Routage distribué de flux RSS

Ngom Bassirou* — Sarr Idrissa** — Ndiaye Samba** — Gueye Modou**

* Département Mathématiques Informatique
Université Cheikh Anta Diop
Dakar
Senegal
bassirou83.ngom@ucad.edu.sn

** Département Mathématiques Informatique
Université Cheikh Anta Diop
Dakar
Senegal
prenom.nom@ucad.edu.sn

RÉSUMÉ. Actuellement le nombre de sources RSS est en très forte croissance. Les besoins des utilisateurs évoluent en conséquence. Ainsi, la charge des demandes d'actualisation de flux générée par les applications devient considérable et pas toujours nécessaire. Cela peut entraîner le ralentissement des systèmes ou induire des surcharges de communication. Les solutions existantes pour gérer les demandes d'actualisation de ces flux s'appuient sur des architectures centralisées ce qui ne favorise pas la scalabilité et la disponibilité des flux. Nous proposons une nouvelle solution pour réduire la charge soumise aux sources RSS. Le protocole s'appuie sur une diffusion ciblée des items à transmettre aux abonnés à travers un réseau structuré de pairs. Il permet aussi aux abonnés de faire des recherches sélectives basées sur des mots clés. Nous avons implémenté notre proposition avec le simulateur Peersim pour étudier sa faisabilité.

ABSTRACT. Currently the number of RSS feeds is very strong growth, therefore user needs evolve. Thus, the load of requests to update RSS feeds generated by applications is enormous and not always necessary. This can result to breakdown of the systems or leads to communication overload. Existing solutions to manage requests to RSS feed updates are based on centralized architectures, thus, do not support the scalability and RSS feed availability. We propose a new solution to reduce the load submitted to the RSS feed providers. The protocol is based on a targeted distribution of items to be transmitted to subscribers through a structured network of peers. It also allows subscribers to do selective keyword-based requests. Moreover, we implemented our proposal with Peersim simulator to study its feasibility.

MOTS-CLÉS : RSS, Routage, syndication de contenu, P2P, polling

KEYWORDS : Keywords

1. Introduction

Les utilisations importantes et diversifiées que l'on peut faire du web (*i.e.*, Customer to Bussiness, Business to Bussiness, ...) requièrent des mises à jour fréquentes du contenu. L'importance de la quantité des informations et leur caractère dynamique rendent impraticable toute forme de mises à jour manuelle faites par un administrateur de site web. C'est ainsi, que de nouvelles technologies ont vu le jour pour faciliter la collecte et la mise en ligne d'informations provenant de diverses sources. Ces technologies offrent la possibilité d'obtenir du contenu par aggrégation de différentes sources d'informations mais aussi de donner l'autorisation à d'autres sites web d'user des mêmes informations, on parle de syndication de contenu web. L'intérêt de la syndication de contenu est de fournir des informations précises, facilement accessibles et mises à jour régulièrement aux utilisateurs intéressés. La syndication web requiert des fournisseurs, d'une part, et des clients, d'autre part, qui s'abonnent aux flux (catégories d'informations) des premiers. L'échange entre fournisseurs (site d'actualités, blogs, ...) et clients peut se faire via des protocoles standards comme le RSS (*Really Simple Syndication*). RSS désigne une famille de formats XML utilisés pour la syndication de contenu web. Le mode de fonctionnement actuel des flux RSS repose sur la scrutation périodique des sources par les applications clientes. Concrètement, chaque application cliente doit envoyer périodiquement une requête HTTP pour visualiser un flux selon un certain timing. Les sites web qui proposent des flux doivent donc répondre à plusieurs requêtes d'utilisateurs durant la journée et ceci même s'il ne dispose pas de contenu nouveau. De manière détaillé, deux problèmes peuvent être soulignés dans le fonctionnement de RSS :

– RSS est un format essentiellement statique donc toutes les entrées sont envoyées simultanément chaque fois que la source est scrutée. Si un flux contient N dernières entrées récentes, ces N entrées sont transmises au client qui a envoyé la requête sans se soucier qu'elles soient nouvelles pour lui ou non ;

– les applications clientes RSS tourne sur les machines des utilisateurs durant toutes les heures de la journée même si l'utilisateur n'est pas présent ou ne désire pas lire les flux. Par conséquent, les clients génèrent une charge inutile et ne respecte pas le modèle interactif du web.

Ce faisant, la charge des demandes d'actualisation de flux générée par les applications devient considérable et pas toujours nécessaire. Cela peut entraîner le ralentissement des systèmes ou induire des surcharges de communication. En outre, les solutions existantes pour gérer les demandes d'actualisation de ces flux s'appuient sur des architectures centralisées, qui ne favorise pas le passage à l'échelle et la disponibilité des flux.

Dans ce papier, nous proposons une nouvelle approche de syndication de web basée sur une architecture distribuée. Notre objectif principal est de réduire d'abord la charge au niveau des fournisseurs de flux, puis de proposer un système flexible qui permettra de répondre aussi bien aux besoins des utilisateurs que des fournisseurs. Pour ce faire, nous nous appuyons sur les principes des systèmes pair-à-pair pour distribuer la charge des fournisseurs de flux. Notre approche se base sur un algorithme que nous appelons "polling between server" : le polling (scrutation) est contrôlé et reste limité au niveau des fournisseurs. A cela s'ajoute une collaboration des clients pour la répartition distribution des flux et une réplication des flux sur plusieurs fournisseur pour assurer la disponibilité du système. Pour valider notre approche, nous avons implémenté notre proposition avec le simulateur Peersim. Le reste de ce travail est structuré comme suit. La Section 1.1 présente l'architecture et les composants de notre solution. La Section 2 détaille notre

protocole de recherche et/ou de routage des flux alors que la Section 3 expose la scrutation entre serveurs. La Section 4 décrit le gestion de la disponibilité du système. La Section 5 présente les résultats obtenus avec notre simulation alors que la Section 6 relate les travaux connexes et la Section 7 conclut.

1.1. Architecture

Nous supposons que les flux sont valides selon la norme RSS ou Atom et que les types de flux sont limités et prédéfinis (ne dépassent pas vingt). Nous ne nous intéressons pas à la manière dont les flux sont créés. Nous supposerons aussi, que la panne d'un serveur web fournissant des flux RSS est rare, que chaque flux dispose d'un seul URL et que l'élément <link> de chaque entrée est renseigné.

L'architecture (*cf* Figure 1) est structurée autour d'un anneau logique et utilise des notions de groupe logique comme la plupart des réseaux P2P [15]. Il comporte trois types d'infrastructures : nous avons d'un coté les serveurs web qui fournissent les flux RSS que nous allons appeler par la suite RSS Peer (RP) et de l'autre côté les utilisateurs désirant consulter les flux qui seront dénommés ici par RSS Client (RC). Entre les deux, nous avons les gestionnaires qui sont des RP particuliers dont le rôle est de coordonner l'offre de flux par les RP et leur demande par les RC. Nous avons classé les flux selon leur type et les identifier chacun par l'url à partir duquel ils sont accessibles. Chaque flux est représenté alors par une entrée sous forme de couple (Title, Link). Title désigne le titre de l'entrée et Link le lien menant vers elle. Ainsi nous allons regrouper les RP qui diffusent le même type de flux en un groupe appelé Groupe RSS qui possède un identifiant GrID unique dans le réseau. Un groupe RSS est une structure logique qui regroupe les RP qui publient des flux d'un type donné et les clients intéressés par ce type de flux. Ce groupe est managé par un Gestionnaire de RP (GRP). Toutes les entités du groupe RSS disposent de l'identifiant GrId par l'intermédiaire duquel elles joignent le réseau.

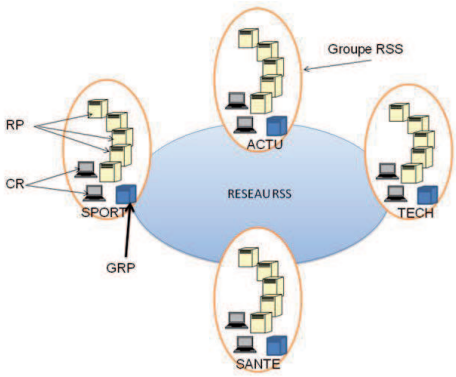

Figure 1. *Architecture*

1.2. Rôle des composants de l'architecture

Nous allons à présent définir le rpole des différents composants de notre système.

1.2.1. Les RSS Peers (RP)

Le RP est composé du serveur web physique désirant publier des flux RSS et d'un module intégré à ce dernier. Le module (application) permet au RP de connaître les autres RP du groupe en maintenant un catalogue. Chaque RP scrute les autres sources du groupe et cachent les dernières entrées diffusées par un serveur sur un flux donné mais garde une historique complète des entrées de ses propres flux. Le fait de stocker les mises à jour des autres dépend d'une politique globale qui sera fixée pour tout le système, par exemple on garde que les mises à jour datant de moins de 24 heures. On appellera ce paramètre *date de fraîcheur*. Pour éviter la scrutation inutile des sources ayant une faible fréquence de mise à jour, nous définirons une politique de scrutation dans la Section 3. Le fait de stocker les dernières entrées d'un flux augmente leur disponibilité et permet leur diffusion rapide. Le RP permettra alors de diffuser ses nouveaux items ainsi que ceux des autres RP du groupe aux clients intéressés. Pour ce faire, le RP maintient une table de diffusion pour chaque flux. Cette table contient les adresses des clients qui sont connectés sur lui et qui désirent recevoir les mises à jour du flux concerné. Le catalogue contient les adresses de tous les autres RP du groupe ainsi que les identifiants des flux fournis par ces derniers. Ainsi Le RP pourra scruter les autres RP pour récupérer les nouvelles entrées.

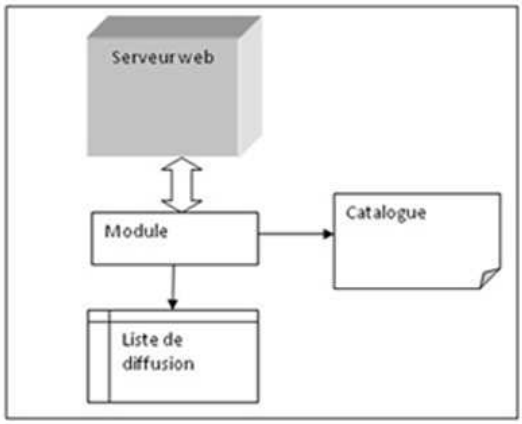

Figure 2. *Structure d'un RSSPeer (RP)*

1.2.2. Les GRP : gestionnaire des RSS Peers

Au niveau de chaque Groupe RSS nous avons un gestionnaire des RSS Peers (GRP) qui est un RP particulier. En plus des fonctionnalités d'un RP qu'il satisfait, il gère aussi le groupe en permettant l'insertion et la localisation des fournisseurs de flux. Le GRP permet la mise à jour des catalogues des autres RP : Tout RP qui joint le groupe est identifié par le gestionnaire et ce dernier, après avoir mis à jour son catalogue local, diffuse les informations relatives au nouveau RP dans le groupe. Grâce au catalogue, il permet de router les demandes des clients aux RPs et par conséquent connaît l'état du groupe à tout moment. A la différence des catalogues des autres RP celui du GRP contient en plus le nombre de clients connectés sur chaque RP ; ce paramètre sera mis à jour de deux manières : quand le GRP route un nouveau client vers un RP, il l'incrémente, chaque RP envoie aussi périodiquement le nombre de clients qui sont connectés sur lui. Il permet aussi l'authentification des nouveaux clients intéressés par le type de flux du groupe et

représente le groupe au niveau du réseau global reliant les groupes et permet à ses clients de visualiser des flux RSS d'un autre type.

1.2.3. Les Clients RSS (CR)

Les CR sont des pairs désirant consulter les flux dqui leur intéressent. Ils appartiennent à un groupe RSS contenant le type de flux désiré et reçoivent régulièrement les mises à jour de ces derniers. Pour ce faire, le client possède une liste contenant ses préférences (liens des flux, les entrées qu'il désire consulter, la date de publication de l'entrée voulue,). Le client pourra spécifier par exemple, lors de sa connexion, l'adresse du fournisseur du flux, la date de dernière mise à jour voulue, etc. En fournissant l'URL du flux, il est possible d'identifier le type de flux. Les CRs peuvent être intéressés par un autre type de flux, dans ce cas ils s'adressent au gestionnaire du groupe qui sera chargé de leur trouver le gestionnaire puis le fournisseur de ce dernier. Le client rejoint le réseau en envoyant une demande d'identification au gestionnaire qui lui permet de trouver un ou plusieurs fournisseurs. Nous remarquons le fait qu'un client peut appartenir à plusieurs groupes. Par ailleurs, le client qui rejoint le réseau possède le(s) type(s) de flux qui l'intéresse(ent). Il dispose alors les GrId des groupes dans lesquels il était connecté la dernière fois. Il dispose aussi d'une liste de mots clés correspondant à ses choix. Cette liste peut se présenter sous la forme suivante *(thème, fournisseur, item, date de publication)*, où *thème* désigne les types de flux voulu, *fournisseur* les fournisseurs de ces flux, *item* un ou plusieurs mots donnant des renseignements sur l'entrée voulue, *date de publication* permet de dire que le client est intéressé par les entrées publiées à partir de la date indiquée. Par exemple un client peut avoir la liste suivante : (news, lemonde, " Europe", " 02/12/08 8h "), (technologie, techcrunch, " web ", 01/12/08 8h). Chaque terme de la liste peut être renseigné ou non : si *fournisseur* n'est pas renseigné cela suppose que le client désire toutes les entées concernant le thème indiqué.

2. Algorithme de gestion et de diffusin des flux

Les spécifications sur les quelles s'appuie notre protocole de recherche ou de routage des flux sont :

– Un Client RSS ne souhaite pas recevoir les actualisations en continu mais seulement lorsqu'il est disponible pour les consulter, c'est-à-dire s'il est connecté au réseau RSS ;

– Un Client RSS souhaite recevoir seulement les actualisations les plus récentes (ex. celles produites il y a moins d'un jour) et correspondant à ses thèmes (liste de mots-clés) ;

– La demande provenant d'un Client RSS peut être traitée sans interroger la source c'est-à dire le RP.

2.1. Recherche de fournisseur

Le client envoie sa requête comprenant ses mots clés ainsi que son identifiant au gestionnaire. A la réception du message de type recherche de flux, le GRP vérifie d'abord le type du flux demandé et s'il correspond à celui du groupe il initie une recherche locale sinon il lance la recherche globale. Nous présentons respectivement les deux algorithmes dans les prochaines sections.

2.1.1. Recherche locale

Notre objectif est de trouver le fournisseur le moins chargé c'est-à-dire ayant le plus petit nombre de clients sur sa liste de diffusion et disposant des entrées recherchées. Le

GRP route les requêtes en se basant sur la date fournie par l'utilisateur et sur la charge des serveurs connecté. Le principe est très simple et se déroule comme suit :

– (a) le GRP qui reçoit la demande et qui constate que la recherche porte sur un flux ayant le type du groupe vérifie la date indiquée par le client ;

– (b) si la date est supérieure à la date de fraicheur, la requête est transmise au RP fournisseur du flux. Si ce dernier n'est pas disponible le client est avisé de l'indisponibilité du flux ;

– (c) sinon la requête est envoyée au RP ayant le moins de clients ;

– (d) le RP accuse réception du message ;

– (e) si l'accusé est reçu, le GRP incrémente le compteur du RP choisi et l'algorithme se termine ;s

– (f) sinon on reprend à partir de (c) ;

– (g) le RP choisit ajoute le client dans sa liste de diffusion.

Dans cet algorithme, nous remarquons que si la demande contient des termes trop contraignants (par exemple si la date indiquée est très ancienne) il y a de faible chance d'avoir une réponse car un seul fournisseur peut répondre, à l'occurrence le propriétaire du flux. Mais si l'entrée est récente, elle se trouve dans le cache des autres RPs et la requête sera traitée par le serveur le moins chargé.

2.1.2. Recherche globale

Pour fair la recherche au niveau gloable nous avons l'algorithme suivant :

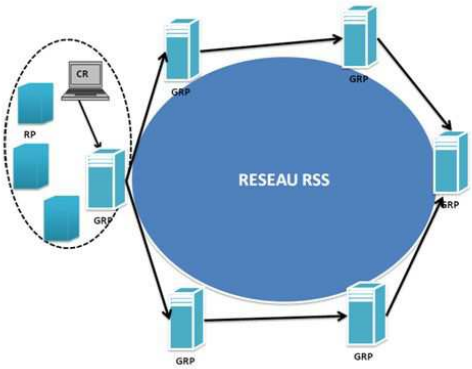

Figure 3. *Recherche d'un fournisseur*

– (a) le gestionnaire qui reçoit une requête de la part d'un client et qui s'aperçoit qu'il ne gère pas le type demandé le transmet à son successeur et son prédécesseur en y joignant un compteur C = (***N***/2) - 1, *N* estle nombre de groupes présents dans le réseau ;

– (b) chaque GRP qui reçoit une demande transmise par un GRP vérifie s'il peut la traiter ;

– (c) si oui, il initie une recherche locale et répond au client ;

– (d) sinon, il transmet la requête au gestionnaire du groupe qui le succède dans le sens de propagation de la requête en décrémentant C ;

– (e) la recherche se termine C = 0. Le noeud ayant le compteur nul avise le client de l'insatisfaction de sa demande.
Cet algorithme offre une bonne complexité en termes de messages car nous avons au pire des cas une réponse avec ((N/2) +1) messages en plus de la recherche locale. Les algorithmes que nous venons de décrire ainsi que notre architecture repose en grande partie sur les gestionnaires, une panne de ces derniers entrainerait alors le disfonctionnement partiel ou total du réseau. Donc il est nécessaire de trouver des mécanismes pour assurer leur disponibilité, c'est ce qui motive la section suivante.

3. Scrutation entre serveurs

Comme nous l'avons spécifié précédemment, notre approche se base sur la scrutation entre les serveurs fournisseurs de flux. Ceci dans le but de réduire la charge de ces derniers et de pouvoir contrôler le polling, véritable talon d'achille des flux RSS. Pour assurer le fonctionnement de la scrutation entre serveurs, nous utilisons une procédure classique connue au niveau des systèmes publication-abonnement qui est la notification. A chaque apparition d'une nouvelle entrée le RP envoie un message *notify* aux autres RP du groupe et ces derniers scrutent le flux. Avant de stocker les entées, le flux récupéré est filtré, seules les entrées apparues après la date de fraicheur sont retenues. Le fait de reprendre le flux entier permet d'avoir toutes les dernières mises à jours même celles qui étaient perdues pour une raison quelconque. Ainsi nous voyons que la fréquence de polling dépend de la fréquence des mises à jour. Un serveur ayant un fort taux de mise à jour se voit scruté plus fréquemment qu'un autre qui en a moins et par conséquent verra ses entrées plus disponibles dans le réseau. Ceci constitue un point fort pour la syndication de contenu car au lieu de surcharger les fournisseurs et de les amener à appliquer des politiques pour limiter la diffusion, et de nuire les principes du RSS, nous les poussons à fournir plus pour mieux bénéficier du système.

4. Maintenance des gestionnaires

Comme dans tous les réseaux P2P, la volatilité des pairs gestionnaires peut entraîner des indisponibilités du système. Pour prendre en compte cette volatitilité, nous avons répliqué le rôle du gestionnaire sur d'autre RPs qui peuvent devenir GRP secondaire (GRPS) en cas d'indisponibilité du GRP. En fait, tout GRP choisit parmi les RPs disponibles, celui le moins chargé qu'il nomme GRPS. Il envoie régulièrement l'état du réseau à ce dernier qui répond par un acquittement. Aini, si le GRP tombe en panne le GRPS peut prendre le relais sans perdre trop d'informations. Pour assurer qu'il y ait toujours un GRP disponible, la démarche suivante est adoptée :

– après une période ***r***, si le GRPS ne reçoit pas de rafraîchissement de la part du GRP, il peut se déclarer GRP en informant aux autres noeuds du groupe de son nouveau rôle (" INTENTION ") et la date du dernier rafraîchissement qu'il a reçu du GRP ;

– si un noeud reçoit cette " INTENTION ", il compare la date envoyée par le prétentieux nouveau GRP (jusqu'ici GRPS) et celle de sa dernière interaction avec le GRP suspecté en panne ;

– si la date de cette dernière interaction est supérieure à celle envoyée dans sa notification par le prétentieux nouveau GRP (ancien GRPS), le noeud renvoie un message au

prétentieux nouveau GRP (ancien GRPS) en lui donnant la date de sa dernière interaction avec l'ancien GRP ;

– si aucun message n'est envoyé au prétentieux nouveau GRP, cela veut dire qu'aucun noeud du groupe n'a pu contacter l'ancien GRP après la date donnée par le prétentieux nouveau GRP, et donc on peut supposer que cet ancien GRP est en panne. Par conséquent le prétentieux nouveau GRP envoie un message de confirmation (" CONFIRMATION ") aux autres noeuds du groupe qui, désormais, le considèrent comme GRP ;

– si toutefois, un message a été envoyé au prétentieux nouveau GRP, ce dernier diffère d'une période ***r***, sa tentative de devenir GRP. Si à la fin de cette période ***r***, il ne reçoit toujours pas de rafraîchissement, il réitère le processus décrit précédemment ;

– après ***k*** tentatives soldées par des échecs, nous concluons qu'il y a une panne de communication entre le GRPS et le GRP et donc le GRPS arrête d'émettre ses intentions et se considère comme un RP simple.
La complexité de cet algorithme est acceptable dans la mesure où nous avons au total (N+2) messages dans le pire des cas, c'est-à-dire le cas où tous les noeuds d'un groupe ont une interaction plus récente avec le GRP que le GRPS ; N étant le nombre de RP dans le groupe. En outre si le GRP ne reçoit plus d'acquittement venant du GRPS pendant une période ***r***, deux cas sont possibles : soit il y a panne de communication entre les deux entités, soit le GRPS est en panne. On procède de la même manière pour détecter la disparition du GRPS. Si le GRP reçoit une réponse lui notifiant que l'ancien GRPS est joignable, il diffère son intention de ***r***. S'il ne reçoit pas de message ou après k tentatives il élit un nouveau GRPS.

5. Expérimentation et validation

L'objet de cette section est d'évaluer l'approche que nous avons présentée dans les sections précédentes. Pour évaluer les performances de notre sytèeme, nous allons :

– montrer la rapidité de notre protocole comparé à la solution naïve où chaque fournisseur dispose ses propres flux qui sont scrutés régulièrement par les clients ;

– montrer le comportement de la charge du système dans le temps et de vérifier s'il y a bien un équilibrage de charge entre les serveurs ;

– prouver le passage à l'échelle de notre système.
Pour ce faire, nous présentons brièvement l'outil de simulation, Peersim, avant de détailler les différentes étapes de notre implémentation. Par la suite nous présentons et commentons les résultats obtenus.

Pour expérimenter et valider notre protocole, nous avons choisi Peersim [14], qui est un outil de simulation d'un environnement pair à pair. Peersim est un simulateur de système pair à pair Open Source développé par le département informatique de l'université de Bologne (Italie). Il a été conçu pour être très dynamique et très évolutif. Il permet de simuler jusqu'à plus d'un million de noeuds et supporte le caractère dynamique des noeuds d'un système P2P. Il est écrit en java et est téléchargeable sur le net. L'héritage et les fonctionnalités offertes par java lui confèrent son caractère modulaire et extensible. Peersim possède deux modes de simulations : la simulation par cycle et la simulation par évènement. Dans la simulation par cycles, tous les noeuds du réseau sont sélectionnés tour à tour et chacun d'eux exécute son comportement (protocole) à chaque cycle. La simulation par événement supporte la concurrence et permet à un noeud d'exécuter un

comportement suite à l'arrivée d'un événement (message) parmi un ensemble planifié. Dans la suite nous utiliserons la simulation par cycle. Ainsi, nous laisserons le système évoluer pendant un certain nombre de cycles pour étudier les résultats obtenus.

5.1. Apport de notre solution

Nous essayons d'abord de comparer notre protocole avec le fonctionnement actuel des flux RSS c'est-à-dire le fonctionnement centralisé. Pour ce faire nous exposons les deux systèmes dans les mêmes conditions : nous avons fixé le nombre de serveur à 80 et le nombre de client à 100, chacun émettant 4 requêtes par cycle. Nous observons le système pendant un certain nombre de cycles (30 cycles). Nous mesurons la variation de la charge des deux systèmes qui correspond au nombre total de requêtes en cours. Nous représentons les résultats obtenus dans la Figure 4. Nous remarquons que la charge des

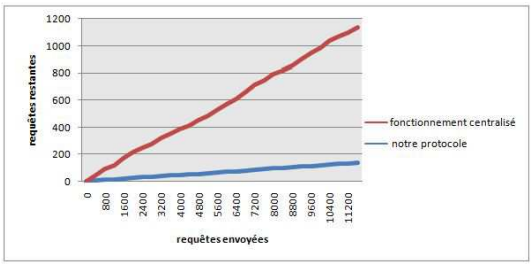

Figure 4. *Evaluation de la charge des serveurs RSS : nombre total de requêtes en cours au niveau des serveurs.*

serveurs est moins importante avec notre protocole qu'avec le fonctionnement centralisé des flux RSS. Ceci s'explique par le fait que si un serveur ne fournit pas de mises à jour récentes, il participe à la diffusion des derniers items des autres fournisseurs, ce qui réduit la charge moyenne des fournisseurs. Ce faisant, nous pouvons dire que la réduction de la charge au niveau des serveurs est assurée dans notre protocole. Par ailleurs on voit sur la Figure 4, que près de 200 requêtes restes dans le système sur un total de 12000 requêtes (400 requêtes par cycle au bout de 30 cycles) avec notre protocole. Nous constatons qu'en 30 cycles et dans les mêmes conditions expérimentales, notre proposition a répondu à prés de 98% des demandes contre 90% pour le régime actuel ; ainsi notre protocole offre un temps de réponse moyen très favorable. Ceci est une déduction logique vue que ce temps dépend en partie de la disponibilité des ressources.

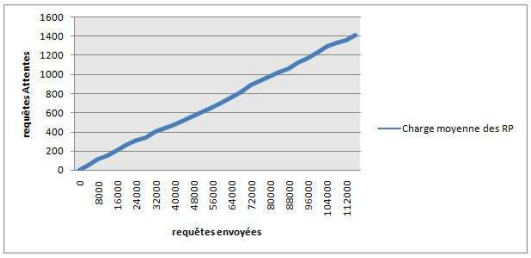

Figure 5. *Variation du nombre de requêtes en attente dans les serveurs en fonction du nombre de requêtes.*

5.2. Passage à l'échelle

Dans un second temps, nous essayons de montrer que notre approche favorise le passage à l'échelle. Ainsi nous soumettons une charge de 4000 requêtes par cycle (chacun des 100 clients émet 40 requêtes) et nous observons la variation de la charge au niveau des serveurs. Nous observons dans la Figure 5, que l'augmentationt progressive du nombre de requêtes dans les serveurs, ne dégrade pas les performances de notre protocole. En effet nous constatons qu'avec 16000 émises, le nombre de requêtes en attentes est de 210 ; ce qui correspond à un pourcentage de 1,3%. Le même résultat est obtenu avec 116000 requêtes émises. Ceci confirme la tendance de la Figure 4 où on avait 100 requêtes par cycles. Ce résultat prouve que le système parvient à fonctionner correctement avec un nombre important de noeud. Ainsi nous pouvons conclure qu'avec notre protocole l'augmentation du nombre de clients n'induit pas de goulôt d'étranglement, et donc favorise le passage à l'échelle.

5.3. Surcharge de notre système

Dans cette partie, nous observons le comportement de la charge des gestionnaires face à l'augmentation du nombre de client dans le réseau. Pour ce faire, nous avons réalisé une série de 40 simulations dans laquelle nous avons fait varier le nombre de client de 100 à 2000 et par des pas de 100, chaque simulation étant répétée deux fois de suite. A la fin nous avons calculé les moyennes et avons obtenu la courbe de la Figure 6.

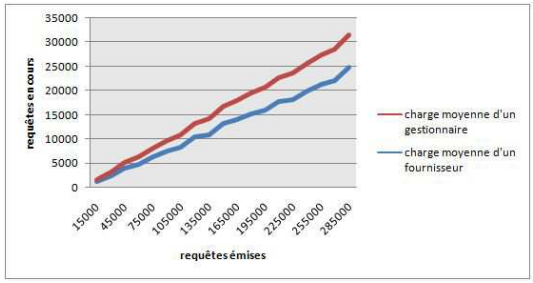

Figure 6. *Variation du nombre de requêtes en cours de traitement dans un noeud (GRP et RP) en fonction du nombre total de requêtes envoyées dans le système.*

Nous observons une légère augmentation de la charge des gestionnaires (courbe en trait rouge) par rapport à celle des fournisseurs (courbe en trait bleu). La charge des gestionnaires représente le nombre moyen de requête en cours sur les gestionnaires. En plus de sa charge en tant que fournisseur, le gestionnaire possède une charge supplémentaire qui correspond au nombre de requêtes en cours de routage. La charge du gestionnaire croît légèrement en fonction du nombre total de requêtes envoyé dans le système. Cette croissance s'explique par le fait qu'au moment où le gestionnaire route les requêtes vers le RP approprié, d'autres requêtes clientes arrivent et ceci en fonction du nombre de clients. Nous négligeons ici les messages inter-groupes c'est-à-dire les messages envoyés entre le gestionnaire et les fournisseurs. Comparé aux fournisseurs nous pouvons dire que la charge supplémentaire du gestionnaire n'est pas conséquente car le passage des requêtes au gestionnaire est éphémère. Nous déduisons de cette dernière expérience que les gestionnaires que nous avons mis en place ne constituent pas de sources de contention. Ce résultat confirme que notre système passe bien à l'échelle.

6. Travaux connexes

La gestion des flux commence à gagner du terrain à cause de l'intérêt majeur que présente la syndication web. Bien que les travaux dans ce domaine ne sont pas encore nombreux, nous avons essayé d'etudier les travuax connexes aux nôtres. Les solutions existantes les plus remarquables tentent de décentraliser la gestion des flux et de diminuer la surcharge au niveau des fournisseurs et du réseau. Pour ce faire, certains ont opté pour l'amélioration du polling en compressant les données à transmettre ou en utilisant d'autres moyens tels que les TTL ou les éléments <skipdate> et <skiphour>[7]. Malheureusement le caractère imprévisible de la production des flux RSS fait que cette approche n'a pas eu beaucoup de succès. D'autres ont tenté d'alléger la charge des serveurs en mettant les flux sur un serveur central dédié connu sous le nom "outsourced aggregation ". Ceci consiste à mettre les flux RSS sur un serveur secondaire qui va scruter le serveur principal et cachent les flux qui seront accédés par les utilisateurs finaux. Bien que cette technique réduise les problèmes de bandes passantes au niveau du serveur principal, elle présente des inconvénients car si le serveur secondaire tombe en panne le système devient inaccessible. Les solutions d'amélioration apportées sur le fonctionnement centralisé de RSS sont toutes confrontées d'une manière ou d'une autre à des problèmes de passage à l'échelle : l'utilisation des flux RSS devient de plus-en-plus importante et nécessite des outils plus scalable. C'est ce qui a motivé les chercheurs à trouver d'autres moyens de communication pour diffuser les flux. Par exemple, FeedTree [8, 9] ont été proposés dans l'optique de décentraliser la gestion des flux. FeedTree est un système de partage de flux en utilisant la technologie Pair-à-Pair. FeedTree est conçu en utilisant SCRIBE [2] qui est un système de communication par groupe basé sur PASTRY [10]. L'hypothèse de base de FeedTree est que tout client intéressé par un flux participe à un réseau overlay basé sur une table de hachage distribuée (DHT). Cet overlay est ensuite exploité pour trouver les autres pairs intéressés par le même flux. En d'autres termes un réseau sous-jacent de diffusion est construit au dessus du réseau basé sur la DHT. Un des avantages de FeedTree est qu'il n'a pas besoin d'une coopération entre les noeuds car ceci entraînerait des points de centralisation dans le système. Il n'existe pas de serveur central, un utilisateur arrivant sur le système n'a qu'à se connecter à n'importe quel noeud du réseau, et donc y faire partie. Ceci fait que FeedTree est un système qui s'adapte au passage à l'échelle et fournit des performances sur l'utilisation de la bande passante au niveau des fournisseurs de flux. Cependant, en dehors des problèmes hérités de la DHT tel que l'initialisation (les problèmes de bootstrap) et ceux des arbres de diffusion (fraîcheur des mises à jour), il reste toujours confronté à certaines difficultés inhérentes au bon fonctionnement des flux RSS tels la superfluidité des messages et le manque de contrôle au niveau des entrées "items " voulues.

FeedEx [11] est aussi un système conçu pour la distribution collaborative des flux RSS (ou Atom) qui repose sur les systèmes pair-à-pair. Il propose une solution qui se base sur le protocole XML-RPC [12] et utilise ainsi les fonctionnalités de ce dernier. A la différance de FeedTree, tout noeud dans FeedEx peut scruter les serveurs hébergeant les flux mais ceci suivant une politique établie par un programme de scrutation (Feed Fetch Scheduler). Chaque noeud dispose d'un nombre limité de voisins et maintient un ensemble d'abonnement (liste des flux auxquels il s'est abonné ainsi que ceux de ses voisins). La transmission des flux s'effectue en fonction de cet ensemble : un noeud ne reçoit et ne transmet que les flux contenus dans son ensemble d'abonnement. Le protocole proposé dans FeedEx fonctionne de manière similaire aux protocoles souvent utilisés dans les réseaux P2P. Au

début le noeud désirant intégrer le système essaie de trouver des noeuds qui sont déjà connectés. Ensuite après avoir trouvé les noeuds, il tente d'établir une connexion avec ces derniers. Puisque le nombre de voisins est limité dans FeedEx, une politique de sélection de voisin a été mise sur pied et permet à un noeud de choisir ses voisins en fonction d'une métrique définie. Afin d'équilibrer la charge entre les fournisseurs et les clients, le protocole définit un mécanisme de récupération adaptive. Ceci permet de planifier la fréquence de scrutation des flux pour répondre au dilemme traditionnel des flux RSS : avoir les mises à jour à temps sans surcharger les serveurs. Ainsi nous voyons que FeedEx a pallié certaines déficiences rencontrées dans les processus de distribution des flux RSS. Il permet une collaboration entre les noeuds qui cachent et diffusent les nouvelles entrées d'un flux. Il permet aussi de faire un contrôle sur les items en offrant la possibilité de faire des recherches suivant des mots clés : titre du flux, titre de l'entrée, date de publication, etc. Bien que FeedEx réduise la charge due au polling, nous constatons que la recherche d'un flux peut prendre plus de temps que nécessaire : car la demande est itérative (peut atteindre une complexité de O(n)) et n'assure pas toujours des réponses.

7. Conclusion

Dans ce papier, nous avons proposé un nouveau protocole pour le routage des flux RSS partant du fait que les solutions existantes ou proposées ne répondaient pas à certains besoins. Ce qui avait conduit certains fournisseurs à limiter leur flux ou appliquer des politiques de défense contre le polling agressif des utilisateurs. Nous avons tenté de répondre à ces problèmes en éliminant le polling des utilisateurs et en leur permettant de faire des recherches à bases de critères. Dans la suite, nous avons éavlué notre protocole pour le comparé avec ceux existant. Comparé aux solutions qui existent, notre approche semble fournir de meilleurs résultats en terme de messages envoyés : pour satisfaire une recherche, nous avons le nombre de message qui tourne autour de (N=2)+n+2, N étant le nombre de groupe et n le nombre de fournisseur dans le groupe choisi. Pour la gestion d'un groupe, l'algorithme de détection des pannes nécessite (n+2) messages. Le protocole a aussi permis la réduction de la charge des serveurs due au polling (seul un groupe réduit scrute suivant un planning déterminé). Il permet aussi la récupération d'un sous ensemble d'item d'un flux donné ce qui résout le problème de la superfluidité.

8. Bibliographie

[1] G. MÜHL, « Large-Scale Content-Based Publish/Subscribe Systems », *PhD thesis*, Darmstadt University of Technology, 2002.

[2] M. CASTRO, P. DRUSCHEL, A.-M. KERMARREC, A. ROWSTRON, « SCRIBE : A large-scale and decentralized application-level multicast infrastructure », *IEEE JSAC*, oct. 2002.

[3] Projet ROSES : http ://www-bd.lip6.fr/roses

[4] http ://www.rssboard.org/

[5] Atom. Atom Syndication Format. http ://www.- atomenabled.org/developers/syndication, Oct. 2005.

[6] R. Scoble. A theory on why RSS traffic is growing out of control.http ://radio.weblogs.com/0001011/2004/09/08.html#a8200, Sept. 2004.

[7] R. C. Morin. HowTo RSS Feed State. http ://www.kbcafe.com/rss/rssfeedstate.html, Sept. 2004.

[8] http ://www.feedtree.net/

[9] D.Sandler, A.Mislove, A. Post, P.Druschel. FeedTree : Sharing Web micronews with peer-to-peer event notification

[10] Antony Rowstron and Peter Druschel, "Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems," in Proc. IFIP/ACM Middleware 2001, Heidelberg, Germany, Nov. 2001.

[11] S. JUN & M. AHAMAD, FeedEx : Collaborative Exchange of News Feeds, Georgia Institute of Technology, may 2006.

[12] D. Winer. XML-RPC specification. http ://xmlrpc.com/spec

[13] Venugopalan Ramasubramanian, Ryan Peterson, and Emin Gun Sirer. Corona : A high performance publish-subscribe system for the world wide web. In In Proceedings of Networked System Design and Implementation, San Jose, California, May 2006.

[14] http ://sourceforge.net/projects/peersim/

[15] http ://gforge.inria.fr/projects/juxmem/