.apache.org/
2. http ://rdf4j.org/sesame/

5.1 Opérateurs d'échantillonnage

Nous présentons ici les opérateurs d'échantillonnage ajoutés à la syntaxe de requête C-SPARQL par extension des clauses **FROM STREAM** (source continue) et **FROM** (source statique).

```
*PREFIX prefixName : <IRI>
SELECT ' ?variables'
*FROM STREAM <StreamIRI> [Window] [SAMPLING   Token]
*FROM <StaticIRI> [SAMPLING   Token]
WHERE { 'Mapping variables' ;|.
                *FILTER ('condition')}
GROUP BY ' ?variables'lexpression
HAVING 'aggregation condition'
ORDER BY ' ?variables'
```

La syntaxe étendue contient les nouveaux opérateurs des méthodes d'échantillonnage à appliquer aux données continues (flux de graphes) et statiques (RDF repository). * indique 0, 1 ou plusieurs occurrences de la clause qu'elle précède dans la requête. Nous utilisons les termes **SAMPLING** et **Token** pour uniquement plus de généralités dans l'écriture de la syntaxe. **SAMPLING** représente l'opérateur utilisé et prend les valeurs **UNIFORM**, **RESERVOIR** ou **CHAIN**. **Token** correspond aux paramètres d'échantillonnage de l'opérateur. En effet, il correspond au pourcentage d'échantillonnage pour l'opérateur UNIFORM, à la taille du reservoir pour l'opérateur RESERVOIR et à la fenêtre puis le pourcentage d'échantillonnage pour l'opérateur CHAIN.

5.2 Extension de l'architecture

Notre approche d'extension est basée sur l'implémentation de méthodes d'échantillonnage dans Esper. La figure 3 montre l'extension proposée de l'architecture de C-SPARQL où ses modules traditionnels restent toujours des plugins indépendants. Trois instances de *Query Translator*, *ContinuousEngine* et *SparqlEngine* sont toujours créées mais cette fois ci étendues.

- Dans ***QueryTranslator***, nous parsons la requête reçue en vérifiant la syntaxe des opérateurs d'échantillonnage des "parties" statique(s) ($FROM$) et continue(s) ($FROM\ STREAM$). Si la requête n'inclut pas un opérateur d'échantillonnage, les flux entrants sont traités en continu sans aucune phase d'échantillonnage préalable. Sinon, après validation, nous créons deux (2) instances de *ContinuousSampQuery* et *SparqlSampQuery* qui seront à exécuter respectivement par *ContinuousEngine* et *SparqlEngine*.
- Le module ***ContinuousEngine*** reçoit une requête continue avec le ou les opérateurs d'échantillonnages associés. Chaque méthode d'échantillonnage invoquée est appliquée en continu sur une fenêtre de flux de graphes entrant (*WindowSampling*). Nous exécutons ensuite la partie CQL (Esper) de la requête C-SPARQL sur les données disponibles dans l'échantillon. Le résultat est alors transmis au troisième module *SPARQLEngine*.
- ***SPARQLEngine***, est le dernier à entrer en jeu. En plus des données issues du module continu, il prend en entrée des échantillons dits statiques (*SampStaticRDF*) issus de graphes sources statiques (repository, fichier). Vient en dernière étape l'exécution

proprement dite de la requête SPARQL sur les graphes statiques et continus déjà échantillonnés.

6 Évaluation

Cette section évalue la qualité et la pertinence de notre extension. Pour ce faire, nous nous sommes intéressés aux performances obtenues en terme de temps d'exécution et de la préservation de la sémantique des données. Pour l'échantillonnage, nous considérons le traitement d'un ensemble de 80000 graphes, chaque graphe étant composé de 10 triplets. Cet ensemble de données est envoyé sous forme de flux de graphes et de triplets respectivement à des débits de 500 graphes et de 5000 triplets par seconde.

Pour l'évaluation des performances en temps d'exécution, nous considérons une requête simple qui retourne, après échantillonnage sur les 1000 derniers graphes reçus, la moyenne des valeurs de pression prélevées par chaque capteur.

```
REGISTER QUERY AvgWaterPressure AS
PREFIX ex : <http ://wtrdi.org/>
SELECT    ?sensorID    (AVG( ?pressureValue)  AS   ?AvgPressure)
FROM STREAM <http ://wtrdi.org/str> [RANGE TRIPLES 1000]
                                    [SAMPLING [window] percent|size]
WHERE { ?sensorID ex :hasPressure ?pressureValue . }
GROUP BY    ?sensorID
```

Pour les opérateurs d'échantillonnage (*UNIFORM*) et (*RESERVOIR*) figures 4 et 5, nous observons les courbes d'évolution du temps de traitement de la requête en faisant varier respectivement le pourcentage d'échantillonnage (*percent*) et la taille du réservoir (*size*). Enfin, pour l'opérateur *CHAIN* nous observons, pour chaque pourcentage d'échantillonnage, la courbe d'évolution du temps de traitement en fonction de la taille de la fenêtre (*Window*). Nous pou-

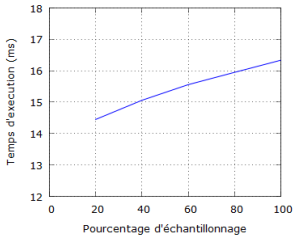

FIG. 4 – *Uniforme.*

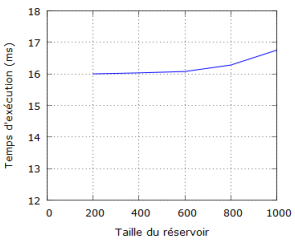

FIG. 5 – *Réservoir.*

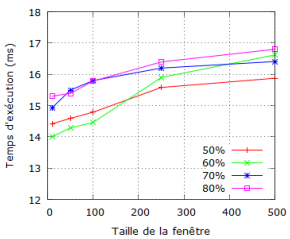

FIG. 6 – *Chain.*

vons noter qu'avec l'échantillonnage uniforme sans remise, le temps de traitement croît en fonction du pourcentage d'échantillonnage. Nous notons également une évolution similaire avec l'échantillonnage réservoir (figure 5) en fonction de la taille du réservoir maintenue fixe en

mémoire durant l'exécution. L'évolution du temps de traitement avec l'échantillonnage chain (figure 6) dépend de la taille de la fenêtre et du ratio. Nous notons que quel que soit le ratio, le temps d'exécution de la requête suit une évolution croissante. Cela s'explique sans doute par la technique de recherche aléatoire de remplaçants (sans duplication) aux éléments de l'échantillon. Ainsi les observations confirment les performances gagnées en réduisant à la volée la charge des flux en entrée.
Pour l'évaluation de la préservation des liens sémantiques des données figurant dans l'échantillon, nous considérons la requête suivante :

```
REGISTER QUERY sourceSensorID AS
PREFIX ex : <http ://wtrdi.org/>
SELECT    ?sensorID    ?pressureValue
FROM STREAM <http ://wtrdi.org/str> [RANGE TRIPLES 10]
                                [SAMPLING [window] percent|size]
WHERE { ?sensorID ex :hasPressure ?pressureMnemonic .
        OPTIONAL{ ?pressureMnemonic ex :value ?pressureValue .} }
ORDER BY    ?sensorID
```

Cette requête est d'abord exécutée en continu sans échantillonnage puis avec échantillonnage orienté triplet et enfin avec échantillonnage orienté graphe. Nous considérons de part et d'autre le même débit et le même temps d'exécution. La requête retourne pour les 10 derniers graphes ou triplets observés dans l'échantillon, le capteur et sa valeur de pression mesurée. Nous considérons un format de graphe où le nœud représentant l'identifiant du capteur est lié à la valeur de la pression par l'intermédiaire d'un autre nœud. Nous avons calculé pour chaque méthode d'échantillonnage basé triplet et graphe, le nombre de résultats corrects et complets (identifiant du capteur et valeur de pression correspondante). Nous avons par la suite calculé le taux de perte par rapport aux résultats corrects et complets sans échantillonnage en utilisant la formule suivante :

$$\textbf{Taux de perte}(\%) = \frac{\textbf{NbrSE - NbrAE}}{\textbf{NbrSE}} * 100 \text{ avec}$$

NbrSE = Nombre de résultats corrects et complets sans échantillonnage
NbrAE = nombre de résultats corrects et complets avec échantillonnage.

Le tableau 2 présente les taux de perte calculés en variant les paramètres d'échantillonnage des méthodes uniforme et réservoir. Le nombre de résultats corrects et complets (**NbrSE**) est égal à **10625**. Nous observons, avec l'échantillonnage orienté graphe un taux de perte largement en de-ssous de celui avec l'orienté triplet. En effet, cela s'explique par l'échantillonnage aléatoire dans le cas de l'orienté triplet, et le fait qu'un identifiant de capteur figurant dans l'échantillon peut ne pas se retrouver avec sa valeur de pression correspondante. En comparaison, l'échantillonnage orienté graphe préserve les liens sémantiques, garantissant ainsi qu'un capteur est toujours associé à sa valeur de pression dans l'échantillon.

7 Conclusion et perspectives

La gestion de flux de données massifs demeure une préoccupation industrielle et un défi scientifique. L'application des technologies du web sémantique aux flux de données reste trou-

Opérateur	Échantillonnage orienté triplet		Échantillonnage orienté graphe	
	Résultats corrects et complets	**Taux de perte (%)**	**Résultats corrects et complets**	**Taux de perte (%)**
UNIFORM				
$P = 20\%$	19	**99,82**	2108	**80,16**
$P = 40\%$	73	**99,31**	3614	**65,98**
$P = 80\%$	181	**98,2**	7137	**32,82**
RESERVOIR				
$Size = 2$	24	**99,77**	2117	**80,07**
$Size = 4$	144	**98,64**	4210	**60,37**
$Size = 8$	522	**95,08**	7240	**31,85**

TAB. 2 – *Taux de perte entre échantillonnage basé graphe et triplet.*

blée par les volumes actuels des flux et leur fréquence de génération rapide. Nous avons, dans cet article, tiré parti des techniques d'échantillonnage existantes en proposant une extension du système C-SPARQL pour l'échantillonnage en temps réel de flux de graphes RDF. L'utilisation de flux de graphes nous a permis de préserver la sémantique des échantillons et d'améliorer ainsi la représentativité de l'échantillon obtenu.
Nous comptons, dans nos travaux futurs échantillonner à la volée les flux RDF en utilisant des méthodes qui prennent en considération les spécificités et le contexte de la requête qui sera exécutée sur l'échantillon. Il s'agit de faire de l'échantillonnage orienté contenu sur des graphes RDF.

Remerciements

Ce travail a été réalisé dans le cadre du projet FUI Waves. Ce projet a pour but la conception et le développement d'une plateforme de traitement distribué de flux de données massifs. Le domaine d'application concerne la supervision en temps réel du réseau de distribution d'eau potable.

Références

Anicic, D., P. Fodor, S. Rudolph, et N. Stojanovic (2011). Ep-sparql : a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, pp. 635–644. ACM.

Arasu, A., S. Babu, et J. Widom (2004). Cql : A language for continuous queries over streams and relations. In *Database Programming Languages*, pp. 1–19. Springer.

Babcock, B., M. Datar, et R. Motwani (2002). Sampling from a moving window over streaming data. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 633–634. Society for Industrial and Applied Mathematics.

Barbieri, D. F., D. Braga, S. Ceri, et M. Grossniklaus (2010). An execution environment for c-sparql queries. In *Proceedings of the 13th International Conference on Extending Database Technology*, pp. 441–452. ACM.

Bolles, A., M. Grawunder, et J. Jacobi (2008). Streaming sparql-extending sparql to process data streams. In *The Semantic Web : Research and Applications : 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain*, Volume 5021, pp. 448. Springer.

Calbimonte, J.-P., O. Corcho, et A. J. Gray (2010). Enabling ontology-based access to streaming data sources. In *The Semantic Web–ISWC 2010*, pp. 96–111. Springer.

Cochran, W. G. (2007). *Sampling techniques*. John Wiley & Sons.

Golab, L. et M. T. Özsu (2003). Issues in data stream management. *ACM Sigmod Record 32*(2), 5–14.

Komazec, S., D. Cerri, et D. Fensel (2012). Sparkwave : continuous schema-enhanced pattern matching over rdf data streams. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pp. 58–68.

Le-Phuoc, D., M. Dao-Tran, J. X. Parreira, et M. Hauswirth (2011). A native and adaptive approach for unified processing of linked streams and linked data. In *The Semantic Web–ISWC 2011*, pp. 370–388. Springer.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS) 11*(1), 37–57.

Summary

Semantic Web technologies are increasingly adopted for data stream management. Several RDF stream processing systems have been proposed: C-SPARQL, CQELS, SPARQL, EP-SPARQL, SPARKWAVE, etc. These all extend the semantic query language SPARQL. Input data are large and continuously generated, thus the storage and processing of the entire stream become expensive and the reasoning is almost impossible. Therefore, the use of technology to reduce the load while keeping the semantics of the data is required to optimize treatments or reasoning. However, none of SPARQL's extensions include this feature. Thus, in this paper, we propose to extend C-SPARQL system to generate samples on the fly on graphs streams while keeping semantics. We add three sampling operators (UNIFORM, RESERVOIR and CHAIN) in C-SPARQL's syntax. These operators have been integrated in Esper, the C-SPARQL's data flow management module. Experiments show the performance of our extension in terms of execution time and preserving data semantics.