Projet ROSES

Programme MDCO – Edition 2007

Livrable no D4.1
Specification of strategies for the evaluation of queries in a P2P network of RSS streams

Identification

Acronyme du projet	ROSES (Really Open and Simple Web Syndication)
Numéro d'identification de l'acte attributif	ANR-07-MDCO-011-01
Coordonnateur	LIP6, Paris 6
Rédacteur (nom, téléphone, email)	YEH, 01 39 25 40 48, Laurent.Yeh@prism.uvsq.fr
No. et titre	D4.1 Spécification de Stratégies pour l'Evaluation de Requêtes dans un Réseau PàP de Flux RSS
Version	V0.1
Date de livraison prévue	Date / t0+10
Date de livraison	Février/ t0+12

Résumé

Ce document présente les spécifications de mécanismes d'exécution de requêtes sur des flux RSS dans un réseau Pair-à-Pair (PàP). Ces mécanismes s'appuient sur l'idée de mutualiser des requêtes ou des sous-parties de requêtes identiques se trouvant sur différents pairs du réseau. Chaque pair comporte un service de syndication de contenus PàP. Un utilisateur voulant exécuter une nouvelle requête la soumet à son pair. Elle est prise en charge par le service de syndication qui détermine si sur le réseau il existe déjà une requête avec laquelle la nouvelle requête peut s'apparier, c'est-à-dire partager des sous-requêtes.

Ce principe de mutualisation s'inscrit dans le paradigme des systèmes PàP où chaque pair est tantôt un serveur offrant son service d'exécution de requêtes aux autres pairs et tantôt un client profitant de résultats de requêtes déjà évaluées par d'autres pairs. Par la mutualisation, l'exécution en PàP évite que plusieurs pairs traitent les mêmes requêtes, augmentant ainsi les performances globales du réseau ; autrement dit, la syndication des requêtes permet d'économiser les ressources de calcul mais aussi d'accès disques. Le réseau assure un équilibrage de la charge au niveau des pairs pour une meilleure qualité de service globale. Le système PàP administre constamment les pairs qui vont et viennent sur le réseau afin d'éviter dans la mesure du possible la présence de pairs saturés ou

sous-chargés.

Pour mutualiser les requêtes, nous proposons dans ce document une extension du système de syndication centralisé proposé dans le cadre du projet RoSeS. Nous introduisons un réseau PàP adapté qui joue le rôle de lien entre les pairs. Nous définissons des stratégies d'exécution de requêtes efficaces sur ce réseau.

1. Introduction

Les flux RSS basés sur les concepts du Web sémantique standardisés par le W3C constituent une nouvelle approche pour la diffusion et le partage des informations sur le web. Cette approche s'appuie sur plusieurs normes telles que RSS 0.91 (Rich Site Summary), RSS 0.90 et 1.0 (RDF Site Summary), RSS 2.0 (Really Simple Syndication), et ATOM [2]. Cette technologie comble un manque pour l'échange des données dans le monde du web. En effet, elle est habituellement utilisée pour signaler les mises à jour des sites dont le contenu change fréquemment, typiquement les sites d'information ou les blogs. L'utilisateur peut s'abonner aux flux et ainsi être informé des dernières mises à jour sans avoir à se rendre sur le site. Un flux RSS est un document XML mis à disposition par un serveur Web. Un flux contient un ensemble d'éléments suivant une des normes susdites. Un flux est structuré comme une séquence d'éléments appelés "items". Chaque item contient une date de publication, un résumé et un lien vers une page Web. Le client s'abonne au flux via un lecteur de flux qui est intégré dans la plupart des navigateurs (IE, Firefox). Le lecteur de flux rafraîchit ensuite régulièrement le flux selon la période spécifiée par le client en réinterrogeant le serveur web qui met à disposition le flux.

Du point de vue utilisateur, la consommation de ces données pose de nouveaux problèmes. En effet, l'utilisateur ne peut ou ne souhaite lire en temps réel l'ensemble des données provenant de ces flux. Un utilisateur peut être rapidement submergé par les données provenant de nombreux flux. Il doit pouvoir se faire aider par des outils qui lui synthétisent l'information. Un exemple de requête sur les flux peut être : « Quels sont les articles traitant de la révolte en Guadeloupe durant les trois derniers jours ? ». Pour exprimer des requêtes temporelles sur flux, il est nécessaire d'utiliser un langage de requête plus puissant que ceux fournis dans les lecteurs de flux, par exemple intégrant une logique temporelle.

Les requêtes ayant la puissance des langages évolués (autre que les simples filtrages) sur flux XML peuvent nécessiter d'importantes ressources machines pour leurs exécutions. Ce problème de ressources est amplifié en raison de l'exécution répétitive d'une même requête (surveillance de l'arrivée de nouveaux items). Ainsi, tout lecteur de flux qui doit périodiquement consulter un site pour détecter les nouveaux items consomme d'importantes ressources (bande passante et accès disques). De plus, le nombre de sujets d'intérêts par utilisateur peut être important. Ces aspects montrent que le coût de requêtes sur flux peut être important.

Une coopération des différents nœuds dans un réseau PàP permet d'alléger la charge globale du réseau à l'aide de stratégies d'exécutions adéquates. Cet allègement se traduit par une meilleure utilisation des ressources du réseau. Outre ce bénéfice, la coopération des pairs dans un réseau PàP, à contrario des systèmes distribués, permet de profiter de propriétés bien connues en PàP comme :

- Une meilleure optimisation des ressources globales. Chaque pair contribue (dans son rôle serveur) aux autres en leur permettant de profiter des calculs qu'il a effectué; en contre partie, chaque pair (dans son rôle client) peut bénéficier des calculs effectués par d'autres pairs.

- Une meilleure fiabilité. Par exemple si un nœud en charge d'aspirer une source de données tombe en panne, un autre pair peut prendre le relais.

- Une administration décentralisée. La mise en œuvre d'un tel réseau ne nécessite pas d'administrateur pour par exemple connecter des sources. Sans administration décentralisée, l'évolution d'un réseau risque d'être difficile.

- Une entrée ou sortie de pairs sans contrainte. Les utilisateurs sont libres de se connecter au réseau PàP à tout moment. Ils peuvent aussi quitter le réseau quand ils le veulent. Les ressources du réseau peuvent ainsi évoluer et notamment s'accroître.

Différentes stratégies d'évaluations des requêtes distribuées peuvent être mise en œuvre dans un réseau PàP. Dans ce domaine, des recherches proposent [5][3][1] de créer un réseau où chaque pair exécute un opérateur algébrique et les pairs sont connectés en « pipeline ». Un arbre de pairs connectés entre eux permet d'exécuter un plan algébrique. Sur le principe, cette approche s'apparente à l'exécution de programmes dans des machines parallèles. Une des difficultés est la mise au point de stratégies pour la répartition des opérateurs algébriques d'un plan d'exécution sur les processeurs du réseau.

Dans le cadre du projet RoSeS, pour la partie PàP, nous explorons une approche d'exécution collaborative de requêtes sur les flux. Cette approche se différencie de l'approche citée ci-dessus par la mutualisation de requêtes entières ou de sous-parties de requêtes réparties sur différents pairs. Etant donné une requête issue d'un pair client, si un pair serveur exécute (ou à exécuté récemment) une requête identique, la stratégie d'exécution de la requête du coté pair client consiste à sous-traiter l'exécution de la requête au pair serveur. De même, si un pair serveur exécute (ou à exécuté récemment) une partie d'une requête, la stratégie d'exécution consiste à demander l'exécution de cette partie de la requête au pair serveur. Ces sous-traitances permettent de profiter d'exécutions déjà effectuées sur d'autres pairs serveurs pour le compte d'autres utilisateurs. Une telle mise en œuvre ne requière pour le pair serveur qu'un coût supplémentaire de transmission des résultats. Soulignons que dans notre approche, on ne distribue pas chaque opérateur algébrique d'un plan d'exécution sur un pair différent, mais la totalité ou un sous-arbre d'opérateurs algébriques d'un plan sur un pair donné.

La mise en œuvre de cette approche nécessite une extension du système de syndication centralisé du projet RoSeS avec un réseau PàP qui joue le rôle de lien entre les pairs, et la définition de stratégies d'exécution de requêtes. Cette mise en œuvre vise à obtenir les propriétés suivantes :

- *Une vision unifiée pour l'utilisateur.* Du point de vue de l'utilisateur, il ne connait que son système de syndication auquel il adresse ses requêtes. Le système PàP doit permettre de déléguer automatiquement et de manière transparente les requêtes des utilisateurs sur des pairs distants pertinents. Lorsqu'un utilisateur veut soumettre une requête, le réseau se charge de le mettre en relation avec un pair adéquat.

- *Eviter la saturation d'un pair.* En supposant qu'une même requête intéresse beaucoup d'utilisateurs sur un système donné, le pair serveur choisi risque d'avoir une charge trop importante pour envoyer le résultat à de nombreux clients. Cette charge importante risque d'entrainer une dégradation des performances globales du système. Dans notre approche, pour pallier à ce problème, la même requête peut être exécutée sur différents systèmes de syndication.

- *Un meilleur équilibrage de la charge du réseau.* Nous obtenons cet équilibrage par une

meilleure répartition des requêtes. A l'échelle d'une multitude de systèmes répartis sur le réseau, des utilisateurs peuvent partager les mêmes centres d'intérêts exprimés par les mêmes requêtes. Sans une vision collaborative, chaque système risque de traiter les mêmes requêtes indépendamment les unes des autres. La mise en commun du traitement des requêtes par un système PàP permet de réduire le nombre global de requêtes sur le réseau. De plus, il est possible de gagner en performance en mettant en place un partitionnement des flux traités par chaque pair sur le réseau. Par exemple, si deux requêtes s'intéressent respectivement au football et aux voitures, et si le réseau PàP permet d'attribuer à un site le traitement des flux concernant le football et à un autre site celui des flux en rapport avec les voitures, chaque système scrutera deux fois moins de flux.

Ainsi, la mutualisation nous permet de proposer une stratégie simple d'exécution de requêtes RSS en PàP et d'autre part d'étendre le système de syndication décrit dans les rapports des lots 1 à 3.

Ce rapport propose et spécifie des stratégies d'évaluation de requêtes dans un réseau constitué de pairs. Pour cela, ces stratégies s'appuient sur les hypothèses suivantes :

- Chaque nœud a des capacités d'évaluation de requêtes sur des flux RSS qui suivent en totalité ou partiellement l'algèbre décrite dans le lot 2 du projet RoSeS.
- Chaque nœud a une capacité de traitement en termes de bases de données dont certaines interfaces sont décrites dans le lot 3.

La suite du rapport est découpée en quatre parties. La section 2 décrit les éléments constituant une architecture de base pour l'exécution des requêtes en PàP. La section 3 présente les techniques de base pour la mutualisation des requêtes : cadre général de traitement des requêtes, équivalence et inclusion de requêtes, publication de métadonnées. La section 4 propose les spécifications de stratégies d'évaluations de requêtes avec mutualisation PàP sur des flux RSS. La section 5 est une brève conclusion.

2. Architecture PàP pour la Mutualisation des Requêtes

2.1 Architecture d'un pair

Cette section décrit les fonctionnalités complémentaires au système de syndication centralisé RoSeS pour un fonctionnement en PàP. Certaines fonctionnalités doivent être étendues, d'autres ajoutées. Cette spécification s'appuie sur la spécification du système de syndication présentée dans le rapport D1.1.

L'objectif recherché à travers cette extension est :

- de préserver l'ensemble des fonctionnalités d'un système centralisé,
- de rendre accessible les fonctionnalités du système centralisé à un client distant.

Déportation des commandes d'utilisateurs

Les abonnements des utilisateurs à des requêtes depuis un pair client sont transférés au système de syndication sur le pair serveur approprié. Ceci est réalisé grâce à un moniteur RPC qui se comporte comme un utilisateur système. Cet utilisateur privilégié peut utiliser les fonctionnalités du système centralisé sur le serveur sélectionné. La figure 1 illustre ces principes.

Le moniteur RPC d'événements est au centre du processus. Ce moniteur peut donc utiliser toutes les fonctionnalités du système de syndication centralisé défini dans le rapport D1.1. Cet utilisateur système fonctionne pour le compte d'utilisateurs distants. Cette approche permet d'effectuer une activation de service à distance par un RPC (Remote Procedure Call) et d'assurer la gestion des utilisateurs déportés. Suivant la figure 1, un utilisateur (représenté par le Pair Client) lance une souscription à une requête R ; celle-ci est envoyée vers un pair serveur via le « moniteur RPC ». Ce dernier active le gestionnaire de requête locale qui engage la souscription

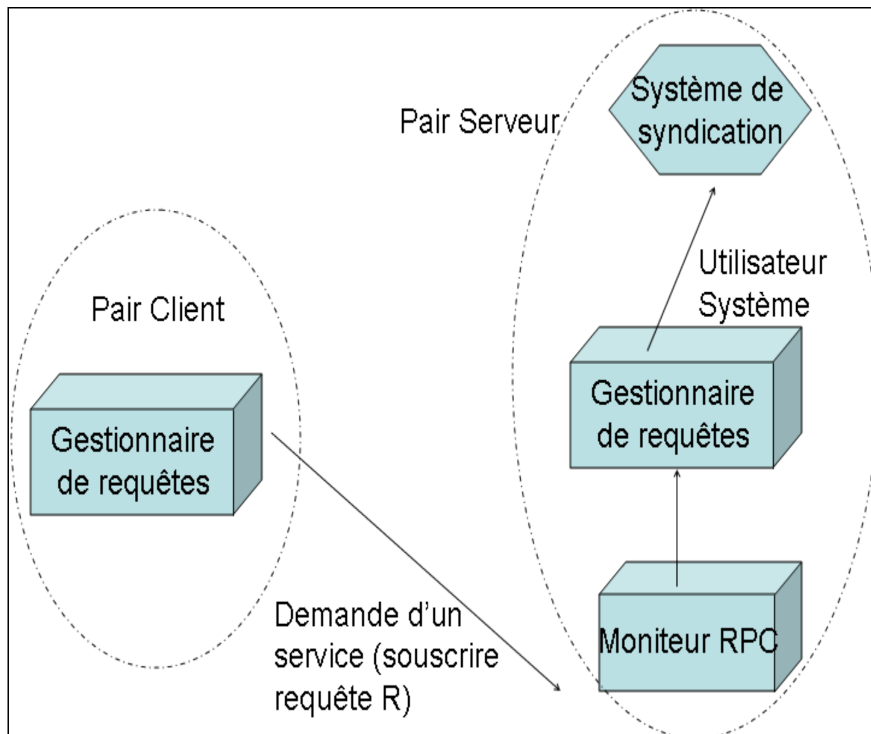

Figure 1 *: Principe de la déportation des commandes*

L'utilisateur distant recevra les résultats suivant deux stratégies possibles :

1. Lorsque le système de syndication évalue la requête, il renvoie directement le résultat aux utilisateurs via le module « diffusion » du système centralisé RoSeS (voir figure 2).

2. Le « diffuseur » du système RoSeS renvoie le résultat au pair distant. Le pair distant reçoit le flux pour être rediffusé à des utilisateurs abonnés ou pour être traité par d'autres processus en local (e.g., le résultat peut être utilisé par une autre requête). Du coté du pair client, la réception du flux utilise les mécanismes existant dans le système de syndication RoSeS pour la lecture de flux.

Ce principe est appliqué pour l'ensemble des services déportés d'un pair vers un autre pair. Les spécifications techniques du mécanisme de RPC basé sur XRPC/SOAP seront détaillées dans le livrable D1.3.

Fonctionnalités additionnelles d'un pair

L'extension de l'architecture est effectuée par couplage faible afin de préserver l'autonomie de chaque pair. Chaque pair assure le fonctionnement du système de syndication centralisé. L'extension consiste à ajouter des modules fonctionnels (en bas de la figure 2) et à les faire interagir avec les modules du système de syndication centralisé. L'interaction se fait en partie via les interfaces

utilisateurs standards (e.g., les souscriptions sous RoSeS, les diffusions, etc.). Pour certaines fonctionnalités, l'interaction se fait via des API spécifiques permettant d'accéder directement aux différents modules.

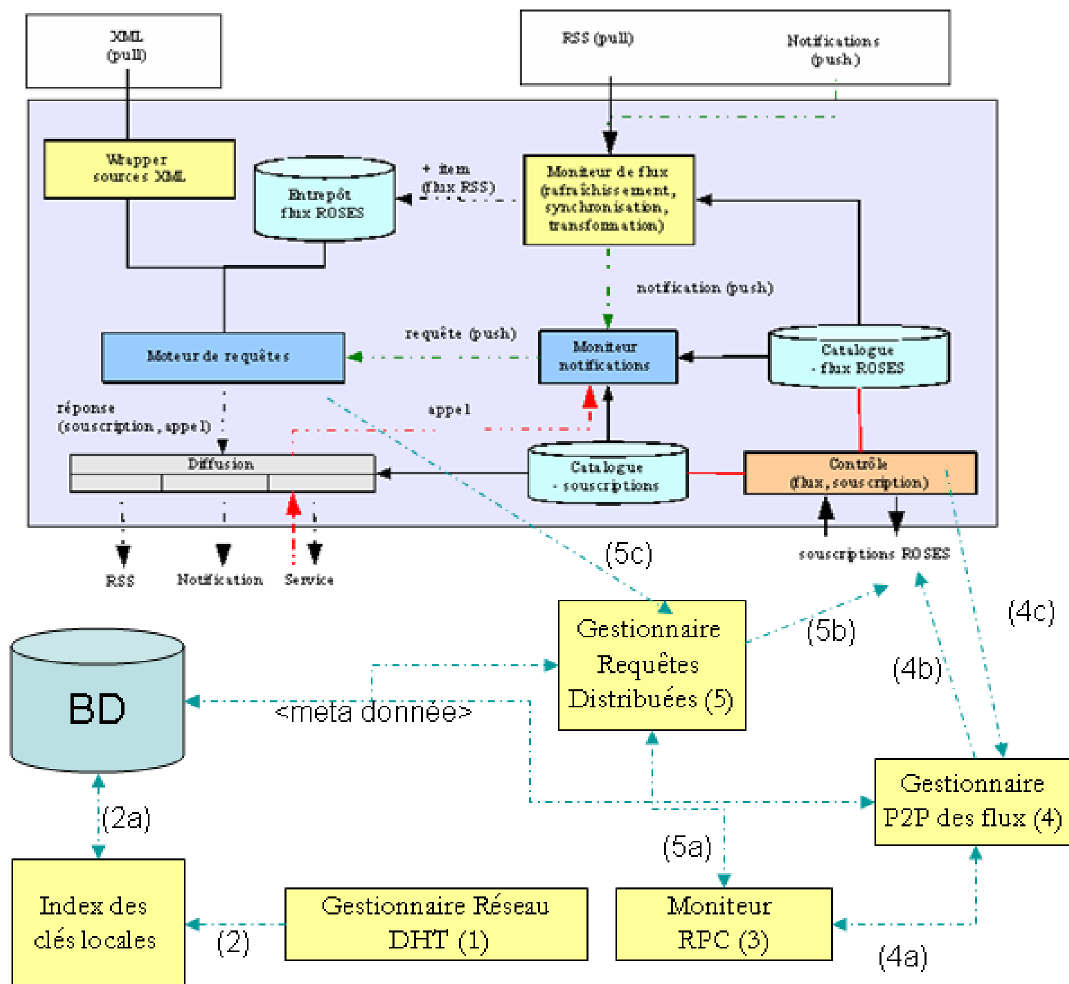

***Figure 2** : Extension de l'architecture du système de syndication*

Suivant la figure 2 et de bas en haut, nous trouvons :

- *Le gestionnaire de réseau (1).* Sa fonction est de gérer :
    - o le `put()` et `get()` d'un réseau DHT.
    - o le routage des messages de localisation (`lookup()`).
    - o l'indexation des clés du réseau (2). Tout pair peut publier des clés (métadonnées) qui peuvent être exploitées par d'autres pairs. Ces clés sont stockées dans une base de données locale (2a).
    - o les problèmes systèmes liés au réseau DHT (connections, etc.).
- *Le moniteur d'événements RPC (3).* Ce moniteur écoute les messages provenant du réseau Web. En fonction des types de messages, il les distribue vers les modules adéquats ou bien il effectue l'action demandée.

- Une liste non exhaustive d'actions est :
  - L'abonnement /désabonnement à un flux,
  - L'abonnement/désabonnement à une requête.

- *Le gestionnaire PàP des flux (4).* Son rôle est de publier sur le réseau PàP les flux qui sont gérée par le pair (4c). Il permet via l'interface (4a) de fournir des métadonnées à un pair distant. Il peut mettre en place un abonnement à un nouveau flux (4b) suivant une demande du « moniteur RPC ».

- *Le gestionnaire PàP des requêtes (5).* Son but est similaire au gestionnaire des flux. Il vise à publier sur le réseau PàP les métadonnées sur les plans d'exécution des requêtes stockés dans le système de syndication. Il peut par l'intermédiaire de l'interface (5a) recevoir une demande d'abonnement à une nouvelle requête (5b). Il peut être averti de l'existence d'une nouvelle requête qui peut être partagée sur le réseau (5c).

- *le système de syndication centralisé étendu* (en haut). Ce système de syndication est décrit dans le rapport D1.1.

Les spécifications techniques de cette extension seront décrites dans le rapport D1.3.

2.2 Architecture du réseau PàP

Les différentes stratégies d'évaluation des requêtes s'inscrivent dans le cadre d'un réseau PàP décrit dans cette section. La figure 3 représente un réseau de trois pairs. Chaque système comporte un portail permettant l'interaction avec les utilisateurs et administrateurs et un système de syndication. Il gère les couches du réseau PàP basé sur une DHT permettant en particulier les recherches de flux et de requêtes en indexant les métadonnées.

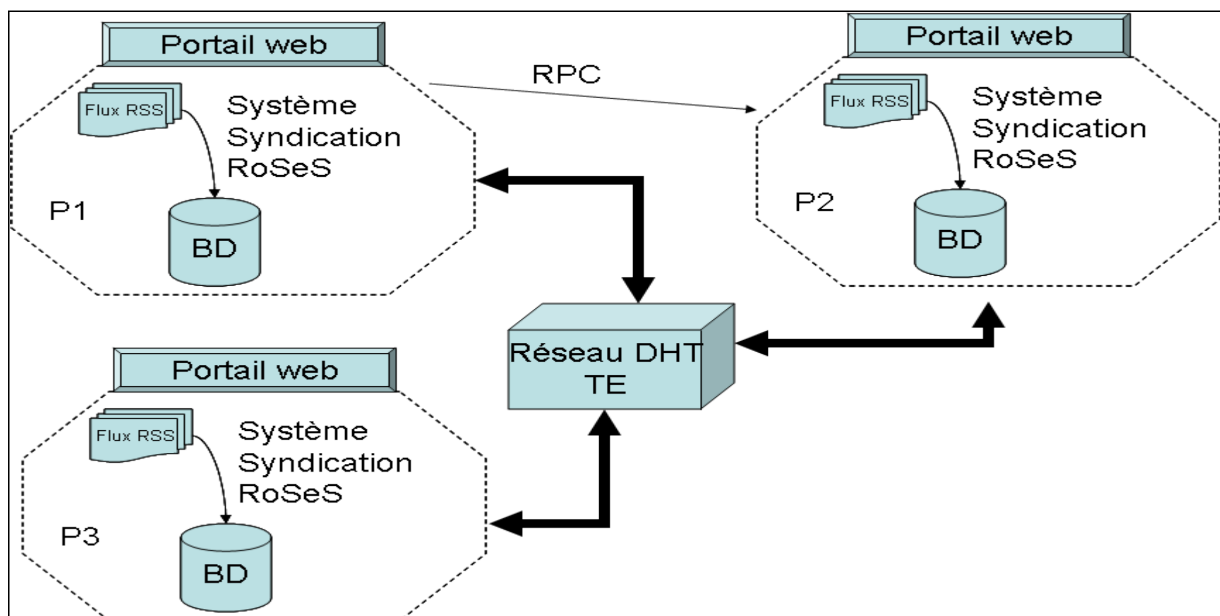

***Figure 3** : Architecture du réseau*

Plus précisément, l'architecture distribuée illustrée figure 3 est constituée :

- D'un réseau pair à pair au centre. Ce réseau est basé sur le principe des DHTs (Distributed Hash Table). Des métadonnées sont publiées dans le réseau (voir les spécifications dans la section suivante) afin qu'un pair puisse récupérer des informations pertinentes pour mettre en œuvre différentes stratégies d'exécutions de requêtes sur le réseau.
- De pairs (trois sur la figure). Chaque pair réalise :
    - les fonctionnalités de base comme aspirer périodiquement des flux RSS,
    - le stockage des flux aspirés dans un cache géré par un wrapper (voir rapport D3.3),
    - l'exécution de requêtes XQuery sur les flux RSS via un wrapper,
    - l'exécution répartie des requêtes décrite dans la section suivante du rapport,
    - l'accès aux données via un portail web.
- Le portail Web permet d'interagir avec le système de syndication. Il implémente une partie des fonctionnalités décrites dans le rapport D1.2. (fonctionnalités de bases comme l'abonnement d'un utilisateur à une requête). Par ailleurs, le portail doit implémenter des fonctionnalités nécessaires à la gestion du réseau et des pairs à distance (depuis un poste). Plus précisément, il permet de commander l'exécution d'un RPC à distance, par exemple, de demander d'exécuter une requête. Il permet aussi de visualiser le comportement des pairs du réseau. Par cette fonctionnalité, le portail permet notamment d'effectuer les mesures comparatives des différentes stratégies d'évaluation des requêtes.

2.3 Tour Eiffel (TE) : un exemple de réseau PàP

Tour Eiffel est un réseau PàP qui permet d'indexer et de router des clés constituées de vecteurs. Ce réseau a été développé au Laboratoire PRiSM dans le cadre de projets comme SemWeb [6]. D'un point de vue technique il répond aux fonctionnalités principales d'un réseau PàP. Certaines fonctionnalités comme la reprise sur panne ne sont cependant pas implémentées.

Une force de Tour Eiffel (TE) est de pouvoir indexer efficacement des vecteurs. La dimension des vecteurs est fixée pour l'ensemble du réseau. TE propose un réseau PàP combinant une structure d'arbre Kd (Kd Tree) combinée à des entrées directes via listes de saut (skip lists) pour indexer des vecteurs (n-uplets). Il est composé de deux niveaux qui collaborent pour router les requêtes vers les pairs contenant les données pertinentes. Une requête consiste typiquement à rechercher les vecteurs ayant une ou plusieurs coordonnées contraintes par une plage de valeur. Le premier niveau maintient la structure d'arbre obtenue par l'inclusion des descripteurs de zones ; lors des insertions, il assure l'éclatement des zones saturées selon les axes multidimensionnels constituant ainsi des hyper-rectangles ordonnés englobant les zones contenant les vecteurs. Le deuxième niveau généralise les « skip lists » de Chord afin d'accélérer le routage des requêtes par une recherche accélérée des voisins. Une description détaillée du réseau a été publiée dans [7]. Nous livrons dans ce document uniquement les éléments essentiels pour la compréhension de la technologie RoSeS.

TE offre des fonctions de base d'insertion (Put) et de recherche (Get), la clé étant donc un vecteur :

- `Put(<clé>, <valeur>)` permet d'insérer dans l'index distribué sur le réseau la clé

associée à la valeur.

- `Get(<clé>) ➟ { <valeur> }` retourne l'ensemble des valeurs correspondant à la clé `<clé>`.

Le réseau fonctionne à l'aide d'un processus système (au sens Unix) indépendant qui est intégré à chaque pair. Ce processus assure les fonctionnalités:

- *de routage* ; en fonction de la valeur de la clé, chaque pair peut rediriger un message de localisation vers un autre pair suivant un nombre de saut en `O(log(n))`, n étant le nombre de pairs présents sur le réseau.

- *d'indexation des valeurs* ; lorsque un pair (appelé **pair index**) reçoit un message avec une valeur à indexer (un pair client a effectué un `Put()`), le pair index stocke la valeur dans un index local ; cette valeur est restituée directement au pair client pour toute demande de `Get() ;`

- *de gestions de l'évolution du réseau* ; par exemple le pair peut demander une connexion ou bien un pair distant peut demander d'enlever les éléments indexés car il compte quitter le réseau.

3. Mutualisation des Requêtes en PàP

Le principe de la mutualisation développé dans cette section est le suivant : si un pair P calcule une requête sur un flux RSS, d'autres pairs voulant calculer la même requête seront adressés au pair P. L'exécution de la requête est entièrement confiée au système de syndication RoSeS. Le réseau PàP n'intervient pas lors de l'exécution, mais seulement lors du routage. Le rôle du réseau est donc d'apparier les requêtes entre elles.

La mise en œuvre de ce principe de fonctionnement nécessite plusieurs phases. Tout d'abord, un pair exécutant la requête R doit publier des métadonnées relatives à la requête R sur le réseau PàP afin de permettre aux autres pairs de la découvrir. La section 3.2 définit les métadonnées nécessaires et la section 3.3 précise le mécanisme de publication de ces métadonnées. En second lieu, un pair voulant exécuter une requête R doit : (i) localiser la requête si elle existe et a été publiée ; ce processus de localisation est défini dans la section 3.4 ; (ii) s'abonner auprès du pair qui a publié la requête afin de récupérer les résultats.

3.1 Traitement des Requêtes

La figure 4 présente une spécification par un diagramme UML du traitement des requêtes. Ce diagramme ne contient que les éléments nécessaires pour les spécifications des stratégies d'évaluations de requêtes présentées dans la suite. Une instance du gestionnaire de requêtes traite les requêtes qui sont présentes sur le réseau PàP. La classe `Requête` se spécialise en des requêtes déportées (`RequêteAppariée`) ou des requêtes non déportées (`RequêteNative`). Une requête appariée est exécutée sur un pair distant. Il peut s'agir de la requête entière (`RequêteAppariée`) ou d'une sous partie de la requête qui est exécutée à distance (`RequêtePartie`). Une requête native est constituée de sous-expressions algébriques décrites dans la section 3.2. Ces sous-expressions proviennent de la décomposition de la requête native. Chaque sous-expression peut potentiellement être appariée à une requête du réseau.

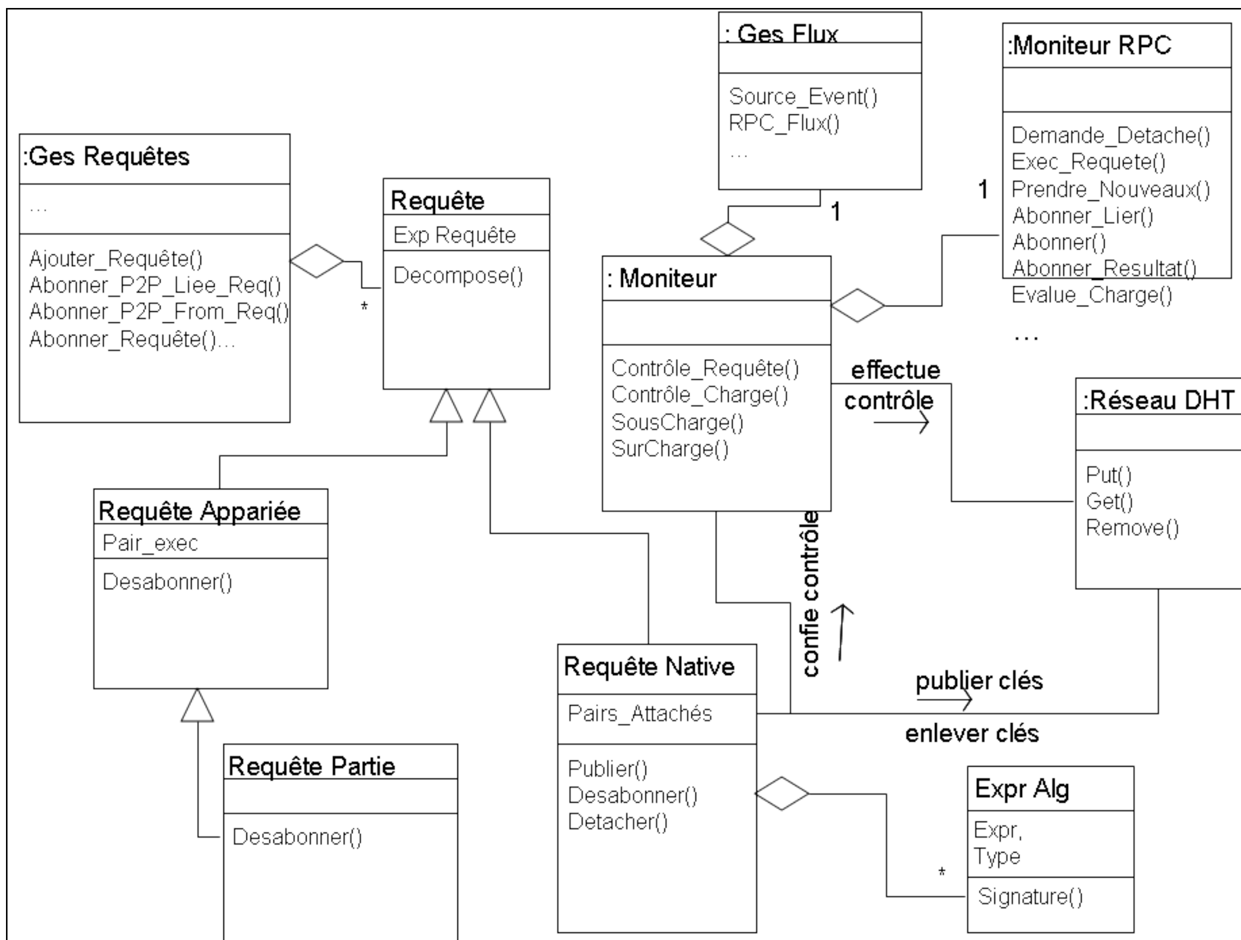

***Figure 4** : Diagramme UML de la gestion des requêtes*

Les instances et les classes de ce diagramme apparaissent sur chaque pair. Par exemple si une requête R d'un pair P1 est déportée sur un pair P2, le pair P1 aura une instance de la classe `RequêteAppariée` et le pair P2 aura une instance de `RequêteNative`. L'inscription d'une requête sur un pair distant met en œuvre le gestionnaire de requête du côté du pair client et le `MoniteurRPC` du côté du pair serveur distant.

Le moniteur est composé d'un gestionnaire de Flux (`GesFlux`) et d'un gestionnaire des demandes distantes (`MoniteurRPC`). Les moniteurs observent des événements et exécutent les méthodes correspondantes. Les événements peuvent être créés en interne par un pair ou bien provenir de pairs distants (par des échanges de messages). Les méthodes du moniteur sont activées suivant trois types d'événements :

1. Les événements provenant du réseau (Une demande de service provenant directement d'un pair distant (`MoniteurRPC`)).

2. Les événements provenant du système de syndication qui déclenchent certaines méthodes (Exemple `Source_Event()`).

3. Les événements provenant du moniteur (par exemple, l'événement qui déclenche la méthode `SurCharge()).` Cette méthode est appelée lorsque le pair n'a plus beaucoup de ressources systèmes à sa disposition. Cette méthode essaye de déporter des requêtes locales sur d'autres pairs.

Nous allons par la suite expliquer les stratégies d'évaluations de requêtes en nous appuyant sur ce

diagramme et sur les méthodes des classes qui seront précisées.

3.2 Equivalence et inclusion de requêtes

Le cœur du système de mutualisation est bâtit autour de la notion d'équivalence syntaxique de requêtes. L'idée est de normaliser les requêtes d'un point de vu syntaxique afin de pouvoir comparer l'égalité de requêtes. Une autre approche utilisant l'équivalence sémantique des requêtes est possible, mais requière des notions plus avancées. L'approche syntaxique ou sémantique n'impacte fondamentalement pas le principe d'évaluations de requêtes en PàP proposé dans ce rapport.

Traduction d'un langage évolué vers une expression algébrique

Afin de comparer les requêtes indépendamment du langage de requêtes, la comparaison de requêtes se base sur les expressions algébriques. Toute expression dans un langage évolué peut se ramener à une expression algébrique grâce aux nombreux travaux dans le domaine des bases de données, plus précisément sur la réécriture de requêtes. L'algèbre sur laquelle nous nous appuyons est celle définie dans le projet (voir livrable D2.2).

Un exemple de requête exprimé par une XQuery étendue est donné ci-dessous. Cette XQuery est basée sur une extension temporelle de [4] ; elle est exécutée sur un cache qui aspire les flux. La requête s'exprime comme suit:

```
For $i in FluxRSS(“http://www.equipe.fr/rss/une.xml",8)/channel/item
Where contains($i/title, ‘Foot’) and contains($i/title, ‘PSG’)
Return $i
Valid Last week;
```

Cette requête peut alors s'exprimer dans l'algèbre RoSeS par:

```
return /rss/channel/item (WindowCurrent last week Filter title contains "Foot" and title contains "PSG"
( FromRSS("http://www.lemonde.fr/rss/une.xml",8))).
```

Une requête un peu plus complexe est représentée figure 5.

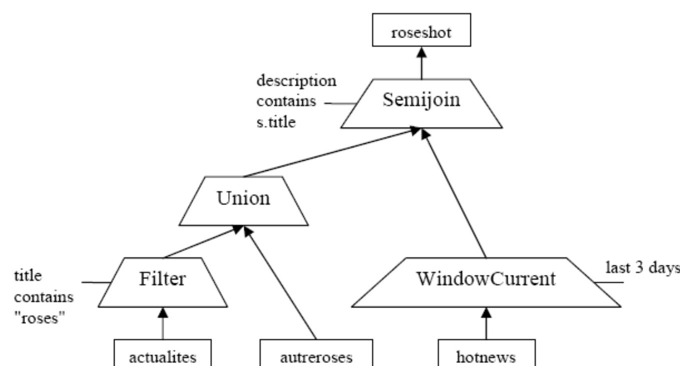

Figure 5 *: Exemple de requête exprimée par un diagramme algébrique*

Par la suite, les exemples seront basés sur l'exemple de la figure 5. Cet exemple de plan est traduit par l'expression exprimée dans l'algèbre RoSeS :

```
Semijoin description contains s.title(Union(
```

```
FromRSS("http://www.roses.fr/rss/une.xml", 8), Filter title contains "roses"(
FromRSS("http://www.lemonde.fr/rss/une.xml",10))), WindowCurrent last 3
days(FromRSS("http://www.roses.fr/rss/une.xml", 8)))
```

Normalisation d'une requête RoSeS

Afin de reconnaitre si deux plans sont identiques indépendamment de l'ordre des opérateurs, les plans sont normalisés. La normalisation concerne les opérateurs qui sont triés suivant un ensemble de critères. Ainsi, deux plans sont égaux s'ils ont la même structure syntaxique et les mêmes opérateurs. Le plan utilisé pour la normalisation est le plan non optimisé par une optimisation logique ou physique. C'est l'expression du plan donnée par un utilisateur ou traduit par un traducteur automatique depuis un langage de haut niveau.

Afin de normaliser un plan, chaque opérateur de la forme OP<arg1>[,…<argn>] (<param1> [, <param2>]) est ordonné au niveau du nom de l'opérateur, puis des paramètres et des arguments. Les arguments apparaissent comme indice de l'opérateur. Pour chaque paramètre qui est un opérateur, le traitement est appliqué récursivement. Le traitement se fait en profondeur d'abord. On applique les règles de transformation suivantes:

- *Règle 1* : si OP est un opérateur binaire commutatif, alors le tri s'applique aux paramètres || param1||<||param2||
- *Règle 2* : si <arg> est un ensemble d'arguments dont l'ordre n'importe pas, alors ces arguments sont triés suivant l'ordre alphabétique.
- *Règle 3* : pour FromRSS, le trie se fait en incluant le nom du flux. Exemple : || FromRSS("http://www.lemonde.fr/rss/une.xml"))|| < || FromRSS ("http://www.roses.fr/rss/une.xml"))||.

L'application de ces règles au plan ci-dessus donne :

```
Semijoin description contains s.title(Union(Filter title contains "roses"(
FromRSS("http://www.lemonde.fr/rss/une.xml",10)),
FromRSS("http://www.roses.fr/rss/une.xml", 8)), WindowCurrent last 3
days(FromRSS("http://www.roses.fr/rss/une.xml", 8)))
```

Figure 6 *: Requête normalisée*

Dans le cas présent, comme l'opérateur Union est commutatif, les opérateurs FromRSS et Filter sont inversés pour respecter l'ordre alphabétique.

Inclusion d'une requête dans une autre

En vue de trouver en PàP une requête pouvant être une partie d'une autre requête, nous allons définir le processus et les éléments nécessaires à la section 3.3 « Publication des méta-informations sur le réseau ». L'idée est de décomposer une requête en parties (ou sous-expressions algébriques). Si une des parties d'une requête est reconnue comme constituant d'une autre requête, elles peuvent être mutualisées.

La décomposition en parties vise à générer toutes les combinaisons de sous-expressions algébriques possibles d'une requête**.** Chaque sous-expression peut alors être utilisée comme clé sur le réseau

PàP pour pouvoir apparier des parties de requêtes. Pour cela, le processus extrait pour un opérateur du type $OP_{<arg1>[,…<argn>]}$ `(<param1> [, <param2>])` tous ses paramètres. En effet, les paramètres peuvent être à leur tour des expressions algébriques. Cette décomposition est récursive et aboutit à une séquence d'expressions algébriques. Par exemple, la figure 7 illustre les niveaux de décomposition et la figure 8 donne la décomposition en partie de la requête exemple.

- Niveau 1 :

```
1. Semijoin description contains s.title(Union(Filter title contains "roses"(
FromRSS("http://www.lemonde.fr/rss/une.xml",10)),
FromRSS("http://www.roses.fr/rss/une.xml", 8)), WindowCurrent last 3
days(FromRSS("http://www.roses.fr/rss/une.xml", 8))) }
```

- Niveau 2 :

```
2. Union(Filter title contains "roses"(
FromRSS("http://www.lemonde.fr/rss/une.xml",10)),
FromRSS("http://www.roses.fr/rss/une.xml", 8)),

3. WindowCurrent last 3 days(FromRSS("http://www.roses.fr/rss/une.xml",
8))
```

- Niveau 3 :

```
4. Filter title contains "roses"( FromRSS("http://www.lemonde.fr/rss/une.xml",10))

5. FromRSS("http://www.roses.fr/rss/une.xml", 8)

6. FromRSS("http://www.roses.fr/rss/une.xml", 8)
```

- Niveau 4 :

```
7. FromRSS("http://www.lemonde.fr/rss/une.xml",10))
```

Figure 7 *: Niveaux de décompositions d'une requête*

```
{ Semijoin description contains s.title(Union(Filter title contains "roses"(
FromRSS("http://www.lemonde.fr/rss/une.xml",10)),
FromRSS("http://www.roses.fr/rss/une.xml", 8)), WindowCurrent last 3
days(FromRSS("http://www.roses.fr/rss/une.xml", 8)));

Union(Filter title contains "roses"(
FromRSS("http://www.lemonde.fr/rss/une.xml",10)),
FromRSS("http://www.roses.fr/rss/une.xml", 8));

WindowCurrent last 3 days(FromRSS("http://www.roses.fr/rss/une.xml",
8));

Filter title contains "roses"( FromRSS("http://www.lemonde.fr/rss/une.xml",10));

FromRSS("http://www.roses.fr/rss/une.xml", 8);

FromRSS("http://www.roses.fr/rss/une.xml", 8);

FromRSS("http://www.lemonde.fr/rss/une.xml",10))

}
```

Figure 8 *: Les parties d'une requête*

Equivalence syntaxique de prédicats

Afin de pouvoir comparer si deux opérateurs ayant des prédicats sont équivalents, nous allons décrire les règles à appliquer sur les prédicats des opérateurs pour les normaliser. Ces règles s'appuient sur la théorie du calcul des prédicats. Cette normalisation des prédicat permet de comparer l'équivalence de deux prédicats d'un point de vue syntaxique. Par exemple, à l'aide de cette normalisation, les deux opérateurs suivants seront reconnus équivalents: *Filter* title contains "roses" and description contains "algèbre" (…) et *Filter* description contains "algèbre" and title contains "roses" (…)

Règle 0 : Normalisation d'un prédicat

Tout prédicat est un terme. Il peut se représenter par une fonction *f* qui est un symbole de fonction *n*-aire. Si $t_1, t_2, \ldots, t_n$ sont des termes, alors $f(t_1, t_2, \ldots, t_n)$ est également un terme.

- Si les termes d'une fonction sont commutatifs, alors les termes sont triés dans l'ordre alphabétique. Exemple : = b a équivalent à = a b (remarque, = a b signifie egal(a,b)).
- Si les termes d'une fonction ne sont pas commutatifs, alors les termes sont triés dans l'ordre alphabétique et la négation de la fonction est appliquée. Exemple < b a équivaut à (¬ <) a b soit (≥ a b)

Règle 1 : Normalisation d'une expression de prédicats

Dans le domaine du calcul des prédicats, toute expression de prédicat peut se ramener à des disjonctions de conjonctions. C'est la propriété de distributivité de l'intersection par rapport à l'union, à savoir : (P1 ∨ (P2 ∧P3)) ⇔ ((P1 ∨ P2) ∧ (P1 ∨ P3)).

Règle 2 : Normalisation d'une expression de disjonction

Comme l'opérateur de disjonction est commutatif, chaque prédicat de l'expression est trié suivant un ordre alphabétique. Par exemple : (P1 ∨ P2 ∨ P3) ⇔ (P2 ∨ P1∨ P3) si ||P2|| ≤ ||P1|| ≤ ||P3||. Ici l'opérateur || || considère le prédicat d'un point de vue chaine de caractères et ainsi l'ordre n'est autre que l'ordre alphabétique.

Règle 3 : Normalisation d'une expression de conjonction

Comme l'opérateur de conjonction est également commutatif, chaque grappe de prédicats (ceux dans une expression de disjonction) de l'expression est triée suivant un ordre alphabétique. Par exemple : ((P1 ∨ P2) ∧ (P1 ∨ P3)) ⇔ ((P1 ∨ P3) ∧ (P1 ∨ P2)) si ||(P1 ∨ P3)|| ≤ ||P1|| ≤ ||(P1 ∨ P2)||.

3.3 Publication des métadonnées sur le réseau PàP

Dans un réseau PàP, si un pair possédant des ressources (requêtes…) veut les partager avec d'autres, il doit publier les métadonnées associées à ces ressources sur le réseau. Ainsi, les autres pairs peuvent découvrir et atteindre les ressources à l'aide de ces métadonnées. Cette sous-section se place du point de vu du pair qui publie ses ressources (pair source), et traite de deux aspects :

1. Le premier concerne les méta-informations publiées.
2. Le second concerne l'équilibrage de charge des pairs. Par exemple, si un pair P possède une requête Q qui s'exécute pour beaucoup de clients, le pair P risque d'être surchargé et avoir

une qualité de service qui se détériore.

Critère de routage

Précisons tout d'abord les clés utilisées par le réseau PàP pour trouver une requête pertinente. La décomposition présentée ci-dessus (voir figure 7) nous permet de publier une requête et ses sous-expressions. Suivant la figure 4, une requête est décomposée par la méthode (`Requête->Decompose()`) en ses sous-expressions algébriques.

Traditionnellement, les DHT routent sur une simple clé mono-valeur. Dans notre cas, la clé peut être générée par une simple fonction de hachage RSA qui transforme une chaine de caractères (expression algébrique) en une valeur numérique (`ExpAlg->Signature()`). La valeur numérique (appelée **signature** de l'expression algébrique) peut alors être utilisée pour le routage dans le réseau PàP.

Publication d'une requête

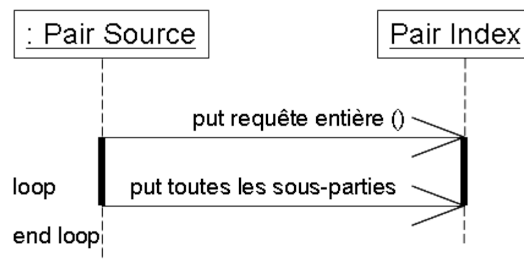

Figure 9 *: Diagramme de séquence pour la publication*

Pour une requête donnée (figure 9), le pair publie l'expression algébrique de la requête avec dans (`ExpAlg->Type`) une indication « requête entière ». L'ensemble de ces données est appelé par la suite une **signature**. C'est une métadonnée associée à la requête. Puis le pair publie chaque sous-partie de la requête avec un autre indicateur. Plus précisément, `ExpAlg->Type` indique soit qu'il s'agit d'un type source si l'expression se résume à `FluxRSS`, soit qu'il s'agit d'une sous-partie de la requête principale, soit qu'il s'agit d'une requête entière. L'objectif est qu'un pair client, lorsqu'il obtient une métadonnée concernant une expression algébrique, soit capable de déterminer le meilleur choix suivant les stratégies développées par la suite. Dans la publication, l'adresse IP du pair serveur de la requête est ajoutée afin qu'un pair client puisse s'adresser au pair serveur directement. D'autres méta-informations comme la charge du pair peuvent également être publiées pour qu'un pair client puisse estimer une qualité de services. Ce processus de publication doit être effectué également à chaque connexion d'un pair sur le réseau.

Contrôle des signatures

Un pair qui prend en charge une requête doit périodiquement vérifier sur le réseau l'existence des signatures concernant les expressions algébriques. Ainsi, les pannes des pairs index peuvent être détectées. Pour cela, chaque pair demande à son moniteur de contrôler le réseau périodiquement (`Moniteur->Contrôle_Requête()`). Au bout d'une certaine durée, le moniteur effectue une demande sur le réseau pour savoir si les signatures sont encore présentes. Si les signatures n'apparaissent plus, elles sont republiées.

Régulation de la charge

Un pair serveur doit pouvoir contrôler sur le réseau le nombre de pairs distants (pairs clients) avec qui il veut bien partager le résultat d'une requête. Si les ressources du pair serveur venait à manquer, le moniteur peut décider de faire baisser la charge du pair (`Moniteur->Contrôle_Charge()`). A l'inverse, si une requête est évaluée uniquement pour le compte de peu de clients, il peut être optimum de décider de confier la requête à un autre pair si il existe un autre pair calculant la même requête.

La fonction `Moniteur->Contrôle_Charge()` peut baser sa décision sur des mesures comme le temps CPU non consommé par unité de temps, la mémoire restante, etc. Son rôle est de maintenir un seuil de disponibilité des ressources fixé par l'utilisateur pour l'arriver de nouvelles demandes. Ainsi, le système de syndication doit toujours être disponible pour servir de nouvelles requêtes de clients.

Inspiré des automates d'autorégulation, la sélection des requêtes à confier à d'autres pairs peut se faire selon l'algorithme suivant:

```
//Phase 1
Pour chaque requête native r de Ordonner(requêtes natives):
  Si r sert plus de NRMax client :
    Si r ne possède pas de requête identique sur le réseau: continuer
    Enlever la signature de r du réseau PàP.
       Détacher Nb(r)-NRMax vers un ou plusieurs autres pairs suivant le principe ci-dessous.
  Sinon Enlever la signature de r du réseau PàP.
  Si seuil NR des ressources est atteint : termine l'algo
// Phase 2
Pour chaque requête native r de Ordonner(requêtes natives):
  Si r sert plus de NRMin client : continuer
    Si r ne possède pas de requête identique sur le réseau: continuer
    Enlever la signature de r du réseau PàP.
     Détacher l'ensemble des clients de r vers un ou plusieurs autres pairs suivant le principe ci-dessous.
    Si seuil NR des ressources est atteint : termine l'algo
```

Cette régulation dépend des paramètres `NR`, `NRMax` et `NRMin`. En fonction des évaluations effectuées dans le cadre du lot 4.5, cet algorithme et/ou ces paramètres pourront être affinés. Rappelons que l'objectif est de rendre le réseau auto régulé suivant le paradigme des réseaux PàP. Cette régulation permet d'éviter qu'il y ait des nœuds inopérants pour les autres à cause d'une certaine surcharge.

L'objectif de la fonction Ordonner(r) est de mettre en tête de liste les requêtes qui s'appliquent sur le moins de sources. En d'autres termes, il s'agit d'ordonner par une fonction pondérée suivant (i) le nombre de requêtes utilisant une source présente sur le pair (en d'autres termes, si beaucoup de requêtes en locales utilisent une source présente sur le pair, il vaut mieux conserver en priorité ces requêtes sur le pair), et (ii) la disponibilité d'une source identique sur le réseau (si une source est rare sur le réseau, il y a peu d'intérêt à envoyer la requête sur le réseau). Par exemple, pour le (i), si un flux est utilisé par beaucoup de requêtes en locale, alors la requête aura tendance à se retrouver

en fin de liste (la source est profitable pour beaucoup de requêtes, il vaut mieux exécuter la requête en local). A l'inverse, si un flux est utilisé par peu de requêtes, alors ces requêtes auront tendance à se retrouver en début de liste. Le principe de cette régulation est de minimiser le nombre de flux en locale en évacuant en priorité toutes les requêtes qui utilisent peu de flux.

Cas d'un pair surchargé.

Lorsqu'un pair exécute la requête pour trop de clients (seuil `NRMax` de l'algorithme ci-dessus), il peut effectuer deux types d'actions :

1. Le pair peut simplement enlever les signatures d'une requête R qu'il a en charge sur le réseau. L'enlèvement des signatures associées à la requête est l'opération inverse de la publication (`RequêteNative->Desabonner()`). Les pairs clients qui sont abonnés à la requête R, reçoivent toujours les résultats de la requête R, mais aucun nouveau client ne peut s'abonner à la requête R et obtenir ses résultats.

2. Un pair peut demander à un client de se connecter sur un autre pair. Pour cela, il doit dans un premier temps effectuer l'action (1) (enlever les signatures de la requête). Puis, il peut détacher un client (`Requête->Detacher(Client)`). En d'autres termes, il demande au client de s'abonner si possible auprès d'un autre pair. Cette action à pour effet d'envoyer sur le réseau un message à destination d'un client précis. Le client par l'intermédiaire de son Moniteur d'événements traite le détachement (`MoniteurRPC->Demande_Detache()`). Le détachement revient au pair client à refaire le processus d'abonnement comme décrit par la suite.

Les méthodes présentées ci-dessus sont également utilisées pour la déconnection d'un pair. Le pair doit appeler la méthode (`RequêteNative->Desabonner()`) suivie de la méthode (`RequêteNative->Detacher()`) pour que le ou les pairs clients (désigné par `RequêteNative->Pairs_Attachés`) puissent exécuter la requête sur d'autres pairs ou lui même.

Cas d'un pair sous-chargé.

Afin d'éviter des situations où il existe une multitude de pairs qui traitent chacun une requête pour lui-même et pour très peu de clients (seuil `NRMin`), le pair peut demander à un autre pair de prendre en charge l'évaluation de la requête. Cette situation peut arriver lorsque pour une actualité donnée (exemple coupe de football), beaucoup de clients partagent la même requête. Puis l'événement étant passé, beaucoup de clients se désabonnement de la requête. Beaucoup de pairs se retrouvent à traiter la requête pour peu de clients. Ainsi, transmettre les requêtes à un autre pair permet de profiter des ressources de l'autre et ainsi libérer des ressources locales pour accueillir de nouveaux traitements. Suivant l'algorithme ci-dessus, le transfert s'effectue que si il existe un autre pair sur le réseau qui traitent déjà la même requête (condition « si r ne possède pas de requête identique sur le réseau : continuer »).

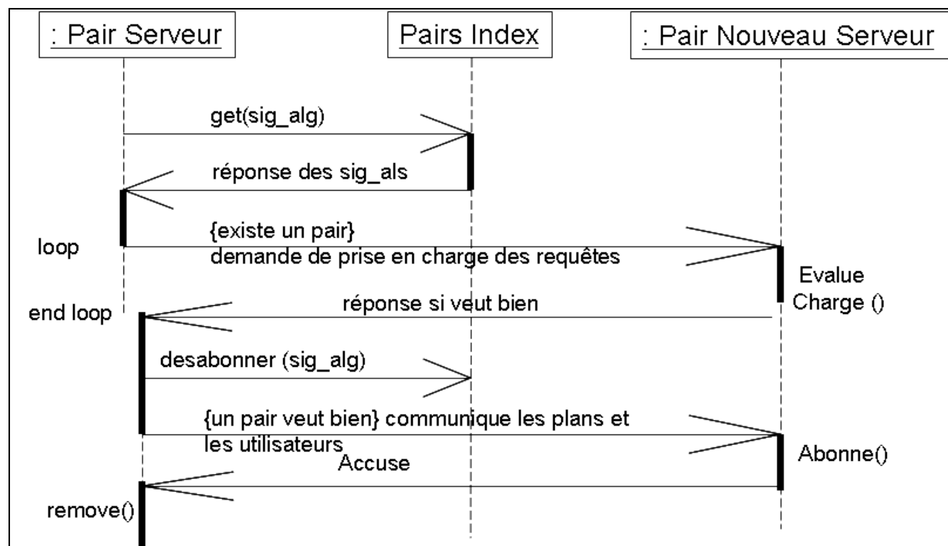

Figure 10 *: Diagramme de séquence pour le déport de requêtes*

Le diagramme de la figure 10 illustre le processus de négociation du pair qui veut déporter une requête avec un nouveau pair serveur dans le réseau PàP. Lorsque l'événement `Moniteur->sous_charge()` ou `Moniteur->Sur_Charge()`est détecté, le pair recherche sur le réseau les possibilités de mutualiser une requête donnée. Si il existe des pairs partageant la même requête ou une sous partie de la requête, alors une négociation s'effectue (`MoniteurRPC->Evalue Charge()`) avec un pair serveur distant pour savoir dans quelle mesure le pair peut prendre en charge le nouveau client. Dans ce processus, un certain nombre de pairs sont contactés pour déterminer la meilleure offre (`loop-end-loop`). Dans le cas ou il existe un pair pouvant prendre en charge la requête, le pair serveur enlève les signatures qui permettaient aux autres pairs de l'atteindre (`RequêteNative->Desabonner()`). Puis la requête est transmise au nouveau pair serveur suivant le procédé décrit ci-dessous.

Pour le transfert d'une requête d'un Pair Serveur Surchargé (PSS) à un Pair Nouveau Serveur (PNS) qui accepte de prendre en charge la requête, lorsque le moniteur du PSS (`PSS->Moniteur->SousCharge()`) envoie sur le réseau une requête de localisation pour une requête R, et qu'un des pairs (appelé PNS) qui traite la même requête répond favorablement, PNS doit verrouille la disponibilité et engage la méthode (`PNS->MoniteurRPC->Prendre_Nouveaux()`) pour dialoguer avec (`PSS->Moniteur->SousCharge()`). Il y a une simple transmission des requêtes au pair PNS. Chaque pair client qui s'attend à recevoir les données de PSS est contacté pour être notifié du changement. Les pairs clients reçoivent à présent de PNS les résultats. Cette information mise à jour sur chaque pair est utile lorsqu'un pair client décide de se déconnecter du réseau.

4 Localisation et Exécution des Requêtes

Pour une requête R donnée, l'objectif est de trouver sur le réseau soit une requête identique, soit une sous-expression de R.

De manière synthétique, une fois un pair serveur distant localisé, le moniteur PàP distant abonne l'utilisateur à la nouvelle requête via un utilisateur système (`MoniteurRPC`). Le gestionnaire de requêtes (`GesRequêtes`) du pair client gère la translation de l'utilisateur réel vers l'utilisateur système et vice versa. De plus, le pair client maintient les informations nécessaires (`Requête_Appariée`) pour la gestion de la requête exécutée sur le pair serveur distant. Sur le pair

serveur, la requête est représentée par une instance de (Requête_Native). Cette instance référence dans Requête_Native->Pairs_Attachés les pairs qui sont clients. Ces principes sont détaillés ci-dessous.

Pour la localisation d'une requête R pouvant s'apparier, trois cas sont distingués :

- Le premier concerne la recherche d'une requête identique. En effet, ce cas est traité en premier pour lui donner une haute priorité. Si une requête identique est trouvée, la localisation s'arrête.

- Le second concerne la localisation d'une requête qui est une sous-expression de la requête R. Si une telle requête est trouvée, la localisation s'arrête.

- Le troisième cas concerne les expressions algébriques de type FromRSS. Même si un pair aspire un flux qui est nécessaire à une requête, on peut décider de ne pas confier la requête au pair serveur distant, mais de l'exécuter sur le pair local avec l'aspiration du flux. En effet, dans le cas contraire, un pair qui aspire déjà un flux donné risque d'agglutiner toutes les requêtes utilisant ce flux au détriment d'une bonne répartition des requêtes sur le réseau.

Dans l'algorithme de localisation, ces trois cas se succèdent et l'on passe d'un cas au cas suivant que si le cas précédent a échoué.

4.1 Localisation d'une requête identique

Lorsqu'un utilisateur veut exécuter une nouvelle requête, il doit en premier lieu interroger le réseau pour connaître les pairs serveurs qui exécutent actuellement la même requête.

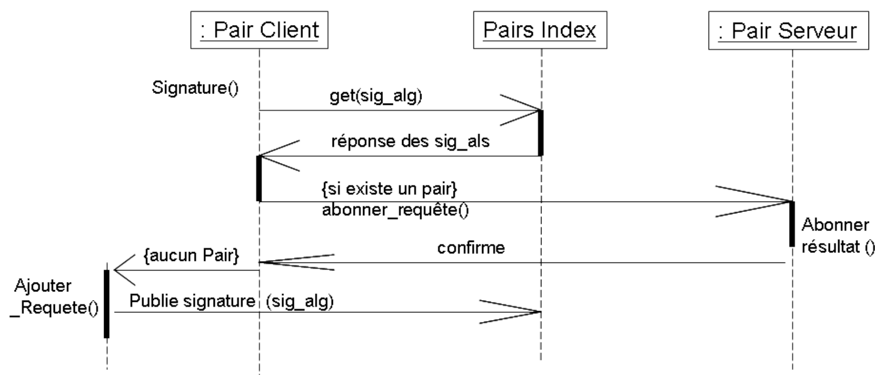

Figure 11 *: Diagramme de séquence pour l'abonnement d'une requête*

Pour cela, suivant la figure 11, le Pair Client (PC) doit d'abord extraire la *signature* de la requête. Si le réseau retourne une réponse négative, il doit alors prendre en charge la requête et effectuer la publication comme décrit ci-dessus (PC->Ajouter_Requête()). Dans le cas contraire, un des pairs choisis suivant une des stratégies décrites ci-dessous est contacté afin de prendre en charge la requête. Plus précisément, comme la requête existe sur le pair serveur, ceci revient à demander au système de syndication RoSeS d'ajouter un utilisateur supplémentaire qui doit recevoir les résultats de la requête.

Pour cela, suivant le diagramme UML de la figure 4 :

- Du coté du client, le client demande un abonnement à une requête (PC->GesRequête-

`>Abonner_Requête()`) au pair serveur. Le pair client crée une instance `RequêteAppariée` avec `Pair_Exec` référençant le pair à distance.

- Du coté du Pair Serveur (PS), le moniteur reçoit un message (`PS->MoniteurRPC->Abonner_Resultat()`) lui demandant d'inscrire un nouveau client. Le pair PS renvoie une confirmation.

Dans le cas où il y a plusieurs pairs pouvant calculer la même requête, plusieurs stratégies de choix du pair serveur peuvent être mise en œuvre, à savoir :

- **Stratégie L1**. Par un choix aléatoire d'un des pairs. Ceci peut constituer un référentiel pour comparer par rapport aux autres approches définies par la suite

- **Stratégie L2**. Par le choix du pair qui a le moins de clients partageant le résultat de la requête.

- **Stratégie L3**. Par le choix du pair qui a le plus de ressources disponibles. Il s'agit d'une fonction de coût qui intègre la capacité du cache disponible, du temps moyens d'exécution d'une requête, etc. C'est une fonction pondérée F=Ccache*Vcache+Creq*Vreq+…. où Ccache, Creq sont des coefficients de pondération et Vcache et Vreq sont des mesures du cache disponible et du temps moyen d'exécutions des requêtes.

Au final, si aucun pair n'a la même requête, alors le processus continue en essayer de localiser un pair qui peut exécuter une partie de la requête. Ceci est décrit dans la sous-section suivante.

4.2 Localisation d'une partie d'une requête

La localisation d'une requête pouvant être une partie d'une autre requête nécessite d'étudier plusieurs cas. Cette localisation est basée sur la publication des parties de requêtes (voir figure 7). La localisation d'une sous partie d'une requête s'effectue lorsqu'il n'est plus possible de trouver la requête entière. Le processus doit rechercher la partie commune la plus large afin de maximiser le nombre d'opérateurs algébriques exécutés sur le pair serveur distant. En d'autres termes, il s'agit de rechercher l'expression algébrique qui comporte le plus d'opérateurs.

Localisation d'une sous-requêtes

Ce processus consiste lors de la décomposition d'une requête (figure 7) à envoyer sur le réseau, pour chaque partie, une demande de localisation. Si une partie est déjà prise en charge par un pair sur le réseau, la méthode `get()` du réseau DHT retourne l'adresse IP du pair traitant la partie. Le choix du pair qui prendra en charge une partie de la requête est effectué lorsque toutes les réponses pour toutes les parties sont obtenues. Ainsi, le processus suit le même diagramme que celui présenté figure 11 mais appliqué sur toutes les parties de la requête (figure 8).

Choix du pair pour l'exécution de la requête

L'ordre dans lequel sont analysées les parties de la requête est important. En suivant l'ordre de décomposition illustré par la figure 7, l'algorithme privilégie les sites capables de résoudre la partie la plus large d'une requête. Ceci permet de profiter du calcul d'un maximum d'opérateurs sur un site distant.

Lorsque la localisation évalue deux parties (expression algébrique) équivalentes en termes de

nombre d'opérateurs, il est nécessaire d'interroger le réseau pour obtenir des paramètres comme la charge d'un pair, puis de prendre la décision suivant une fonction de coût. Par exemple, si une requête donnée nécessite l'évaluation des flux A, B et C, qu'un pair serveur distant peut évaluer les flux A et B et qu'un autre pair peut évaluer les flux C et A, quel pair choisir ? En effet, sans données supplémentaires, les deux pairs semblent offrir les mêmes possibilités. Pour cela, le processus doit interroger le réseau PàP pour obtenir les signatures et les coûts associés à toutes les parties possible d'une requête. Avec toutes ces données, le processus peut prendre sa décision suivant une fonction de coût. La fonction peut être identique à celles utilisées pour les stratégies L1 à L3 décrites ci-dessus.

Lorsque toutes les parties de la décomposition ont été explorées et qu'aucun pair ne partage une partie en commun de la requête, le pair doit alors se résoudre à prendre en charger la requête (`Ges_Requêtes->Ajouter_Requête()`). La requête devient une requête native (`Requête_Native`). Pour que d'autres puissent partager les résultats de la requête, elle sera publiée sur le réseau comme décrit ci-dessus.

Si un pair serveur distant est déterminé, le pair serveur et le pair client doivent alors se coordonner pour prendre en charge l'exécution déléguée de la nouvelle requête. L'exécution coordonnée s'articule autour du découpage de la requête en deux. Dans le contexte de l'algèbre, ce découpage revient à créer un résultat intermédiaire. A titre d'illustration, à partir de la requête de la figure 6 et en supposant que le traitement PàP a réussi à isoler la partie 2 de la figure 7, l'exécution de la requête est composée des deux parties suivantes :

- Première partie :

```
$r= Union(Filter title contains "roses"(
FromRSS("http://www.lemonde.fr/rss/une.xml",10)),
FromRSS("http://www.roses.fr/rss/une.xml", 8))
```

- Seconde partie:

```
Semijoin description contains s.title($r, WindowCurrent last 3
days(FromRSS("http://www.roses.fr/rss/une.xml", 8)))
```

La première partie de la requête correspond à la requête découverte sur un pair serveur distant. Pour la seconde partie, deux stratégies de répartition sont possibles :

- *Stratégie 1* : Elle consiste à confier la seconde partie de la requête au pair serveur distant. Pour cela, suivant la figure 4, le gestionnaire de requêtes (`GesRequêtes->Abonner_PàP_Liee_Req ()`) du pair client se charge de créer une nouvelle instance de `Requête_Apariée` avec `Pair_Exec` pointant sur le pair distant qui a en charge l'exécution.

  Du côté du pair serveur distant, `MoniteurRPC->Abonner_Lier()` effectue l'inscription de la nouvelle requête dans le système de syndication. Cette inscription revient à enregistrer pour un utilisateur distant une nouvelle requête avec comme particularité la liaison à une requête existante (`$r`) présente sur le système de syndication RoSeS. La méthode effectue également le nécessaire pour le renvoie du résultat à l'utilisateur ou au pair client.

- *Stratégie 2* : Une seconde stratégie consiste à demander au pair serveur distant d'envoyer le résultat de l'exécution de la première partie (`$r`) au pair client. Puis le pair client effectue la

seconde partie du traitement. Pour cela, le gestionnaire de requêtes (`GesRequêtes->Abonner_PàP_From_Req ()`) du pair client se charge de créer une nouvelle instance de `Requête_Partie` avec `Pair_Exec` pointant sur le pair serveur distant qui a en charge l'exécution. Une instance `Requête_Partie` hérite de toutes les propriétés d'une `Requête_Appariée`, mais le traitement du désabonnement est différent car il ne doit effacer que la seconde partie de la requête.

Du côté du pair serveur distant, celui-ci effectue un simple abonnement pour un utilisateur qui est le pair client (`MoniteurRPC->Abonner()`). La méthode indique le nécessaire au système de syndication pour le renvoi du résultat au pair client. Cet utilisateur reçoit les données calculées par la première partie de la requête via `GesFlux->RPC_Flux()`. La méthode enclenche les requêtes qui doivent être exécutées.

Chacune des deux stratégies présentent des avantages et inconvénients. En effet, la première surcharge le pair en exécution, alors que la seconde surcharge le réseau en communication. Notons que la seconde stratégie évite le processus couteux d'observation d'un flux pour détecter l'arrivé de nouveaux items. Un simple protocole de type RPC permet de lier le pair serveur au pair client de manière efficace. Des mesures viseront à comparer et qulifier les comportements des deux stratégies.

4.3 Localisation des sources pertinentes

Lorsqu'un pair P calcule une requête qui requière les items en provenance de n Flux (F1,…Fn) et qu'un ensemble de pairs possèdent qu'une partie des flux, plusieurs stratégies d'exécution de la requête sont possibles :

- *Stratégie 1* : la première stratégie consiste à placer la requête sur le pair serveur distant de sorte que la requête profite de l'aspiration des flux déjà effectués sur le site. Cette stratégie devient moins évidente lorsque plusieurs pairs serveurs possèdent plusieurs flux pertinents pour une requête donnée, par exemple, si une requête requière les flux F1, F2 et F3. En supposant qu'un pair P1 aspire déjà les flux F1 et F2 et que par ailleurs, un pair P2 aspire les flux F2 et F3, quel pair doit-on choisir ?

  Parmi les approches possibles pour déterminer le pair serveur distant qui va prendre en charge la requête, citons :

  - une approche qui consiste à choisir le pair qui a le plus de flux pouvant répondre à la requête. Si plusieurs choix sont possibles comme dans notre exemple ci-dessus, le pair peut être choisi de manière aléatoire.
  - une évaluation plus fine sur un ou plusieurs paramètres (exemple, la charge d'un pair, etc.) est possible. Le choix pourra être optimisé suivant des techniques d'analyses de données multi-paramètres (exemple, les Skylines).

- *Stratégie 2* : Une autre solution est de recevoir les nouveaux items des flux pour calculer la requête sur le pair client local. Cette stratégie s'apparente à la seconde stratégie pour le choix d'un pair pour une partie d'une requête (voir ci-dessus). Dans cette approche, le bénéfice pour le pair local est d'éviter le coût de détection de nouveaux items. Ce coût étant déjà payer par le site distant, le site local récupère simplement les nouveaux items suivant le principe du « push » via un mécanisme de type RPC. Il y a mutualisation pour la détection des nouveaux items dans le réseau PàP.

- *Stratégie 3* : Cette troisième stratégie crée en local l'aspiration du flux. Le pair à l'avantage de l'autonomie mais requière d'aspirer un flux supplémentaire.

4.5 Phase d'exécution des requêtes

Dans cette spécification du système PàP, l'exécution des requêtes est confiée au système de syndication RoSeS et les résultats sont directement renvoyés aux utilisateurs. Par conséquent, l'impact sur l'extension requise pour le PàP dans l'architecture d'un système de syndication centralisé est minimal.

Exécution d'une requête sur le pair serveur

L'exécution des requêtes est entièrement assurée par le système de syndication centralisé. Les clients reçoivent les résultats de requêtes. L'exécution des requêtes définies par les utilisateurs est déclenchée périodiquement en fonction de l'arrivé de nouveaux items en provenance des flux. Suivant la figure 2, ces déclanchement sont pris en charge par le « moniteur de notification » et les requêtes sont exécutées par le « moteur de requêtes ». Le réseau PàP n'intervient pas pour l'exécution des requêtes. Le système de syndication peut fonctionner en autonome.

Exécution d'une requête distribuée sur deux pairs

Dans ce cas, le pair qui a en charge la sous partie de la requête l'exécute normalement. Puis, le résultat est publié par le module « Diffusion » de la figure 2. Celui-ci crée un flux qui va être envoyé sur le pair distant. Le pair distant recevant le résultat via le module « Moniteur de flux » va lancer l'exécution de la seconde partie par le « moteur de requêtes ». Dans ce cas aussi, le réseau PàP n'intervient pas pour l'exécution des requêtes. Le système de syndication peut fonctionner en autonome.

5. Conclusion

Ce rapport décrit différentes stratégies d'exécution de requêtes RSS dans un réseau PàP. Ces stratégies sont basées sur le principe de la mutualisation d'exécution des requêtes. Cette approche par mutualisation permet :

1. d'économiser les ressources au niveau des pairs car les pairs partagent le traitement de parties de requêtes,
2. de réduire la charge globale de calculs et d'entrées-sorties nécessaires pour évaluer périodiquement une requête,
3. de mieux équilibrer la charge des traitements sur les différents systèmes.

Ces stratégies ont été décrites dans un cadre où chaque système de syndication centralisé est préservé en autonomie et en fonctionnalités. Le réseau PàP gère des métadonnées (signature des requêtes) dont le but est de permettre à chaque pair de localiser des requêtes identiques ou en partie identiques à une nouvelle requête utilisateur. Le réseau est un lien pour apparier les requêtes.

Ces techniques sont particulièrement pertinentes lorsque les temps de calculs de certaines sous-requêtes fréquentes sont importants. Ce peut être le cas dans le cadre de séries temporelles numériques, par exemple pour la recherche de tendance ou le groupage, plus généralement pour des

calculs statistiques. Ceci reste à étudier au vu des cas d'usage du système RoSeS.

Ce rapport sera suivi du livrable D1.3 qui décrit de manière plus détaillée l'architecture PàP mise en œuvre pour l'exécution de ces requêtes en mutualisation. Il doit être suivi d'une implémentation du réseau PàP dans le livrable D4.3 et par des mesures qui seront rapportées dans le livrable D4.5.

6. Bibliographie

[1] Arvind Arasu, and al, "Stream: The stanford data stream management system", Technical Report. Stanford InfoLab. 2004

[2] "Atom Enabled" http://www.atomenabled.org/

[3] "Borealis Distributed Stream Processing Engine" http://www.cs.brown.edu/research/borealis/public/

[4] Dengfeng Gao, Richard T. Snodgrass, "Temporal slicing in the evaluation of XML queries", In VLDB, 2003, 632-643

[5] Richard Kuntschke, Bernhard Stegmaier, Alfons Kemper, Angelika Reiser, "StreamGlobe: Processing and Sharing Data Streams in Grid-Based PàP Infrastructures", In proceeding of VLDB 2005,pp 1259-1262

[6] "Interrogation du Web Sémantique avec XQuery", projet ACI MD, http://bat710.univ-lyon1.fr/~semweb/, 2007

[7] Laurent Yeh, Georges Gardarin, Florin Dragan "Multidimensional vector routing in a p2p network". ICEIS 2007: 486-489